Partially Observable Concurrent Kleene Algebra

Jana Wagemaker
Radboud University Nijmegen, The Netherlands
j.wagemaker@cs.ru.nl

Paul Brunet
University College London, UK

Simon Docherty
University College London, UK

Tobias Kappé
University College London, UK

Jurriaan Rot
Radboud University Nijmegen, The Netherlands

Alexandra Silva
University College London, UK

Abstract

We introduce partially observable concurrent Kleene algebra (POCKA), an algebraic framework to reason about concurrent programs with variables as well as control structures, such as conditionals and loops, that depend on those variables. We illustrate the use of POCKA through concrete examples. We prove that POCKA is a sound and complete axiomatisation of a model of partial observations, and show the semantics passes an important check for sequential consistency.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Theory of computation → Concurrency; Theory of computation → Formal languages and automata theory

Keywords and phrases Concurrent Kleene algebra, Kleene algebra with tests, observations, axiomatisation, completeness, sequential consistency

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.20

Related Version A version with detailed proofs is available at `https://arxiv.org/abs/2007.07593`.

Funding ERC Starting Grant ProFoundNet (679127).
Paul Brunet: EPSRC grant IRIS (EP/R006865/1).
Simon Docherty: EPSRC grant (EP/S013008/1).

1 Introduction

Kleene Algebra (KA) was originally proposed as the algebra of regular languages [21, 4, 16, 13], but its well-developed meta-theory facilitates applications in the analysis and verification of sequential programs. Many extensions of KA were studied in the last decades, notably Kleene Algebra with Tests (KAT) [14], which enables reasoning about control structures such as if-statements and while-loops. Orthogonally, Concurrent Kleene Algebra (CKA) was proposed as an extension of KA to analyse concurrent program behaviour [9].

It is a natural question whether concurrent Kleene algebra can be extended with tests as in KAT. This question was studied by Jipsen [10] and later by Kappé et al. [11, 12], who proposed Concurrent Kleene Algebra with Observations (CKAO). Observations are tests in a concurrent setting, and they are governed by different axioms than tests, hence justifying their name change. It was illustrated that extending CKA with tests in a naive way results in an algebraic framework that is unusable in program verification. In a nutshell, the interactions of parallel threads are lost if we identify the conjunction of observations

with their sequential composition, as is done in KAT. Instead, an algebra where conjunction and sequential composition are kept distinct is essential to capture concurrent interaction between conditionals in different threads – this distinguishes tests from observations.

In this paper we demonstrate how this class of techniques can be used for a more fine-grained analysis of concurrent programs. We focus our development around the issue of *sequential consistency*, i.e., whether programs behave as if memory accesses taking place were interleaved and executed sequentially [17]. A standard way of testing this property is the so-called *store buffering litmus test* [1]. Consider the following program with two threads:

$$\begin{array}{lll} \mathsf{T0}\colon & x \leftarrow 1; & \Big\| \quad \mathsf{T1}\colon \quad y \leftarrow 1; \\ & r_0 \leftarrow y; & \Big\| \qquad\quad r_1 \leftarrow x; \end{array}$$

A sequentially consistent implementation should satisfy the following property: if initially both registers r_0 and r_1 are set to 0, after running the program one of them should have value 1. Therefore, we can detect failures of sequential consistency by observing behaviour that deviates from this specification. This test can be encoded algebraically [15] as:

$$\Big((r_0 = 0 \wedge r_1 = 0); (\mathsf{T0} \parallel \mathsf{T1}); \neg(r_0 = 1 \vee r_1 = 1)\Big) \equiv 0. \tag{$\dagger$}$$

That is, the program that asserts that r_0 and r_1 are both 0, executes $\mathsf{T0}$ and $\mathsf{T1}$ in parallel, and then asserts that neither r_0 nor r_1 is 1, is equivalent to the program 0, which has no valid behaviour. To reason in this fashion, our algebraic framework should include *observations* of the shape $v = n$ as well as *assignments* $v \leftarrow n$ and $v \leftarrow v'$, where v, v' and n range over some fixed sets of variables and values. To that end, we propose *Partially Observable Concurrent Kleene Algebra* (*POCKA*), an algebraic theory built on top of CKA that allows for an analysis of concurrent programs manipulating memory, such as the simple program above. POCKA has a natural interpretation in terms of pomset languages over assignments and memory states, encoded as partial functions, similarly to separation logic [20], which describe the behaviour of concurrent programs that can access variables and values (Section 3). We prove soundness and completeness with respect to this interpretation (Section 4).

POCKA deviates from KAT and CKAO by using *partial* observations in its semantics. These are crucial in a concurrent setting, where a single thread may have only a partial view of the memory. Whilst memory as a whole depends on the combined action of all threads, these partial views may be analysed on a thread-by-thread basis. This shift from total to partial observations thus allows for a richer compositional semantic model. Formally, this means that we move from a Boolean algebra of observations, as in CKAO or KAT, to a pseudocomplemented distributive lattice (PCDL) [3], as proposed by Jipsen and Moshier [10].

To ensure compositionality, semantics of concurrent programs should capture not only isolated program behaviour, but also all *possible* behaviours of the program when run in parallel with another program. For example, take the program $P = (x = 1); (x = 2)$, which asserts that x has value 1 and then value 2, and the program $Q = (x \leftarrow 2)$, which assigns the value 2 to x. In an interpretation that captures isolated program behaviour, the semantics of P would be empty, as x cannot change between the tests. In contrast, the program $P \parallel Q$ (i.e., P and Q in parallel) *does* have behaviour, because the assignment may be interleaved between the two observations. Hence, the isolated semantics of P is not sufficient.

Thus, the semantics of a POCKA term accommodates possible interference by an outside context. As a result, the test ($\dagger$) fails at this stage, meaning this semantics is not sequentially consistent. This raises the question of how to study the isolated program behaviour. To this end, we identify a subset of the semantics that captures isolated program behaviour, and show that this fragment coincides with guarded pomsets [10] (Section 5). This turns out to fix the defect in sequential consistency we observe earlier, as we show in Section 6.

2 Preliminaries

Throughout this section we fix a finite alphabet Σ. We recall pomsets [7, 8], a generalisation of words that model concurrent traces. First, a *labelled poset* over Σ is a tuple $\mathbf{u} = \langle S_{\mathbf{u}}, \leq_{\mathbf{u}}, \lambda_{\mathbf{u}} \rangle$, where $S_{\mathbf{u}}$ is a finite set (the *carrier* of $\mathbf{u}$), $\leq_{\mathbf{u}}$ is a partial order on $S_{\mathbf{u}}$ (the *order* of $\mathbf{u}$), and $\lambda_{\mathbf{u}} \colon S_{\mathbf{u}} \to \Sigma$ is a function (the *labelling* of $\mathbf{u}$). Pomsets are labelled posets up to isomorphism:

▶ **Definition 2.1** (Poset isomorphism, pomset). *Let $\mathbf{u}, \mathbf{v}$ be labelled posets over Σ. We say $\mathbf{u}$ is* isomorphic *to $\mathbf{v}$, denoted $\mathbf{u} \cong \mathbf{v}$, if there exists a bijection $h \colon S_{\mathbf{u}} \to S_{\mathbf{v}}$ that preserves labels, and preserves and reflects ordering. More precisely, we require that $\lambda_{\mathbf{v}} \circ h = \lambda_{\mathbf{u}}$, and $s \leq_{\mathbf{u}} s'$ if and only if $h(s) \leq_{\mathbf{v}} h(s')$. A* pomset *over Σ is an isomorphism class of labelled posets over Σ, i.e., the class $[\mathbf{v}] = \{\mathbf{u} \mid \mathbf{u} \cong \mathbf{v}\}$ for some labelled poset $\mathbf{v}$.*

When two pomsets are in scope, we tacitly assume that they are represented by labelled posets with disjoint carriers. We write $\mathsf{Pom}(\Sigma)$ for the set of pomsets over Σ, and 1 for the empty pomset. When $\mathsf{a} \in \Sigma$, we write a for the pomset represented by the labelled poset whose sole element is labelled by a. Pomsets can be composed in sequence and in parallel:

▶ **Definition 2.2** (Pomset composition). *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$ be pomsets over Σ.*

We write $U \parallel V$ for the parallel composition *of U and V, which is the pomset over Σ represented by the labelled poset $\mathbf{u} \parallel \mathbf{v}$, where $S_{\mathbf{u}\parallel\mathbf{v}} = S_{\mathbf{u}} \cup S_{\mathbf{v}}$, $\leq_{\mathbf{u}\parallel\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}}$ and for $x \in S_{\mathbf{u}}$ we have $\lambda_{\mathbf{u}\parallel\mathbf{v}}(x) = \lambda_{\mathbf{u}}(x)$ and for $x \in S_{\mathbf{v}}$ we let $\lambda_{\mathbf{u}\parallel\mathbf{v}}(x) = \lambda_{\mathbf{v}}(x)$.*

We write $U \cdot V$ for the sequential composition *of U and V, that is, the pomset represented by the labelled poset $\mathbf{u} \cdot \mathbf{v}$, where $S_{\mathbf{u}\cdot\mathbf{v}} = S_{\mathbf{u}\parallel\mathbf{v}}$, $\leq_{\mathbf{u}\cdot\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}} \cup (S_{\mathbf{u}} \times S_{\mathbf{v}})$ and $\lambda_{\mathbf{u}\cdot\mathbf{v}} = \lambda_{\mathbf{u}\parallel\mathbf{v}}$.*

The pomsets that we use can be built using sequential and parallel composition.

▶ **Definition 2.3** (Series-parallel pomsets). *The set of* series-parallel pomsets (sp-pomsets) *over Σ, denoted $\mathsf{SP}(\Sigma)$, is the smallest subset of $\mathsf{Pom}(\Sigma)$ such that $1 \in \mathsf{SP}(\Sigma)$ and $\mathsf{a} \in \mathsf{SP}(\Sigma)$ for every $\mathsf{a} \in \Sigma$, and is furthermore closed under parallel and sequential composition.*

One way of comparing pomsets is to see whether they have the same events and labels, except that one is "more sequential" in the sense that more events are ordered. This is captured by the notion of *subsumption* [7], defined as follows.

▶ **Definition 2.4** (Subsumption). *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$. We say U* is subsumed by *V, written $U \sqsubseteq V$, if there exists a label- and order-preserving bijection $h \colon S_{\mathbf{v}} \to S_{\mathbf{u}}$. That is, h is a bijection such that $\lambda_{\mathbf{u}} \circ h = \lambda_{\mathbf{v}}$ and if $s \leq_{\mathbf{v}} s'$, then $h(s) \leq_{\mathbf{u}} h(s')$.*

In the rest of this paper we only consider the relation $\sqsubseteq$ restricted to series-parallel pomsets. We will also need the notion of *pomset contexts* [12].

▶ **Definition 2.5.** *Let $*$ be a symbol not in Σ. The set of* pomset contexts*, denoted $\mathsf{PC}(\Sigma)$, is the smallest subset of $\mathsf{SP}(\Sigma \cup \{*\})$ satisfying*

$$\frac{}{* \in \mathsf{PC}(\Sigma)} \qquad \frac{X \in \mathsf{SP}(\Sigma \cup \{*\}) \qquad C \in \mathsf{PC}(\Sigma)}{X \cdot C \in \mathsf{PC}(\Sigma) \qquad C \cdot X \in \mathsf{PC}(\Sigma)} \qquad \frac{X \in \mathsf{SP}(\Sigma \cup \{*\}) \qquad C \in \mathsf{PC}(\Sigma)}{X \parallel C \in \mathsf{PC}(\Sigma)}$$

Alternatively, $\mathsf{PC}(\Sigma)$ consists of the sp-pomsets over $\Sigma \cup \{*\}$ with exactly one occurrence of $*$.

One can think of $*$ as a gap where another pomset can be inserted: given $C \in \mathsf{PC}$ and $U \in \mathsf{Pom}$, we can insert U into the gap in C to obtain $C[U]$. More precisely, we define

$$*[U] = U \qquad (C \cdot X)[U] = C[U] \cdot X \qquad (X \cdot C)[U] = X \cdot C[U] \qquad (X \parallel C)[U] = X \parallel C[U]$$

This insertion is well-defined, and can in fact be extended to pomsets in general [12]. We extend the notation to a set of pomsets $L \subseteq \mathsf{Pom}$ by $C[L] = \{C[U] \mid U \in L\}$.

