Automating Memory Model Metatheory with Intersections

Aristotelis Koutsouridis
MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Michalis Kokologiannakis
MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Viktor Vafeiadis
MPI-SWS, Kaiserslautern and Saarbrücken, Germany

Abstract

In the weak memory consistency literature, the semantics of concurrent programs is typically defined as a constraint on execution graphs, expressed in relational algebra. Prior work has shown that basic metatheoretic questions about memory models are decidable as long as they can be expressed as irreflexivity and emptiness constraints over Kleene Algebra with Tests (KAT), a condition that rules out practical memory models such the C/C++ and the Linux kernel models.

In this paper, we extend these results to memory models containing arbitrary intersections with uninterpreted relations. We can thus automatically establish compilation correctness and derive efficient incremental consistency checkers for RC11, LKMM, and other memory models.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Concurrency

Keywords and phrases Kleene Algebra, Weak Memory Models

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2024.33

Supplementary Material *Text (Full paper with technical appendix)*: `https://plv.mpi-sws.org/kater`

Funding This work was supported by a European Research Council (ERC) Consolidator Grant for the project "PERSIST" under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101003349).

Acknowledgements We would like to thank the anonymous reviewers for their feedback.

1 Introduction

In the weak memory consistency literature, the semantics of a concurrent and/or distributed program is typically defined as a set of labeled directed graphs, each representing a single possible execution of the program. These execution graphs comprise a set of nodes recording the individual memory accesses performed and a set of edges recording various ordering constraints among them. Example constraints [2] include the *program order* (`po`), the *reads-from* relation (`rf`), and the *coherence order* (`co`).

Each memory model defines a "consistency" constraint on execution graphs, asserting which graphs are possible outcomes of any program. These constraints are conveniently expressed in relational algebra with the help of some additional built-in sets (e.g., the set of read events `R`, and the set of write events `W`) and relations (e.g., `sameloc` relating events accessing the same memory location, and `diffthread` relating events originating from different threads). For example, *sequential consistency* (SC) [18] can be defined as the constraint (SC) in Fig. 1; *coherence* (a.k.a., SC-per-location) as (COH), or equivalently as (COH2); *release-acquire* (RA) as (RA), or equivalently as (RA2), or equivalently as the conjunction of (COH) and (RA3); and *Total Store Order* (TSO) [22] as the conjunction of (COH) and (TSO).

35th International Conference on Concurrency Theory (CONCUR 2024).
Editors: Rupak Majumdar and Alexandra Silva; Article No. 33; pp. 33:1–33:16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$$\mathsf{irreflexive}((\mathsf{po} \cup \mathtt{rf} \cup \mathtt{co} \cup \mathtt{fr})^+) \quad \text{where } \mathtt{fr} \triangleq \mathtt{rf}^{-1}; \mathtt{co} \tag{SC}$$

$$\mathsf{irreflexive}((\mathsf{po} \cap \mathtt{sameloc} \cup \mathtt{rf} \cup \mathtt{co} \cup \mathtt{fr})^+) \tag{COH}$$

$$\mathsf{irreflexive}(\mathsf{po}^?; (\mathtt{rf} \cup \mathtt{co} \cup \mathtt{fr})^+) \tag{COH2}$$

$$\mathsf{irreflexive}(((\mathsf{po} \cup \mathtt{rf})^+ \cap \mathtt{sameloc} \cup \mathtt{co} \cup \mathtt{fr})^+) \tag{RA}$$

$$\mathsf{irreflexive}((\mathsf{po} \cup \mathtt{rf})^+; (\mathtt{co} \cup \mathtt{fr})^?) \tag{RA2}$$

$$\mathsf{irreflexive}((\mathsf{ppo} \cup \mathtt{rf})^+; (\mathtt{co} \cup \mathtt{fr})^?) \quad \text{where } \mathsf{ppo} \triangleq \mathsf{po} \setminus (\mathtt{W} \times \mathtt{R}) \tag{RA3}$$

$$\mathsf{irreflexive}((\mathsf{ppo} \cup \mathtt{rf} \cap \mathtt{diffthread} \cup \mathtt{co} \cup \mathtt{fr})^+) \tag{TSO}$$

Figure 1 Sample consistency constraints.

Kokologiannakis et al. [14] present KATER, a framework that can automatically answer certain fundamental questions about such definitions, but only for the case where the models are expressed purely as irreflexivity constraints over *Kleene Algebra with Tests* (KAT) [16]. This restriction to KAT, however, is a severe limitation of KATER: many common model definitions do not fall into this fragment (e.g., COH, RA, TSO), and although some of the simpler definitions can be equivalently expressed in KAT, more advanced practical models such as RC11 [17] and the Linux kernel memory model (LKMM) [1], cannot.

In response, we present KATI, an extension of KAT with *intersections* with *uninterpreted relations*, as well as a *top element*. KATI can express terms like $- \cap \mathtt{sameloc}$ and $- \cap \mathtt{diffthread}$ in Fig. 1, and supports all the aforementioned memory models. However, KATI also makes answering the following questions more difficult:

(Incremental) consistency checking: Is a given execution graph G consistent according to a model M? Moreover, given an execution graph G and an event $e \in G$ such that $G \setminus \{e\}$ is M-consistent, is G also M-consistent?

Inclusion: Is memory model A *stronger* than a memory model B, i.e., does the consistency predicate of A imply that of B?

Incremental consistency checking is important for testing and automated verification of concurrent programs (e.g., via stateless model checking [7, 15]). The problem admits a straightforward cubic solution (in the size of the execution graph) that calculates the relation appearing in the irreflexivity constraints in a bottom-up fashion. For acyclicity constraints of KAT expressions, KATER provides a better solution of linear complexity: it performs a custom DFS of the cross product of the execution graph with a finite state automaton corresponding to the KAT expression. We extend KATER's linear-time solution to KATI with register automata [11], which extend standard finite state automata with a finite set of registers, which can store arbitrary values and compare them for equality.

Inclusion is not only an important metatheoretical question, but it actually also underlies the correctness proofs of compilation from one model to another and of local program transformations (compiler optimizations). Unfortunately, however, we cannot simply use our encoding into register automata because inclusion between register automata is generally undecidable [11]. We therefore follow another approach, and reduce relational intersection to KAT expressions over an extended alphabet with additional "bracket" letters. We prove that the resulting inclusion algorithm remains decidable (PSPACE-complete for a bounded number of intersections).

Our contributions can be summarized as follows:

§2 We review KAT and show how it encodes consistency constraints of weak memory models.

§3–§5 We present KATI, an extension of KAT that supports intersections with primitive relations, prove equivalence between its relational and language interpretation, and provide a decision procedure for language inclusion based on NFAs.

§6 We show how KATI can be used to check consistency of execution graphs in linear time. We conclude the paper with a presentation of related work (§7) and a note about future work (§8).

2 Kleene Algebra with Tests

In this section, we review the syntax and semantics of *Kleene Algebra with Tests* (KAT) [16].

2.1 Syntax and Interpretation

Syntax. KAT has two kinds of terms: tests and expressions.

Tests, $t \in \mathsf{Test}$, form a boolean algebra over a set of primitive predicates, $p \in \mathsf{P}$, i.e., they are constructed using the standard boolean/set operators: true ($\top$), false ($\bot$), union ($\cup$), intersection ($\cap$), and complement ($\bar{\ }$).

$$t ::= \top \mid \bot \mid p \mid t_1 \cup t_2 \mid t_1 \cap t_2 \mid \bar{t}$$

Expressions, $e \in \mathsf{KAT}$, form a Kleene algebra over primitive relations, $r \in \mathsf{R}$, and tests; i.e., they are constructed using relational composition (sequencing), union, and repetition.

$$e ::= r \mid [t] \mid e_1 \,;\, e_2 \mid e_1 \cup e_2 \mid e^*$$

Unlike plain Kleene Algebra, KAT does not need special constructs for the empty string and the empty set, as these are given by the KAT expressions $[\top]$ and $[\bot]$ respectively.

Relational Interpretation. KAT terms can be interpreted in the context of a graph G, which defines the interpretations of primitive tests and relations. Formally, a graph G is a tuple $\langle \mathtt{E}_G, \mathcal{I}_G^{\mathsf{P}}, \mathcal{I}_G^{\mathsf{R}} \rangle$ where $\mathtt{E}_G$ is a set of nodes (events) and $\mathcal{I}_G^{\mathsf{P}}$ is a function interpreting primitive tests over subsets of $\mathtt{E}_G$ and $\mathcal{I}_G^{\mathsf{R}}$ primitive relations over binary relations on $\mathtt{E}_G$.

$$\mathcal{I}_G^{\mathsf{P}} : \mathsf{P} \to \mathcal{P}(\mathtt{E}_G) \qquad \mathcal{I}_G^{\mathsf{R}} : \mathsf{R} \to \mathcal{P}(\mathtt{E}_G \times \mathtt{E}_G)$$

We extend these interpretations to arbitrary KAT terms as follows:

$$\begin{aligned}
[\![p]\!]_G &\triangleq \mathcal{I}_G^{\mathsf{P}}(p) & [\![r]\!]_G &\triangleq \mathcal{I}_G^{\mathsf{R}}(r) \\
[\![\top]\!]_G &\triangleq \mathtt{E}_G & [\![[t]]\!]_G &\triangleq \big\{ \langle a, a \rangle \;\big|\; a \in [\![t]\!]_G \big\} \\
[\![\bot]\!]_G &\triangleq \emptyset & [\![e_1; e_2]\!]_G &\triangleq \big\{ \langle a, c \rangle \;\big|\; \exists b.\, \langle a, b \rangle \in [\![e_1]\!]_G \wedge \langle b, c \rangle \in [\![e_2]\!]_G \big\} \\
[\![\bar{t}]\!]_G &\triangleq \mathtt{E}_G \setminus [\![t]\!]_G & [\![e^*]\!]_G &\triangleq ([\![e]\!]_G)^* \\
[\![t_1 \cup t_2]\!]_G &\triangleq [\![t_1]\!]_G \cup [\![t_2]\!]_G & [\![e_1 \cup e_2]\!]_G &\triangleq [\![e_1]\!]_G \cup [\![e_2]\!]_G \\
[\![t_1 \cap t_2]\!]_G &\triangleq [\![t_1]\!]_G \cap [\![t_2]\!]_G & &
\end{aligned}$$

Language Interpretation. The main property of KAT is that inclusion and equivalence between KAT expressions is *decidable* (PSPACE-complete). This can be shown either with an algebraic axiomatization of KAT [16] or, as we show below, via an equivalent model of KAT expressions as a regular language.

