Concurrent Kleene algebra with tests

Peter Jipsen

Chapman University, Orange, California 92866, USA
jipsen@chapman.edu

Abstract. Concurrent Kleene algebras were introduced by Hoare, Möller, Struth and Wehrman in [HMSW09,HMSW09a,HMSW11] as idempotent bisemirings that satisfy a concurrency inequation and have a Kleene-star for both sequential and concurrent composition. Kleene algebra with tests (KAT) were defined earlier by Kozen and Smith [KS97]. *Concurrent Kleene algebras with tests* (CKAT) combine these concepts and give a relatively simple algebraic model for reasoning about operational semantics of concurrent programs. We generalize guarded strings to *guarded series-parallel strings*, or gsp-strings, to provide a concrete language model for CKAT. Combining nondeterministic guarded automata [Koz03] with branching automata of Lodaya and Weil [LW00] one obtains a model for processing gsp-strings in parallel, and hence an operational interpretation for CKAT. For gsp-strings that are simply guarded strings, the model works like an ordinary nondeterministic guarded automaton. If the test algebra is assumed to be $\{0,1\}$ the language model reduces to the regular sets of bounded-width sp-strings of Lodaya and Weil.
Since the concurrent composition operator distributes over join, it can also be added to relation algebras with transitive closure to obtain the variety CRAT. We provide semantics for these algebras in the form of coalgebraic arrow frames expanded with concurrency.

Keywords: Concurrent Kleene algebras, Kleene algebras with tests, parallel programming models, series-parallel strings, relation algebras with transitive closure

1 Introduction

Relation algebras and Kleene algebras with tests have been used to model specifications and programs, while automata and coalgebras have been used to model state based systems and object-oriented programs. To compensate for plateauing processor speed, multi-core architectures and cluster-computing are becoming widely available. However there is little agreement on how to efficiently develop software for these technologies or how to model them with suitably abstract and simple principles. The recent development of concurrent Kleene algebra [HMSW09,HMSW09a,HMSW11] builds on a computational model that is well understood and has numerous applications. Hence it is useful to explore which aspects of Kleene algebras can be lifted fairly easily to the concurrent setting,

and whether the simplicity of regular languages and guarded strings can be preserved along the way. For the nonguarded case many interesting results have been obtained by Lodaya and Weil [LW00] using labeled posets (or pomsets) of Pratt [Pra86] and Gisher [Gis88], but restricted to the class of series-parallel pomsets called sp-posets. This is a special case of the set-based traces and dependency relation used in [HMSW09,HMSW09a,HMSW11] to motivate the laws of CKA. Here we investigate how to extend guarded strings to handle concurrent composition with the same approach as for sp-posets in [LW00].

Recall from [KS97] that a *Kleene algebra with tests* (KAT) is an idempotent semiring with a Boolean subalgebra of tests and a unary Kleene-star operation that plays the role of reflexive-transitive closure. More precisely, it is a two-sorted algebra of the form $\mathbf{A} = (A, A', +, 0, \cdot, 1, \bar{}, ^*)$ where A' is a subset of A, $(A, +, 0, \cdot, 1, ^*)$ is a Kleene algebra and $(A', +, 0, \cdot, 1, \bar{})$ is a Boolean algebra (the complementation operation is only defined on A').

Let Σ be a set of *basic program symbols* $p, q, r, p_1, p_2, \ldots$ and T a set of *basic test symbols* $t, t_1, t_2, \ldots$, where we assume that $\Sigma \cap T = \emptyset$. Elements of T are Boolean generators, and we write 2^T for the set of *atomic tests*, given by characteristic functions on T and denoted by $\alpha, \beta, \gamma, \alpha_1, \alpha_2, \ldots$

The collection of guarded strings over $\Sigma \cup T$ is $GS_{\Sigma,T} = 2^T \times \bigcup_{n<\omega} (\Sigma \times 2^T)^n$, and a typical guarded string is denoted by $\alpha_0 p_1 \alpha_1 p_2 \alpha_2 \ldots p_n \alpha_n$, or by $\alpha_0 w \alpha_n$ for short, where $\alpha_i \in 2^T$ and $p_i \in \Sigma$. Note that for finite T the members of $2^T \subseteq GS_{\Sigma,T}$ can be identified with the atoms of the free Boolean algebra generated by T.

Concatenation of guarded strings is via the coalesced product: $w\alpha \diamond \beta w' = w\alpha w'$ if $\alpha = \beta$ and undefined otherwise. For subsets L, M of $GS_{\Sigma,T}$ define

- $L + M = L \cup M$,
- $LM = \{v \diamond w : v \in L, w \in M \text{ and } v \diamond w \text{ is defined}\}$,
- $0 = \emptyset$, $1 = 2^T$, $\bar{L} = GS_{\Sigma,T} \setminus L$ and
- $L^* = \bigcup_{n<\omega} L^n$ where $L^0 = L$ and $L^n = LL^{n-1}$ for $n > 0$.