Bi-Kleene Algebra (BKA): syntax and semantics. *Bi-Kleene Algebra* [18] adds a binary operator, denoted $\|$, to KA, which satisfies a few basic axioms but does not interact with the other KA operators. *BKA-terms* over Σ, denoted $\mathcal{E}_\Sigma$ (the subscript is omitted if it is clear from the context), also called series-rational expressions [19], are generated by the grammar

$$e, f ::= 0 \mid 1 \mid \mathtt{a} \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^*$$

The semantics of a BKA-term is a *pomset language*, i.e., an element of 2^{SP}. Formally, the BKA-semantics is a function $[\![-]\!] \colon \mathcal{E} \to 2^{\mathsf{SP}}$ defined inductively, as follows:

$$[\![0]\!] = \emptyset \qquad [\![1]\!] = \{1\} \qquad [\![e + f]\!] = [\![e]\!] \cup [\![f]\!] \qquad [\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!]$$
$$[\![e^*]\!] = [\![e]\!]^* \qquad [\![\mathtt{a}]\!] = \{\mathtt{a}\} \qquad [\![e \parallel f]\!] = [\![e]\!] \parallel [\![f]\!]$$

In this definition we use the pointwise lifting of sequential and parallel composition from pomsets to pomset languages, e.g., $L \cdot K = \{U \cdot V \mid U \in L,\ V \in K\}$. The Kleene star of a pomset language L is defined as $L^* = \bigcup_{n \in \mathbb{N}} L^n$, where $L^0 = \{1\}$ and $L^{n+1} = L^n \cdot L$.

We write $\equiv_{\mathsf{BKA}}$ or simply $\equiv$ for the smallest congruence on $\mathcal{E}$ generated by the Kleene algebra axiom together with the additional bi-Kleene algebra axioms, which govern the parallel operator $\|$; it is associative, commutative, has a unit and distributes over $+$ (Table 1). Soundness and completeness of $\equiv_{\mathsf{BKA}}$ w.r.t. the pomset language semantics was proved in [18]:

▶ **Theorem 2.6** (Soundness and Completeness BKA). *Let $e, f \in \mathcal{E}$. Then $e \equiv f \Leftrightarrow [\![e]\!] = [\![f]\!]$.*

Given alphabets Σ and Γ, a function $h \colon \Sigma \to \mathcal{E}_\Gamma$ extends inductively to a map $\hat{h} \colon \mathcal{E}_\Sigma \to \mathcal{E}_\Gamma$ (e.g., $\hat{h}(e + f) = \hat{h}(e) + \hat{h}(f)$) which we refer to as the *homomorphism generated by h*.

Concurrent Kleene Algebra with Hypotheses (CKAH). *Concurrent Kleene algebra with Hypotheses* [12] (see also [5] for the case of KA), allows for a set of additional axioms, called *hypotheses*, to be added to the axioms of BKA. Based on these hypotheses, one can then derive a sound model. This facilitates a modular completeness proof of POCKA based on the completeness of BKA, as POCKA extends BKA with additional axioms.

▶ **Definition 2.7.** *A* hypothesis *is an inequation $e \leq f$ where $e, f \in \mathcal{E}$. When H is a set of hypotheses, we write $\equiv^H$ for the smallest congruence on $\mathcal{E}$ generated by the hypotheses in H as well as the axioms and implications that build the equational theory of BKA. More concretely, whenever $e \leq f \in H$, also $e \leqq^H f$.*

▶ **Definition 2.8.** *Let $L \subseteq \mathsf{Pom}$. We define the H-closure of L, written $L{\downarrow^H}$, as the smallest language containing L such that for all $e \leq f \in H$ and $C \in \mathsf{PC}$, if $C[[\![f]\!]] \subseteq L{\downarrow^H}$, then $C[[\![e]\!]] \subseteq L{\downarrow^H}$. We stress here the use of the BKA-semantics for defining the H-closure of any language. Formally, $L{\downarrow^H}$ may be described as the smallest language satisfying:*

$$\frac{}{L \subseteq L{\downarrow^H}} \qquad \frac{e \leq f \in H \qquad C \in \mathsf{PC} \qquad C[[\![f]\!]] \subseteq L{\downarrow^H}}{C[[\![e]\!]] \subseteq L{\downarrow^H}}$$

The H-closure adds those pomsets that are needed to ensure soundness of the axioms generated by H. This yields a sound model for BKA with the set of hypotheses H [12]:

▶ **Lemma 2.9** (Soundness). *If $e \equiv^H f$, then $[\![e]\!]{\downarrow^H} = [\![f]\!]{\downarrow^H}$.*

An axiom often added to BKA is the *exchange law*, and together with BKA it axiomatises Concurrent Kleene Algebra (CKA). It can be added in the form of a set of hypotheses [12]:

▶ **Definition 2.10.** *We write* exch *for the set* $\{(e \parallel f)\cdot(g \parallel h) \leq (e\cdot g) \parallel (f\cdot h) \mid e,f,g,h \in \mathcal{E}\}$.

These hypotheses encode the interleavings of a program: when $e \cdot g$ runs in parallel with $f \cdot h$, one possible behaviour is that e first runs in parallel with f, followed by g in parallel with h.

The exch-closure coincides with the downwards closure w.r.t. the subsumption order [12].

▶ **Lemma 2.11.** *Let* $L \subseteq \mathsf{SP}$ *and* $U \in \mathsf{SP}$. $U \in L{\downarrow^{\mathsf{exch}}} \Leftrightarrow$ *there exists a* $V \in L$ *s.t.* $U \sqsubseteq V$.

▶ **Definition 2.12.** *A map* $c\colon \mathcal{E} \to \mathcal{E}$ *is a* syntactic closure *for* H *when for all* $e \in \mathcal{E}$ *it holds that* $e \equiv^H c(e)$ *and* $[\![e]\!]{\downarrow^H} = [\![c(e)]\!]$.

Syntactic closures are used in modular constructions of completeness proofs: their existence implies a completeness result for H, by reducing it to completeness of BKA, i.e. Theorem 2.6.

3 Partially Observable Concurrent Kleene Algebra

In this section we define *partially observable concurrent Kleene algebra* (*POCKA*). The syntax of POCKA is given by BKA terms over an alphabet tailor-made to reason about programs that can access variables and values. Specifically, this alphabet holds *assignments* of the form $(v \leftarrow n)$ and $(v \leftarrow v')$, and *observations* of the form $(v = n)$. We say $(v \leftarrow n)$ assigns the value n to variable v, $(v \leftarrow v')$ copies the value of variable v' to v, and $(v = n)$ asserts that v must have value n. Formally, we define the alphabets

$$\mathsf{Act} = \{(v \leftarrow n), (v \leftarrow v') \mid v, v' \in \textsc{Var}, n \in \textsc{Val}\} \qquad \mathsf{Obs} = \{(v = n) \mid v \in \textsc{Var}, n \in \textsc{Val}\}$$

where Var and Val are finite sets of variables and values, respectively (see Remark 3.12 for a discussion on the finiteness assumption). An example POCKA term would be $(x = 1) \cdot (x \leftarrow 2) \cdot (x = 2)$, which asserts that x must start with value 1, assigns the value 2 to x, and then asserts that x holds the value 2.

We will later give semantics to POCKA terms using program states, which are partial functions from Var to Val: $\mathsf{State} = \{\alpha \mid \alpha\colon \textsc{Var} \rightharpoonup \textsc{Val}\}$. The domain of a state α is denoted $\mathsf{dom}(\alpha)$. State carries a partial order $\leq$, where $\alpha \leq \beta$ iff $\mathsf{dom}(\beta) \subseteq \mathsf{dom}(\alpha)$ and for all $x \in \mathsf{dom}(\beta)$ we have $\alpha(x) = \beta(x)$, which we will use to generate the algebra of observations.

3.1 Observation algebra: axiomatisation and semantics

To obtain POCKA, we define the *observation* algebra (OA) that will be added to CKA as the algebraic structure of observations. This is similar to how a Boolean algebra enriches Kleene algebra into Kleene algebra with tests. In contrast with KAT, the observation algebra of POCKA is a pseudocomplemented distributive lattice, which is a generalisation of Boolean algebra in which the law of excluded middle does not necessarily hold.

▶ **Definition 3.1** (Pseudocomplemented Distributive Lattice)**.** *A* pseudocomplemented distributive lattice *(PCDL) is a tuple* $(A, \wedge, \vee, \overline{\cdot}, \top, \bot)$ *such that* $(A, \wedge, \vee, \top, \bot)$ *is a bounded distributive lattice and* $\overline{\cdot}\colon A \to A$ *is such that for* $p, q \in A$ *we have* $p \wedge q = \bot$ *iff* $p \leq \overline{q}$.

For a poset $(X, \leq)$ and a set $S \subseteq X$, define the *downwards-closure* of S by $S_{\leq} ::= \{x \mid \exists y \in S \text{ s.t. } x \leq y\}$ and $P_{\leq}(X) ::= \{Y \subseteq X \mid Y = Y_{\leq}\}$. It is well-known that $P_{\leq}(X)$ carries the structure of a bounded distributive lattice, with intersection as meet, union as join, X as top and $\emptyset$ as bottom. Further, if $(X, \leq)$ is finite, the lattice is itself finite and thus carries a (necessarily unique) pseudocomplement defined by $\overline{Y} ::= \bigcup\{Z \in P_{\leq}(X) \mid Y \cap Z = \emptyset\}$. This simply reifies that the pseudocomplement of an element is the largest element incompatible with it, which is guaranteed to exist in any complete lattice with bottom.

Table 1 Axioms of POCKA, built over an alphabet of actions Act and observations Obs. The left column contains the axioms of Concurrent Kleene Algebra. The right column axiomatises the partial observations: they form a pseudocomplemented distributive lattice, subject to constraints on the interface axioms that connect the lattice operators to the Kleene algebra ones. The last group of axioms applies to the observation alphabet Obs. We write $e \leqq f$ as a shorthand for $e + f \equiv f$.

Kleene Algebra Axioms

$$e + (f + g) \equiv (f + g) + h$$
$$e + f \equiv f + e$$
$$e + 0 \equiv e$$
$$e + e \equiv e$$
$$e \cdot (f \cdot g) \equiv (e \cdot f) \cdot g$$
$$e \cdot 1 \equiv e \equiv 1 \cdot e$$
$$e \cdot 0 \equiv 0 \equiv 0 \cdot e$$
$$e \cdot (f + g) \equiv e \cdot f + e \cdot h$$
$$(e + f) \cdot g \equiv e \cdot g + f \cdot g$$
$$e^* \equiv 1 + ee^*$$
$$e + f \cdot g \leqq f \Rightarrow e \cdot g^* \leqq f$$
$$e^* \equiv 1 + e^*e$$
$$e + f \cdot g \leqq g \Rightarrow f^* \cdot e \leqq g$$

Additional Bi-Kleene Algebra Axioms

$$e \parallel 1 \equiv e$$
$$e \parallel (f \parallel g) \equiv (e \parallel f) \parallel g$$
$$e \parallel 0 \equiv 0$$
$$e \parallel (f + g) \equiv e \parallel f + e \parallel g$$
$$e \parallel f \equiv f \parallel e$$

Exchange law

$$(e \parallel f) \cdot (g \parallel h) \leqq (e \cdot g) \parallel (f \cdot h)$$

Bounded Distributive Lattice Axioms

$$p \vee \bot \equiv p \equiv p \wedge \top$$
$$p \vee q \equiv q \vee p$$
$$p \wedge q \equiv q \wedge p$$
$$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$$
$$p \vee (q \vee r) \equiv (p \vee q) \vee r$$
$$p \vee (p \wedge q) \equiv p \equiv p \wedge (p \vee q)$$
$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

Pseudocomplement

$$p \leqq \overline{q} \Leftrightarrow p \wedge q \equiv \bot$$

Observation Axioms

$$v = n \wedge v = m \equiv \bot \qquad (n \neq m)$$
$$\overline{v = n} \leqq \bigvee_{n \neq m} v = m$$
$$\overline{\bigwedge_i v_i = n_i} \leqq \bigvee_i \overline{v_i = n_i} \qquad (\forall i \neq j . v_i \neq v_j)$$

Interface Axioms

$$p \wedge q \leqq p \cdot q$$
$$p \vee q \equiv p + q$$
$$0 \equiv \bot$$
$$\top \cdot p \leqq p \qquad p \cdot \top \leqq p \qquad (p \in \mathcal{O})$$
$$\top \cdot a \leqq a \qquad a \cdot \top \leqq a \qquad (a \in \mathsf{Act})$$

▶ **Definition 3.2** (Observation Algebra)**.** *The* Observation Algebra *is the PCDL* OA $::= (P_{\leq}(\mathsf{State}), \cap, \cup, \overline{\cdot}, \mathsf{State}, \emptyset)$ *generated by* $(\mathsf{State}, \leq)$.

Taking Obs as our set of propositions, we generate a term language $\mathcal{O}$ over the signature of PCDLs as follows:

$$p, q ::= \bot \mid \top \mid o \in \mathsf{Obs} \mid p \vee q \mid p \wedge q \mid \overline{p}.$$

This language is interpreted in OA by the homomorphic extension of the assignment

$$[\![v = n]\!] ::= \{\{v \mapsto n\}\}_{\leq} = \{\alpha \in \mathsf{State} \mid \alpha(v) = n\}.$$

Intuitively, the behaviour of an observation p consists of all partial functions that agree with p. This is captured algebraically below, in Lemma 3.7. For instance, $[\![v = n]\!]$ is the set containing all partial functions assigning n to v, and this is downwards closed because any partial function with a larger domain that also assigns n to v is included in this set.

