Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 265 (2010) 177–195
www.elsevier.com/locate/entcs

Fairness, Resources, and Separation

Stephen Brookes

Department of Computer Science
Carnegie Mellon University
Pittsburgh, USA

Abstract

Fair interleaving plays a fundamental rôle in denotational semantic models for shared-memory parallel programs, beginning with Park's trace semantics, based on a *fairmerge* relation designed so that $(\alpha, \beta, \gamma) \in$ *fairmerge* if and only if γ can be obtained by interleaving α and β. Park's formulation of *fairmerge* used nested greatest and least fixed points of monotone functions over traces, but he remarked that fixed point induction principles seem unsuitable for proving natural algebraic properties such as *associativity*. Such properties are needed to validate intuitive laws of program equivalence and to support hierarchical analysis of programs. Recent models and logics for shared-memory programs with mutable state and pointers build on and extend Park's foundations, with emphasis on *resources* and logical rules that embody *separation principles*. For example, *concurrent separation logic* is based on a race-detecting, resource-sensitive variant of *fairmerge*. For the kinds of interleaving employed in these models, and other more sophisticated variants of *fairmerge*, the algebraic difficulties are exacerbated. Rather than search for *ad hoc* techniques, we develop here a general framework for defining k-ary fairmerge operators, parameterized first by a choice of a *resource model* and then refined by a choice of a *conflict* or *interference relation*. Our formulation avoids nested fixed points, and supports inductive reasoning based on the length of finite prefixes of a trace. We prove a *generalized associativity* property, and obtain associativity proofs for prior models as a by-product.

Keywords: concurrency, shared memory, denotational semantics, fixed point induction, fairness, resources, separation logic

1 Introduction

Fair execution is a crucial assumption for reasoning about *liveness properties* of concurrent programs [18]. Indeed, fairness has played a fundamental rôle in the development of denotational semantic models for shared-memory parallel programs, beginning with early work of Keller [14] and Park's trace model [21]. In Park's semantics the meaning of a parallel program $c_1 \| c_2$ is defined to be the set of all traces obtainable as a *fair interleaving* of a trace of c_1 with a trace of c_2. More formally, Park introduced a (ternary) *fairmerge* relation on traces, characterized as a nested combination of greatest and least fixed points of monotone functions on a cartesian product of the complete lattice of trace sets, ordered by componentwise set inclusion. This relation was designed so that $(\alpha, \beta, \gamma) \in$ *fairmerge* if and only if γ is an

[1] Email: brookes@cs.cmu.edu

doi:10.1016/j.entcs.2010.08.011

interleaving of all of α with all of β. As Park noted, fixed point induction principles do not appear suitable in attempting to establish natural algebraic properties, such as *associativity* of *fairmerge*. Such properties are needed to validate natural laws of program equivalence, such as $c_1 \| (c_2 \| c_3) = (c_1 \| c_2) \| c_3$, as well as laws concerning the interaction between parallel composition and the scope of local declarations. Such laws help to support hierarchical analysis of networks of processes. Park derived an equivalent formula for the *fairmerge* relation, based on iterated concatenation, but remarked that additional (or alternative) insights were needed to prove associativity and other properties [22].

Park's fairmerge relation serves as the foundation for much later work, for example [6,7,16]. Recent models and logics for shared-memory languages involving mutable state and pointers also build on Park's foundations, but emphasize *resources*, *separation principles* and *race detection* [4,17]. This has led to the development of semantic models that rely on resource-sensitive and race-detecting forms of fair interleaving [8,9]. Such models are vital in establishing the soundness of *concurrent separation logic* [8,17]. These models, as originally presented, also suffer from algebraic problems, exacerbated by the additional technical sophistication required to deal with resource accounting and race detection. Furthermore, an associativity proof for Park's fairmerge would not immediately imply associativity for these more sophisticated variants, and it seems wasteful to have to treat each variant on an *ad hoc* basis. Ideally we need a coherent tractable mathematical account of fairness that facilitates algebraic reasoning and better supports the validation of laws of program equivalence.

We address these problems here by developing an abstract and general fairness framework, parameterized by a choice of a *resource model* and an *interference* or *confict relation* on actions, assumed to have some intuitively natural properties. A key ingredient in our approach is a generalization from Park's 2-ary fairmerge to a uniform family of k-ary fairmerge relations with closely related properties. We avoid dealing with fixed point operators, instead working with finite prefixes of traces and supporting a style of reasoning based on induction on trace length [2]. We prove a general associativity theorem applicable to all instances of this framework. The earlier semantic models can be recast as instances of our fairness framework, so we obtain associativity proofs for these models as a by-product. The framework is robust enough to encompass resource models based on semaphores and other synchronization mechanisms, as well as models that involve permissions.

2 Background

We develop the main ingredients of our framework in an abstract setting, by working with an arbitrary (non-empty) "alphabet" Σ whose elements represent "actions". We will use λ as a meta-variable ranging over Σ. Let Σ^* be the set of finite sequences

[2] Our emphasis on prefixes does not contradict the fact that fairmerge is non-monotone with respect to the prefix ordering on traces [20]. Our results do run counter to the longstanding tendency to conclude from non-monotonicity that fairness is beyond the denotational pale. Lack of monotonicity turns out to be irrelevant in our approach.

with elements from Σ, Σ^+ be the set of non-empty finite sequences, and Σ^ω be the set of (countably) infinite sequences. Let ϵ denote the empty sequence. A *trace* is a finite or infinite sequence of actions. Let $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ be the set of all traces. We use α, β, γ as meta-variables ranging over Σ^∞.

When α is a non-empty finite trace, let $last(\alpha)$ be the singleton trace consisting of the final element of α, and let $last(\epsilon) = \epsilon$. For $\alpha, \beta \in \Sigma^\infty$ we let $\alpha\beta$ denote the concatenation of α and β; when α is infinite, $\alpha\beta = \alpha$. Concatenation is associative, and the empty sequence is a unit: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$ and $\alpha\epsilon = \epsilon\alpha = \alpha$.