Then $\mathcal{P}(GS_{\Sigma,T})$ is a KAT under these operations, and one defines a map G from KAT terms over $\Sigma \cup T$ to this concrete model by

- $G(t) = \{\alpha \in 2^T : \alpha(t) = 1\}$ for $t \in T$,
- $G(p) = \{\alpha p \beta : \alpha, \beta \in 2^T\}$ for $p \in \Sigma$,
- $G(p + q) = G(p) + G(q)$, $G(pq) = G(p)G(q)$, $G(p^*) = G(p)^*$, for any terms p, q and
- $G(0) = 0$, $G(1) = 1$, $G(\bar{b}) = \overline{G(b)}$ for any Boolean term b.

The *language theoretic model* $\mathbf{G}_{\Sigma,T}$ is the subalgebra of $\mathcal{P}(GS_{\Sigma,T})$ generated by $\{G(t) : t \in T\} \cup \{G(p) : p \in \Sigma\}$. In fact $\mathbf{G}_{\Sigma,T}$ is the free KAT and its members are the *regular guarded languages*. Subsets of 2^T are called Boolean tests, and other members of $\mathbf{G}_{\Sigma,T}$ are called programs.

A *nondeterministic guarded automaton* is a coalgebra

$$\mathcal{A} : X \to \mathcal{P}(X)^{\Sigma \cup \mathcal{P}(2^T)} \times 2$$

where X is a set of states, $\mathcal{A}_0(x)(y)$ is the *set of successor states* of $x \in X$ for symbol $y \in \Sigma \cup \mathcal{P}(2^T)$, and $F = \{x : \mathcal{A}_1(x) = 1\}$ is the set of *final states*. Alternatively one can describe these automata in the more traditional way as a tuple $\mathcal{A}' = (X, \delta, F)$ where $\delta \subseteq X \times (\Sigma \cup \mathcal{P}(2^T)) \times X$ is the transition relation and $F \subseteq X$ is the set of final states. Acceptance of a guarded string w by $\mathcal{A}$ starting from initial state x_0 and ending in state x_f is defined recursively by:

- If $w = \alpha \in 2^T$ then w is accepted iff for some $n \geq 1$ there is a path $x_0 t_1 x_1 t_2 \ldots x_{n-1} t_n x_f$ in $\mathcal{A}$ of n test transitions $t_i \in \mathcal{P}(2^T)$ such that $\alpha \in t_i$ for $i = 1, \ldots, n$.
- If $w = \alpha p v$ then w is accepted iff there exist states x_1, x_2 such that α is accepted ending in state x_1, there is a transition labeled p from x_1 to x_2 (i.e., $x_2 \in \mathcal{A}_0(x_1)(p)$) and v is accepted by $\mathcal{A}$ starting from initial state x_2.

Finally, w is *accepted by $\mathcal{A}$ starting from* x_0 if the ending state x_f is indeed a final state, i.e., satisfies $x_f \in F$.

Kozen [Koz03] proved that the equational theory of KAT is decidable in PSPACE. Moreover KAT is much more versatile that Kleene algebra since it can faithfully express "if b then p else q" by the term $bp + \bar{b}q$ and "while b do p" using $(bp)^*\bar{b}$, as well as several other standard programming constructs. It also interprets Hoare logic and properly distinguishes between simple Boolean tests and complex assertions.

2 Adding concurrency

After this rather brief discussion of the language semantics and operational semantics of KAT, we now describe how these definitions generalize to handle concurrency. Intuitively, elements P, Q of a concurrent Kleene algebra with tests can be thought of as programs or program fragments, and they are represented by sets of "computation paths". The operation that needs to be added to KAT is the concurrent composition $P||Q$. Whereas in the sequential model the computation paths are guarded strings, we now need to be able to place two such sequential strings "next to each other", and then we also need to be able to sequentially compose such "concurrent strings" etc. A convenient way to visualize the semantic objects that we would like to construct is to view sequential composition as vertical concatenation (top to bottom) and concurrent composition as horizontal concatenation.