Specifically, KAT expressions can be seen as regular languages over *guarded strings*, which we shall define below. To do so, we first define the *atoms* of a set of primitive tests.

▶ **Definition 1** (Atom). *An atom over* $\mathsf{P} = \{p_1, \dots, p_k\}$ *is a string of literals* $c_1 c_2 \dots c_k$ *such that* $c_i \in \{p_i, \overline{p_i}\}, 1 \leq i \leq k$*. Furthermore, the set of all* 2^k *atoms over* P *is denoted* A_{P}*.*

We use the greek lowercase letters $\alpha, \beta, \ldots$ to denote atoms. For an atom α and a test t we write $\alpha \leq t$ to denote that $\alpha \rightarrow t$ is a propositional tautology.

▶ **Definition 2** (Guarded String). *A guarded string is a string over* $\mathsf{GS} \triangleq (\mathsf{A_P}; \mathsf{R})^*; \mathsf{A_P}$*, i.e., consists of a non-empty, alternating sequence of atoms and primitive relations, starting and ending with an atom.*

Concatenation and Kleene closure can be lifted to languages of guarded strings:

$$X \fatsemi Y \triangleq \{u \cdot \alpha \cdot v \mid u \cdot \alpha \in X, \alpha \cdot v \in Y\}$$
$$X^{(0)} \triangleq \mathsf{A_P} \qquad X^{(n+1)} \triangleq X \fatsemi X^{(n)} \qquad X^{\circledast} \triangleq \bigcup_{n \geq 0} X^{(n)}$$

Observe that concatenation is guarded, i.e., it is only defined if the two strings are composable.

The language interpretation, $[\![.]\!]_\mathsf{L}$, maps tests to sets of atoms and KAT expressions to (regular) sets of guarded strings.

$$
\begin{aligned}
[\![p]\!]_\mathsf{L} &\triangleq \{\alpha \in \mathsf{A_P} \mid \alpha \leq p\} & [\![r]\!]_\mathsf{L} &\triangleq \{\alpha \cdot r \cdot \beta \mid \alpha, \beta \in \mathsf{A_P}\} \\
[\![\top]\!]_\mathsf{L} &\triangleq \mathsf{A_P} & [\![[t]]\!]_\mathsf{L} &\triangleq [\![t]\!]_\mathsf{L} \\
[\![\bot]\!]_\mathsf{L} &\triangleq \emptyset & [\![e_1 \cup e_2]\!]_\mathsf{L} &\triangleq [\![e_1]\!]_\mathsf{L} \cup [\![e_2]\!]_\mathsf{L} \\
[\![\bar{t}]\!]_\mathsf{L} &\triangleq \mathsf{A_P} \setminus [\![t]\!]_\mathsf{L} & [\![e_1 \,;\, e_2]\!]_\mathsf{L} &\triangleq [\![e_1]\!]_\mathsf{L} \fatsemi [\![e_2]\!]_\mathsf{L} \\
[\![t_1 \cup t_2]\!]_\mathsf{L} &\triangleq [\![t_1]\!]_\mathsf{L} \cup [\![t_2]\!]_\mathsf{L} & [\![e^*]\!]_\mathsf{L} &\triangleq ([\![e]\!]_\mathsf{L})^{\circledast} \\
[\![t_1 \cap t_2]\!]_\mathsf{L} &\triangleq [\![t_1]\!]_\mathsf{L} \cap [\![t_2]\!]_\mathsf{L}
\end{aligned}
$$

2.2 Interpretation Equivalence

The language and relational interpretations of KAT expressions are equivalent in the sense that e_1 is included in e_2 according to the one interpretation if and only if it is included according to the other.

▶ **Theorem 3** (Interpretation Equivalence). $[\![e_1]\!]_\mathsf{L} \subseteq [\![e_2]\!]_\mathsf{L}$ *if and only if* $\forall G.\, [\![e_1]\!]_G \subseteq [\![e_2]\!]_G$.

Proof sketch. For the "⇒" direction, we define a function $\rho_G : \mathsf{GS} \rightarrow \mathcal{P}(\mathtt{E}_G \times \mathtt{E}_G)$ that interprets guarded strings as relations on a graph G as follows:

$$\rho_G(\alpha) \triangleq \big\{\langle a, a\rangle \bigm| a \in [\![\alpha]\!]_G\big\} \qquad\qquad \rho_G(\alpha \cdot r \cdot w) \triangleq \rho_G(\alpha) \,;\, [\![r]\!]_G \,;\, \rho_G(w)$$

Here, $[\![\alpha]\!]_G$ interprets the atom α as the composition of its primitive tests. We show that $[\![e]\!]_G = \bigcup_{w \in [\![e]\!]_\mathsf{L}} \rho_G(w)$ (by induction on e). Then,

$$[\![e_1]\!]_G = \bigcup_{w \in [\![e_1]\!]_\mathsf{L}} \rho_G(w) \subseteq \bigcup_{w \in [\![e_2]\!]_\mathsf{L}} \rho_G(w) = [\![e_2]\!]_G\,.$$

For the "⇐" direction, from a word $w \in [\![e_1]\!]_\mathsf{L}$, we construct a "canonical" graph G_w as a sequence of nodes $n_0, \ldots, n_k$, such that the only guarded string w' such that $\langle n_0, n_k\rangle \in \rho_{G_w}(w')$ is $w' = w$. Then it follows that $\langle n_0, n_k\rangle \in [\![e_1]\!]_{G_w} \subseteq [\![e_2]\!]_{G_w}$, and thus $w \in [\![e_2]\!]_\mathsf{L}$. ◀

Deciding Language Inclusion with NFAs. When deciding the inclusion $[\![e_1]\!]_\mathsf{L} \subseteq [\![e_2]\!]_\mathsf{L}$, it is convenient to use NFAs that accept guarded strings.

▶ **Definition 4.** *An NFA over an alphabet* Σ *is a tuple* $\langle Q, \iota, F, \delta\rangle$*, where* Q *is the set of states,* $\iota \in Q$ *is the initial state,* $F \subseteq Q$ *is the set of final states, and* $\delta \subseteq Q \times \Sigma \times Q$ *is the transition relation.*

Given an NFA, we abuse notation and write $\delta(S,a)$ for the set $\{q \in Q \mid \exists s \in S.\, \langle s,a,q\rangle \in \delta\}$. We also lift the transition relation to words as follows $\delta(S,\epsilon) \triangleq S$, and $\delta(S,aw) \triangleq \delta(\delta(S,a),w)$.

The *language* accepted by an NFA contains all words accepted by the NFA: $\mathsf{L}(\langle Q,\iota,F,\delta\rangle) \triangleq \{w \in \Sigma \mid \delta(\{\iota\},w) \cap F \neq \emptyset\}$.