When $\alpha \in \Sigma^\infty$ and $h : \Sigma \to \Sigma$, let $map\, h\, \alpha$ be the sequence obtained by applying h to the element occurrences in α. The function map has the usual properties: $map\, h\, \epsilon = \epsilon$, $map\, h\, (\lambda\alpha) = (h\, \lambda)\, (map\, h\, \alpha)$, and $map\, h\, (\alpha\beta) = (map\, h\, \alpha)(map\, h\, \beta)$.

For a trace α and a subset A of Σ, we write $\alpha\lceil A$ for the subsequence of α consisting of actions from A.

Definition 2.1 For a trace α, let $pre(\alpha)$ be the set of finite prefixes of α, ordered by the prefix relation: $\alpha_1 < \alpha_2$ iff α_1 is a proper prefix of α_2. We also write $\alpha_1 \leq \alpha_2$, and say that α_1 is a prefix of α_2, when $\alpha_1 = \alpha_2$ or $\alpha_1 < \alpha_2$.

Clearly, $(pre(\alpha), <)$ is a linear ordering, with least element ϵ; α belongs to $pre(\alpha)$ if and only if α is finite. We may recover α from its prefixes, by listing the members of $pre(\alpha)$ in prefix order and concatenating their last elements: $\alpha = map\, last\, (pre(\alpha))$. Note that if $pre(\alpha) = pre(\beta)$ then $\alpha = \beta$.

Definition 2.2 A function $f : pre(\alpha) \to pre(\beta)$ is *strictly monotone* if and only if $f(\epsilon) = \epsilon$ and for all $\alpha_1 < \alpha_2 \in pre(\alpha)$, $f(\alpha_1) < f(\alpha_2)$.

For any function $h : \Sigma \to \Sigma$ it is easy to see that $pre(map\, h\, \alpha) = \{map\, h\, \alpha' \mid \alpha' \in pre(\alpha)\}$, and when $\alpha_1 < \alpha_2$ we also have $map\, h\, \alpha_1 < map\, h\, \alpha_2$. Thus $map\, h : pre(\alpha) \to pre(map\, h\, \alpha)$ is strictly monotone.

Definition 2.3 When $f : pre(\alpha) \to pre(\beta)$ is strictly monotone, we define the "inverse" function $f^{-1} : pre(\beta) \to pre(\alpha)$ such that, for all $\beta' \in pre(\beta)$,

$$f^{-1}(\beta') = max\{\alpha' \in pre(\alpha) \mid f(\alpha') \leq \beta'\}.$$

If f is strictly monotone, then f^{-1} is *monotone* (but not necessarily strictly monotone): whenever $\beta_1 \leq \beta_2 \in pre(\beta)$, $f^{-1}(\beta_1) \leq f^{-1}(\beta_2)$. Also note that for all $\alpha' \in pre(\alpha)$, $f^{-1}(f(\alpha')) = \alpha'$; and for all $\beta' \in pre(\beta)$, $f(f^{-1}(\beta')) \leq \beta'$.

Using these ingredients, we can now define a family $\langle FM_k \mid k \geq 1\rangle$ of k-ary fair merge relations, with the intuitive reading that $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$ iff β is an interleaving of the sequences $\alpha_1, \ldots, \alpha_k$.

Definition 2.4 The k-ary *fair merge* relations $FM_k \subseteq (\Sigma^\infty)^k \times \Sigma^\infty$, for $k \geq 1$, are given by:

$$((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$$

iff there are strictly monotone functions

$$f_1 : pre(\alpha_1) \to pre(\beta), \ldots, f_k : pre(\alpha_k) \to pre(\beta)$$

with the following properties:

(a) For all $i \neq j$, $rge(f_i) \cap rge(f_j) = \{\epsilon\}$.

(b) $pre(\beta) = rge(f_1) \cup \cdots \cup rge(f_k)$.

(c) For all i and all $\alpha' \in pre(\alpha_i)$, $last(f_i(\alpha')) = last(\alpha')$.

We will refer to a tuple of strictly monotone functions $(f_1, \ldots, f_k)$ with the indicated types and properties as a *schedule* for merging $\alpha_1, \ldots, \alpha_k$ into β. The scheduling functions specify an interleaving of $\alpha_1, \ldots, \alpha_k$ that produces β, by describing for each element occurrence in β, a corresponding element occurrence of one of $\alpha_1 \ldots, \alpha_k$. The constraints imposed by monotonicity and (a), (b) and (c) ensure that every element occurrence in β comes from one of $\alpha_1, \ldots, \alpha_k$, each element occurrence from each α_i occurs in α, and the relative order of the element occurrences from each α_i is preserved.

Example 2.5 Let $\Sigma = \{0, 1\}$.

(i) $((0^\omega, 1^\omega), \beta) \in FM_2$ if and only if β is an infinite sequence of 0's and 1's containing infinitely many 0's and infinitely many 1's. This follows easily from the definition of FM_2, since each non-empty finite prefix 0^n of 0^ω must generate a distinct prefix of β ending with 0, and each non-empty prefix 1^m of 1^ω must generate a distinct prefix of β ending with 1. In this simple example, for each such trace β there is a unique schedule.

(ii) $((0^\omega, 0^\omega), \beta) \in FM_2$ if and only if $\beta = 0^\omega$. In this case, there are infinitely many possible schedules, since each occurrence of 0 in β can be chosen either from the first input sequence or the second; each choice of schedule produces the same result trace.

(iii) $((0^\omega, 1), 0^\omega)$ does not belong to FM_2, since 0^ω has no prefix ending with 1, so there is no way to define a scheduling function from $pre(1)$ to $pre(0^\omega)$.

Next we identify some obvious properties of the family $\langle FM_k \mid k > 0 \rangle$ of relations. Informally, we can summarize by saying that fairmerge is invariant under permutation of the "input" traces and invariant under *map*. Moreover, the empty trace makes no contribution in fair merges. We also isolate a pair of properties that express a natural connection between fair merging and the concatenation operation on traces.

Lemma 2.6 *The relations FM_k satisfy the following properties:*

(i) If $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$ and $\pi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ is a permutation, then $((\alpha_{\pi 1}, \ldots, \alpha_{\pi k}), \beta) \in FM_k$.