If threads have only partial information about the machine state, an observation should be satisfied only if there is *positive evidence* for it. Hence, $\overline{v = n}$ should be satisfied only when v has a value that is different from n. To see why a Boolean algebra does not capture

the intended meaning of $\overline{v = n}$, consider using a BA over sets of partial functions, with negation as set-complement. This entails that $[\![\overline{v = n}]\!]$ will include all partial functions where v either gets a different value than n or no value at all. In the latter case, were we to obtain more information about the machine state we may discover that the actual value of v is in fact n, and it was therefore incorrect to assert $\overline{v = n}$. Our pseudocomplement provides a notion of negation that correctly excludes states for which this error could manifest, and this motivates our use of a PCDL rather than a Boolean algebra. This can be calculated directly:

▶ **Example 3.3.** Consider the semantics of $\overline{v = n}$:

$$\begin{aligned}[\![\overline{v = n}]\!] &= \bigcup\{Z \in P_{\leq}(\mathsf{State}) \mid [\![v = n]\!] \cap Z = \emptyset\} \\ &= \{\alpha \mid \alpha \in Z \text{ and } \{\beta \mid \beta(v) = n\} \cap Z = \emptyset\} \\ &= \{\alpha \mid \alpha(v) = m \text{ and } m \neq n\}\end{aligned}$$

In the last step we use that Z is downwards closed: if $\alpha(v)$ were undefined, then the partial function α' which is the same as α except that $\alpha'(v) = n$ would also occur in Z, making the intersection with $[\![v = n]\!]$ non-empty. Thus $\alpha \in [\![\overline{v = n}]\!]$ only if $\alpha(v)$ is defined, and evaluates to a value distinct from n. This witnesses the failure of the law of excluded middle.

▶ **Definition 3.4** (Axiomatisation). *$\equiv_{\mathsf{OA}}$, or simply $\equiv$, is the smallest congruence on $\mathcal{O}$ generated by the distributive lattice, pseudocomplement and observation axioms in Table 1.*

This axiomatisation supplements a standard axiomatisation of PCDLs with domain-specific axioms to capture the propositional theory of observations. For instance, the axiom $(v = n \wedge v = m \equiv \bot)$ states that a variable cannot have two different values at the same time. The axiom $(\overline{v = n} \leqq \bigvee_{n \neq m} v = m)$ tells us that the pseudocomplement of a variable having a value n is the assertion that the variable holds *some* distinct value m (the axiom is an inequality, but the other way around also holds). The last domain-specific axiom enforces specific instances of a De Morgan law that does not hold generally hold in arbitrary PCDLs.

Soundness of this axiomatisation follows straightforwardly from the fact that OA is a PCDL, together with basic consequences of the definition of the poset $(\mathsf{State}, \leq)$.

▶ **Lemma 3.5** (Soundness OA). *For all $p, q \in \mathcal{O}$, if $p \equiv q$ then $[\![p]\!] = [\![q]\!]$.*

Let $\pi_\alpha ::= \bigwedge_{\alpha(v)=n} v = n$. Note that if α is the empty function, then $\pi_\alpha = \bigwedge \emptyset = \top$.

▶ **Lemma 3.6.** *For all $\alpha, \beta \in \mathsf{State}$: $\alpha \in [\![\pi_\beta]\!]$ iff $\alpha \leq \beta$ iff $\pi_\alpha \leqq \pi_\beta$.*

In the following sections, we will silently assume that $\mathsf{State} \subseteq \mathcal{O}$. This is possible because π_- provides us with a sound way of injecting State inside $\mathcal{O}$. In order to prove completeness for $[\![-]\!]$ w.r.t. $\equiv$, we need an intermediary result, which allows us to syntactically rewrite any OA-expression in terms of elements of State.

▶ **Lemma 3.7.** *For all $p \in \mathcal{O}$, we have $p \equiv \bigvee\{\alpha \in \mathsf{State} \mid \pi_\alpha \leqq p\}$.*

With this result, we can then prove completeness of $[\![-]\!]$ w.r.t. $\equiv$ on terms from $\mathcal{O}$. In short, from $[\![p]\!] = [\![q]\!]$, $\pi_\alpha \leqq p$ iff $\pi_\alpha \leqq q$ can be established, from which $p \equiv q$ follows.

▶ **Theorem 3.8** (Completeness OA). *For all $p, q \in \mathcal{O}$, we have $p \equiv q$ if and only if $[\![p]\!] = [\![q]\!]$.*

3.2 POCKA: axiomatisation and semantics

▶ **Definition 3.9.** *The POCKA-terms, denoted $\mathcal{T}$, are formed by the following grammar:*

$$e, f ::= 0 \mid 1 \mid a \in \mathsf{Act} \mid p \in \mathcal{O} \mid e + f \mid e \cdot f \mid e \parallel f \mid e^* \ .$$

Note that $\mathcal{T} = \mathcal{E}_{\mathsf{Act} \cup \mathcal{O}}$.

The language model for POCKA consists of pomset languages over $\mathsf{Act} \cup \mathsf{State}$. When using pomsets to reason about behaviours of programs, we would like actions and states to alternate, because the states allow one to take stock of the configuration of the machine in between actions. However, imposing such an alternation in the semantics can be problematic with the exchange law [11]. Imagine the program $(\alpha \cdot \beta) \parallel \mathtt{a}$, where $\alpha, \beta \in \mathsf{State}$ and $\mathtt{a} \in \mathsf{Act}$. We can derive the following:

$$\begin{aligned}
\alpha \cdot \mathtt{a} \cdot \beta &\equiv (\alpha \parallel 1) \cdot (1 \parallel \mathtt{a}) \cdot (\beta \parallel 1) && \text{(Unit axiom)} \\
&\leqq (\alpha \parallel 1) \cdot ((1 \cdot \beta) \parallel (\mathtt{a} \cdot 1)) && \text{(Exchange Law)} \\
&\equiv (\alpha \parallel 1) \cdot (\beta \parallel \mathtt{a}) && \text{(Unit Axiom)} \\
&\leqq (\alpha \cdot \beta) \parallel (1 \cdot \mathtt{a}) \equiv (\alpha \cdot \beta) \parallel \mathtt{a} && \text{(Exchange Law, Unit Axiom)}
\end{aligned}$$

However, if the semantics $(\!|-|\!)$ contain only pomsets with alternating assignments and states, $(\!|\alpha \cdot \beta|\!)$ would have to consist of one state, and hence be empty if $\alpha \neq \beta$. This would make $(\!|(\alpha \cdot \beta) \parallel \mathtt{a}|\!)$ empty as well. As $(\!|\alpha \cdot \mathtt{a} \cdot \beta|\!)$ should not be empty, the exchange law is unsound. Thus, the POCKA-semantics is not restricted to pomsets with alternating states and actions.

▶ **Definition 3.10** (Semantics). *Let $(\!|-|\!) \colon \mathcal{T} \to 2^{\mathsf{SP}}$, where 2^{SP} are pomset languages over $\mathsf{Act} \cup \mathsf{State}$. For $p \in \mathcal{O}$, $(v \leftarrow n), (v \leftarrow v') \in \mathsf{Act}$ and $e, f \in \mathcal{T}$ we have:*

$$\begin{aligned}
(\!|v \leftarrow n|\!) &= \mathsf{State}^* \cdot \{v \leftarrow n\} \cdot \mathsf{State}^* & (\!|e + f|\!) &= (\!|e|\!) \cup (\!|f|\!) & (\!|e^*|\!) &= (\!|e|\!)^* \\
(\!|v \leftarrow v'|\!) &= \mathsf{State}^* \cdot \{v \leftarrow v'\} \cdot \mathsf{State}^* & (\!|e \cdot f|\!) &= (\!|e|\!) \cdot (\!|f|\!) & (\!|0|\!) &= \emptyset \\
(\!|p|\!) &= \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* & (\!|e \parallel f|\!) &= (\!|e|\!) \parallel (\!|f|\!) & (\!|1|\!) &= \{1\}
\end{aligned}$$

We define the POCKA-semantics of $e \in \mathcal{T}$ as $(\!|e|\!)\!\downarrow = (\!|e|\!)\!\downarrow^{\mathsf{exch} \cup \mathsf{contr}}$, where we use the closure definition from Definition 2.8, and $\mathsf{contr} = \{\alpha \leq \alpha \cdot \alpha \mid \alpha \in \mathsf{State}\}$*, referred to as* contraction.

We briefly explain closure under exch and contr. These closures are not part of the axiomatisation, but exist to ensure soundness of some of the axioms. The set of hypotheses exch closes the POCKA-semantics under subsumption and ensures soundness for the exchange law familiar from CKA. The set contr encodes that one way of observing α twice is to make both observations on the same state. This provides soundness for the axiom $p \wedge q \leqq p \cdot q$, which was introduced in [11]. This axiom captures that if p and q hold simultaneously in some state, it is possible to observe p and q in sequence (the converse should not hold as some action could happen in between the two obervations in a parallel thread).

▶ Remark 3.11. The assignment $(v \leftarrow v')$ cannot be simulated. In a sequential setting, we could express $(v \leftarrow v')$ as $\sum_{n \in \mathrm{VAL}}((v' = n) \cdot (v \leftarrow n))$. However, in a parallel setting this does not work, since some action can change the value of v' in between the observation that $(v' = n)$ and the assignment $(v \leftarrow n)$, meaning that v does not get assigned the value of v'.

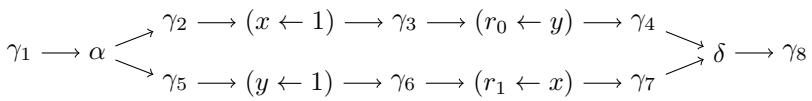
▶ Remark 3.12. In this paper we assume the set of variables VAR and the set of values VAL are both finite, in keeping with other verification frameworks, e.g. in model-checking.

The restriction on VAR could be lifted, since the finite set of variables that appear syntactically in a term completely determine its semantics. However, this is not the case for the set of values: for instance, the term $\overline{v = 0}$ evaluates to $\emptyset$ if the set of values is $\textsc{Val} = \{0\}$, but contains the partial function $[v \mapsto n]$ if VAL contains some value $n \neq 0$.

It is possible that a more sophisticated reduction could still work: indeed it seems unlikely that our finite terms would be able to manipulate non-trivially infinitely many values. For now though, this question is left open for future investigations.

The POCKA-semantics of a program e contains the possible behaviours of e in any possible context, where the context refers to any expression that could be put in parallel with e. For instance, $(\!|(v \leftarrow n)|\!)\downarrow$ contains pomsets that consist of a string of possible states of the machine, where the state of the machine can have been influenced by other parallel threads, followed at some point by the assignment $(v \leftarrow n)$, followed by another string of states. In Section 5, we will show how to reason about programs in isolation, i.e., under the hypothesis that there is no outside context to prompt state-modifying actions.

▶ **Example 3.13.** Let $t = (r_0 = 0 \wedge r_1 = 0) \cdot (\mathsf{T0} \parallel \mathsf{T1}) \cdot \overline{(r_0 = 1 \vee r_1 = 1)}$ as in (†) be our litmus test. A pomset in $(\!|t|\!)$ may look as follows, where we depict a pomset graphically with nodes labelled by actions or observations and their ordering with arrows.



Here, $\gamma_i \in \mathsf{State}$, $\alpha(r_0) = 0 = \alpha(r_1)$ and $\delta(r_0) = 0 = \delta(r_1)$. However, as stated in the introduction, if POCKA is sequentially consistent, this litmus test should pass, which means that the semantics of t should instead be empty. The reason it is not empty is that our semantics gives the behaviour of a program in any possible context, and indeed, if we put the litmus test in parallel with a program such as $(r_0 \leftarrow 0) \cdot (r_1 \leftarrow 0)$, the final assertion becomes satisfiable. In Sections 5 and 6, we look at how to execute the litmus test in isolation.

We also have axioms to algebraically describe equivalence between POCKA-terms, including some domain-specific axioms tailored to the alphabet. We define $\equiv$ as the smallest congruence on $\mathcal{T}$ generated by the axioms in Table 1.

▶ **Theorem 3.14** (Soundness POCKA). *For all $e, f \in \mathcal{T}$, if $e \equiv f$ then $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$.*

4 Completeness

In this section we prove completeness of the POCKA-semantics w.r.t. the axioms provided in Section 3.2. First, we show that POCKA terms can be normalised to a simpler form, where the only observations that appear are states. Next, we show that the resulting POCKA-terms can be used to describe the POCKA-semantics using BKA-semantics and closure. We then use the techniques from [12] to obtain a completeness result with respect to this semantics. Finally, we put all of the above together to obtain a completeness result for POCKA proper.