Let us now define the function $[\![-]\!]_{\mathsf{NFA}}$ to convert an expression $e \in \mathsf{KAT}$ to an NFA over the alphabet of atoms and primitive relations: $\Sigma \triangleq \mathsf{A_P} \cup \mathsf{R}$.

$$
\begin{aligned}
[\![r]\!]_{\mathsf{NFA}} &\triangleq \langle \{q_0,q_1,q_2,q_3\}, q_0, \{q_3\}, \{\langle q_1,r,q_2\rangle\} \cup \bigcup_{\alpha\in\mathsf{A_P}} \{\langle q_0,\alpha,q_1\rangle, \langle q_2,\alpha,q_3\rangle\}\rangle \\
[\![[t]]\!]_{\mathsf{NFA}} &\triangleq \langle \{q_0,q_1\}, q_0, \{q_1\}, \{\langle q_0,\alpha,q_1\rangle \mid \alpha \in [\![t]\!]_{\mathsf{L}}\}\rangle \\
[\![e_1;e_2]\!]_{\mathsf{NFA}} &\triangleq \langle Q_1 \uplus Q_2, \iota_1, F_2, \delta_1 \cup \delta_2 \cup \{\langle q_1,\alpha,q_2\rangle \mid \delta_1(q_1,\alpha) \in F_1 \wedge (\iota_2,\alpha,q_2) \in \delta_2\}\rangle \\
&\qquad \textbf{where } [\![e_i]\!]_{\mathsf{NFA}} = \langle Q_i,\iota_i,F_i,\delta_i\rangle \textbf{ for } i \in \{1,2\} \\
[\![e_1 \cup e_2]\!]_{\mathsf{NFA}} &\triangleq \langle Q_1 \uplus Q_2, \iota_1, F_1 \cup F_2, \delta_1 \cup \delta_2 \cup \{\langle \iota_1,\alpha,q_2\rangle \mid \langle \iota_2,\alpha,q_2\rangle \in \delta_2\}\rangle \\
&\qquad \textbf{where } [\![e_i]\!]_{\mathsf{NFA}} = \langle Q_i,\iota_i,F_i,\delta_i\rangle \textbf{ for } i \in \{1,2\} \\
[\![e^*]\!]_{\mathsf{NFA}} &\triangleq \langle Q \uplus \{q\}, \iota, F \cup \{q\}, \delta \cup \{\langle q_2,\alpha,q_1\rangle \mid \langle \iota,\alpha,q_1\rangle \in \delta, \langle q_2,\alpha,q_F\rangle \in \delta, q_F \in F\} \\
&\qquad\qquad \cup \{\langle \iota,\alpha,q\rangle \mid \alpha \in \mathsf{A_P}\}\rangle \\
&\qquad \textbf{where } [\![e]\!]_{\mathsf{NFA}} = \langle Q,\iota,F,\delta\rangle
\end{aligned}
$$

By construction, the function $[\![-]\!]_{\mathsf{NFA}}$ creates an NFA that accepts only guarded strings. In fact, $[\![e]\!]_{\mathsf{NFA}}$ accepts precisely the words in $[\![e]\!]_{\mathsf{L}}$.

▶ **Proposition 5** (NFA Equivalence). *For all $e \in \mathsf{KAT}$, $[\![e]\!]_{\mathsf{L}} = \mathsf{L}([\![e]\!]_{\mathsf{NFA}})$.*

Language inclusion between KAT expressions can thus be checked via NFA automata and is PSPACE-complete.

2.3 Memory Models as KAT Constraints

Kokologiannakis et al. [14] observe that declarative *memory models M* can be formulated as a pair $\langle e_\emptyset, e_{\mathrm{irr}}\rangle$ of an emptiness and an irreflexivity constraint over KAT. A memory model is interpreted as a set of execution graphs as follows $[\![\langle e_\emptyset, e_{\mathrm{irr}}\rangle]\!] \triangleq \{G \mid [\![e_\emptyset]\!]_G \cup [\![e_{\mathrm{irr}}]\!]_G \cap \mathsf{id} = \emptyset\}$, where $\mathsf{id} \triangleq \{\langle x,x\rangle \mid x \in \mathsf{E}_G\}$ is the identity relation.

Crucially, Kokologiannakis et al. [14] prove that various metatheoretic properties about memory models (such properties boil down to irreflexivity implications) can be decided in a sound and complete fashion:

▶ **Theorem 6** (Kater). *For every $e_1,e_2 \in \mathsf{KAT}$, $\mathsf{sameEnds}([\![e_1]\!]_{\mathsf{L}}) \subseteq \mathsf{DEDUP}(\mathsf{ROT}([\![e_2]\!]_{\mathsf{L}}))$ if and only if for all G, $\mathsf{irreflexive}([\![e_2]\!]_G)$ implies $\mathsf{irreflexive}([\![e_1]\!]_G)$.*

In the theorem above, $\mathsf{sameEnds}(L) \triangleq \{\alpha\cdot v\cdot\alpha \mid \alpha\cdot v\cdot\alpha \in L\}$ restricts L so that its end-points are compatible, $\mathsf{ROT}(L) \triangleq \{\alpha\cdot u\cdot\beta\cdot v\cdot\alpha \mid \beta\cdot v\cdot\alpha\cdot u\cdot\beta \in L\}$ is the rotation closure of L, and $\mathsf{DEDUP}(L) \triangleq \{\alpha\cdot w\cdot\alpha \mid \exists n.\, (\alpha\cdot w)^n\cdot\alpha \in L\}$ the deduplication closure.

Kokologiannakis et al. [14] further observe that the deduplication closure is never needed in practice, and so their tool, Kater, simply checks $\mathsf{sameEnds}([\![e_1]\!]_{\mathsf{L}}) \subseteq \mathsf{ROT}([\![e_2]\!]_{\mathsf{L}})$.

3 KATI: Kleene Algebra with Tests and Intersections

In this section, we present our extension of KAT with relational intersection. KATI (Kleene Algebra with Tests and Intersections) extends KAT with relational intersection with *intersection relations*, $ir \in \mathsf{IR}$, with the standard relational interpretation.

$$e \in \mathsf{KATI} ::= \ldots \mid e \cap ir \qquad\qquad [\![e \cap ir]\!]_G \triangleq [\![e]\!]_G \cap [\![ir]\!]_G$$

In this section, for simplicity, we assume that the set of primitive relations R and the set of intersection relations IR are disjoint. We will later lift this assumption in §5.

3.1 Language Interpretation

To show that inclusion between KATI expressions remains decidable, we need to suitably extend the language interpretation. To do so, we cannot employ the usual interpretation of intersection between formal languages because $[\![r]\!]_{\mathsf{L}} \cap [\![ir]\!]_{\mathsf{L}} = \{\alpha \cdot r \cdot \beta \mid \alpha, \beta \in \mathsf{A_P}\} \cap \{\alpha \cdot ir \cdot \beta \mid \alpha, \beta \in \mathsf{A_P}\} = \emptyset$.

Our idea is to introduce a set of bracket symbols $\mathsf{IR}_{()} \triangleq \bigcup_{ir \in \mathsf{IR}} \{(_{ir},)_{ir}\}$ and interpret intersections as well-bracketed words over $\mathsf{IR}_{()} \cup \mathsf{R} \cup \mathsf{A_P}$. Note, however, that we cannot simply interpret $e \cap ir$ as $\{(_{ir} \cdot w \cdot)_{ir} \mid w \in [\![e]\!]_{\mathsf{L}}\}$, as such an interpretation fails to validate the following four important equivalences between KATI expressions that hold according to the relational interpretation.

$$(e \cap ir) \cap ir = e \cap ir \qquad\qquad (e \cap ir) \cap ir' = (e \cap ir') \cap ir$$
$$([t] \,;\, e) \cap ir = [t] \,;\, (e \cap ir) \qquad\qquad (e \,;\, [t]) \cap ir = (e \cap ir) \,;\, [t]$$

Idempotence fails because the LHS has more brackets the RHS, while the commutativity properties fail because the brackets (and the tests) appear in different orders. In addition, we sometimes want intersection relations, such as $\mathsf{sameloc}$, to be reflexive, in which case we would like to support the equivalence $[t] \cap ir = [t]$.

To resolve these problems, we assume a total order $\prec$ on IR and a function[1] $\mathsf{id} : \mathsf{IR} \to \mathsf{Test}$ such that $[\![[\mathsf{id}(ir)]]\!]_G = [\![[\top] \cap ir]\!]_G$. Then, we can extend the notion of guarded strings to enforce a number of well-formed properties: (1) ignoring bracket symbols, words form an non-empty alternating sequence of atoms and primitive relations, starting and ending with an atom; (2) brackets are properly nested; (3) words inside brackets do not start or end with an atom, (4) directly nested brackets are sorted according to $\prec$. To do so, we introduce a set PGS of non-empty words indexed by a set $S \subseteq \mathsf{IR}$ constraining any end-to-end bracket symbol to be indexed by an intersection relation in S,

$$\begin{aligned} \mathsf{PGS}_S &\triangleq \{r \mid r \in \mathsf{R}\} \cup \{w_1 \cdot \alpha \cdot w_2 \mid w_1, w_2 \in \mathsf{PGS_{IR}}, \alpha \in \mathsf{A_P}\} \\ &\quad \cup \{(_{ir} \cdot w \cdot)_{ir} \mid ir \in S, w \in \mathsf{PGS}_{\{\succ ir\}}\} \\ \mathsf{GS} &\triangleq \{\alpha \mid \alpha \in \mathsf{A_P}\} \cup \{\alpha \cdot w \cdot \beta \mid \alpha, \beta \in \mathsf{A_P}, w \in \mathsf{PGS_{IR}}\} \end{aligned}$$

where $\{\succ ir\} \triangleq \{ir' \mid ir \prec ir'\}$.

Note that given $w \in \mathsf{PGS_{IR}}$ and $ir \in \mathsf{IR}$, there exist $u \in (_{\{\prec ir\}}{}^*$, $v \in (_{ir}^?$, $w' \in \mathsf{PGS}_{\{\succ ir\}}$ such that $w = u \cdot v \cdot w' \cdot \overline{v} \cdot \overline{u}$, where for a sequence of opening brackets u, we write $\overline{u}$ for the corresponding sequence of closing brackets such that $u \cdot \overline{u}$ is well-nested.