(ii) If $h : \Sigma \to \Sigma$ and $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$, then $((map\, h\, \alpha_1, \ldots, map\, h\, \alpha_k), map\, h\, \beta) \in FM_k$.

Proof. (i) is obvious. For (ii), if $(f_1, \ldots, f_k)$ is a schedule for $((\alpha_1, \ldots, \alpha_k), \beta)$ we can define a schedule $(g_1, \ldots, g_k)$ for $((map\, h\, \alpha_1, \ldots, map\, h\, \alpha_k), map\, h\, \beta)$ by letting $g_i(map\, h\, \alpha') = map\, h\, (f_i(\alpha'))$, for $1 \leq i \leq k$ and $\alpha' \in pre(\alpha_i)$. □

Lemma 2.7 *The empty sequence plays a trivial role in merges.*

(i) $((\epsilon, \alpha_1, \ldots, \alpha_k), \beta) \in FM_{k+1}$ *iff* $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$.

(ii) $((\alpha_1, \ldots, \alpha_k), \epsilon) \in FM_k$ *if and only if each* α_i *is* ϵ.

Proof. These results follow trivially from the fact that $pre(\epsilon) = \{\epsilon\}$. □

Theorem 2.8 *The k-ary fairmerge relations* FM_k*, for* $k \geq 1$*, satisfy the following "prefix/suffix" and "concatenation" properties:*

(i) If $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$*,* $\beta' \in pre(\beta)$ *and* $\beta = \beta'\beta''$*, then there are prefixes* $\alpha'_1 \in pre(\alpha_1), \ldots, \alpha'_k \in pre(\alpha_k)$*, with suffixes* $\alpha''_1, \ldots, \alpha''_k$ *such that* $\alpha_1 = \alpha'_1\alpha''_1, \ldots, \alpha_k = \alpha'_k\alpha''_k$*, for which* $((\alpha'_1, \ldots, \alpha'_k), \beta') \in FM_k$ *and* $((\alpha''_1, \ldots, \alpha''_k), \beta'') \in FM_k$.

(ii) If $((\alpha'_1, \ldots, \alpha'_k), \beta') \in FM_k$*,* $((\alpha''_1, \ldots, \alpha''_k), \beta'') \in FM_k$*, and* β' *is finite, then* $((\alpha'_1\alpha''_1, \ldots, \alpha_k\alpha''_k), \beta'\beta'') \in FM_k$.

Proof. For (i), suppose $(f_1, \ldots, f_k)$ is a schedule for $((\alpha_1, \ldots, \alpha_k), \beta)$ and let $\beta = \beta'\beta''$, with β' finite. If $\beta' = \beta$ (so $\beta'' = \epsilon$) we can take $\alpha'_i = \alpha_i$ and $\alpha''_i = \epsilon$ for $i = 1, \ldots, k$ and the result holds trivially by Lemma 2.7. Otherwise, let i be the (unique) index such that $\beta' \in rge(f_i)$, and let $\alpha'_j = f_j^{-1}(\beta')$ for $j = 1, \ldots, k$. Then $f_i(\alpha'_i) = \beta'$ and for $j \neq i$ we have $f_j(\alpha'_j) < \beta'$. The restrictions of $f_1, \ldots, f_k$ to $pre(\alpha'_1), \ldots, pre(\alpha'_k)$ form a schedule for $((\alpha'_1, \ldots, \alpha'_k), \beta')$, so $((\alpha'_1, \ldots, \alpha'_k), \beta') \in FM_k$. Let $\alpha''_1, \ldots, \alpha''_k$ and β'' be the suffixes corresponding to $\alpha'_1, \ldots, \alpha'_k$ and β', so that $\alpha_1 = \alpha'_1\alpha''_1, \ldots, \alpha_k = \alpha'_k\alpha''_k$, and $\beta = \beta'\beta''$. The functions $g_1 : pre(\alpha''_1) \to pre(\beta''), \ldots, g_k : pre(\alpha''_k) \to pre(\beta'')$ such that $g_j(\alpha'') = f_j(\alpha'_j\alpha'')$, for each j and each $\alpha'' \in pre(\alpha''_j)$, are well defined, strictly monotone, and satisfy conditions (a), (b) and (c). So $(g_1, \ldots, g_k)$ is a schedule for $((\alpha''_1, \ldots, \alpha''_k), \beta'')$, showing that $((\alpha''_1, \ldots, \alpha''_k), \beta'') \in FM_k$, as required.

For (ii), suppose β' is finite, $(f_1, \ldots, f_k)$ is a schedule for $((\alpha'_1, \ldots, \alpha'_k), \beta')$, and $(g_1, \ldots, g_k)$ is a schedule for $((\alpha''_1, \ldots, \alpha''_k), \beta'')$. Then each α'_i is finite, and we can define functions $h_i : pre(\alpha'_i\alpha''_i) \to pre(\beta'\beta'')$ by: $h_i(\alpha') = f_i(\alpha')$ when $\alpha' \leq \alpha'_i$, $h_i(\alpha'_i\alpha'') = g_i(\alpha'')$ when $\alpha'' \in pre(\alpha''_i)$. These functions are well defined, strictly monotone, and satisfy (a), (b) and (c). So $(h_1, \ldots, h_k)$ is a schedule for $((\alpha'_1\alpha''_1, \ldots, \alpha'_k\alpha''_k), \beta'\beta'')$, as required. □

The next two results show that for $k = 1$ and $k = 2$ our definition yields the intended fairmerge relation at the corresponding arity: unary fairmerge FM_1 is the identity relation on traces, and binary fairmerge FM_2 is the same as Park's fairmerge.

Theorem 2.9 $FM_1 = \{((\alpha), \alpha) \mid \alpha \in \Sigma^\infty\}$.

Proof. [3]: Clearly the identity function on $pre(\alpha)$ is a schedule for $((\alpha), \alpha)$. So $\{((\alpha), \alpha) \mid \alpha \in \Sigma^\infty\} \subseteq FM_1$. For the converse, note that if $((\alpha), \beta) \in FM_1$ and $f : pre(\alpha) \to pre(\beta)$ is strictly monotone, $rge(f) = pre(\beta)$, and for all $\alpha' \in pre(\alpha)$,

[3] Perhaps pedantically, we distinguish notationally between a 1-tuple (α) and a trace α.