So for example, given two guarded strings $\alpha_0 v \alpha_m$ and $\beta_0 w \beta_n$ we would like to construct

$$\begin{matrix} \alpha_0 \\ v \\ \alpha_m \end{matrix} \Bigg\| \begin{matrix} \beta_0 \\ w \\ \beta_n \end{matrix}$$

As with sequential composition, this operation is not always defined. In order for these type of objects to be sequentially (vertically) composable, we impose the condition that $\alpha_0 = \beta_0$ and $\alpha_m = \beta_n$. So in fact we have $\alpha_0 v \alpha_m || \alpha_0 w \alpha_m$ and the resulting object is denoted by $\alpha_0 \{\!|v, w|\!\} \alpha_m$ or vertically by

$$\begin{array}{c}\alpha_0\\ v|w\\ \alpha_m\end{array}$$

In particular, if α, β are distinct atomic tests then $\alpha||\beta$ is undefined and $\alpha||\alpha = \alpha$. Similarly, $\alpha||\beta w\gamma$ is undefined for all atomic tests α, β, γ. Also, we define concurrent composition to be commutative, which is already reflected in our choice of notation: $\{\!|v, w|\!\} = \{\!|w, v|\!\}$ is a multiset. Moreover it is associative, which means that in these "strings", multisets are not members of multisets, i.e., $\{\!|\{\!|u, v|\!\}, w|\!\}$ is normalized to $\{\!|u, v, w|\!\}$. This ensures that $(\alpha p\beta||\alpha q\beta)||\alpha r\beta = \alpha\{\!|p, q, r|\!\}\beta = \alpha p\beta||(\alpha q\beta||\alpha r\beta)$. Via successive concurrent and sequential compositions we obtain *guarded series-parallel strings*, or *gsp-strings* for short. Formally the set of gsp-strings generated by Σ, T is the smallest set $GSP_{\Sigma,T}$ that has 2^T and $2^T \times \Sigma \times 2^T$ as subsets and is closed under the coalesced product $\diamond$ as well as the concurrent product $||$. For example, if $\Sigma = \{p, q\}$ and $T = \{t\}$ then, abbreviating 2^T by $\{\alpha, \beta\}$, the following expressions are gsp-strings: α, $\alpha p\alpha$, $\alpha p\beta$, $\alpha\{\!|p, q|\!\}\alpha$, $\alpha\{\!|p, q|\!\}\alpha q\beta$, $\alpha\{\!|p, \{\!|p, q|\!\}\alpha q|\!\}\beta$, ...

The language model over gsp-strings is defined as in the case of guarded strings, except that we now have an additional operation. For $L, M \in \mathcal{P}(GSP_{\Sigma,T})$ let

- $L||M = \{v||w : v \in L, w \in M \text{ and } v||w \text{ is defined}\}$.

This makes $\mathcal{P}(GSP_{\Sigma,T})$ into a complete bisemiring with a Kleene-star for sequential composition. The map G from the previous section is extended to all terms of KAT with $||$, by defining $G(p||q) = G(p)||G(q)$. The bi-Kleene algebra of *series-rational gsp-languages*[1], denoted by $\mathbf{C}_{\Sigma,T}$, is the subalgebra generated by $\{G(t) : t \in T\} \cup \{G(p) : p \in \Sigma\}$.

Note that for $b \in \mathcal{P}(2^T)$ and for any subset p of $GSP_{\Sigma,T}$ the concurrent composition $b||p$ is equal to $b \cap p$. In particular, concurrent and sequential composition coincide on tests. However, in general $||$ is not idempotent for sets of gsp-strings and the identity 1 of sequential composition is not an identity of concurrent composition.

With this language model as guide, we now define a *concurrent Kleene algebra with tests* (CKAT) as an algebra $\mathbf{A} = (A, A', +, 0, ||, \cdot, 1, ^*, \bar{})$ where

- $(A, A', +, 0, \cdot, 1, ^*, \bar{})$ is a Kleene algebra with tests,
- $(A, +, 0, ||)$ is a commutative semiring with 0 (but possibly no unit), and
- $b||c = bc$ for all $b, c \in A'$.

We do not include iterated parallel composition (i.e., parallel star) in the definition of a CKAT since this operation prevents the generalization of Kleene's theorem to gsp-languages ([LW00], see Section 3 for further discussion).

The language model also shows that the concurrency inequation $(x||y)(z||w) \leq (xz)||(yw)$ of CKA is not satisfied under the present definition of CKAT. Take for

[1] Lodaya and Weil used the name series-rational sp-language for the members of their language model

example $x = \{\alpha p \beta\}$, $y = \{\alpha q \beta\}$, $z = \{\beta p \gamma\}$, and $w = \{\beta q \gamma\}$, then $(x||y)(z||w) = \{\alpha\{\!|p, q|\!\}\beta\{\!|p, q|\!\}\gamma\}$ whereas $(xz)||(yw) = \{\alpha\{\!|p\beta p, q\beta q|\!\}\gamma\}$. So each expression produces a singleton set, but the two elements are distinct, hence the two expressions are not comparable. However one can impose the concurrency inequation on the generators of the regular gsp-languages to obtain a homomorphic image that satisfies this condition. Not all forms of concurrency satisfy this inequation (in some cases the reverse inequality is applicable), so having a more general axiomatization could be advantageous.