We extend the language interpretation of KAT to KATI as follows:

$$\begin{aligned} [\![e \cap ir]\!]_{\mathsf{L}} &\triangleq \left\{ \alpha \cdot u \cdot (_{ir} \cdot w \cdot)_{ir} \cdot \overline{u} \cdot \beta \;\middle|\; \begin{array}{l} \alpha \cdot u \cdot v \cdot w \cdot \overline{v} \cdot \overline{u} \cdot \beta \in [\![e]\!]_{\mathsf{L}}, \\ u \in (_{\{\prec ir\}}{}^*, v \in (_{ir}^?, w \in \mathsf{PGS}_{\{\succ ir\}} \end{array} \right\} \\ &\quad \cup \{\alpha \mid \alpha \in [\![e]\!]_{\mathsf{L}},\ \alpha \in [\![\mathsf{id}(ir)]\!]_{\mathsf{L}}\} \end{aligned}$$

Using this definition, one can show that inclusion of the language interpretation implies inclusion of the relational interpretation.

[1] Such a function can always be defined by extending P with additional primitive tests if necessary.

▶ **Proposition 7.** *For all KATI expressions* e_1, e_2*, if* $[\![e_1]\!]_{\mathsf{L}} \subseteq [\![e_2]\!]_{\mathsf{L}}$*, then* $\forall G.\, [\![e_1]\!]_G \subseteq [\![e_2]\!]_G$.

Proof sketch. The conclusion follows by showing that $[\![e]\!]_G = \bigcup_{w \in [\![e]\!]_{\mathsf{L}}} \rho_G(w)$, where the function $\rho_G : (\mathsf{GS} \cup \mathsf{PGS}_{\mathsf{IR}}) \to \mathcal{P}(\mathtt{E}_G \times \mathtt{E}_G)$ is defined recursively as follows:

$$\rho_G(\alpha) \triangleq [\![[\alpha]]\!]_G \qquad \rho_G(\alpha \cdot u \cdot \beta) \triangleq [\![[\alpha]]\!]_G \,;\, \rho_G(u) \,;\, [\![[\beta]]\!]_G$$
$$\rho_G(r) \triangleq [\![r]\!]_G \qquad \rho_G(u \cdot \alpha \cdot v) \triangleq \rho_G(u) \,;\, [\![[\alpha]]\!]_G \,;\, \rho_G(v)$$
$$\rho_G((_{ir} \cdot u \cdot)_{ir}) \triangleq \rho_G(u) \cap [\![ir]\!]_G$$ ◀

The other direction, however, does not hold because $r \cap ir \subseteq r$ clearly holds according to the relational interpretation, but not according to the language interpretation. More generally, the issue is that RHS can have fewer intersections than the LHS and so its language interpretation can have fewer brackets than that of the LHS.

We therefore define a partial order $\lesssim_{\mathsf{B}}$ on guarded strings $(\mathsf{GS} \cup \mathsf{PGS}_{\mathsf{IR}})$ that allows the LHS to contain more brackets than the RHS as the least structure-preserving partial order relating $(_{ir} \cdot w \cdot)_{ir} \lesssim_{\mathsf{B}} w$ for all $w \in \mathsf{PGS}_{\mathsf{IR}}$, where we call an order $\lesssim_{\mathsf{B}}$ *structure-preserving* if:

$$\frac{u \lesssim_{\mathsf{B}} w}{\alpha \cdot u \cdot \beta \lesssim_{\mathsf{B}} \alpha \cdot w \cdot \beta} \qquad \frac{u_1 \lesssim_{\mathsf{B}} w_1 \quad u_2 \lesssim_{\mathsf{B}} w_2}{u_1 \cdot \alpha \cdot u_2 \lesssim_{\mathsf{B}} w_1 \cdot \alpha \cdot w_2} \qquad \frac{u \lesssim_{\mathsf{B}} w}{(_{ir} \cdot u \cdot)_{ir} \lesssim_{\mathsf{B}} (_{ir} \cdot w \cdot)_{ir}}$$

We can easily define the bracketed saturation of a language $\mathsf{BR}(L) \triangleq \{u \mid w \in L \wedge u \lesssim_{\mathsf{B}} w\}$, and write $L_1 \lesssim_{\mathsf{B}} L_2$ when $L_1 \subseteq \mathsf{BR}(L_2)$, i.e., when for all $u \in L_1$, there exists $w \in L_2$ such that $u \lesssim_{\mathsf{B}} w$.

With the above building blocks in place, we can prove the following equivalence between the two KATI representations.

▶ **Theorem 8** (Interpretation Equivalence). $[\![e_1]\!]_{\mathsf{L}} \lesssim_{\mathsf{B}} [\![e_2]\!]_{\mathsf{L}}$ *if and only if* $\forall G.\, [\![e_1]\!]_G \subseteq [\![e_2]\!]_G$.

Proof sketch. The "⇒'" direction follows from Prop. 7 and the observation that $u \lesssim_{\mathsf{B}} v$ implies $\rho_G(u) \subseteq \rho_G(v)$.

In the "⇐'" direction, from a guarded string $w \in [\![e_1]\!]_{\mathsf{L}}$, we construct a "canonical'" graph G_w as a sequence of nodes $n_0, \ldots, n_k$, such that a guarded string u has $\langle n_0, n_k\rangle \in \rho_{G_w}(u)$ iff $w \lesssim_{\mathsf{B}} u$. Then, it follows that $\langle n_0, n_k\rangle \in [\![e_1]\!]_{G_w} \subseteq [\![e_2]\!]_{G_w}$, and thus $w \in \mathsf{BR}([\![e_2]\!]_{\mathsf{L}})$. ◀

Theorem 8 provides a way to use language-based techniques to reason about inclusion of KATI expressions. There are two remaining questions:

- How can we finitely represent $[\![e]\!]_{\mathsf{L}}$?
- How can we finitely represent the bracketing closure, $\mathsf{BR}(L)$?

We first tackle the former question in § 3.2, and relegate the second to § 3.3.

Before we do so, we present an improvement of the bracketing closure that does not blindly add further brackets, but only ones that appear in the LHS of the inclusion. We say that the *nesting context* at given index of a guarded string is the sequence of relations corresponding to unmatched open brackets up to that index. We will be mainly interested in the set of all nesting contexts of a string, $\mathsf{c}(w)$, which can be defined inductively as follows:

$$\mathsf{c}(\alpha) \triangleq \mathsf{c}(r) \triangleq \{\epsilon\} \qquad \mathsf{c}((_{ir} \cdot w \cdot)_{ir}) \triangleq \{\epsilon\} \cup \{ir \cdot u \mid u \in \mathsf{c}(w)\}$$
$$\mathsf{c}(w_1 \cdot \alpha \cdot w_2) \triangleq \mathsf{c}(w_1) \cup \mathsf{c}(w_2) \qquad \mathsf{c}(\alpha \cdot w \cdot \beta) \triangleq \mathsf{c}(w)$$

Given a set of nesting contexts C and a language of guarded strings L, its restricted bracketing closure is $\mathsf{BR}_C(L) \triangleq \{u \mid w \in L \wedge u \lesssim_{\mathsf{B}} w \wedge \mathsf{c}(u) \subseteq C\}$. Using the restricted bracketing closure suffices to show inclusion.

▶ **Proposition 9.** $L_1 \lesssim_{\mathsf{B}} L_2$ *if and only if* $L_1 \subseteq \mathsf{BR}_{\mathsf{c}(L_1)}(L_2)$.

3.2 Converting KATI Expressions to Automata

As in §2, we will again use NFAs to compute $[\![.]\!]_{\mathsf{L}}$ albeit with a much more complex construction. As it is difficult to provide a direct NFA construction corresponding to $[\![e \cap ir]\!]_{\mathsf{L}}$, we will first put e in a normal form that enables a straightforward construction.

Normalization. The idea of the normal form is to ensure that (1) there are no tests immediately inside a bracket, and (2) directly nested brackets appear in $\prec$-order. To arrive at such a form, we first convert an expression e into a form that makes all possible tests at the beginning and the end of a string explicit. For this, we define $\mathsf{pred}(e)$, which returns a test t such that $[t] = e \cap [\top]$, and $\mathsf{pull}(e)$, which makes explicit any tests at the beginning of e.

$$
\begin{aligned}
\mathsf{pred}([t]) &\triangleq t & \mathsf{pull}([t]) &\triangleq \varnothing \\
\mathsf{pred}(r) &\triangleq \bot & \mathsf{pull}(r) &\triangleq r \\
\mathsf{pred}(e \cap ir) &\triangleq \bot & \mathsf{pull}(e \cap ir) &\triangleq e \cap ir \\
\mathsf{pred}(e_1 \cup e_2) &\triangleq \mathsf{pred}(e_1) \cup \mathsf{pred}(e_2) & \mathsf{pull}(e_1 \cup e_2) &\triangleq \mathsf{pull}(e_1) \cup \mathsf{pull}(e_2) \\
\mathsf{pred}(e_1 \mathbin{;} e_2) &\triangleq \mathsf{pred}(e_1) \cap \mathsf{pred}(e_2) & \mathsf{pull}(e_1 \mathbin{;} e_2) &\triangleq [\mathsf{pred}(e_1)] \mathbin{;} \mathsf{pull}(e_2) \cup \mathsf{pull}(e_1) \mathbin{;} e_2 \\
\mathsf{pred}(e^*) &\triangleq \top & \mathsf{pull}(e^*) &\triangleq \mathsf{pull}(e) \mathbin{;} e^*
\end{aligned}
$$

▶ **Definition 10.** *The converse of an expression $e \in \mathsf{KATI}$, written e^{-1}, is defined as follows:*

$$
\begin{aligned}
[t]^{-1} &\triangleq [t] & (e_1 \cup e_2)^{-1} &\triangleq e_1^{-1} \cup e_2^{-1} & (e_1 \mathbin{;} e_2)^{-1} &\triangleq e_2^{-1} \mathbin{;} e_1^{-1} \\
(e^*)^{-1} &\triangleq (e^{-1})^* & (e \cap ir)^{-1} &\triangleq e^{-1} \cap ir^{-1} & (x^{-1})^{-1} &\triangleq x \quad \textit{for } x \in \mathsf{R} \cup \mathsf{IR}.
\end{aligned}
$$

▶ **Lemma 11.** $[\![e]\!]_{\mathsf{L}} = [\![[\mathit{pred}(e)] \cup \mathit{pull}(e)]\!]_{\mathsf{L}} = [\![[\mathit{pred}(e)] \cup \mathit{pull}((\mathit{pull}(e^{-1}))^{-1})]\!]_{\mathsf{L}}$.