In order to normalise POCKA terms, we replace every observation by the summation of states to which it corresponds. This is done using the homomorphism $\hat{r}$ generated by

$$r(a) = \begin{cases} \sum_{\alpha \leq_{\mathsf{OA}} a} \alpha & a \in \mathcal{O} \\ a & a \in \mathsf{Act} \end{cases}$$

As a straightforward consequence of Lemma 3.7 and the interface axioms, we then obtain:

▶ **Lemma 4.1.** *For all $e \in \mathcal{T}$, it holds that $e \equiv \hat{r}(e)$.*

Proof. This can be proven by induction on the structure of e. If $e = \mathsf{a} \in \mathsf{Act}$, then $\hat{r}(\mathsf{a}) = \mathsf{a} \equiv \mathsf{a}$ immediately. Otherwise, if $p \in \mathcal{O}$, then we derive $\hat{r}(p) = \sum_{\alpha \leqq_{\mathsf{OA}} p} \alpha \equiv \bigvee_{\alpha \leqq_{\mathsf{OA}} p} \alpha \equiv p$, where we apply Lemma 3.7 in the last step. The inductive step follows trivially. ◀

The effect of $\hat{r}$ is to bridge the gap between the semantics of BKA and that of observation algebra: indeed for an observation $p \in \mathcal{O}$, we have $[\![\hat{r}(p)]\!] = [\![p]\!]_{\mathsf{OA}}$. However, this does not bring us fully to the unclosed POCKA-semantics $(\!|-|\!)$, as the latter inserts state-nodes in between actions and observations. For instance, $(\!|v \leftarrow n|\!)$ includes pomsets like $\alpha \cdot (v \leftarrow n) \cdot \beta$, while $[\![v \leftarrow n]\!] = \{v \leftarrow n\}$. We cover for this by means of the following set of hypotheses:

$$\mathsf{top} = \{\alpha \cdot c \leq c, c \cdot \alpha \leq c \mid \alpha \in \mathsf{State}, c \in \mathsf{Act} \cup \mathsf{State}\}$$

The hypotheses in top allow us to connect the unclosed POCKA semantics to the BKA-semantics, by filling in surrounding or preceding state-labelled nodes as necessary.

▶ **Lemma 4.2.** *For all $e \in \mathcal{T}$, we have $(\!|e|\!) = [\![\hat{r}(e)]\!]{\downarrow^{\mathsf{top}}}$.*

Sketch. Proceed by induction on the construction of e; the case where $e \in \mathsf{Act} \cup \mathcal{O}$ is fairly straightforward. The inductive case follows from the fact that closure w.r.t. top is compatible with pomset language composition, i.e., that $(L \cup K){\downarrow^{\mathsf{top}}} = L{\downarrow^{\mathsf{top}}} \cup K{\downarrow^{\mathsf{top}}}$ as well as $(L \cdot K){\downarrow^{\mathsf{top}}} = L{\downarrow^{\mathsf{top}}} \cdot K{\downarrow^{\mathsf{top}}}$, and similarly for the other operators defining $[\![-]\!]$. ◀

The next step is to provide syntactic closures for the sets of hypotheses involved. First, we note that there exists a syntactic closure for $\mathsf{exch} \cup \mathsf{contr}$, as shown in [12, Theorem 5.6].

▶ **Lemma 4.3.** *There exists a syntactic closure k for $\mathsf{exch} \cup \mathsf{contr}$.*

For the set top we still need to provide a syntactic closure. To this end, we simply take every action or observation in a term and surround it by a sequence of states of arbitrary length, which can be done using the homomorphism generated by

$$s(a) = \Big(\sum_{\alpha \in \mathsf{State}} \alpha\Big)^* \cdot a \cdot \Big(\sum_{\alpha \in \mathsf{State}} \alpha\Big)^*$$

It is fairly straightforward to show that this gives rise to a syntactic closure.

▶ **Lemma 4.4.** *The homomorphism generated by s is a syntactic closure for top.*

Sketch. The proof proceeds by induction on the construction of a term e. In the base, we can show for $a \in \mathsf{Act} \cup \mathcal{O}$ that $s(a) \equiv^{\mathsf{top}} a$ and $[\![a]\!]{\downarrow^{\mathsf{top}}} = [\![s(a)]\!]$. The inductive step follows by an argument similar to that in Lemma 4.2. ◀

The final step needed for the completeness proof of POCKA is a relation between the axioms that generate $\equiv$ and the hypotheses found in top and contr.

▶ **Lemma 4.5.** *For all $e, f \in \mathcal{T}$, if $e \leq f \in \mathsf{top} \cup \mathsf{contr}$, then $e \leqq f$.*

We now have all the ingredients in place for the desired completeness proof.

▶ **Theorem 4.6.** *For all $e, f \in \mathcal{T}$, we have $e \equiv f$ if and only if $(\!|e|\!){\downarrow} = (\!|f|\!){\downarrow}$.*

Proof. The direction from left to right was already established in Theorem 3.14. For the other direction, suppose that $e, f \in \mathcal{T}$ such that $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$. We can then derive that

$$\begin{aligned}
(\!|e|\!)\downarrow &= (\!|e|\!)\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(def. } (\!|-|\!)\downarrow) \\
&= \left(\llbracket \hat{r}(e) \rrbracket\downarrow^{\mathsf{top}}\right)\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(Lemma 4.2)} \\
&= \llbracket \hat{s} \circ \hat{r}(e) \rrbracket\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(Lemma 4.4)} \\
&= \llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket && \text{(Lemma 4.3)}
\end{aligned}$$

Similarly, $(\!|f|\!)\downarrow = \llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket$. Since $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$, also $\llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket = \llbracket k \circ \hat{s} \circ \hat{r}(f) \rrbracket$; by Theorem 2.6, it then follows that $k \circ \hat{s} \circ \hat{r}(e) \equiv_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(e)$. We then derive that

$$\begin{aligned}
e &\equiv \hat{r}(e) && \text{(Lemma 4.1)} \\
&\equiv^{\mathsf{top}}_{\mathsf{BKA}} \hat{s} \circ \hat{r}(e) && \text{(Lemma 4.4)} \\
&\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(e) && \text{(Lemma 4.3)} \\
&\equiv_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(f) && \text{(Observation above)} \\
&\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}} \hat{s} \circ \hat{r}(f) && \text{(Lemma 4.3)} \\
&\equiv^{\mathsf{top}}_{\mathsf{BKA}} \hat{r}(f) && \text{(Lemma 4.4)} \\
&\equiv f && \text{(Lemma 4.1)}
\end{aligned}$$

By Lemma 4.5, $\equiv^{\mathsf{top}}_{\mathsf{BKA}}$ and $\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}}$ are contained in $\equiv$; we conclude that $e \equiv f$. ◀

5 Guarded Pomsets

We now identify a fragment of the semantics that we use in the analysis of the litmus test from (†), namely the *guarded* pomsets. This term comes from Jipsen and Moshier [10], and was meant to define guarded pomsets in analogy to guarded strings in KAT.

We need two pieces of notation. First, we define the result of a state after updating it for one value. Let $\mathsf{a} \in \mathsf{Act}$ and $\alpha \in \mathsf{State}$. We say that $\alpha[\mathsf{a}]$ *exists* if $\mathsf{a} = v \leftarrow n$ for some $n \in \textsc{Val}$ or $\mathsf{a} = v \leftarrow v'$ and $v' \in \mathsf{dom}(\alpha)$. If $\alpha[\mathsf{a}]$ exists, we define it for all $w \in \textsc{Var}$ as follows:

$$\alpha[v \leftarrow n](w) = \begin{cases} n & \text{if } w = v \\ \alpha(w) & \text{otherwise} \end{cases} \qquad \alpha[v \leftarrow v'](w) = \begin{cases} \alpha(v') & \text{if } w = v \\ \alpha(w) & \text{otherwise} \end{cases}$$

Second, we define a binary operator $\oplus$ on State to combine states. For $\alpha, \beta \in \mathsf{State}$:

$$\alpha \oplus \beta = \begin{cases} \alpha \cup \beta & \text{if } \alpha(v) = \beta(v) \text{ for all } v \in \mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta) \\ \text{undefined} & \text{otherwise} \end{cases}$$

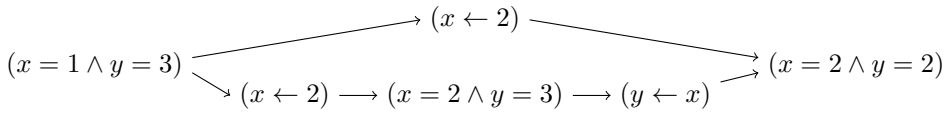
▶ **Definition 5.1.** *The set of* guarded pomsets*, denoted $\mathcal{G}$, is the smallest set satisfying:*

$$\frac{\alpha \in \mathsf{State}}{\alpha \in \mathcal{G}} \qquad \frac{\alpha \in \mathsf{State} \qquad \mathsf{a} \in \mathsf{Act} \qquad \alpha[\mathsf{a}] \textit{ exists}}{\alpha \cdot \mathsf{a} \cdot \alpha[\mathsf{a}] \in \mathcal{G}} \qquad \frac{U \cdot \alpha, \alpha \cdot V \in \mathcal{G} \qquad \alpha \in \mathsf{State}}{U \cdot \alpha \cdot V \in \mathcal{G}}$$

$$\frac{\alpha \cdot U \cdot \beta \qquad \gamma \cdot V \cdot \delta \in \mathcal{G} \qquad \alpha \oplus \gamma \textit{ defined} \qquad \beta \oplus \delta \textit{ defined} \qquad \alpha, \beta, \gamma, \delta \in \mathsf{State}}{\alpha \oplus \gamma \cdot (U \parallel V) \cdot \beta \oplus \delta \in \mathcal{G}}$$

This definition is close to [10]. The definition in op. cit. is not catered to a specific alphabet, and the operator $\oplus$ to combine states does not allow for the two states to have any shared variable in their domains. We deliberately deviate from this, allowing threads to share variables as long as they do so in a consistent manner.

▶ **Example 5.2.** Consider the guarded pomsets $(x = 1) \cdot (x \leftarrow 2) \cdot (x = 2)$ and $(x = 1 \wedge y = 3) \cdot (x \leftarrow 2) \cdot (x = 2 \wedge y = 3) \cdot (y \leftarrow x) \cdot (x = 2 \wedge y = 2)$. The final rule for the construction of $\mathcal{G}$ guarantees that the following pomset is again guarded:



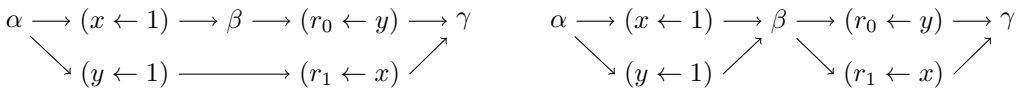
Note how $(x = 2) \oplus (x = 2 \wedge y = 3)$ is defined, because both states agree on the value of x.

In the execution of parallel threads in pomsets, no interaction between the threads takes place: the threads execute "truly" concurrently. To account for interactions, we consider the interleavings that result from closure w.r.t. the exchange law (c.f. Lemma 2.11).

▶ **Example 5.3.** Consider the a slightly adjusted version of the litmus test t discussed earlier:

$$t' = (r_0 = 0 \wedge r_1 = 0); (\mathsf{T0} \parallel \mathsf{T1}); (r_0 = 1 \vee r_1 = 1)$$

The *unclosed* semantics of t' includes (but is not limited to) the pomset below on the left, for all $\alpha, \beta, \gamma \in \mathsf{State}$, where $\alpha(r_0) = \alpha(r_1) = 0$, and $\gamma(r_0) = 1$ or $\gamma(r_1) = 1$. As a result of the exchange law, the *closed* semantics includes the pomset below on the right.



In the special case where $\alpha = \{r_0 \mapsto 0, r_1 \mapsto 0\}$, $\beta = \{r_0 \mapsto 0, r_1 \mapsto 0, x \mapsto 1, y \mapsto 1\}$, $\gamma = \{r_0 \mapsto 1, r_1 \mapsto 1, x \mapsto 1, y \mapsto 1\}$, the latter is a guarded pomset.

Guardedness in pomsets can be characterised by the conjunction of seven properties, which we will discuss now. On the one hand, these properties have an intuitive explanation as characteristics of behaviours of (possibly concurrent) programs running in isolation. Hence, if a pomset represents some execution of an isolated program, it is guarded. On the other hand, the characterisation in terms of these properties provides a proof method to show that a pomset is *not* guarded, by demonstrating the failure of one such property.

We start by observing that guarded pomsets alternate states and actions. Formally, we can capture this in three properties. Let $U = [\mathbf{u}] \in \mathsf{SP}(\mathsf{Act} \cup \mathsf{State})$. We say that $s' \in S_{\mathbf{u}}$ is a *predecessor* of s if it is the latest node ordered strictly before s – i.e., $s\ s' <_{\mathbf{u}} s$ and for all $s'' \in S_{\mathbf{u}}$ such that $s'' <_{\mathbf{u}} s$ it holds that $s'' \leq_{\mathbf{u}} s'$. The notion of *successor* is defined dually. A node is a *state-node* if it is labelled by an element of State, and an *action-node* otherwise.
(A1) U admits a unique minimum and maximum, $*_{\min}, *_{\max} \in S_{\mathbf{u}}$, labelled by states.
(A2) Every two related state-nodes are separated by an action-node.
(A3) Action-nodes have unique state-nodes as neighbours (their predecessor and successor).