$last(f(\alpha')) = last(\alpha')$, it is easy to prove by induction on the length of α' that $f(\alpha') = \alpha'$. It follows that $\beta = \alpha$. □

Theorem 2.10 *FM_2 coincides with Park's fairmerge relation.*

Proof. Park's relation *fairmerge* $\subseteq \Sigma^\infty \times \Sigma^\infty \times \Sigma^\infty$ can be characterized as follows [22]. First extend the concatenation operation $\alpha\beta$ on traces to triples of traces, componentwise: $(\alpha_1, \alpha_2, \alpha_3)(\beta_1, \beta_2, \beta_3) = (\alpha_1\beta_1, \alpha_2\beta_2, \alpha_3\beta_3)$; then extend, pointwise, to sets of triples. For a set of triples T, let $T^0 = \{(\epsilon, \epsilon, \epsilon)\}$, $T^{n+1} = T^n T$ for $n \geq 0$, and $T^* = \bigcup_{n=0}^{\infty} T^n$. Then T^* satisfies the following equation:

$$T^* = \{(\alpha_1 \ldots \alpha_n, \beta_1 \ldots \beta_n, \gamma_1 \ldots \gamma_n) \mid \exists n \geq 0.\ \forall i. 1 \leq i \leq n \Rightarrow (\alpha_i, \beta_i, \gamma_i) \in T\}.$$

If every triple in T has non-empty components, i.e. whenever $(\alpha, \beta, \gamma) \in T$ we have $\alpha \neq \epsilon, \beta \neq \epsilon$ and $\gamma \neq \epsilon$, we can define

$$T^\omega = \{(\alpha_1 \ldots \alpha_n \ldots, \beta_1 \ldots \beta_n \ldots, \gamma_1 \ldots \gamma_n \ldots) \mid \forall i \geq 0.\ (\alpha_i, \beta_i, \gamma_i) \in T\}.$$

(The non-emptiness constraint on T means we avoid the degenerate term ϵ^ω.)

Let $A, B, C \subseteq \Sigma^\infty \times \Sigma^\infty \times \Sigma^\infty$ be given by:

$$A = \{(\alpha, \epsilon, \alpha) \mid \alpha \in \Sigma^\infty\} \cup \{(\epsilon, \beta, \beta) \mid \beta \in \Sigma^\infty\}$$

$$B = \{(\lambda, \epsilon, \lambda) \mid \lambda \in \Sigma\} \qquad C = \{(\epsilon, \lambda, \lambda) \mid \lambda \in \Sigma\}.$$

As shown by Park, $fairmerge = (B \cup C)^* A \ \cup\ (B^* C C^* B)^\omega$. (Note that every triple $(\alpha, \beta, \gamma) \in B^* C C^* B$ has non-empty components, so the infinite iteration used here is well defined.)

We argue that $((\alpha, \beta), \gamma) \in FM_2$ iff $(\alpha, \beta, \gamma) \in fairmerge$, as follows.

Suppose $((\alpha, \beta), \gamma) \in FM_2$. Let $f : pre(\alpha) \to pre(\gamma), g : pre(\beta) \to pre(\gamma)$ form a schedule for $((\alpha, \beta), \gamma)$. It is easy to show that if at least one of $rge(f)$ and $rge(g)$ is finite then $(\alpha, \beta, \gamma) \in (B \cup C)^* A$, and if both $rge(f)$ and $rge(g)$ are infinite, then $(\alpha, \beta, \gamma) \in (B^* C C^* B)^\omega$. Thus $(\alpha, \beta, \gamma) \in fairmerge$, as required.

Conversely, if $(\alpha, \beta, \gamma) \in fairmerge$ we can construct a schedule (f, g) for $((\alpha, \beta), \gamma)$, showing that $((\alpha, \beta), \gamma) \in FM_2$. The details are straightforward. □

We now come to the crucial result concerning associativity, expressed in a suitably generalized manner to deal with k-ary merges. We use notation such as $\overline{\alpha}$ to denote a k-tuple of traces, relying on the context to specify the intended value of k. When $\overline{\alpha_1}, \ldots, \overline{\alpha_n}$ are tuples of length $k_1, \ldots, k_n$, we write $\overline{\alpha_1 \ldots \alpha_n}$ for the tuple of length $k_1 + \cdots + k_n$ obtained by concatenating and flattening, so for instance $\overline{(\alpha, \beta)(\gamma, \delta)} = (\alpha, \beta, \gamma, \delta)$.

Theorem 2.11 *The relations FM_k have the following "general associativity" properties.*

(i) If $(\overline{\alpha_1}, \beta_1) \in FM_{k_1}, \ldots, (\overline{\alpha_n}, \beta_n) \in FM_{k_n}$ and $((\beta_1, \ldots, \beta_n), \gamma) \in FM_n$, then $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in FM_{k_1 + \cdots + k_n}$.

(ii) If $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in FM_{k_1 + \cdots + k_n}$ and for each i, $len(\overline{\alpha_i}) = k_i$, then there exist $\beta_1, \ldots, \beta_n$ such that $(\overline{\alpha_1}, \beta_1) \in FM_{k_1}, \ldots, (\overline{\alpha_n}, \beta_n) \in FM_{k_n}$ and

$((\beta_1, \ldots, \beta_n), \gamma) \in FM_n$.