To convert an expression into normal form, we apply the following rewrite rules in a bottom-up fashion. The first rule is applied only once for each intersection in the KATI expression; the remaining rules as much as possible.

$$
\begin{aligned}
e \cap ir &= [\mathsf{pred}(e) \cap \mathsf{id}(ir)] \ \cup \ \mathsf{pull}((\mathsf{pull}(e^{-1}))^{-1}) \cap ir \\
([t] \mathbin{;} e) \cap ir &= [t] \mathbin{;} (e \cap ir) \\
(e \mathbin{;} [t]) \cap ir &= (e \cap ir) \mathbin{;} [t] \\
(e_1 \cup e_2) \cap ir &= e_1 \cap ir \ \cup \ e_2 \cap ir \\
((e_1 \cup e_2) \mathbin{;} e) \cap ir &= (e_1 \mathbin{;} e) \cap ir \ \cup \ (e_2 \mathbin{;} e) \cap ir \\
(e \mathbin{;} (e_1 \cup e_2)) \cap ir &= (e \mathbin{;} e_1) \cap ir \ \cup \ (e \mathbin{;} e_2) \cap ir \\
(e \cap ir) \cap ir &= e \cap ir \\
(e \cap ir') \cap ir &= (e \cap ir) \cap ir' \text{ if } ir' \prec ir
\end{aligned}
$$

It is easy to show that all these rules are equivalences according to the language interpretation, and thus $[\![\mathsf{normalize}(e)]\!]_{\mathsf{L}} = [\![e]\!]_{\mathsf{L}}$. We observe that the size of the normalized expression increases exponentially with the nesting depth of the expression. However, if we assume a bounded nesting of intersections in KATI expressions (as in all memory models), then our decision procedure for inclusion remains PSPACE-complete.

NFA Conversion. Once e is in normal form, conversion to an NFA is fairly straightforward. The only new case is that of the intersection of an automaton with ir, which adds $(_{ir}$ and $)_{ir}$ transitions at the start and end of the automaton, and ensures that any α-transition from the initial to a final state satisfies $\alpha \in [\![\mathsf{id}(ir)]\!]_{\mathsf{L}}$. Assuming that $[\![e]\!]_{\mathsf{NFA}} = \langle Q, \iota, F, \delta \rangle$, $[\![e \cap ir]\!]_{\mathsf{NFA}}$ returns the NFA $\langle Q', \iota, F, \delta' \rangle$, where:

$$
\begin{aligned}
Q' &= Q \uplus \left\{ q_{open} \;\middle|\; \langle \iota, \_, q \rangle \in \delta \right\} \uplus \left\{ q_{close} \;\middle|\; \langle q, \_, q_F \rangle \in \delta, q_F \in F \right\} \\
\delta' &= \left\{ \langle q, \sigma, q' \rangle \in \delta \;\middle|\; q \neq \iota, q' \notin F \right\} \\
&\quad \cup \left\{ \langle \iota, \alpha, q_{open} \rangle, \langle q_{open}, (_{ir}, q \rangle \;\middle|\; \langle \iota, \alpha, q \rangle \in \delta \right\} \\
&\quad \cup \left\{ \langle q,)_{ir}, q_{close} \rangle, \langle q_{close}, \alpha, q_F \rangle \;\middle|\; \langle q, \alpha, q_F \rangle \in \delta, q_F \in F \right\} \\
&\quad \cup \left\{ \langle \iota, \alpha, q_F \rangle \;\middle|\; \langle \iota, \alpha, q_F \rangle \in \delta, q_F \in F, \alpha \in [\![\mathsf{id}(ir)]\!]_{\mathsf{L}} \right\}
\end{aligned}
$$

The correctness of the conversion is captured by the following proposition.

▶ **Proposition 12.** *For all KATI expressions e,* $\mathsf{L}([\![\mathsf{normalize}(e)]\!]_{\mathsf{NFA}}) = [\![e]\!]_{\mathsf{L}}$.

3.3 Saturating NFAs with Brackets

We move on to define the bracketing saturation of an NFA. We begin by making an observation about the structure of the automaton $[\![e]\!]_{\mathsf{NFA}}$ corresponding to a KATI expression e. Observe that every state q in $[\![e]\!]_{\mathsf{NFA}}$ has a unique nesting context: all runs from the initial state(s) to q go through the same sequence of unmatched brackets. As such, we first define the function $\mathsf{c}(\cdot) : Q \to \mathsf{IR}^*$ returning the nesting context of each state.

▶ **Definition 13** (Nesting context). *Given an NFA $\langle Q, \iota, F, \delta \rangle$ and a state $q \in Q$, the* nesting context *of q, written $\mathsf{c}(q)$, is the word $ir_1 \cdots ir_k$ corresponding to the unmatched open bracket symbols $(_{ir_1} \cdots (_{ir_k}$ along any run from an initial state ι to q.*

Then, we define the notion of *nesting context completion* (or nesting completion for short). Intuitively, a nesting completion is used to saturate a KATI expression with matching brackets. In practice, we want to saturate the right-hand side of an inclusion with brackets that exist in the left-hand side, and as such we define the nesting completion of a context d w.r.t. a set of nesting contexts C.

▶ **Definition 14** (Nesting completion). *Given a nesting context $d = ir_1 \cdots ir_k$ and a set of nesting contexts C, the sequence $N = [w_1, \ldots, w_{k+1}]$ of $k+1$ words $w_i \in \mathsf{IR}^*$ is called a* nesting completion *of d with respect C, written $d \rightsquigarrow^N C$, if $w_1 \cdot ir_1 \cdots ir_k \cdot w_{k+1} \in C$.*

Given a sequence N of words $w_i \in \mathsf{IR}^*$, we write:

- $N.\epsilon$ for the sequence that appends the empty string at the end of N: $[w_1, w_2, \ldots, w_{k+1}, \epsilon]$.
- N/ir for the sequence that appends $ir \in \mathsf{IR}$ at the last word of N: $[w_1, \ldots, w_k, (w_{k+1} \cdot ir)]$.

At this point we are ready to define our bracketed substring saturation on NFAs. Using nesting completions, we can construct the saturated automaton. Given an NFA $\langle Q, \iota, F, \delta \rangle$ we define its *bracketed saturation* w.r.t. a set of nesting contexts C, written $\mathsf{BR}_C(\langle Q, \iota, F, \delta \rangle)$, as the automaton $\langle Q_{sat}, \iota_{sat}, F_{sat}, \delta_{sat} \rangle$, where:

$$
\begin{aligned}
Q_{sat} &\triangleq \{(q, N) \mid q \in Q, \mathsf{c}(q) \rightsquigarrow^N C\} \\
\iota_{sat} &\triangleq (\iota, [\epsilon]) \\
F_{sat} &\triangleq \{(q, [\epsilon]) \mid q \in F\} \\
\delta_{sat} &\triangleq \{((q, N), a, (q', N)) \mid (q, a, q') \in \delta, (q, N), (q', N) \in Q_{sat}\} \\
&\quad \cup \{((q, N), (_{ir}, (q', N.\epsilon)) \mid (q, (_{ir}, q') \in \delta, (q, N), (q', N.\epsilon) \in Q_{sat}\} \\
&\quad \cup \{((q, N.\epsilon),)_{ir}, (q', N)) \mid (q,)_{ir}, q') \in \delta, (q, N), (q', N.\epsilon) \in Q_{sat}\} \\
&\quad \cup \{((q, N), (_{ir}, (q, N/ir)) \mid (q, N/ir), (q, N) \in Q_{sat}\} \\
&\quad \cup \{((q, N/ir),)_{ir}, (q, N)) \mid (q, N/ir), (q, N) \in Q_{sat}\}
\end{aligned}
$$

As can be seen, the saturated NFA has the initial and final states of the original NFA with the empty completion as its initial states and final states, while its transition relation has three kinds of edges: (a) those maintaining the same nesting completion (modulo adding or removing an empty word at the end), when the original NFA performs the corresponding transition, (b) those incrementing the last word of the current nesting completion by reading an open bracket, and (c) those decrementing the last word of the current nesting completion by closing a bracket.

Correctness. We next prove that bracketing saturation at the level of NFAs is a sound and complete method for proving inclusion between KATI expressions, and thus inclusion is decidable.