The next property formalises the idea that two related observations cannot contradict each other, such as in the program $(x = 1) \cdot (x = 2)$. To this end, we need the notion of a *path*. A path for a variable v from a state-node u to another state-node s is a chain such that the changes in the value of v between u and s are explained by the actions between them and recorded in all the states between u and s.

▶ **Definition 5.4** (Path). *Let $U = [\mathbf{u}] \in \mathsf{Pom}(\mathsf{Act} \cup \mathsf{State})$ and $u_1, u_2 \in S_{\mathbf{u}}$ such that $u_1 \leq_{\mathbf{u}} u_2$ and $\lambda_{\mathbf{u}}(u_1), \lambda_{\mathbf{u}}(u_1) \in \mathsf{State}$. We say a* path *$p_v$ from u_1 to u_2 for variable $v \in \textsc{Var}$ is a sequence of nodes $q_1, a_1, \ldots, a_n, q_{n+1} \in S_{\mathbf{u}}$ that satisfy the following conditions:*

(P1) *For all* $1 \leq i \leq n$*, we have* $\lambda_{\mathbf{u}}(a_i) \in \mathsf{Act}$ *and* $u_1 \leq_{\mathbf{u}} a_i \leq_{\mathbf{u}} u_2$ *for all* i*. Additionally we require that* $a_i \leq_{\mathbf{u}} a_{i+1}$ *for* $1 \leq i < n$*.*

(P2) *For all* $1 \leq i \leq n+1$ *it holds that* $\lambda_{\mathbf{u}}(q_i) \in \mathsf{State}$*, and for all* $1 \leq i \leq n$*, the predecessor of* a_i *is* q_i*, and the successor of* a_i *is* q_{i+1}*. Additionally we have that* $\lambda_{\mathbf{u}}(q_1) = \lambda_{\mathbf{u}}(u_1)$*,* $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u_1))$ *and* $\lambda_{\mathbf{u}}(q_{n+1}) = \lambda_{\mathbf{u}}(u_2)$*. Lastly, for* $1 \leq i \leq n$ *we have:*

$$\lambda_{\mathbf{u}}(q_{i+1})(v) = \begin{cases} n & \lambda_{\mathbf{u}}(a_i) = v \leftarrow n \text{ for some } n \in \textsc{Val} \\ \lambda_{\mathbf{u}}(q_i)(v') & \lambda_{\mathbf{u}}(a_i) = v \leftarrow v' \text{ for some } v' \in \textsc{Var}, v' \in \mathsf{dom}(\lambda_{\mathbf{u}}(q_i)) \\ \lambda_{\mathbf{u}}(q_i)(v) & \text{otherwise} \end{cases}$$

▶ **Example 5.5.** The following is a path for x:

$$(x = 1) \cdot (y \leftarrow 3) \cdot (x = 1) \cdot (x \leftarrow 2) \cdot (x = 2 \wedge y = 3) \cdot (x \leftarrow y) \cdot (x = 3)$$

Note that this is not a path for y, because it is not assigned a value by the final atom.

We can now formulate another criterion for a pomset executing in isolation: for every variable in the domain of a state-node, there is a path explaining the changes in value of that variable between the state-node and the maximum node of the pomset.

(A4) For all state-nodes $u \in S_{\mathbf{u}}$ and $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u))$, there is a path for v from u to $*_{\max}$.

▶ **Example 5.6.** The first pomset below satisfies (A4), and the second pomset does not, as there is no path from beginning to end for x: the value of x in the second observation is not in accordance to the previous assignment.

$$(x = 2 \wedge y = 2) \begin{array}{l} \nearrow (x \leftarrow 4) \searrow \\ \searrow (y \leftarrow 3) \nearrow \end{array} (x = 4 \wedge y = 3)$$

$$(x = 2 \wedge y = 4) \longrightarrow (x \leftarrow 4) \longrightarrow (x = 5 \wedge y = 4) \longrightarrow (y \leftarrow 2) \longrightarrow (x = 4 \wedge y = 2)$$

If a pomset represents an isolated program, an action has an effect on its successor. If that action is of the form $v \leftarrow n$, then the sucessor should assign n to v; likewise, if the action is of the form $v \leftarrow v'$, then the successor should assign the value of v' to v, but the predecessor should also be aware of a value for v'.

(A5) If $u \in S_{\mathbf{u}}$ such that $\lambda_{\mathbf{u}}(u) = v \leftarrow n$ for some $v \in \textsc{Var}$ and $n \in \textsc{Val}$, we require that the successor of u is s s.t. $\lambda_{\mathbf{u}}(s)(v) = n$.

(A6) Let $u \in S_{\mathbf{u}}$ s.t. $\lambda_{\mathbf{u}}(u) = v \leftarrow v'$ for some $v, v' \in \textsc{Var}$ and let p and s be the predecessor, resp. successor, of u. Then $v' \in \mathsf{dom}(\lambda_{\mathbf{u}}(p))$ and $\lambda_{\mathbf{u}}(s)(v) = \lambda_{\mathbf{u}}(s)(v') = \lambda_{\mathbf{u}}(p)(v')$.

▶ **Example 5.7.** The pomset below on the left violates (A5), because the successor of $(x \leftarrow 1)$ does not assign 2 to x. On the other hand, the pomset on the right satisfies (A6), because the predecessor of $(x \leftarrow y)$ has a value for y, and that value is assigned to x in the successor.

$$(x = 0) \longrightarrow (x \leftarrow 1) \longrightarrow (x = 2) \qquad (x = 1 \wedge y = 2) \longrightarrow (x \leftarrow y) \longrightarrow (x = 2 \wedge y = 2)$$

Finally, isolated programs cannot observe variables that have not been assigned a value anywhere in the program. On the pomset-level, this translates to:

(A7) Let $u \in S_{\mathbf{u}}$ be a state-node. Then for all $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u))$, there exists a path for v from $s \in S_{\mathbf{u}}$ to u such that either $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(s))$ and $s = *_{\min}$ or s is the successor of an assignment-node with label $v \leftarrow k$ with $k \in \textsc{Var} \cup \textsc{Val}$.

▶ **Example 5.8.** The pomset on the left does not satisfy (A7), but the one on the right does.

$$(x = 1) \longrightarrow (x \leftarrow 2) \longrightarrow (x = 2 \wedge y = 2) \qquad (x = 1) \longrightarrow (y \leftarrow 2) \longrightarrow (x = 1 \wedge y = 2)$$

Guarded pomsets satisfy (A1)–(A7). In fact, there exists an equivalence:

▶ **Theorem 5.9.** *For $U \in \mathsf{SP}$, U is guarded if and only if U satisfies (A1)–(A7).*

Sketch. The forward implication is proved by induction on the construction of $\mathcal{G}$. For the other direction, we perform induction on the size of U (which is possible because U is series-parallel and therefore finite). The induction hypothesis then states that whenever V is strictly smaller than U, and V satisfies the seven properties, then V is guarded. Since U satisfies (A1), we know that either it consists of one node labelled by a state, in which case U is immediately guarded, or $U = \alpha \cdot V \cdot \beta$ for $\alpha, \beta \in \mathsf{State}$ and $V \in \mathsf{SP}$. This gives us four cases to consider: $V = 1$, $V = \mathsf{a}$ for some $\mathsf{a} \in \mathsf{Act} \cup \mathsf{State}$, $V = V_0 \cdot V_1$ or $V = V_0 \parallel V_1$. The first case can be disregarded as U would then violate (A2). In the second case, (A4)–(A7) can be used to show that $\beta = \alpha[\mathsf{a}]$. In the latter two cases we show that U is built out of two strictly smaller pomsets that satisfy (A1)–(A7), making them guarded by the induction hypothesis. When these two pomsets are combined to form U, this is done according to the rules of guarded pomsets, making U guarded as well. The details can be found in the full version of this paper [22]. ◀

6 Litmus Test

The POCKA-semantics of a program captures all *possible* behaviours of the program, including all behaviours that could arise when it is put in parallel with other programs. In this section we look at the behaviour of the litmus test when it is executed in isolation. In the previous section we saw that if a pomset represents an execution of a program in isolation, it is guarded, and hence it is sufficient to look at the guarded pomsets. We demonstrate that there are in fact no guarded pomsets in the semantics of the litmus test, which shows that it passes. This suggests the guarded fragment of the POCKA-semantics is sequentially consistent: the programs behave as if memory accesses performed concurrently are interleaved and executed sequentially and writes to memory are broadcasted to all threads instantaneously.

Recall the litmus test t we considered before, with $\textsc{Var} = \{x, y, r_0, r_1\}$ and $\textsc{Val} = \{0, 1\}$:

$$t := (r_0 = 0 \wedge r_1 = 0) \cdot ((x \leftarrow 1 \cdot r_0 \leftarrow y) \parallel (y \leftarrow 1 \cdot r_1 \leftarrow x)) \cdot \overline{(r_0 = 1 \vee r_1 = 1)}$$

Our strategy for showing that the semantics of t does not contain guarded pomsets, is to first show that all pomsets in the semantics of t have certain property. We then claim that if a pomset has this property, then it is not guarded, using (A1)–(A7) from Section 5.

▶ **Definition 6.1** (Litmus Pomsets). *Let $x, y, r_0, r_1, w \in \textsc{Var}$ be distinct and $0, 1 \in \textsc{Val}$. A pomset $U = [\mathbf{u}]$ has property P, denoted $P(U)$, if there exists $u_1, u_2, v_1, v_2, w \in S_{\mathbf{u}}$ s.t.*

1. *the following conditions hold:*

$$\lambda_{\mathbf{u}}(u_1) = (x \leftarrow 1) \qquad \lambda_{\mathbf{u}}(u_2) = (y \leftarrow 1) \qquad \lambda_{\mathbf{u}}(v_1) = (r_0 \leftarrow y) \qquad \lambda_{\mathbf{u}}(v_2) = (r_1 \leftarrow x)$$

$$\lambda_{\mathbf{u}}(w)(r_0) = 0 = \lambda_{\mathbf{u}}(w)(r_1) \qquad u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} w \qquad u_2 \leq_{\mathbf{u}} v_2 \leq_{\mathbf{u}} w$$

Graphically, we can represent these conditions as the following diagram:

$$\begin{array}{ccc} u_1 : x \leftarrow 1 & \longrightarrow & v_1 : r_0 \leftarrow x \\ & \searrow & \\ & & w : \left[\begin{array}{c} r_0 \mapsto 0 \\ r_1 \mapsto 0 \end{array}\right] \\ & \nearrow & \\ u_2 : y \leftarrow 1 & \longrightarrow & v_2 : r_1 \leftarrow y \end{array}$$

2. *For other assignment-nodes in U, we have the following conditions. Let $k \in \textsc{Val} \cup \textsc{Var}$.*

$$\forall z. \lambda_{\mathbf{u}}(z) = (x \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} u_1 \qquad \forall z. \lambda_{\mathbf{u}}(z) = (y \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} u_2$$

$$\forall z. \lambda_{\mathbf{u}}(z) = (r_0 \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} v_1 \qquad \forall z. \lambda_{\mathbf{u}}(z) = (r_1 \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} v_2$$

The property P describes the actions and observations found in the litmus test, and their relative ordering. For instance, $\forall z.\lambda_{\mathbf{u}}(z) = (x \leftarrow n) \Rightarrow z \leq_{\mathbf{u}} u_1$ states that all action-nodes that change the value of x, occur before node u_1 Hence, the maximal node that alters the value of x, changes x to 1. The other requirements are explained similarly.

▶ **Lemma 6.2.** *Let $U = [\mathbf{u}] \in \mathsf{SP}$. If $P(U)$ then U is not guarded.*

We show that P is an invariant under closure w.r.t. exch and contr. To this end, it is useful to study the effect of the contraction order on the level of pomsets; we introduce the following partial order relation on pomsets, analogous to the subsumption order.

▶ **Definition 6.3** (Contraction Order)**.** *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$ be pomsets over $\mathsf{Act} \cup \mathsf{State}$. We write $U \preceq V$ holds iff there exists a surjection $h\colon S_{\mathbf{v}} \to S_{\mathbf{u}}$ satisfying: (i) $\lambda_{\mathbf{u}} \circ h = \lambda_{\mathbf{v}}$; (ii) $v \leq_{\mathbf{v}} v'$ implies $h(v) \leq_{\mathbf{u}} h(v')$; (iii) if $h(v) \leq_{\mathbf{u}} h(v')$, then $\lambda_{\mathbf{v}}(v), \lambda_{\mathbf{v}}(v') \in \mathsf{State}$ implies $v \leq_{\mathbf{v}} v'$ or $v' \leq_{\mathbf{v}} v$, and $\lambda_{\mathbf{v}}(v)$ or $\lambda_{\mathbf{v}}(v') \notin \mathsf{State}$ implies $v \leq_{\mathbf{v}} v'$.*

We then prove the analogue of Lemma 2.11, relating $\preceq$ to closure w.r.t. contr as follows.