Proof. For (i), suppose that $(\overline{\alpha_1}, \beta_1) \in FM_{k_1}, \ldots, (\overline{\alpha_n}, \beta_n) \in FM_{k_n}$ and $((\beta_1, \ldots, \beta_n), \gamma) \in FM_n$. Let $\overline{f_1} = (f_{11}, \ldots, f_{1k_1})$ be a schedule for $(\overline{\alpha_1}, \beta_1)$, and let $\overline{f_2}, \ldots, \overline{f_n}$ be schedules for $(\overline{\alpha_2}, \beta), \ldots, (\overline{\alpha_n}, \beta_n)$ respectively, with similar notations f_{ij} for the component functions. Let $(g_1, \ldots, g_n)$ be a schedule for $((\beta_1, \ldots, \beta_n), \gamma)$. Then define $F_{ij} = g_i \circ f_{ij}$, for $1 \leq i \leq n$ and $1 \leq j \leq k_i$, and let $\overline{F_i} = (F_{i1}, \ldots, F_{ik_i})$. The functions $F_{ij} : pre(\alpha_{ij}) \to pre(\gamma)$ are strictly monotone and satisfy the intended properties (a), (b) and (c), since the f_{ij} and g_i have the analogous properties. Thus $\overline{F_1 \ldots F_n}$ is a schedule for $(\overline{\alpha_1 \ldots \alpha_n}, \gamma)$, and $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in FM_{k_1+\cdots+k_n}$.

For (ii), suppose $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in FM_{k_1+\cdots+k_n}$ and for each i, $len(\overline{\alpha_i}) = k_i$. Let $\overline{F_1 \ldots F_n}$ be a schedule for $(\overline{\alpha_1 \ldots \alpha_n}, \gamma)$, where for $1 \leq i \leq n$ each F_i has the form $(F_{i1}, \ldots, F_{ik_i})$, and for $1 \leq j \leq k_i$, each F_{ij} is a strictly monotone function from $pre(\alpha_{ij})$ to $pre(\gamma)$. Let $rge(F_i)$ be an abbreviation for $\bigcup_{j=1}^{k_i} rge(F_{ij})$. For $1 \leq i \leq n$ let $\beta_i = map\ last\,(pre(\gamma)\lceil rge(F_i))$, where $pre(\gamma)\lceil rge(F_i)$ denotes the set of finite prefixes of γ that belong to the range of one of the scheduling functions $F_{i1}, \ldots, F_{ik_i}$, enumerated in prefix order. Then for each i we have $(\overline{\alpha_i}, \beta_i) \in FM_{k_i}$, using $\overline{F_i}$ as a schedule. Moreover, we can obtain a schedule for $((\beta_1, \ldots, \beta_n), \gamma)$ by defining for $1 \leq i \leq n$ the functions $g_i : pre(\beta_i) \to pre(\gamma)$ so that when $\beta' \in pre(\beta_i)$, $g_i(\beta') = F_{ij}(\alpha')$, where j is the unique index and $\alpha' \in pre(\alpha_{ij})$ is the unique prefix such that $\beta' = map\ last\,(pre(F_{ij}(\alpha'))\lceil rge(\overline{F_i}))$. The relevant existence properties, uniqueness, strict monotonicity, and properties (a), (b) and (c) follow from the assumed properties of the F_{ij} and the definition of the traces β_i. □

Corollary 2.12 *Park's fairmerge is associative.*

Proof. If $((\alpha_1, \alpha_2), \beta) \in FM_2$ and $((\beta, \alpha_3), \gamma) \in FM_2$ then by the previous result, $((\alpha_1, \alpha_2, \alpha_3), \gamma) \in FM_3$. Hence there is a δ such that $((\alpha_2, \alpha_3), \delta) \in FM_2$ and $((\alpha_1, \delta), \gamma) \in FM_2$. The converse argument is similar. Since FM_2 coincides with Park's fairmerge, the result follows. □

It seems more difficult to find a direct proof of associativity using Park's original fixed point formulation of *fairmerge*, or even from the derived formula for *fairmerge* based on iterated concatenation [21]. Furthermore, an attempt to generalize Park's fixed point recipe to produce a fixed point characterization of k-ary fairmerge (perhaps using Bekic's Theorem on iterated fixed points [2]) seems unlikely to yield the right algebraic insights; and it seems more difficult still to characterize k-ary fairmerge using iterated concatenation in a tractable manner, because of the obvious combinatorial explosion. Instead our approach shows the benefits of dealing explicitly with *scheduling functions*, which give a more intensional prefix-by-prefix account of the way the interleaving is built up from the component traces.

We have also proven a number of general results such as Lemma 2.6, Lemma 2.7, and Theorem 2.8, from which analogous results for Park's *fairmerge* can be deduced. While the analogues of Lemma 2.6 and Lemma 2.7 are easy to prove directly in Park's setting, the prefix/suffix properties (Theorem 2.8) are not so straightforward there.

Park's work has served as foundation for many denotational models of concurrent programs, each employing some form of "fair merge" on traces of some kind. The choice of "alphabet" Σ and the kind of interleaving may vary, but the underlying ideas have much in common with the general setting outlined above. For example, in Park's traces the steps are pairs of (global) states representing the effect of a single atomic action, so that $\Sigma = S \times S$, where S is the set of states; in the *transition traces* model [6] each step represents the effect of a finite sequence of atomic actions, and trace sets are closed under *stuttering* and *mumbling*. In *action traces* semantic models such as [7] actions include reads $i{=}v$ and writes $i{:=}v$ to program variables, and the effect of actions on the underlying state is handled separately.

3 Resources and separation

Recent models and logics for shared-memory languages involving mutable state and pointers emphasize *resources* and *separation principles* [17], and rely on resource-sensitive forms of fair interleaving [8]. The tasks of defining a suitable fairmerge relation and establishing associativity and other natural properties are exacerbated by the need to account for resources.

Next we introduce a simple model of *resources*, phrased in abstract terms and with intuitive axioms along lines suggested by [11]. (A *resource model* corresponds to a *separation algebra* in [11].) We extend our framework to incorporate resources, defining a family $\langle RFM_k \mid k > 0\rangle$ of resource-sensitive fairmerge relations that embody *resource separation*: at all stages, the resources "owned" by parallel processes are "compatible". We can recast the prior resource-sensitive models as instances of this framework.

We will assume that resources belong to a set M that is equipped with a partial binary operation $\oplus$, and we say that m_1, m_2 in M are *compatible* iff $m_1 \oplus m_2$ is defined. We also assume that M contains a "zero" element 0, intuitively representing "no resources". When e_1, e_2 are meta-language expressions involving partial operations we write $e_1 \simeq e_2$ to mean that e_1 and e_2 are either both undefined, or both defined and equal.