3 Automata over guarded series-parallel strings

The notion of nondeterministic automaton for gsp-strings is based on the one for guarded strings, but it is expanded with fork and join transitions taken from the branching automata of Lodaya and Weil [LW00]. Specifically a *guarded branching automaton* is a coalgebra for the functor $F(X) = \mathcal{P}(X)^{\Sigma \cup \mathcal{P}(2^T)} \times \mathcal{P}(\mathcal{M}(X)) \times \mathcal{P}(\mathcal{M}(X)) \times 2$ defined on the category of sets, where $\mathcal{M}(X)$ is the collection of multisets of X with more than one element. This means that an automaton is a map $\alpha : X \to F(X)$ where X is the set of states. As for guarded automata, the transition function is given by the first component of α and the set of final states is given by the last component in the form of a characteristic function. The second and third component are the *fork* and *join* relations respectively. In traditional notation, the automaton can also be specified by the tuple $\alpha' = (X, \delta, \delta_{\text{fork}}, \delta_{\text{join}}, F)$, where

- (X, δ, F) is a guarded automaton,
- $\delta_{\text{fork}} \subseteq X \times \mathcal{M}(X)$ and
- $\delta_{\text{join}} \subseteq \mathcal{M}(X) \times X$.

Fork transitions in δ_{fork} are denoted $(x, \{\!|x_1, x_2, \ldots, x_n|\!\})$, and if the multiset has n elements they are called forks of arity n. The join transitions of arity n are defined similarly, but with the order of the two components reversed.

While coalgebraic automata do not have an explicit initial state, they can be augmented with such a state whenever this is required. The advantages of the coalgebraic point of view is that it turns the class of all automata for this functor into a concrete category, and allows many standard results on bisimulation and coalgebraic modal logic to be applied to this setting. We will not make use of it at this point, but in the later part of this paper we again use the coalgebraic perspective to define frame semantics for concurrent relation algebras with transitive closure.

The acceptance condition for gsp-strings does have to be defined carefully since it substantially extends the one for guarded strings. Intuitively one can think of an automaton as evaluating the acceptance condition for parallel parts of the input string concurrently on separate processors. In many cases, when large scale parallel programs are run on a distributed cluster of computers, (part of) the program code is distributed to all the available processors and executes in separate environments until at an appropriate point results are communicated

back to a subset of the processors (perhaps a single one) and combined into a new state. This fork and join paradigm is of course a fairly restricted model of concurrent programming, but it has the merit of being quite simple and algebraic since it avoids syntactic annotations for named channels and other more architecture-dependent features. It also meshes well with our generalization of guarded strings and with the laws of concurrent Kleene algebra.

For the actual definition of acceptance we do not need to have separate copies of automata, instead we simply map the parallel parts of a gsp-string into the same automaton. Looking back at the recursive definition of acceptance for a (non-concurrent) guarded string relative to an initial state x_0, it is apparent that this condition is equivalent to finding a path from x_0 to some final state x_f such that the atomic program symbols in the string match with symbols along the path in the same order, and if $p_{i-1}\alpha_i p_i$ occurs in the guarded string then there is a path $\beta_1 \dots \beta_{n_i}$ of Boolean tests $\beta_k \geq \alpha_i$ along edges of the automaton that lie between the edges matched by p_{i-1} and p_i. For gsp-strings we define a similar "embedding" into the automaton where parallel branches correspond to a fork transition, followed by parallel (not necessarily disjoint) paths along matching edges until they reach a join transition. The precise recursive definition is as follows: A *weak guarded series parallel string* (or wgsp-string for short) is a gsp-string but possibly without the first and/or last atomic test. Acceptance of a wgsp-string w by $\mathcal{A}$ starting from initial state x_0 and ending at state x_f, is defined recursively by:

- If $w = \alpha \in 2^T$ then w is accepted iff for some $n \geq 1$ there is a sequential path $x_0 t_1 x_1 t_2 \dots x_{n-1} t_n x_f$ in $\mathcal{A}$ (i.e., (x_{i-1}, t_i, x_i) is an edge in $\mathcal{A}$) of n test transitions $t_i \in \mathcal{P}(2^T)$ such that $\alpha \in t_i$ for $i = 1, \dots, n$.
- If $w = p \in \Sigma$ then w is accepted iff there exist a transition labelled p from x_0 to x_f.
- If $w = \{\!|u_1, \dots, u_m|\!\}v$ for $m > 1$ then w is accepted iff there exist a fork $(x_0, \{\!|x_1, \dots, x_m|\!\})$ and a join $(\{\!|y_1, \dots, y_m|\!\}, y_0)$ in $\mathcal{A}$ such that u_i is accepted starting from x_i and ending at y_i for all $i = 1, \dots, m$, and furthermore βv is accepted by $\mathcal{A}$ starting at y_0 and ending at x_f.
- If $w = uv$ then w is accepted iff there exist a state x such that u is accepted ending in state x and v is accepted by $\mathcal{A}$ starting from initial state x and ending at x_f.

Finally, w is *accepted by $\mathcal{A}$ starting from* x_0 if the ending state x_f is indeed a final state, i.e., satisfies $\mathcal{A}_2(x_f) = 1$.

In the second recursive clause the fork transition corresponds to the creation of n separate processes that can work concurrently on the acceptance of the wgsp-strings $u_1, \dots, u_n$. The matching join-operation then corresponds to a communication or merging of states that terminates these processes and continues in a single thread.

The sets of gsp-strings that are accepted by a finite automaton are called *regular gsp-languages*. For sets of (unguarded) strings, the regular languages and the series-rational languages (i.e., those built from Kleene algebra terms) coincide. However, Loyala and Weil pointed out that this is not the case for sp-posets

(defined like gsp-strings except without using atomic tests), since for example the language $\{p, p||p, p||p||p, \ldots\}$ is regular, but not a series-rational language. The *width* of an sp-poset or a gsp-string is the maximal cardinality of an antichain in the underlying poset. A (g)sp-language is said to be of *bounded width* if there exists $n < \omega$ such that every member of the language has width less than n. Intuitively this means that the language can be accepted by a machine that has no more than n processors. The series-rational languages are of bounded width since concurrent iteration was not included as one of the operations of CKAT. For languages of bounded width we regain familiar results such as Kleene's theorem which states that a language is series-rational if and only if it is regular (i.e., accepted by a finite automaton) and has bounded width.

We now use a method from Kozen and Smith [KS97] to relate the bounded-width regular languages of Lodaya and Weil [LW00] to guarded bounded-width regular languages. Let $\overline{T} = \{\bar{t} : t \in T\}$ be the set of negated basic tests. From now on we will assume that $T = \{t_1, \ldots, t_n\}$ is finite, and we consider atomic tests α to be (sequential) strings of the form $b_1 b_2 \ldots b_n$ where each b_i is either the element t_i or $\bar{t}_i$. Every term p can be transformed into a term p' in negation normal form using DeMorgan laws and $\bar{\bar{b}} = b$, so that negation only appears on t_i.

Hence the term p' is also a CKA term over the set $\Sigma \cup T \cup \overline{T}$. Let $R(p')$ be the result of evaluating p' in the set of sp-posets of Lodaya and Weil. In [KS97] it is shown how to transform p' further to a sum $\hat{p}$ of *externally guarded* terms such that $p = p' = \hat{p}$ in KAT and $R(\hat{p}) = G(\hat{p})$. This argument extends to terms of CKAT since $||$ distributes over $+$. Therefore the completeness result of Lodaya and Weil [LW00] can be lifted to the following result.

Theorem 1. $CKAT \models p = q \iff G(p) = G(q)$

It follows that $\mathbf{C}_{\Sigma,T}$ is indeed the free algebra of CKAT. With the same approach one can also deduce the next result from [LW00].

Theorem 2. *A set of gsp-strings is series-rational (i.e. an element of $\mathbf{C}_{\Sigma,T}$) if and only if it is accepted by a finite guarded branching automaton and has bounded width.*

The condition of bounded width can be rephrased as a restriction on the automaton. A run of $\mathcal{A}$ is called *fork-acylic* if a matching fork-join pair never occurs as a matched pair nested within itself. The automaton is fork-acylic if all the accepted runs of $\mathcal{A}$ are fork-acyclic. Lodaya and Weil prove that if a language is accepted by a fork-acyclic automaton then it has bounded width, and their proof applies equally well to gsp-languages.

At this point it is not clear whether this correspondence can be used as a decision procedure for the equational theory of concurrent Kleene algebras with tests.

4 Trace semantics for concurrent Kleene algebras with tests

Kozen and Tiuryn [KT03] (see also [Koz03]) show how to provide trace semantics for programs (i.e. terms) of Kleene algebra with tests. This is based on an elegant connection between computation traces in a Kripke structure and guarded strings. Here we point out that this connection extends very simply to the setting of concurrent Kleene algebras with tests, where traces are related to labeled Hasse diagrams of posets and these objects in turn are associated with guarded series-parallel strings.