▶ **Lemma 6.4.** *Let $L \subseteq \mathsf{SP}$ and $U \in \mathsf{SP}$. Now $U \in L{\downarrow^{\mathsf{contr}}}$ iff $U \preceq V$ for some $V \in L$.*

With this characterisation in hand, we can prove that P is invariant under closure.

▶ **Lemma 6.5.** *Let $e \in \mathcal{T}$. If $\forall U \in (\!|e|\!)$ we have $P(U)$, then $\forall V \in (\!|e|\!)\downarrow$ it holds that $P(V)$.*

Sketch. By [12, Lemma 5.4] we know that $(\!|e|\!)\downarrow = \left((\!|e|\!)\downarrow^{\mathsf{exch}}\right)\downarrow^{\mathsf{contr}}$. It then follows, by Lemma 2.11 and Lemma 6.4, that if $V \in (\!|e|\!)\downarrow$, then there must exist $W, X \in \mathsf{SP}$ with $X \in (\!|e|\!)$ and $V \preceq W \sqsubseteq X$. We then show that P is preserved by both of these orders. ◀

▶ **Corollary 6.6.** *The semantics of the litmus test contains no guarded pomsets: $(\!|t|\!)\downarrow \cap \mathcal{G} = \emptyset$.*

Proof. All pomsets in $(\!|t|\!)$ have property P if we pick for u_1 the node with label $(x \leftarrow 1)$, for v_1 the node with label $(r_0 \leftarrow y)$, and same for u_2 and v_2 (see Example 3.13). Lastly, we pick for w the node with label δ. By Lemma 6.5 we can conclude that all pomsets in $(\!|t|\!)\downarrow$ have property P, and by Lemma 6.2 we infer that t has no guarded pomsets in its semantics. ◀

We showed that we can correctly analyse the litmus test in our algebraic framework. In the next example we show that addition of one extra axiom, which is a commonly made assumption in programming languages, makes the litmus test fail on the guarded semantics.

▶ **Example 6.7.** We add the following axiom, which states that assignments to different variables can be swapped as long as the assigned values are none of the involved variables:

$$v \leftarrow k \cdot v' \leftarrow k' \equiv v' \leftarrow k' \cdot v \leftarrow k \text{ for } v, v' \in \textsc{Var},\ k, k' \in \textsc{Var} \cup \textsc{Val},\ k' \neq v \neq v',\ k \neq v \neq v'$$

We show that with this assumption, which is commonly made in programming languages, we get guarded pomsets in the semantics of the litmus program. We can derive:

$$\begin{aligned}
((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) &\equiv ((r_0 \leftarrow y) \parallel 1) \cdot (1 \parallel (r_1 \leftarrow x)) && \text{(Unit axiom)} \\
&\leqq ((r_0 \leftarrow y) \cdot 1) \parallel (1 \cdot (r_1 \leftarrow x)) && \text{(Exchange Law)} \\
&\equiv (r_0 \leftarrow y) \parallel (r_1 \leftarrow x) && \text{(Unit axiom)}
\end{aligned}$$

Similarly, we can derive that $(x \leftarrow 1) \cdot (y \leftarrow 1) \leqq (x \leftarrow 1) \parallel (y \leftarrow 1)$. Hence, we have

$$\begin{aligned}
&((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \cdot (y \leftarrow 1)) \leqq ((r_0 \leftarrow y) \parallel (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \parallel (y \leftarrow 1)) \\
&\leqq ((r_0 \leftarrow y) \cdot (x \leftarrow 1)) \parallel ((r_1 \leftarrow x) \cdot (y \leftarrow 1)) && \text{(Exchange law)} \\
&\equiv ((x \leftarrow 1) \cdot (r_0 \leftarrow y)) \parallel ((y \leftarrow 1) \cdot (r_1 \leftarrow x)) && \text{(New axiom)}
\end{aligned}$$

Let $e = ((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \cdot (y \leftarrow 1))$. We can conclude that

$$(r_0 = 0 \wedge r_1 = 0) \cdot e \cdot \overline{(r_0 = 1 \vee r_1 = 1)} \leqq t$$

From soundness, we infer that $(\!|(r_0 = 0 \wedge r_1 = 0) \cdot e \cdot \overline{(r_0 = 1 \vee r_1 = 1)}|\!)\downarrow \subseteq (\!|t|\!)\downarrow$. In the left set we find at least one guarded pomset. Let $\alpha = (r_0 = 0 \wedge r_1 = 0 \wedge x = 0 \wedge y = 0)$, $\beta = (r_0 = 0 \wedge r_1 = 0 \wedge x = 1 \wedge y = 0)$ and $\gamma = (r_0 = 0 \wedge r_1 = 0 \wedge x = 1 \wedge y = 1)$.
$\alpha \longrightarrow (r_0 \leftarrow y) \longrightarrow \alpha \longrightarrow (r_1 \leftarrow x) \longrightarrow \alpha \longrightarrow (x \leftarrow 1) \longrightarrow \beta \longrightarrow (y \leftarrow 1) \longrightarrow \gamma$

It is easy to show that this pomset is guarded by observing that $\alpha \cdot (r_0 \leftarrow y) \cdot \alpha$, $\alpha \cdot (r_1 \leftarrow x) \cdot \alpha$, $\alpha \cdot (x \leftarrow 1) \cdot \beta$ and $\beta \cdot (y \leftarrow 1) \cdot \gamma$ are all guarded. Hence, by adding this one extra axiom, we find guarded pomsets in the semantics of the litmus test, meaning that this axiom breaks sequential consistency.

7 Discussion

We presented POCKA, a sound and complete algebraic framework that can be used to analyse concurrent programs that manipulate variables. We identified the guarded fragment of the semantics, and showed this fragment captures the behaviour of programs executing in isolation. We demonstrated reasoning in POCKA by analysing a litmus test, also suggesting that the guarded fragment of the POCKA-semantics is sequentially consistent.

This work is built on Kleene algebra and extensions thereof. It is closest to Concurrent Kleene algebra with Observations [11, 12], which was proposed to integrate concurrency with a form of tests (i.e., observations). We deviate from CKAO by using partial observations and accordingly changing the algebraic structure of observations (a PCDL instead of a Boolean algebra), and by incorporating explicit assignments and tests to manipulate variables. Programs such as the litmus test that we analyse in POCKA are outside the scope of CKAO.

The idea of using a PCDL and partial functions in the semantics comes from Jipsen and Moshier [10]. In the current paper we establish completeness w.r.t. the partial function model, which is missing in *loc. cit.* A further contrast is that POCKA includes as basic syntax atomic programs and assertions pertaining to variable assignment, as occur in the litmus test. The definition of guarded pomset that we used is close to the one proposed in [10]. We provided an extensive analysis of guarded pomsets and showed how they can be used to study concrete program behaviour: our new characterisation in terms of concrete properties of pomsets (Theorem 5.9) is essential for the analysis of the litmus test in Section 6.

We suggest three avenues for future research. Firstly, the concrete observations and assignments that we have used are reminiscent of NetKAT [2, 6], an algebraic framework based on Kleene algebra with tests that allows for reasoning about networks. POCKA is thus suggestive of a concurrent version of NetKAT, in which algebraic reasoning about concurrent networks could be studied. While NetKAT arises as a particular instance of KAT, POCKA is *not* an instance of its closest relative in the Kleene algebra family, CKAO, due to the aforementioned move from an *arbitrary* Boolean algebra of observations to a *concrete* PCDL. It would therefore be of interest to formulate the necessary metatheory for the analogous framework of CKA with partial observations (where partial observations are given by an arbitrary PCDL), and situate POCKA within it.

This naturally leads to a third line of research. We have used the CKAH framework to obtain a completeness proof, and it turned out that the proof technique was perfectly amenable to a replacement of the Boolean algebra structure of observations with our observation algebra. This raises the question: which conditions are necessary on the algebraic structure of observations to be able to prove completeness in a similar manner? In particular, what

conditions are needed for a result similar to Lemma 4.1 to hold? Our conjecture is that the observation algebra needs to be such that all elements can be written as a finite sum of join-irreducible elements of the algebra (cf. Lemma 3.7).

References

1 Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. Litmus: Running tests against hardware. In *TACAS*, pages 41–44, 2011. doi:10.1007/978-3-642-19835-9_5.

2 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126, 2014. doi:10.1145/2535838.2535862.

3 Thomas S. Blyth. *Lattices and Ordered Algebraic Structures*. Springer-Verlag London, 2005.

4 John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, Ltd., London, 1971.

5 Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. Kleene algebra with hypotheses. In *FOSSACS*, pages 207–223, 2019. doi:10.1007/978-3-030-17127-8_12.

6 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, pages 343–355, 2015. doi:10.1145/2676726.2677011.

7 Jay L. Gischer. The equational theory of pomsets. *Theor. Comput. Sci.*, 61:199–224, 1988. doi:10.1016/0304-3975(88)90124-7.

8 Jan Grabowski. On partial languages. *Fundam. Inform.*, 4(2):427, 1981.

9 Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene algebra. In *CONCUR*, pages 399–414, 2009. doi:10.1007/978-3-642-04081-8_27.

10 Peter Jipsen and M. Andrew Moshier. Concurrent Kleene algebra with tests and branching automata. *J. Log. Algebr. Meth. Program.*, 85(4):637–652, 2016. doi:10.1016/j.jlamp.2015.12.005.

11 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene algebra with observations. In *CONCUR*, pages 41:1–41:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.41.

12 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene algebra with observations: From hypotheses to completeness. In *FOSSACS*, pages 381–400, 2020. doi:10.1007/978-3-030-45231-5_20.

13 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

14 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In *TACAS*, pages 14–33, 1996. doi:10.1007/3-540-61042-1_35.

15 Dexter Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

16 Daniel Krob. A complete system of B-rational identities. In *ICALP*, pages 60–73, 1990. doi:10.1007/BFb0032022.

17 Leslie Lamport. How to make a correct multiprocess program execute correctly on a multiprocessor. *IEEE Trans. Computers*, 46(7):779–782, 1997. doi:10.1109/12.599898.

18 Michael R. Laurence and Georg Struth. Completeness theorems for bi-Kleene algebras and series-parallel rational pomset languages. In *RAMiCS*, pages 65–82, 2014. doi:10.1007/978-3-319-06251-8_5.

19 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.

20 John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *LiCS*, July 2002. doi:10.1109/LICS.2002.1029817.

21 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

22 Jana Wagemaker, Paul Brunet, Simon Docherty, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. Partially observable concurrent Kleene algebra, 2020. arXiv:2007.07593.

A Proofs about observation algebra

▶ **Lemma 3.5** (Soundness OA). *For all $p, q \in \mathcal{O}$, if $p \equiv q$ then $[\![p]\!] = [\![q]\!]$.*

Proof. The fact that OA is a PCDL and the assignment $[\![-]\!]$ is well-defined establishes the soundness of the PCDL axioms. We thus only have the domain-specific axioms left to verify.

- If $n \neq m$, it is immediate that

$$[\![v = n \wedge v = m]\!] = [\![v = n]\!] \cap [\![v = m]\!] = \{\alpha \mid \alpha(v) = n\} \cap \{\alpha \mid \alpha(v) = m\} = \emptyset = [\![\bot]\!]$$

- Next we show $[\![\overline{v = n}]\!] \subseteq \left[\!\!\left[\bigvee_{m \neq n} v = m\right]\!\!\right]$. Assume

$$\alpha \in [\![\overline{v = n}]\!] = \bigcup \{B \in P_{\leq}(\mathsf{State}) \mid B \cap [\![v = n]\!] = \emptyset\}.$$

  We have some downwards-closed $B \subseteq \mathsf{State}$ such that $\alpha \in B$ and $B \cap [\![v = n]\!] = \emptyset$. Suppose towards a contradiction that $\alpha(v)$ is undefined, or that $\alpha(v) = n$. We can then choose $\alpha' \in \mathsf{State}$ to be n on v, and identical to α elsewhere; in that case, $\alpha' \leq \alpha$, which means that $\alpha' \in B$. But then, since $\alpha' \in [\![v = n]\!]$ by construction, we have a contradiction with the fact that $B \cap [\![v = n]\!] = \emptyset$. Thus, there exists an $m \in \textsc{Val}$ such that $\alpha(v) = m$ and $m \neq n$. It then follows that $\alpha \in \left[\!\!\left[\bigvee_{m \neq n} v = m\right]\!\!\right]$.