Definition 3.1 A *resource model* is a partial commutative cancellative monoid $(M, \oplus, 0)$, with $\oplus : M \times M \rightharpoonup M$ and $0 \in M$, satisfying:

- $m_1 \oplus m_2 \simeq m_2 \oplus m_1$
- $m_1 \oplus (m_2 \oplus m_3) \simeq (m_1 \oplus m_2) \oplus m_3$
- $m \oplus m_1 = m \oplus m_2 \Rightarrow m_1 = m_2$
- $m \oplus 0 = 0 \oplus m = m$.

Example 3.2 The following are examples of resource models.

(i) The *trivial* model: the set $\{0\}$, with $0 \oplus 0 = 0$.

(ii) The *resource sets* model: $M = \mathcal{P}_{fin}(R)$, where R is a set of *resource names*, $\oplus$ is disjoint union, and $0 = \{\}$.

(iii) The *local states* model: Let $M = \mathbf{Ide} \rightharpoonup_{fin} V$ be a set of *states*, where **Ide** is the set of identifiers and V is a set of denotable *values*. Define $\oplus : M \times M \rightharpoonup M$ by: $s_1 \oplus s_2 = s_1 \cup s_2$ iff $dom(s_1) \cap dom(s_2) = \{\}$, and let 0 be the empty state.

The reader may consult [11] for further examples of resource models and their utility in formalizing intuitive notions of separation.

The ability of a process to perform actions depends on the resources being held by the process and also on the resources being held by the "environment", i.e. by other processes assumed to be running concurrently; when an action occurs it may affect the resources being held by the process. To describe the interplay between resources and actions abstractly we introduce a notion of *resource enabling*, an infix relation of form $m \vdash m_1 \xrightarrow{\lambda} m_1'$, interpreted as saying that a process with resources m_1, in an environment with resources m, can do λ and then have resources m_1'. The enabling relation is assumed to satisfy some intuitively natural axioms that ensure *separation* and *frame* properties: process and environment resources stay separate, extra process resources are preserved, and extra environment resources that do not disable an action have no effect on process resources.

Definition 3.3 Given a resource model $(M, \oplus, 0)$, and a set of actions Σ, an *enabling relation* is a subset $\vdash \subseteq M \times (M \times \Sigma \times M)$ with the following properties, expressed using infix notation: when $(m, (m_1, \lambda, m_1')) \in \rightarrow$ we write $m \vdash m_1 \xrightarrow{\lambda} m_1'$.

(i) If $m \vdash m_1 \xrightarrow{\lambda} m_1'$ then $m \oplus m_1$ and $m \oplus m_1'$ are defined.

(ii) If $m \vdash m_1 \xrightarrow{\lambda} m_1'$ and $m \vdash m_1 \oplus m_2 \xrightarrow{\lambda} m'$ then $m' = m_1' \oplus m_2$.

(iii) If $m \vdash m_1 \xrightarrow{\lambda} m_1'$ and $m \oplus m_2 \vdash m_1 \xrightarrow{\lambda} m'$, then $m' = m_1'$.

(iv) If $m \oplus m_2 \vdash m_1 \xrightarrow{\lambda} m_1'$ then $m \vdash m_1 \xrightarrow{\lambda} m_1'$.

Definition 3.4 We extend enabling inductively, to finite sequences of actions. When $m \oplus m_1$ is defined, and $\lambda \in \Sigma, \alpha \in \Sigma^*$:

- $m \vdash m_1 \xrightarrow{\epsilon} m_1$.
- $m \vdash m_1 \xrightarrow{\lambda\alpha} m_2'$ if $m \vdash m_1 \xrightarrow{\lambda} m_1'$ and $m \vdash m_1' \xrightarrow{\alpha} m_2'$, for some $m_1' \in M$.

Enabling in an environment with resources 0 is a special case corresponding intuitively to execution without competition for resources. We say that a finite trace α is *executable from* m if and only if there is an m' such that $0 \vdash m \xrightarrow{\alpha} m'$. These notions extend to infinite traces in the obvious way. Finally, we say that a trace α is simply *executable* if it is executable from 0.

Example 3.5 Let $M = \{0\}$ be the trivial resource model. For any set Σ we can define a trivial enabling relation such that $0 \vdash 0 \xrightarrow{\lambda} 0$ holds for all $\lambda \in \Sigma$. For this combination of data, every trace is executable.

Example 3.6 Let $\Sigma = \Delta \cup \{acq(r), rel(r) \mid r \in R\}$ and $M = \mathcal{P}_{fin}(R)$ be the resource sets model. Define the relation $\vdash$ as follows: whenever $A, B \subseteq_{fin} R$ and

$A \cap B = \{\}$,

$$B \vdash A \xrightarrow{acq(r)} A \cup \{r\} \quad \text{if } r \notin A \cup B$$
$$B \vdash A \xrightarrow{rel(r)} A - \{r\} \quad \text{if } r \in A$$
$$B \vdash A \xrightarrow{\lambda} A \quad \text{if } \lambda \in \Delta.$$

This is indeed an enabling relation, and the rules enforce mutual exclusion: each resource name r can be acquired by at most one process, after which other processes trying to acquire r must wait until its release. Traces in which the *acquires* and *releases* for named resources alternate, constrained by mutual exclusion, are common in semantic clauses for synchronization constructs such as conditional critical regions [7,8]. With this enabling relation, a trace α is *executable* if and only if for each $r \in R$ the sequence $\alpha\lceil\{acq(r), rel(r)\}$ is a prefix of $(acq(r)\, rel(r))^\omega$, so at each stage during the execution of the trace, each resource r is "owned" by at most one process.