Exactly as for KAT, a Kripke frame over Σ, T is a structure (K, m_K) where K is a set of *states*, $m_K : \Sigma \to \mathcal{P}(K \times K)$ and $m_K : T \to \mathcal{P}(K)$. An sp-trace τ in K is essentially a gsp-string with the atomic guards replaced by states in K, such that whenever a triple $spt \in K \times \Sigma \times K$ is a subtrace of τ then $(s,t) \in m_K(p)$. As with gsp-strings one can form the coalesced product $\sigma \diamond \tau$ of two sp-traces σ, τ (if σ ends at the same state as where τ starts) as well as the parallel product $\sigma || \tau$ (if σ and τ start at the same state and end at the same state). These partial operations lift to sets X, Y of sp-traces by

- $XY = \{\sigma \diamond \tau : \sigma \in X, \tau \in Y \text{ and } \sigma \diamond \tau \text{ is defined}\}$
- $X||Y = \{\sigma||\tau : \sigma \in X, \tau \in Y \text{ and } \sigma||\tau \text{ is defined}\}$.

Programs (terms of CKAT) are interpreted in K using the inductive definition of Kozen and Tiuryn [KT03] extended by a clause for $||$:

- $[\![p]\!]_K = \{spt | (s,t) \in m_K(p)\}$ for $p \in \Sigma$
- $[\![0]\!]_K = \emptyset$ and $[\![b]\!]_K = m_K(b)$ for $b \in T$
- $[\![\bar{b}]\!]_K = K \setminus m_K(b)$ and $[\![p+q]\!]_K = [\![p]\!]_K \cup [\![q]\!]_K$
- $[\![pq]\!]_K = ([\![p]\!]_K)([\![q]\!]_K)$ and $[\![p^*]\!]_K = \bigcup_{n<\omega} [\![p]\!]_K^n$
- $[\![p||q]\!]_K = [\![p]\!]_K || [\![q]\!]_K$.

Each sp-trace τ has an associated gsp-string $\mathrm{gsp}(\tau)$ obtained by replacing every state s in τ with the corresponding unique atomic test $\alpha \in 2^T$ that satisfies $s \in [\![\alpha]\!]_K$. It follows that $\mathrm{gsp}(\tau)$ is the unique guarded string over Σ, T such that $\tau \in [\![\mathrm{gsp}(\tau)]\!]_K$. As a result the connection between sp-trace semantics and gsp-strings is the same as in [KT03] (the proof is also by induction on the structure of p).

Theorem 3. *For a Kripke frame K, program p and sp-trace τ, we have $\tau \in [\![p]\!]_K$ if and only if $gsp(\tau) \in G(p)$, whence $[\![p]\!]_K = gsp^{-1}(G(p))$. In fact gsp^{-1} is a CKAT homomorphism from the free algebra $\mathbf{C}_{\Sigma,T}$ to the algebra of series-rational sets of sp-traces over K.*

The trace model for guarded strings has many applications since each trace in $[\![p]\!]_K$ can be interpreted as a sequential run of the program p starting from the first state of the trace. The sp-trace model provides a similar interpretation for programs that fork and join threads during their runs. Each sp-trace in $[\![p]\!]_K$ is a representation of the basic programs and tests that were performed during the

possibly concurrent execution of the program p. Note that there are no explicit fork and join transitions in an sp-trace since, unlike a gsp-automaton (which has to allow for nondeterministic choice), whenever a state in an sp-trace has several immediate successor states, this is the result of a fork, and similarly states with several immediate predecessors represent a join.

While series-parallel traces are more complex than linear traces, they can, like the gsp-strings in Section 2, still be represented by planar diagrams where parallel composition is denoted by placing traces next to each other (with only one copy of the start state and end state), and sequential composition is given by placing traces vertically above each other (with only one connecting state between them).

The sp-trace semantics are useful for analysing the behavior of threads that communicate only indirectly with other concurrent threads via joint termination in a single state. While this is a restricted model of concurrency, it has a simple algebraic model based on Kleene algebras with tests, and it satisfies most of the laws of concurrent Kleene algebra.