- Next we prove that $\left[\!\!\left[\overline{\bigwedge_i v_i = n_i}\right]\!\!\right] \subseteq [\![\bigvee_i \overline{v_i = n_i}]\!]$, if the v_i are distinct. Let $\alpha \in B$ such that $B \cap [\![\bigwedge_i v_i = n_i]\!] = \emptyset$. We claim that for some i, $\alpha(v_i) = m \neq n_i$. Suppose otherwise: then for each v_i either $\alpha(v_i) = n_i$ or $\alpha(v_i)$ is undefined. Define:

$$\alpha'(v) ::= \begin{cases} n_i & \text{if } v = v_i \text{ and } \alpha(v_i) \text{ undefined;} \\ \alpha(v) & \text{otherwise.} \end{cases}$$

  This is well-defined by the assumption that the v_i are all distinct. By construction, $\alpha' \leq \alpha$ and $\alpha' \in \bigcap_i \{\beta \mid \beta(v_i) = n_i\} = [\![\bigwedge_i v_i = n_i]\!]$. As B is downwards-closed, we get $\alpha' \in B$, contradicting $B \cap [\![\bigwedge_i v_i = n_i]\!] = \emptyset$. Hence for some i, $\alpha(v_i) = m \neq n_i$. Hence $\alpha \in [\![\overline{v_i = n_i}]\!] \subseteq [\![\bigvee_i \overline{v_i = n_i}]\!]$ as required.

For the inductive step, we verify that the closure rules for congruence preserve soundness. This is all immediate from the definition of $[\![-]\!]$. For instance, if $e = e_0 \vee e_1$, $f = f_0 \vee f_1$, $e_0 \equiv f_0$ and $e_1 \equiv f_1$, then $[\![e]\!] = [\![e_0]\!] \cup [\![e_1]\!] = [\![f_0]\!] \cup [\![f_1]\!] = [\![f]\!]$, where we have used that $[\![e_0]\!] = [\![f_0]\!]$ and $[\![e_1]\!] = [\![f_1]\!]$ by the induction hypothesis. ◀

▶ **Lemma 3.6.** *For all $\alpha, \beta \in \mathsf{State}$: $\alpha \in [\![\pi_\beta]\!]$ iff $\alpha \leq \beta$ iff $\pi_\alpha \leqq \pi_\beta$.*

Proof. Assume $\alpha \in [\![\pi_\beta]\!]$. Then for all $v \in \mathsf{dom}(\beta)$, $\alpha(v)$ is defined and $\alpha(v) = \beta(v)$, hence $\alpha \leq \beta$. Assume $\alpha \leq \beta$. Then $\pi_\alpha \leqq \pi_\beta$ is established from $\pi_\alpha \wedge \pi_\beta \equiv \pi_\alpha$: by the assumption, every conjunct in $\bigwedge_{\beta(v)=n} v = n$ appears as a conjunct in $\bigwedge_{\alpha(v)=n} v = n$, so by idempotence $\pi_\alpha \wedge \pi_\beta \equiv \left(\bigwedge_{\alpha(v)=n} v = n\right) \wedge \left(\bigwedge_{\beta(v)=n} v = n\right) \equiv \bigwedge_{\alpha(v)=n} v = n \equiv \pi_\alpha$. Finally, assume $\pi_\alpha \leqq \pi_\beta$. By soundness $[\![\pi_\alpha]\!] \subseteq [\![\pi_\beta]\!]$, and it is trivial to establish $\alpha \in [\![\pi_\alpha]\!]$. ◀

▶ **Lemma 3.7.** *For all $p \in \mathcal{O}$, we have $p \equiv \bigvee \{\alpha \in \mathsf{State} \mid \pi_\alpha \leqq p\}$.*

Proof. Noting that $\bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq p\} \leqq p$ by definition, we focus on the other inequality, proceeding by induction on p. For the base cases, $\bot \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq \bot\}$ by definition; $\top \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq \top\}$ as $\top \equiv \pi_\emptyset \in \{\alpha \in \mathsf{State} \mid \alpha \leqq \top\}$; and $v = n \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq v = n\}$ as $v = n \equiv \pi_{\{v \mapsto n\}} \in \{\alpha \in \mathsf{State} \mid \alpha \leqq v = n\}$.

In the induction step we have three cases.

- If $p = p_0 \wedge p_1$, by the inductive hypothesis and distributivity we obtain

$$p_0 \wedge p_1 \leqq \bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\}.$$

We claim that $\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\} \subseteq \{\alpha \mid \alpha \leqq p_0 \wedge p_1\} \cup \{\bot\}$. Call $\alpha, \beta \in \mathsf{State}$ *compatible* if, for any $v \in \mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta)$, $\alpha(v) = \beta(v)$. Take $\alpha \leqq p_0$ and $\beta \leqq p_1$. There are two cases. In the first, α and β are compatible. Then define γ by

$$\gamma(v) ::= \begin{cases} \alpha(v) & \text{if } \alpha(v) \text{ defined;} \\ \beta(v) & \text{if } \beta(v) \text{ defined;} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This is well-defined by compatability. Then $\gamma \leq \alpha$ as well as $\gamma \leq \beta$, and hence by Lemma 3.6 we find $\gamma \leqq \alpha \wedge \beta \leqq p_0 \wedge p_1$. In the other case, α and β are not compatible: hence for some distinct n and m, $v = n$ and $v = m$ are among the conjuncts of $\alpha \wedge \beta$. By the axiom $v = n \wedge v = m \equiv \bot$, it then follows that $\alpha \wedge \beta \equiv \bot$. We obtain

$$\bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\} \leqq \bigvee(\{\alpha \mid \alpha \leqq p_0 \wedge p_1\} \cup \{\bot\}) \equiv \bigvee\{\alpha \mid \alpha \leqq p_0 \wedge p_1\}.$$

- If $p = p_0 \vee p_1$, we derive

$$\begin{aligned} p_0 \vee p_1 &\leqq \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p_0\} \vee \bigvee\{\beta \in \mathsf{State} \mid \beta \leqq p_1\} && \text{(IH)} \\ &\leqq \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p_1 \vee p_2\} && (\alpha \leqq p_0 \leqq p_0 \vee p_1, \text{ similar for } \beta) \end{aligned}$$

- If $p = \overline{p_0}$, we derive

$$\begin{aligned} \overline{p_0} &\equiv \overline{\bigvee\{\alpha \mid \alpha \leqq p_0\}} && \text{(IH)} \\ &\equiv \bigwedge\{\overline{\alpha} \mid \alpha \leqq p_0\} && \text{(De Morgan)} \\ &\equiv \bigwedge\{\overline{\bigwedge_{\alpha(v)=n} v = n} \mid \alpha \leqq p_0\} && \text{(Definition of } \pi_\alpha = \alpha) \\ &\leqq \bigwedge\{\bigvee_{\alpha(v)=n} \overline{v = n} \mid \alpha \leqq p_0\} && \text{(De Morgan-like domain-specific axiom)} \\ &\leqq \bigwedge\{\bigvee_{\substack{\alpha(v)=n \\ m \neq n}} v = m \mid \alpha \leqq p_0\} && \text{(Pseudocomplement domain-specific axiom)} \end{aligned}$$

Note that the De Morgan law applied in the second step is indeed satisfied by PCDLs [3]. Now, define $K ::= \{\alpha \in \mathsf{State} \mid \alpha \leqq p_0\}$, $J_\alpha ::= \{(v, m) \mid \alpha(v) = n \neq m\}$, $J ::= \bigcup_{\alpha \in K} J_\alpha$ and $F ::= \{f : K \to J \mid \forall \alpha \in K, f(\alpha) \in J_\alpha\}$. Further, let

$$p_{\alpha,(v,m)} ::= \begin{cases} v = m & \text{if } \alpha(v) = n \neq m; \\ \bot & \text{otherwise} \end{cases}$$

Then

$$\bigwedge\{\bigvee_{\substack{\alpha(v)=n \\ m \neq n}} v = m \mid \alpha \leqq p_0\} \equiv \bigwedge_{\alpha \in K} \bigvee_{(v,m) \in J_\alpha} p_{\alpha,(v,m)} \equiv \bigvee_{f \in F} \bigwedge_{\alpha \in K} p_{\alpha, f(\alpha)}$$

by distributivity. For each $f \in F$, if the $p_{\alpha,f(\alpha)}$ are compatible, $\bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \equiv \beta_f$, for a β_f with the property that for every $\alpha \leqq p_0$, $\beta_f \wedge \alpha = \bot$ (as by definition, for each such α, β_f has some $v = m$ as a conjunct, where α has a conjunct $v = n$ for $n \neq m$). If they are incompatible, $\bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \equiv \bot$. Hence

$$\bigvee_{f\in F} \bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \leqq \bigvee\{\beta \mid \text{for all } \alpha, \alpha \leqq p_0 \text{ implies } \alpha \wedge \beta \equiv \bot\}.$$

For any β satisfying the property that for all $\alpha, \alpha \leqq p_0$ implies $\alpha \wedge \beta \equiv \bot$, we have $\beta \wedge p_0 \equiv \beta \wedge \bigvee\{\alpha \mid \alpha \leqq p_0\} \equiv \bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0\} \equiv \bot$, so $\beta \leqq \overline{p_0}$ and

$$\bigvee\{\beta \mid \text{for all } \alpha \leqq p_0 \text{ implies } \alpha \wedge \beta \equiv \bot\} \leqq \bigvee\{\beta \mid \beta \leqq \overline{p_0}\},$$

completing the proof. ◀

▶ **Theorem 3.8** (Completeness OA). *For all $p, q \in \mathcal{O}$, we have $p \equiv q$ if and only if $[\![p]\!] = [\![q]\!]$.*

Proof. The left-to-right direction follows from Lemma 3.5. For the right-to-left direction, suppose that $[\![p]\!] = [\![q]\!]$. By Lemma 3.7, we obtain that

$$[\![p]\!] = \left[\!\!\left[\bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p\}\right]\!\!\right] = \left[\!\!\left[\bigvee\{\beta \in \mathsf{State} \mid \beta \leqq q\}\right]\!\!\right] = [\![q]\!].$$

We prove $\alpha \leqq p$ if and only if $\alpha \leqq q$. Take $\alpha \leqq p$. Then $\alpha \in [\![\alpha]\!] \subseteq [\![p]\!] = \bigcup_{\beta\leqq q} [\![\beta]\!]$ by Lemmas 3.5 and 3.6. Hence for some $\beta \leqq q$, $\alpha \in [\![\beta]\!]$, and by Lemma 3.6 once more, $\alpha \leqq \beta \leqq q$. The other direction is symmetric. It follows that

$$p \equiv \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p\} \equiv \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq q\} \equiv q,$$

as required. ◀

B Proofs towards completeness

The following three results are all needed in the proofs that follow, and come from [12]. First of all, we can prove the following useful properties about the interaction between closure and other operators on pomset languages:

▶ **Lemma B.1.** *Let $L, K \subseteq \mathsf{Pom}$ and $C \in \mathsf{PC}$. The following hold.*

1. $L \subseteq K{\downarrow^H}$ *iff* $L{\downarrow^H} \subseteq K{\downarrow^H}$.
2. *If* $L \subseteq K$, *then* $L{\downarrow^H} \subseteq K{\downarrow^H}$.
3. $(L \cup K){\downarrow^H} = \left(L{\downarrow^H} \cup K{\downarrow^H}\right){\downarrow^H}$
4. $(L \cdot K){\downarrow^H} = \left(L{\downarrow^H} \cdot K{\downarrow^H}\right){\downarrow^H}$
5. $(L \parallel K){\downarrow^H} = \left(L{\downarrow^H} \parallel K{\downarrow^H}\right){\downarrow^H}$
6. $(L^*){\downarrow^H} = \left(\left(L{\downarrow^H}\right)^*\right){\downarrow^H}$
7. *If* $L{\downarrow^H} \subseteq K{\downarrow^H}$, *then* $C[L]{\downarrow^H} \subseteq C[K]{\downarrow^H}$.
8. *If* $L \subseteq \mathsf{SP}$, *then* $L{\downarrow^H} \subseteq \mathsf{SP}$.

Second, we can note the following result about the interaction between exch and contr:

▶ **Lemma B.2.** *For any $L \in 2^{\mathsf{SP}}$, we have $L{\downarrow^{\mathsf{contr}\cup\mathsf{exch}}} = (L{\downarrow^{\mathsf{exch}}}){\downarrow^{\mathsf{contr}}}$.*

We can then prove soundness of the POCKA semantics w.r.t. the axioms.