▶ **Proposition 15** (Bracketing Saturation Correctness). *Let A be an automaton accepting only guarded strings and C be a set of nesting contexts. Then,* $\mathsf{BR}_C(\mathsf{L}(A)) = \mathsf{L}(\mathsf{BR}_C(A)) \cap \mathsf{GS}$.

Proof sketch. In the "$\supseteq$'" direction, let $w \in \mathsf{L}(\mathsf{BR}_C(A)) \cap \mathsf{GS}$, and $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ the respective accepting run on $\mathsf{BR}_C(A)$. By induction on the structure of w, we show that there exists a corresponding run $s \xrightarrow{u} t$ in A such that $w \lesssim_{\mathsf{B}} u$. This run is accepting on A, since s and t are an initial and final state of A, respectively, so $w \in \mathsf{BR}_C(\mathsf{L}(A))$.

In the "$\subseteq$'" direction, let $s \xrightarrow{u} t$ be an accepting path in A and $w \lesssim_{\mathsf{B}} u$ with $\mathsf{c}(w) \subseteq C$. By induction on the structure of $\lesssim_{\mathsf{B}}$, we show that there exists a corresponding path $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ in $\mathsf{BR}_C(A)$. Since $(s, [\epsilon]) \xrightarrow{w} (t, [\epsilon])$ are initial/final in $\mathsf{BR}_C(A)$ by construction, we obtain the desired result. ◀

Putting Propositions 9, 12, and 15 together, we can derive the soundness and completeness of the NFA-based checking of inclusion.

▶ **Theorem 16** (Decidability of Inclusion). *For all* $e_1, e_2 \in \mathsf{KATI}$, $[\![e_1]\!]_{\mathsf{L}} \lesssim_{\mathsf{B}} [\![e_2]\!]_{\mathsf{L}}$ *if and only if* $\mathsf{L}([\![\mathsf{normalize}(e_1)]\!]_{\mathsf{NFA}}) \subseteq \mathsf{L}(\mathsf{BR}_{\mathsf{c}(e_1)}([\![\mathsf{normalize}(e_2)]\!]_{\mathsf{NFA}}))$.

Proof. We show that the LHS is equivalent to the RHS:

$$\begin{aligned}
\mathsf{L}([\![\mathsf{normalize}(e_1)]\!]_{\mathsf{NFA}}) &= [\![e_1]\!]_{\mathsf{L}} && \text{by Prop. 12}\\
&\subseteq \mathsf{BR}_{\mathsf{c}(e_1)}([\![e_2]\!]_{\mathsf{L}}) && \text{by Prop. 9 and the LHS}\\
&= \mathsf{BR}_{\mathsf{c}(e_1)}(\mathsf{L}([\![\mathsf{normalize}(e_2)]\!]_{\mathsf{NFA}})) && \text{by Prop. 12}\\
&= \mathsf{L}(\mathsf{BR}_{\mathsf{c}(e_1)}([\![\mathsf{normalize}(e_2)]\!]_{\mathsf{NFA}})) && \text{by Prop. 15}
\end{aligned}$$ ◀

4 Memory Models as KATI Constraints

Let us now revisit §2, and see how irreflexivity implications between model definitions in KATI can be proved in a sound fashion. Recall from Theorem 6 that KATER reduces irreflexivity implications to a language inclusion problem, after taking some closures on the involved expressions. We would of course like to follow the same strategy in KATI, but unfortunately the deduplication closure $\mathsf{DEDUP}(L)$ cannot be easily adjusted to bracketed strings.

Nonetheless, we can adjust the rotation closure $\mathsf{ROT}(L)$ which raises a problem when applied to bracketed strings. Indeed, assuming the previous definition of $\mathsf{ROT}(L)$, if the language L contains the string $\alpha \cdot u_1 \cdot \beta \cdot (_{ir} \cdot w_1 \cdot \gamma \cdot w_2)_{ir} \cdot \alpha$, $\mathsf{ROT}(L)$ will include strings that are not well-bracketed like $\gamma \cdot w_2 \cdot)_{ir} \cdot \alpha \cdot u_1 \cdot \beta \cdot (_{ir} \cdot w_1 \cdot \gamma$.

To retain well-bracketedness, we have to redefine $\mathsf{ROT}(L)$. To that end, we first define a helper function $\mathsf{split}()$ that splits a string into a prefix and a suffix, and inverts the unmatched brackets of each substring.

$$\begin{aligned}
\mathsf{split}(r) &\triangleq \emptyset \\
\mathsf{split}((_{ir} \cdot w \cdot)_{ir}) &\triangleq \left\{ \langle ()_{ir^{-1}} \cdot u, \beta, v \cdot (_{ir^{-1}} \rangle \;\middle|\; \langle u, \beta, v \rangle \in \mathsf{split}(w) \right\} \\
\mathsf{split}(w_1 \cdot \alpha \cdot w_2) &\triangleq \left\{ \langle u, \beta, v' \cdot r \cdot \alpha \cdot (_S \cdot w_2 \rangle \;\middle|\; \langle u, \beta, v' \cdot r \cdot (_S \rangle \in \mathsf{split}(w_1) \right\} \\
&\quad \cup \left\{ \langle w_1 \cdot)_S \cdot \alpha \cdot r \cdot u', \beta, v \rangle \;\middle|\; \langle)_S \cdot r \cdot u', \beta, v \rangle \in \mathsf{split}(w_2) \right\} \\
&\quad \cup \{ \langle w_1, \alpha, w_2 \rangle \}
\end{aligned}$$

Inverting a bracket, inverts the corresponding intersection relation; if the relation is symmetric, then $ir^{-1} = ir$. In the definition above, $(_S,)_S$ denotes a sequence of zero or more opening and closing brackets respectively and S is the sequence of intersection relations ir that appear in the bracket subscripts. We can easily verify that if $\langle u, \alpha, v \rangle \in \mathsf{split}(w)$, then $u, v \in ((\mathsf{R} \cup \mathsf{IR}_{()}) \cdot \mathsf{A_P})^* \cdot (\mathsf{R} \cup \mathsf{IR}_{()})$, i.e., they are in guarded form.

Given $\mathsf{split}()$, we define $\mathsf{ROT}(L)$ as follows:

$$\begin{aligned}
\mathsf{ROT}(\alpha) &\triangleq \{\alpha\} \\
\mathsf{ROT}(\alpha \cdot w \cdot \alpha) &\triangleq \left\{ \beta \cdot v \cdot r' \cdot \alpha \cdot r \cdot u \cdot \beta \;\middle|\; \begin{array}{l} \langle)_S \cdot r \cdot u, \beta, v \cdot r' \cdot (_S \rangle \in \mathsf{split}(w) \\ \forall ir \in S.\, \alpha \leq \mathsf{id}(ir) \end{array} \right\} \cup \{\alpha \cdot w \cdot \alpha\} \\
\mathsf{ROT}(L) &\triangleq \left\{ u \in \mathsf{ROT}(w) \;\middle|\; w \in L \right\}
\end{aligned}$$

Observe that rotation produces only guarded strings because it commutes tests outside of brackets and $\mathsf{split}()$ inverts the direction of brackets.

We obtain the following equivalences.

▶ **Proposition 17** (Irreflexivity Equivalence). *Given a graph G and a language $L \subseteq \mathsf{GS}$:*

$$\begin{aligned}
\mathsf{irreflexive}(\rho_G(L)) &\Leftrightarrow \mathsf{irreflexive}(\rho_G(\mathsf{sameEnds}(L))) \Leftrightarrow \mathsf{irreflexive}(\rho_G(\mathsf{BR}(L))) \\
&\Leftrightarrow \mathsf{irreflexive}(\rho_G(\mathsf{ROT}(L)))\,,
\end{aligned}$$

where $\rho_G(L) \triangleq \bigcup_{w \in L} \rho_G(w)$ and $\rho_G(w)$ is defined in the proof sketch of Prop. 7.

Proof sketch. The first equivalence can be shown in a similar fashion to that in [14]. The second equivalence follows directly from the observation that $w \lesssim_{\mathsf{B}} u$ implies $\rho_G(w) \subseteq \rho_G(u)$. For the final one, the "$\Leftarrow$'" direction is trivial because $L \subseteq \mathsf{ROT}(L)$.