5 Expanding relation algebras with concurrency

Kleene algebra with tests provides a reasonable operational semantics for imperative programs, but for specification purposes it would be useful to also have the full language of binary relations available when reasoning about concurrent software. In this section we show how coalgebraic arrow frames of relation algebras can be augmented with an additional component that corresponds to the $||$ operation. Recall that a relation algebra is of the form $\mathbf{A} = (A, +, 0, \wedge, \top, \bar{}, ;, 1, ^\smile)$ where $(A, +, 0, \wedge, \top, \bar{})$ is a Boolean algebra, $(A, ;, 1)$ is a monoid and for all $x, y, z \in A$

$$x; y \leq \bar{z} \iff x^\smile; z \leq \bar{y} \iff z; y^\smile \leq \bar{x}.$$

It follows that both ; and $^\smile$ distribute over the Boolean join, and that $^\smile$ is an involution, i.e., $x^{\smile\smile} = x$ and $(x; y)^\smile = y^\smile; x^\smile$. Jónsson and Tarski showed that every relation algebra $\mathbf{A}$ can be embedded in a complete and atomic relation algebra, and one can define a relational structure on the set of atoms from which the algebra can be reconstructed as a complex (powerset) algebra. The structure is known as *atom structure* or *ternary Kripke frame* or *arrow frame*, but it is in fact a coalgebra. Hence we define an *arrow coalgebra* to be of the form $\gamma : X \to \mathcal{P}(X^2) \times X \times 2$ such that for all $x, y, z \in X$,

- $(x \circ y) \circ z = x \circ (y \circ z)$ where $x \circ y = \gamma_0^{-1}\{(x, y)\}$ and $A \circ z = \{a \circ z : a \in A\}$,
- $I \circ x = x = x \circ I$ where $I = \gamma_2^{-1}\{1\}$ and
- $(x, y) \in \gamma_0(z) \iff (x^\smile, z) \in \gamma_0(y) \iff (z, y^\smile) \in \gamma_0(x)$ where $x^\smile = \gamma_1(x)$.

For $A, B \subseteq X$, define $A; B = \{a \circ b : a \in A, b \in B\}$ and $A^\smile = \{a^\smile : a \in A\}$ and $1 = I$. Then the *complex algebra* over γ, denoted

$$\mathcal{C}m(\gamma) = (\mathcal{P}(X), \cup, \emptyset, \cap, X, \bar{}, ;, ^\smile, 1')$$

is a complete relation algebra and $;, ^\smile$ distribute over arbitrary unions. Hence we can expand this algebra to a relation algebra with reflexive transitive closure (or RAT for short):

- $x^* = \bigcup_{n<\omega} x^n$, where $x^0 = 1'$ and $x^n = x; x^{n-1}$ for $n > 0$.

The variety generated by these algebras has a finite equational axiomatization, and has been studied by Tarski and Ng [NT77,Ng84]. We now expand arrow coalgebras further by adding another factor $\mathcal{P}(X^2)$ to the type functor. A *concurrent arrow coalgebra* is of the form $\gamma : X \to \mathcal{P}(X^2) \times X \times 2 \times \mathcal{P}(X^2)$ such that the projection onto the first three components is an arrow coalgebra and for all $x, y \in X$,

- $(x||y)||z = x||(y||z)$ and $x||y = y||x$ where $x||y = \gamma_3^{-1}\{x, y\}$
- $x \in \gamma_2^{-1}(1)$ implies $x||x = x$ and if $x \neq y$ then $x||y$ is undefined.

The complex algebra of a concurrent arrow coalgebra is a relation algebra with an additional binary operation $||$ defined on subsets A, B of X by $A||B = \{a||b : a \in A, b \in B\}$. Adding reflexive transitive closure is done as before. Based on this concrete model we have the following definition:

A *concurrent relation algebra with reflexive transitive closure* (or CRAT) is an algebra of the form

$$\mathbf{A} = (A, +, 0, \wedge, \top, ^-, ||, ;, 1, ^\smile, ^*)$$

where $\mathbf{A} = (A, +, 0, \wedge, \top, ^-, ;, 1, ^\smile, ^*)$ is a RAT, $(A, +, 0, ||)$ is a commutative semiring with zero and $(x \wedge 1)||y = x \wedge y \wedge 1$ holds for all $x, y \in A$. The result below follows from the theory of Boolean algebras with operators.

Theorem 4. *The complex algebra of a concurrent arrow coalgebra is a complete and atomic CRAT, and every CRAT can be embedded into such a complex algebra.*

The next result establishes a connection between CRAT and concurrent Kleene algebras with test.

Theorem 5. *Let* $\mathbf{A} = (A, +, 0, \wedge, \top, ^-, ||, ;, 1, ^\smile, ^*)$ *be a CRAT and define* $A' = \{b \in A : b \leq 1\}$. *Then* $\mathbf{A}'' = (A, A', +, 0, ||, \cdot, 1, ^-, ^*)$ *is a CKAT.*

The proof is simply a matter of checking that the axioms of CKAT hold for $\mathbf{A}''$. It is currently not known if every CKAT is embeddable into an algebra of the form $\mathbf{A}''$. Some related results about KAT can be found in [Koz06].