▶ **Theorem 3.14** (Soundness POCKA). *For all $e, f \in \mathcal{T}$, if $e \equiv f$ then $(\!|e|\!){\downarrow} = (\!|f|\!){\downarrow}$.*

Proof. By construction it is immediate that $(\!|-|\!){\downarrow}$ is closed under closure with respect to $\mathsf{exch} \cup \mathsf{contr}$. We then proceed by induction on $\equiv$. For all the pairs from $\equiv_{\mathsf{BKA}}$, it follows from Theorem 2.6 that $(\!|e|\!) = (\!|f|\!)$. Then immediately their POCKA-semantics also coincide. For all the pairs from $\equiv_{\mathsf{OA}}$, we make use of Theorem 3.8. Note that $(\!|-|\!){\downarrow}$ almost coincides

with $[\![-]\!]_{\mathsf{OA}}$ on $\mathcal{O}$, so the proof is very straightforward. For instance, take $p \vee (p \wedge q) \equiv_{\mathsf{OA}} p$. Then $(\!|p \vee (p \wedge q)|\!) = \mathsf{State}^* \cdot [\![p \vee (p \wedge q)]\!]_{\mathsf{OA}} \cdot \mathsf{State}^*$. From Theorem 3.8, we know that $[\![p \vee (p \wedge q)]\!]_{\mathsf{OA}} = [\![p]\!]_{\mathsf{OA}}$, and thus we obtain $(\!|p \vee (p \wedge q)|\!) = \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* = (\!|p|\!)$. Then from this we can conclude that $(\!|p \vee (p \wedge q)|\!)\downarrow = (\!|p \vee (p \wedge q)|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = (\!|p|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = (\!|p|\!)\downarrow$. We can prove soundness of the other observation algebra axioms analogously.

The next axiom is the exchange law. We show that $(\!|(e \parallel f) \cdot (g \parallel h)|\!)\downarrow \subseteq (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$. By Lemma B.1(1), it suffices to prove that $(\!|(e \parallel f) \cdot (g \parallel h)|\!) \subseteq (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$. Take an element in $(\!|(e \parallel f) \cdot (g \parallel h)|\!)$. This is thus a pomset of the form $(X \parallel Y) \cdot (V \parallel W)$ for $X \in (\!|e|\!)$, $Y \in (\!|f|\!)$, $V \in (\!|g|\!)$ and $W \in (\!|h|\!)$. Thus we immediately obtain that $(X \cdot V) \parallel (Y \cdot W) \in (\!|(e \cdot g) \parallel (f \cdot h)|\!)$. From Lemma B.2, we know that $(\!|-|\!)\downarrow = ((\!|-|\!)\downarrow^{\mathsf{exch}})\downarrow^{\mathsf{contr}}$. We know that $(X \parallel Y) \cdot (V \parallel W) \sqsubseteq (X \cdot V) \parallel (Y \cdot W)$ and that $(X \cdot V) \parallel (Y \cdot W) \in ((\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow^{\mathsf{exch}})\downarrow^{\mathsf{contr}}$. Then we can apply Lemmas 2.11 and B.2 to obtain that $(X \parallel Y) \cdot (V \parallel W) \in (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$.

Left to verify are the interface axioms. To check $p \wedge q \leqq p \cdot q$ it again suffices to prove that $(\!|p \wedge q|\!) \subseteq (\!|p \cdot q|\!)\downarrow$ by Lemma B.1(1). We take an element in $(\!|p \wedge q|\!)$. This a pomset of the form $U \cdot \alpha \cdot V$ such that $U, V \in \mathsf{State}^*$ and $\alpha \in [\![p]\!]_{\mathsf{OA}} \cap [\![q]\!]_{\mathsf{OA}}$. We can establish that $U \cdot \alpha \cdot \alpha \cdot V \in (\!|p \cdot q|\!)$. Now take the pomset context $C = U \cdot * \cdot V$. We have that $C[[\![\alpha \cdot \alpha]\!]] = \{U \cdot \alpha \cdot \alpha \cdot V\} \subseteq (\!|p \cdot q|\!) \subseteq (\!|p \cdot q|\!)\downarrow$. Then by closure we find $C[[\![\alpha]\!]] = \{U \cdot \alpha \cdot V\} \subseteq (\!|p \cdot q|\!)\downarrow$.

Since $(\!|\bot|\!) = \emptyset = (\!|0|\!)$, it follows that $(\!|\bot|\!)\downarrow = \emptyset = (\!|0|\!)\downarrow$. Similarly, since $(\!|p + q|\!) = (\!|p|\!) \cup (\!|q|\!) = (\!|p \vee q|\!)$, we also have that $(\!|p + q|\!)\downarrow = (\!|p \vee q|\!)\downarrow$

Now we have four axioms left. The first one we verify is $\top \cdot p \leqq p$ for $p \in \mathcal{O}$. We immediately obtain that $(\!|\top \cdot p|\!) = \mathsf{State} \cdot \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* \subseteq \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* = (\!|p|\!) \subseteq (\!|p|\!)\downarrow$. The axioms $\top \cdot p \leqq p$, $a \cdot \top \leqq a$, and $\top \cdot a \leqq a$ for $a \in \mathsf{Act}$ are all verified in a similar manner.

In the inductive step we need to check whether the closure rules for congruence have been preserved. We distinguish four cases.

- If $e = e_0 + e_1$ and $f = f_0 + f_1$ with $e_0 \equiv f_0$ and $e_1 \equiv f_1$, then by induction we know that $(\!|e_0|\!)\downarrow = (\!|f_0|\!)\downarrow$ and $(\!|e_1|\!)\downarrow = (\!|f_1|\!)\downarrow$. By Lemma B.1(3), we can then derive that

$$\begin{aligned}
(\!|e|\!)\downarrow &= ((\!|e_0|\!) \cup (\!|e_1|\!))\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|e_0|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \cup (\!|e_1|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}})\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|f_0|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \cup (\!|f_1|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}})\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|f_0|\!) \cup (\!|f_1|\!))\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= (\!|f|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}}
\end{aligned}$$

- The cases for $\cdot$, $\parallel$ and $*$ are argued similarly. ◀

▶ **Lemma 4.5.** *For all $e, f \in \mathcal{T}$, if $e \leq f \in \mathsf{top} \cup \mathsf{contr}$, then $e \leqq f$.*

Proof. If $e \leq f \in \mathsf{contr}$, then $e = \alpha$ and $f = \alpha \cdot \alpha$ for some $\alpha \in \mathsf{State}$. We can then derive that $\alpha \equiv \alpha \wedge \alpha \leqq \alpha \cdot \alpha$, using the PCDL and the Interface axioms. Hence $e \leqq f$. If $e \leq f \in \mathsf{top}$, then we distinguish two cases.

1. Let $e = \alpha \cdot c$ and $g = c$ for $c \in \mathsf{Act}$ and $\alpha \in \mathsf{State}$. Then we derive

$$\alpha \cdot c \leqq \top \cdot c \leqq c \qquad \text{(PCDL and Interface axioms)}$$

   The case where $e = c \cdot \alpha$ is similar.
2. Let $e = \alpha \cdot \beta$ and $g = \beta$ for $\alpha, \beta \in \mathsf{State}$. Then we derive

$$\alpha \cdot \beta \leqq \top \cdot \beta \leqq \beta \qquad \text{(PCDL and Interface axioms)}$$

   The case where $e = \beta \cdot \alpha$ is similar. Hence, we can conclude that $e \leqq f$. ◀

C Proofs about the litmus test

▶ **Lemma 6.2.** *Let $U = [\mathbf{u}] \in \mathsf{SP}$. If $P(U)$ then U is not guarded.*

Proof. We prove by contradiction; assume that $P(U)$ and that U is guarded. Via Theorem 5.9 we conclude that U satisfies (A1)–(A7). From $P(U)$ we infer that there exists $u_1, u_2, w \in S_{\mathbf{u}}$ such that $\lambda_{\mathbf{u}}(u_1) = (x \leftarrow 1)$, $u_1 \leq w$ and $\lambda_{\mathbf{u}}(w)(r_0) = 0 = \lambda_{\mathbf{u}}(w)(r_1)$. From (A2) and (A5), we infer that u_1 has a unique successor node $s_1 \in S_{\mathbf{u}}$ such that s_1 is state-labelled, and $\lambda_{\mathbf{u}}(s_1)(x) = 1$. From (A4) there exists a path for x from s_1 to w. Hence, if $\lambda_{\mathbf{u}}(w)(x) \neq 1$, there must be at least one assignment between s_1 and w altering the value of x, as the path must explain how the value of x changed from 1 to 0. Hence, there exists a node $u_3 \in S_{\mathbf{u}}$ such that $s_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{u}} w$ and $\lambda_{\mathbf{u}}(u_3) = (x \leftarrow n)$ for $n \in \textsc{Var} \cup \textsc{Val}$. However, from property P, we know that all such assignments occur before u_1, and thereby strictly before s_1. From this we can conclude that $\lambda_{\mathbf{u}}(w)(x) = 1$. Similarly, we obtain $\lambda_{\mathbf{u}}(w)(y) = 1$.

From (A2) and (A6), we know that v_1 has a unique successor node t_1, such that $\lambda_{\mathbf{u}}(t_1)(r_0) = \lambda_{\mathbf{u}}(t_1)(y)$. Then from (A4), there must be a path for r_0 from t_1 to w. With similar reasoning as for x above, we obtain $\lambda_{\mathbf{u}}(t_1)(r_0) = \lambda_{\mathbf{u}}(w)(r_0) = 0$. Similarly, we obtain a successor node t_2 of v_2 such that $\lambda_{\mathbf{u}}(t_2)(r_1) = \lambda_{\mathbf{u}}(t_2)(x) = 0$.

As we have that $\lambda_{\mathbf{u}}(t_1)(y) = 0$ and $\lambda_{\mathbf{u}}(w)(y) = 1$ and $t_1 \leq_{\mathbf{u}} w$, we can conclude from (A4) that there must be a path from t_1 to w for y such that this path contains at least one assignment that alters the value for y. Thus, there exists a node u_3 such that $t_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{w}} w$ and u_3 has a label that changes the value of y. Similarly, we obtain a node u_4 such that $t_2 \leq_{\mathbf{u}} u_4 \leq_{\mathbf{w}} w$ and u_4 changes the value of x. From property P, we obtain $u_3 \leq_{\mathbf{u}} u_2$ and $u_4 \leq_{\mathbf{u}} u_1$. Then, making use of the fact that t_1 and t_2 are the successors of v_1 and v_2 respectively, we can derive: $v_2 \leq_{\mathbf{u}} t_2 \leq_{\mathbf{u}} u_4 \leq_{\mathbf{u}} u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} t_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{u}} u_2 \leq_{\mathbf{u}} v_2$ Then, by antisymmetry, all these nodes are equivalent. As they cannot be, we have a contradiction. Hence, U is not a guarded pomset. Hence, U is not a guarded pomset. ◀

▶ **Lemma 6.5.** *Let $e \in \mathcal{T}$. If $\forall U \in (\!|e|\!)$ we have $P(U)$, then $\forall V \in (\!|e|\!)\downarrow$ it holds that $P(V)$.*

Proof. First, note that $(\!|e|\!)\downarrow = (\!|e|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = \big((\!|e|\!)\downarrow^{\mathsf{exch}}\big)\downarrow^{\mathsf{contr}}$ by Lemma B.2. Thus, if $V \in (\!|e|\!)\downarrow$, we apply Lemma 6.4, to infer that there exists a pomset $W \in (\!|e|\!)\downarrow^{\mathsf{exch}}$ such that $V \preceq W$. Next we can apply Lemma 2.11, to obtain a pomset $U \in (\!|e|\!)$ such that $W \sqsubseteq U$. We know that U has property P. We first show that W also has property P, and then that the same holds for V. From the definition of $\sqsubseteq$ we get that there exists a bijective pomset morphism h from W to U. Thus we have $U = [\mathbf{u}]$ and $W = [\mathbf{w}]$ and a bijective function $h \colon S_{\mathbf{u}} \to S_{\mathbf{w}}$ such that $\lambda_{\mathbf{w}} \circ h = \lambda_{\mathbf{u}}$ and if $u \leq_{\mathbf{u}} u'$ then $h(u) \leq_{\mathbf{w}} h(u')$. Now we need to verify the two properties of Definition 6.1.

1. As $\lambda_{\mathbf{w}}(h(u_1)) = \lambda_{\mathbf{u}}(u_1)$, we get $\lambda_{\mathbf{w}}(h(u_1)) = (x \leftarrow 1)$. The same for the other existential statements of Item 1. For the ordering: from $u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} w$ we immediately obtain that $h(u_1) \leq_{\mathbf{w}} h(v_1) \leq_{\mathbf{w}} h(w)$ and similarly for $h(u_2) \leq_{\mathbf{w}} h(v_2) \leq_{\mathbf{w}} h(w)$.
2. Take a z such that $\lambda_{\mathbf{w}}(z) = (x \leftarrow n)$ for $n \in \textsc{Val} \cup \textsc{Var}$. As h is surjective, we know there exists a node $s \in S_{\mathbf{u}}$ such that $h(s) = z$ and $\lambda_{\mathbf{w}}(h(s)) = \lambda_{\mathbf{u}}(s)$. As $P(U)$, we get that $s \leq_{\mathbf{u}} u_1$. Hence, $h(s) \leq_{\mathbf{w}} h(u_1)$ and thus $z \leq_{\mathbf{w}} h(u_1)$. An analogue argument can be given for the other conditions in property (2).

This demonstrates that W has property P. We know that $V \preceq W$, and we will show this implies that V also has property P. From the definition of $\preceq$ we know that there exists a pomset morphism h from V to W. The argument to verify the two properties of Definition 6.1 is exactly the same as above. Hence we can conclude that V has property P. ◀