Given a resource model M and enabling relation $\vdash$ based on Σ and M, we will define a family of "resource-sensitive" fairmerge functions

$$RFM_k : M^k \to \mathcal{P}((\Sigma^\infty)^k \times \Sigma^\infty)$$

as follows. The idea is that when $((\alpha_1, \ldots, \alpha_k), \beta) \in RFM_k(m_1, \ldots, m_k)$ there is a schedule for merging $\alpha_1, \ldots, \alpha_k$ and producing β, that assumes that for each i the process executing trace α_i initially holds resources m_i; at each stage the process taking the next step is enabled by its current resources, and not disabled by the resources of the remaining processes; and the resources held by the processes remain compatible. When these conditions hold we say that β is a resource-sensitive fair merge of traces $(\alpha_1, \ldots, \alpha_k)$ from the initial resource allocation $(m_1, \ldots, m_k)$.

We represent RFM_k as a function from M^k to subsets of $(\Sigma^\infty)^k \times \Sigma^\infty$. The set $RFM_k(m_1, \ldots, m_k)$ will be empty if $m_1, \ldots, m_k$ are incompatible or at some point there is no way to maintain resource compatibility. We may refer to the set $RFM_k(m_1, \ldots, m_k) \subseteq (\Sigma^\infty)^k \times \Sigma^\infty$ as the (resource-sensitive) k-ary fairmerge relation with initial resources $(m_1, \ldots, m_k)$.

Definition 3.7 For each $k \geq 1$ the *resource-sensitive fairmerge* function

$$RFM_k : M^k \to \mathcal{P}((\Sigma^\infty)^k \times \Sigma^\infty)$$

is given by:

$$((\alpha_1, \ldots, \alpha_k), \beta) \in RFM_k(m_1, \ldots, m_k)$$

iff there are strictly monotone functions

$$f_1 : pre(\alpha_1) \to pre(\beta), \ldots, f_k : pre(\alpha_k) \to pre(\beta)$$

and a function $res : pre(\beta) \to M^k$, satisfying the following properties:

(a) For all $i \neq j$, $rge(f_i) \cap rge(f_j) = \{\epsilon\}$.

(b) $pre(\beta) = rge(f_1) \cup \cdots \cup rge(f_k)$.

(c) For all i and all $\alpha' \in pre(\alpha_i)$, $last(f_i(\alpha')) = last(\alpha')$.

(d) $res(\epsilon) = (m_1, \ldots, m_k)$, and for all $\beta' \in pre(\beta)$, if $res(\beta') = (m'_1, \ldots, m'_k)$ then $m'_1 \oplus \cdots \oplus m'_k$ is defined.

(e) For all i and all β' and λ, if $f_i(\alpha'\lambda) = \beta'\lambda$, $res(\beta') = (m'_1, \ldots, m'_k)$, and $res(\beta'\lambda) = (m''_1, \ldots, m''_k)$, then $m \vdash m'_i \xrightarrow{\lambda} m''_i$, where for all $j \neq i$, $m''_j = m'_j$, and $m = \oplus\{m_j \mid 1 \leq j \leq k \;\&\; j \neq i\}$.

In (e) we may also use the abbreviation $\oplus_{j \neq i} m_j$ for m; when $k = 1$ the formula for m degenerates to 0. Condition (d) ensures the desired property that $RFM_k(m_1, \ldots, m_k)$ is empty when $m_1, \ldots, m_k$ are incompatible.

Intuitively, when β is a resource-sensitive fair merge of $(\alpha_1, \ldots, \alpha_k)$ from $(m_1, \ldots, m_k)$, β is a fair merge of $(\alpha_1, \ldots, \alpha_k)$ as before, and at each stage of the merge there is a compatible assignment of resources to processes, such that the process making the next step has enough resource to enable that step, given the resources of the other processes. The scheduling functions $f_1, \ldots, f_k$, together with the initial constraint $res(\epsilon) = (m_1, \ldots, m_k)$, and the enabling relation, determine $res(\beta')$ for all $\beta' \in pre(\beta)$, by induction on trace length.

Theorem 3.8 *For all $m \in M$, $RFM_1(m) = \{((\alpha), \alpha) \mid \alpha$ executable from $m\}$.*

Proof. Follows directly from the definitions. □

Theorem 3.9 *For the trivial resource model (Example 3.5), $RFM_k(0, \ldots, 0)$ coincides with FM_k.*

Proof. In this case let $res(\beta') = (0, \ldots, 0)$ for all $\beta' \in pre(\beta)$. (Indeed, this is the only way to define a resource mapping here.) Then (d) and (e) are vacuous. Conditions (a), (b) and (c) for RFM_k are exactly as in FM_k. □

Theorem 3.10 *For the resource sets model (Example 3.6), when $A \cap B = \{\}$ and $\gamma \in \Sigma^\infty$, $((\alpha, \beta), \gamma) \in RFM_2(A, B)$ iff $\gamma \in \alpha_A \| _B \beta$ as defined in [8].* [4]

Proof. The definition of $\cdot_A \| _B \cdot$ in [8] uses a resourceful "enabling" relation of form $A \xrightarrow[B]{\lambda} A'$, which coincides exactly with $B \vdash A \xrightarrow{\lambda} A'$ as formulated here. If $((\alpha, \beta), \gamma) \in RFM_2(A, B)$ use the schedule and resource assignment to show that $\gamma \in \alpha_A \| _B \beta$. If $\gamma \in \alpha_A \| _B \beta$ use the Axiom of Choice to derive a schedule and resource assignment for $((\alpha, \beta), \gamma)$. □

Theorem 3.10 shows that for the resource sets model $\mathcal{P}_{fin}(R)$, the associated resource-sensitive fairmerge operation RFM_2 maintains the intended mutual exclusion property for each named resource. For example, the only fairmerges of $\alpha = acq(r)\, \lambda_1 \lambda_2\, rel(r)$ and $\beta = acq(r)\, \lambda\, rel(r)$ are $\alpha\beta$ and $\beta\alpha$.

The analogue of 2.8 needs to be phrased carefully to account for resources.

[4] The proviso that $\gamma \in \Sigma^\infty$ here restricts to the *abort*-free subset of the trace model from [8], reflecting the fact that RFM deals with resources but not yet race detection. This mismatch will be repaired in Theorem 4.6 of the next section.