To prove that $\mathsf{irreflexive}(\rho_G(L)) \Rightarrow \mathsf{irreflexive}(\rho_G(\mathsf{ROT}(L)))$, consider $\langle b, b \rangle \in \rho_G(w)$ for some $w \in \mathsf{ROT}(L) \setminus L$. (If $w \in L$, the conclusion holds trivially.) Expanding the definition of rotation, $w = \beta \cdot v \cdot \alpha \cdot u \cdot \beta$ with $\langle u, \alpha, v \rangle \in \mathsf{split}(w)$, where u, v are the result of inverting the unmatched brackets of u', v' respectively, and $w' = \alpha \cdot u' \cdot \beta \cdot v' \cdot \alpha \in L$. Here, b is the node of G that corresponds to the atom β, and let a be the node that corresponds to the atom α in the cycle $\langle b, b \rangle$. Let γ_1, γ_2 be the atom adjacent to a possible unmatched bracket (originating from a matching pair of brackets $(_{ir},)_{ir}$) in v and u respectively and g_1, g_2 the corresponding nodes of G for these atoms in the cycle $\langle b, b \rangle$. Also, since $\langle b, b \rangle \in \rho_G(w)$, we know that $\langle g_1, g_2 \rangle \in [\![ir]\!]_G$. When calculating $\rho_G(w')$ we would interpret this pair of brackets with an intersection of the tuple $\{\langle g_2, g_1 \rangle\}$ with $[\![ir^{-1}]\!]_G$, which includes $\{\langle g_2, g_1 \rangle\}$. Therefore, $\langle a, a \rangle \in \rho_G(w')$ contradicting that $\rho_G(L)$ is irreflexive. ◀

▶ **Theorem 18** (Irreflexivity Implications). *For every $e_1, e_2 \in \mathsf{KATI}$, if $\mathsf{sameEnds}([\![e_1]\!]_\mathsf{L}) \subseteq \mathsf{ROT}(\mathsf{BR}([\![e_2]\!]_\mathsf{L}))$ then for all G, $\mathsf{irreflexive}([\![e_2]\!]_G) \Rightarrow \mathsf{irreflexive}([\![e_1]\!]_G)$.*

Proof sketch. Follows by repeated application of Prop. 17. ◀

5 KATI: Adding a "Top" Element

In this section, we extend KATI so that any relation $r \in \mathsf{R}$ can be used in intersections (and not only some dedicated relations).

The problem when doing so is that KATI's language interpretation is inadequate when it comes to prove certain relational properties. For instance, even though $[\![r_1 \cap r_2]\!]_G = [\![r_2 \cap r_1]\!]_G$, our bracketed language interpretation will yield $[\![r_1 \cap r_2]\!]_{\mathsf{L}} = ({}_{r_2} \cdot r_1 \cdot)_{r_2}$ which in turn is not equal to $({}_{r_1} \cdot r_2 \cdot)_{r_1} = [\![r_2 \cap r_1]\!]_{\mathsf{L}}$. Of course, this particular case could be handled as part of our normalization procedure, but more complicated relational inclusions (e.g., $[\![(r_1; r_2) \cap r_3]\!]_G \subseteq [\![r_3]\!]_G$) cannot be handled with said normalization.

To remedy this, we introduce a *top relation*, top, and express all primitive relations as intersections with top as follows:

$$[\![\mathsf{top}]\!]_{\mathsf{L}} \triangleq \left\{\alpha \cdot \mathsf{top} \cdot \beta \;\middle|\; \alpha, \beta \in \mathsf{A_P}\right\}$$
$$[\![r]\!]_{\mathsf{L}} \triangleq [\![\mathsf{top} \cap r]\!]_{\mathsf{L}} = \left\{\alpha \cdot ({}_{r} \cdot \mathsf{top} \cdot)_{r} \cdot \beta \;\middle|\; \alpha, \beta \in \mathsf{A_P}\right\} \cup \{\alpha \mid \alpha \in [\![\mathsf{id}(r)]\!]_{\mathsf{L}}\}$$

Observe that using the definition above and assuming that $\prec$ totally orders $\mathsf{R}_{()}$, we can already easily prove inclusions like $[\![r_1 \cap r_2]\!]_{\mathsf{L}} = [\![r_2 \cap r_1]\!]_{\mathsf{L}}$, since KATI's language interpretation of intersections already imposes a total order on brackets: the language interpretation of both expressions is $({}_{r_1} \cdot ({}_{r_2} \cdot \mathsf{top} \cdot)_{r_2} \cdot)_{r_1}$.

To be able to prove inclusions like $(r_1; r_2) \cap r_3 \subseteq r_3$, we introduce the top-closure $\lesssim_{\mathsf{T}}$ as the least structure-preserving partial order on $\mathsf{GS} \cup \mathsf{PGS_{IR}}$ containing $w \lesssim_{\mathsf{T}} \mathsf{top}$ for all $w \in \mathsf{PGS}_\emptyset$, and define $\lesssim_{\mathsf{BT}} \triangleq (\lesssim_{\mathsf{B}} \cup \lesssim_{\mathsf{T}})^+$, which is in fact equivalent to $\lesssim_{\mathsf{B}} ; \lesssim_{\mathsf{T}}$. The top closure of a language $L \subseteq \mathsf{GS}$ is $\mathsf{T}(L) \triangleq \left\{u_1 \cdot w \cdot u_2 \;\middle|\; u_1 \cdot \mathsf{top} \cdot u_2 \in L, w \in \mathsf{PGS}_\emptyset\right\}$.

With the above definition for $\lesssim_{\mathsf{T}}$ we can prove equivalence between the language and the relational interpretation of KATI (Theorem 8).

As far as the decision procedure of §3.2 and §3.3 is concerned, we can extend it to handle the new top element by modifying the NFA conversion of expressions consisting of a single primitive relation r, and our bracketed saturation. For the former, we redefine $[\![r]\!]_{\mathsf{NFA}}$ as the automaton $\langle\{q_0, q_1, q_2, q_3, q_4, q_5\}, q_0, \{q_5\}, \delta_{\mathsf{top}\cap r}\rangle$ where

$$\delta_{\mathsf{top}\cap r} \triangleq \textstyle\bigcup_{\alpha\in\mathsf{A_P}}\{\langle q_0, \alpha, q_1\rangle\} \cup \{\langle q_1, ({}_{r}, q_2\rangle, \langle q_2, \mathsf{top}, q_3\rangle, \langle q_3,)_{r}, q_4\rangle\} \cup \bigcup_{\alpha\in\mathsf{A_P}}\{\langle q_4, \alpha, q_5\rangle\}\,.$$

For the latter, given an NFA $A = \langle Q, \iota, F, \delta\rangle$, we define its top-closure $\mathsf{T}(A)$ as the automaton $\langle Q, \iota, F, \delta \cup \delta_{\mathsf{top}}\rangle$ where $\delta_{\mathsf{top}} = \{\langle q', \alpha, q\rangle \mid \langle q, \mathsf{top}, q'\rangle \in \delta, \alpha \in \mathsf{A_P}\}$

▶ **Proposition 19** (Top Closure Correctness). *For every automaton A accepting only guarded strings,* $\mathsf{T}(\mathsf{L}(A)) = \mathsf{L}(\mathsf{T}(A))$*.*

Then, we take the combined bracketing-top closure as $\mathsf{BR}_C^{\mathsf{top}}(A) \triangleq \mathsf{BR}_C(\mathsf{T}(A))$, and we obtain as corollary of Theorem 16 and Prop. 19 our main decidability result.

▶ **Theorem 20.** $[\![e_1]\!]_{\mathsf{L}} \lesssim_{\mathsf{BT}} [\![e_2]\!]_{\mathsf{L}}$ *iff* $\mathsf{L}([\![\mathsf{normalize}(e_1)]\!]_{\mathsf{NFA}}) \subseteq \mathsf{L}(\mathsf{BR}^{\mathsf{top}}_{\mathsf{c}(e_1)}([\![\mathsf{normalize}(e_2)]\!]_{\mathsf{NFA}}))$.

6 Consistency Checking

Similarly to Kater, KATI can also be used to generate consistency-checking code for a memory model's acyclicity constraints. In this section, we briefly recall Kater's code-generating infrastructure, and then show this infrastructure can be extended for the KATI language.

6.1 Consistency Checking with Kater

The key idea behind KATER's consistency-checking infrastructure is twofold. First, given a constraint demanding that a KAT expression e be acyclic, any e-cycle in a given graph G will ultimately be composed of primitive relations and predicates $r \in \mathsf{R}$ and $\pi \in \mathsf{P}$, i.e., the same primitives used to express e in KAT. As such, to find e-cycles in G, one only has to find some cyclic path in G, a permutation of which is accepted by $[\![e]\!]_{\mathsf{NFA}}$.

To determine whether a cyclic path is accepted by $[\![e]\!]_{\mathsf{NFA}}$, KATER treats G as another automaton, and takes its intersection with $[\![e]\!]_{\mathsf{NFA}}$[2]. Given the intersection, KATER searches for strongly connected components (SCCs) that contain at least one accepting state of $[\![e]\!]_{\mathsf{NFA}}$. (Observe that such SCCs are guaranteed to represented cycles in G that are accepted by $[\![e]\!]_{\mathsf{NFA}}$.) By using a depth-first-search algorithm (e.g., Tarjan's SCC algorithm [5]), the complexity of the generated consistency-checking code is $\mathcal{O}(nm)$, where $n = |G|$ and $m = |[\![e]\!]_{\mathsf{NFA}}|$.

6.2 Consistency Checking in KATI

When generating code for KATI expressions, we can employ the language representation of §3, as in the weak memory literature there is a disjoint set of relations used in intersections.

As such, we can extend KATER's code-generating infrastructure by making the following observation: the language representation of the KATI expressions $[\![e]\!]_{\mathsf{L}}$ and $[\![e \cap ir]\!]_{\mathsf{L}}$ is the same, modulo the $(_{ir}$ symbols. This observation implies that in order to check for acyclicity of $e \cap ir$, we can use the procedure of § 6.1 to enumerate all e-paths, and then simply restrict to paths whose endpoints are ir-matching (e.g., have the same location, if $ir = \texttt{sameloc}$).