The concurrency inequality $(x||y); (z||w) \leq (x; z)||(y; w)$ can be added to CRAT and defines a proper subvariety. In the language of concurrent arrow coalgebras the inequality takes the following form: for all $t, u, v, w, x, y, z \in X$

- $t \in u \circ v$ and $u \in x||y$ and $v \in z||w \implies \exists r, s \in X$ ($t \in r||s$ and $r \in x \circ z$ and $s \in y \circ w$).

Other inequations that could be considered are $x||x = x$ or $x;y \leq x||y$ or $x||y \leq x;y$.

Unlike Kleene algebras with tests, the equational theory of relation algebras is known to be undecidable. This is a consequence of having complementation defined on the whole algebra, together with the associativity of a join-preserving operation (see [KNSS93] for such general results). However Andreka, Mikulas and Nemeti [AMN11] have recently proved that the theory of Kleene lattices is decidable. It is an interesting question whether their result can be extended to Kleene lattices with tests or concurrent Kleene lattices (with tests).

6 Conclusion

Many theoretical models of concurrency have been proposed and studied during the last five decades. Here we have taken an algebraic approach starting from Kleene algebras with tests and adapting them to concurrent Kleene algebras of Hoare et. al. and bounded-width series-parallel language models. This provides semantics for concurrency based on standard notions such as regular languages and automata. The addition of tests allows KAT to express standard imperative programming constructs such as if-then-else and while-do. Adding concurrency into this elegant algebraic model is likely to lead to new applications such as verifying compiler optimizations targeting multicore architectures or modeling computations on large distributed clusters. In the last section we have also shown how to add concurrency to relation algebras with reflexive and transitive closure, thus making concurrent composition part of this well-known and expressive algebraic setting.

References

[AMN11] Andréka, H., Mikulás, S.: Németi, I., The equational theory of Kleene lattices. Theoret. Comput. Sci. 412 (2011), no. 52, 7099–7108.

[Gis88] Gisher, L.: The equational theory of pomsets. Theoretical Computer Science 62 (1988) 299–224

[HMSW11] Hoare, C. A. R., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene algebra and its foundations. J. Log. Algebr. Program. 80 (2011), no. 6, 266–296.

[HMSW09] Hoare, C. A. R., Möller, B., Struth, G., Wehrman, I.: Foundations of concurrent Kleene algebra. Relations and Kleene algebra in computer science, 166–186, Lecture Notes in Comput. Sci., 5827, Springer, Berlin, 2009.

[HMSW09a] Hoare, C. A. R., Möller, B., Struth, G., Wehrman, I.: Concurrent Kleene algebra. CONCUR 2009—concurrency theory, 399–414, Lecture Notes in Comput. Sci., 5710, Springer, Berlin, 2009.

[KNSS93] Kurucz, Á., Németi, I., Sain, I., Simon, A.: Undecidable varieties of semilattice-ordered semigroups, of Boolean algebras with operators, and logics extending Lambek calculus. Logic Journal of IGPL, 1(1) (1993) 91–98.

[Koz03] Kozen, D.: Automata on guarded strings and applications. 8th Workshop on Logic, Language, Informations and Computation—WoLLIC'2001 (Brasília). Mat. Contemp. 24 (2003), 117–139.

[Koz06] Kozen, D.: On the representation of Kleene algebras with tests. Mathematical foundations of computer science 2006, 73–83, Lecture Notes in Comput. Sci., 4162, Springer, Berlin, 2006.

[KS97] Kozen, D., Smith, F.: Kleene algebra with tests: completeness and decidability. Computer science logic (Utrecht, 1996), 244–259, Lecture Notes in Comput. Sci., 1258, Springer, Berlin, 1997.

[KT03] Kozen, D., Tiuryn, J.: Substructural logic and partial correctness. ACM Trans. Computational Logic, 4(3) (2003) 355–378.

[LW00] Lodaya, K., Weil, P.: Series-parallel languages and the bounded-width property. Theoret. Comput. Sci. 237 (2000), no. 1-2, 347–380.

[Ng84] Ng, K. C.: Relation Algebras with Transitive Closure. PhD thesis, University of California, Berkeley, 1984.

[NT77] Ng, K. C., Tarski, A.: Relation algebras with transitive closure, Abstract 742-02-09, Notices Amer. Math. Soc. 24 (1977), A29–A30.

[Pra86] Pratt, V.: Modelling concurrency with partial orders. Internat. J. Parallel Prog. 15 (1) (1986) 33–71.