Such a restriction can easily be performed by using dedicated variables $v_{c,ir}$ for $ir \in \mathsf{R}$ and $0 < c \leq \mathsf{c}(e)$. Whenever the intersection of $[\![e \cap ir]\!]_{\mathsf{NFA}}$ and G encounters the symbol $(_{ir}$, the corresponding information of the respective graph event is saved in v_{ir} (e.g., the event's location, if $ir = \texttt{sameloc}$), and the exploration proceeds as normal. Subsequently, when the intersection encounters the matching $)_{ir}$, the exploration only proceeds if the corresponding information of the respective graph event matches the information stored in v_{ir}.

Incremental Consistency Checking

In certain scenarios like testing or stateless model checking [15], we know that a given graph G' is consistent, and we want to check whether an event a can be added in a particular way maintaining consistency.

Even though we can use the algorithm of § 6.2 to check whether the newly constructed graph G is consistent, we can devise a more efficient procedure for checking G's consistency, inspired by the respective algorithm of Kokologiannakis et al. [14]. The key idea is that, since G' is consistent, any inconsistency in G will be caused by a (cyclic) path that passes through a. As such, we only have to find a cyclic paths in G that starts from a and is also a word accepted by $[\![e \cap ir]\!]_{\mathsf{NFA}}$. The only problem is that the word accepted by $[\![e \cap ir]\!]_{\mathsf{NFA}}$ might not have a in the beginning, but rather in the middle of the word.

To solve this, we perform a variation of the algorithm using the following construction. First, we enforce that $[\![e \cap ir]\!]_{\mathsf{NFA}}$ has a single starting/accepting state q_0 (e.g., by taking its reflexive-transitive closure), and we assume that G has a as its single starting/accepting state. Then, we run the algorithm, but instead of following the algorithm of § 6.2 and look

[2] In this construction, all of G's states are considered starting/final.

for SCCs starting from any state of the product (i.e., for each state $\langle e, q_0 \rangle$, where $e \in G$), we can instead only look for SCCs starting from the states $\langle a, q \rangle$ of the product, where q is a state in $[\![e \cap ir]\!]_{\mathsf{NFA}}$.

Observe that any such SCC that we find represents a consistency violation, as some permutation of the respective path in G is guaranteed to be accepted by $[\![e \cap ir]\!]_{\mathsf{NFA}}$. Such an algorithm leads to better performance, as it essentially corresponds to taking all rotations of $[\![e \cap ir]\!]_{\mathsf{NFA}}$ (instead of taking all rotations of G), and typically $|[\![e \cap ir]\!]_{\mathsf{NFA}}| \ll |G|$.

7 Related Work and Conclusion

There has been an abundance of work building on Kleene Algebra (with Tests) [16].

Many works focus on extending KA(T) to particular program domains. [8] support more program transformations than plain KAT by adding mutable tests. Anderson et al. [3] develop an instance of KAT called NetKAT to model packet transmission in networks, Wagemaker et al. [23] extend NetKAT for concurrency. Hoare et al. [9] presents Concurrent KA (CKA), an extension of Kleene Algebra with a built-in operator modeling parallel composition, and Jipsen [10] extends CKA with tests. Kappé et al. [12] present an alternative foundation for the concurrent setting called KA with Observations (KAO), to which they subsequently add tests [13]. Pous et al. [19] show that a lot of KA variants that have extra assumptions or impose additional structure (e.g., KAO, NetKAT) fit into the framework of KA with Hypotheses, and provide modular proofs for various such variants.

Others focus on handling a richer algebraic structure. Pous and Wagemaker [21] present two variants of KAT with an additional top element: one that only supports $[\![e]\!]_G \subseteq [\![\mathsf{top}]\!]_G$, and one that has the additional property that $[\![e]\!]_G \subseteq [\![e; \mathsf{top}; e]\!]_G$. Ésik and L. Bernátsky [6] extend KA with a converse operator, and prove equivalence between the language, relational and algebraic models. Brunet and Pous [4] prove that the equational theory of relation algebras that support union, intersection (with arbitrary relations) and concatenation, but do not support converse or the identity relation is decidable. Pous and Vignudelli [20] show that the equational theory of relation algebras that support concatenation, converse, arbitrary intersections and the identity relation (but neither union nor star!) is decidable.

As Pous and Wagemaker [21] note, however: *"The case of intersection (with or without converse or the various constants) is significantly more difficult, and remains partly open [...]"*. KATI attempts to tackle a useful instance of this problem by providing a decision procedure for KAT with intersections, assuming that intersections are restricted to primitive relations. Such a restriction is common when using KAT to describe weak memory consistency models, as per the work of Kokologiannakis et al. [14], which forms the basis for KATI.

8 Conclusion

In this paper, we have extended the results of Kokologiannakis et al. [14] to handle memory models containing intersections with uninterpreted relations. While this restriction on intersections appears sufficient for existing memory model definitions, it would definitely be nice to devise a more general technique that can handle arbitrary intersections. We leave the exploration of such a technique for future work.

References

1 Jade Alglave, Luc Maranget, Paul E. McKenney, Andrea Parri, and Alan Stern. Frightening small children and disconcerting grown-ups: Concurrency in the Linux kernel. In *ASPLOS 2018*, pages 405–418, New York, NY, USA, 2018. ACM. `doi:10.1145/3173162.3177156`.

2 Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Trans. Program. Lang. Syst.*, 36(2):7:1–7:74, July 2014. `doi:10.1145/2627752`.

3 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: Semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *POPL 2014, 2014*, pages 113–126. ACM, 2014. `doi:10.1145/2535838.2535862`.

4 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS 2015*, pages 68–79. IEEE Computer Society, 2015. `doi:10.1109/LICS.2015.17`.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. URL: `http://mitpress.mit.edu/books/introduction-algorithms`.

6 Zoltán Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theor. Comput. Sci.*, 137(2):237–251, 1995. `doi:10.1016/0304-3975(94)00041-G`.

7 Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *POPL 2005*, pages 110–121, New York, NY, USA, 2005. ACM. `doi:10.1145/1040305.1040315`.

8 Niels Bjørn Bugge Grathwohl, Dexter Kozen, and Konstantinos Mamouras. KAT + B! In Thomas A. Henzinger and Dale Miller, editors, *LICS 2014*, pages 44:1–44:10. ACM, 2014. `doi:10.1145/2603088.2603095`.

9 C. A. R. Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene algebra. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009*, volume 5710 of *LNCS*, pages 399–414. Springer, 2009. `doi:10.1007/978-3-642-04081-8_27`.

10 Peter Jipsen. Concurrent Kleene algebra with tests. In Peter Höfner, Peter Jipsen, Wolfram Kahl, and Martin Eric Müller, editors, *RAMiCS 2014*, volume 8428 of *LNCS*, pages 37–48. Springer, 2014. `doi:10.1007/978-3-319-06251-8_3`.

11 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. `doi:10.1016/0304-3975(94)90242-9`.

12 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene algebra with observations. In Wan J. Fokkink and Rob van Glabbeek, editors, *CONCUR 2019*, volume 140 of *LIPIcs*, pages 41:1–41:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.41`.

13 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene algebra with observations: From hypotheses to completeness. *CoRR*, abs/2002.09682, 2020. `doi:10.48550/arXiv.2002.09682`.

14 Michalis Kokologiannakis, Ori Lahav, and Viktor Vafeiadis. Kater: Automating weak memory model metatheory and consistency checking. *Proc. ACM Program. Lang.*, 7(POPL), January 2023. `doi:10.1145/3571212`.

15 Michalis Kokologiannakis, Azalea Raad, and Viktor Vafeiadis. Model checking for weakly consistent libraries. In *PLDI 2019*, New York, NY, USA, 2019. ACM. `doi:10.1145/3314221.3314609`.

16 Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3), 1997. `doi:10.1145/256167.256195`.

17 Ori Lahav, Viktor Vafeiadis, Jeehoon Kang, Chung-Kil Hur, and Derek Dreyer. Repairing sequential consistency in C/C++11. In *PLDI 2017*, pages 618–632, New York, NY, USA, 2017. ACM. `doi:10.1145/3062341.3062352`.

18 Leslie Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, September 1979. doi:10.1109/TC.1979.1675439.

19 Damien Pous, Jurriaan Rot, and Jana Wagemaker. On tools for completeness of Kleene algebra with hypotheses. *CoRR*, abs/2210.13020, 2022. doi:10.48550/arXiv.2210.13020.

20 Damien Pous and Valeria Vignudelli. Allegories: Decidability and graph homomorphisms. In Anuj Dawar and Erich Grädel, editors, *LICS 2018*, pages 829–838. ACM, 2018. doi:10.1145/3209108.3209172.

21 Damien Pous and Jana Wagemaker. Completeness theorems for Kleene algebra with tests and top. *CoRR*, abs/2304.07190, 2023. doi:10.48550/arXiv.2304.07190.

22 SPARC International Inc. *SPARC architecture manual - version 8*. Prentice Hall, 1992.

23 Jana Wagemaker, Nate Foster, Tobias Kappé, Dexter Kozen, Jurriaan Rot, and Alexandra Silva. Concurrent NetKAT - modeling and analyzing stateful, concurrent networks. In Ilya Sergey, editor, *ESOP 2022*, volume 13240 of *LNCS*, pages 575–602. Springer, 2022. doi:10.1007/978-3-030-99336-8_21.