Theorem 3.11 *The functions RFM_k have the following resource-sensitive "prefix/suffix" and "concatenation" properties:*

(i) Let $((\alpha_1, \ldots, \alpha_k), \beta) \in RFM_k(m_1, \ldots, m_k)$, and let $res : pre(\beta) \to M^k$ be the associated resource mapping. If $\beta' \in pre(\beta)$ and $\beta = \beta'\beta''$, then there are prefixes $\alpha'_1 \in pre(\alpha_1), \ldots, \alpha'_k \in pre(\alpha_k)$, and suffixes $\alpha''_1, \ldots, \alpha''_k$, such that $\alpha_1 = \alpha'_1\alpha''_1, \ldots, \alpha_k = \alpha'_k\alpha''_k$, $((\alpha'_1, \ldots, \alpha'_k), \beta') \in RFM_k(m_1, \ldots, m_k)$ and $((\alpha''_1, \ldots, \alpha''_k), \beta'') \in RFM_k(m'_1, \ldots, m'_k)$, where $(m'_1, \ldots, m'_k) = res(\beta')$.

(ii) Suppose $((\alpha'_1, \ldots, \alpha'_k), \beta') \in RFM_k(m_1, \ldots, m_k)$, with resource mapping $res : pre(\beta') \to M^k$, and suppose β' is finite. Let $res(\beta') = (m'_1, \ldots, m'_k)$. If $((\alpha''_1, \ldots, \alpha''_k), \beta'') \in RFM_k(m'_1, \ldots, m'_k)$, then $((\alpha'_1\alpha''_1, \ldots, \alpha_k\alpha''_k), \beta'\beta'') \in RFM_k(m_1, \ldots, m_k)$.

Proof. Adapt the proof of Theorem 2.8 to incorporate resource accounting. For (i) the assumptions, and the properties built into Definition 3.7, allow us to use *res* to construct resource mappings for the prefix merge and for the suffix merge. For (ii) the assumptions ensure that it is possible to combine the resource mappings from the prefix merges and the suffix merges, to obtain a resource mapping for $((\alpha'_1\alpha''_1, \ldots, \alpha'_k\alpha''_k), \beta'\beta'')$. □

We also have the appropriate analogue of Theorem 2.11, adapted to deal with resources. We use abbreviations such as $\overline{m} = (m_1, \ldots, m_k)$ for k-tuples of resources, and $\oplus\overline{m}$ for the combination $m_1 \oplus \cdots \oplus m_k$.

Theorem 3.12 *The functions RFM_n satisfy the following resource-sensitive "general associativity" properties:*

(i) If $(\overline{\alpha_1}, \beta_1) \in RFM_{k_1}(\overline{m_1}), \ldots, (\overline{\alpha_n}, \beta_n) \in RFM_{k_n}(\overline{m_n})$, and $((\beta_1, \ldots, \beta_n), \gamma) \in RFM_n(\oplus\overline{m_1}, \ldots, \oplus\overline{m_n})$, then $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in RFM_{k_1+\cdots+k_n}(\overline{m_1 \ldots m_n})$.

(ii) If $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in RFM_{k_1+\cdots+k_n}(\overline{m_1 \ldots m_n})$ and $len(\overline{\alpha_i}) = len(\overline{m_i}) = k_i$ for $1 \le i \le n$, then there are traces $\beta_1, \ldots, \beta_n$ such that $(\overline{\alpha_1}, \beta_1) \in RFM_{k_1}(\overline{m_1}), \ldots, (\overline{\alpha_n}, \beta_n) \in RFM_{k_n}(\overline{m_n})$ and $((\beta_1, \ldots, \beta_n), \gamma) \in RFM_n(\oplus\overline{m_1}, \ldots, \oplus\overline{m_n})$.

Proof. Adapt the proof of Theorem 2.11 to include resource assignment. The assumptions in (i) and (ii), and properties of M and the enabling relation, are crucial in ensuring that the resource accounting from the subsidiary merges can be dovetailed together properly. In particular, in (i) the components of $\overline{m_1 \ldots m_n}$ are compatible because $\overline{m_1}, \ldots \overline{m_n}$ are compatible by assumption. □

4 Race detection

So far we have avoided explicitly dealing semantically with race conditions, such as an attempt by one process to write to a piece of state being used concurrently by another process. This avoidance is typical of early semantic accounts of shared-memory programs, which assume that assignments and expression evaluations are

executed atomically (as in Park [21]), or that reads and writes to shared variables are atomic (as in [7]). Yet race conditions may lead to unpredictable program behavior and dependence on hardware implementation details beyond the control or knowledge of the programmer.

More recent models incorporate race detection, and treat a potential race as a disaster [8,9]. We now expand our framework in an analogous manner. This expansion meshes well with a subtle gap in the formulation given above of the resource-sensitive fairmerge relations. The definition of RFM_k does not specify what to do when we reach a stage in which two processes, with resources m_i and m_j, have next actions λ_i and λ_j such that each process's resource disables the other process's action. Phrased in terms of our enabling relation, this arises when $m_i \oplus m_j$ is defined but there are no m_i' and m_j' such that $m_j \vdash m_i \xrightarrow{\lambda_i} m_i'$ and $m_i \vdash m_j \xrightarrow{\lambda_j} m_j'$. When this happens condition (e) in the merge definition will be impossible to fulfill. Even when this situation does not hold, there may be pairs of actions, such as two writes to the same variable or heap cell, that we wish to treat as a disaster. We can easily adapt the prior framework to deal with such problems, by introducing an action *abort* to represent runtime error, and a *conflict* (or *interference*) relation on pairs of actions, and extending the merge construction so that a potential conflict produces a trace ending with *abort*. We assume some intuitive axioms that link conflict and the enabling relation.

We extend Σ with a special action *abort* used to represent error. Let $\hat{\Sigma} = \Sigma \cup \{abort\}$. We will treat *abort* as a zero for concatenation on the right: $abort\beta = abort$. Let $\hat{\Sigma}^\infty = \Sigma^\infty \cup \Sigma^* abort$.

Definition 4.1 Given a set of actions Σ, a resource model $(M, \oplus, 0)$, and an enabling relation $\vdash$, a *conflict relation* is a subset $\bowtie \subseteq \hat{\Sigma} \times \hat{\Sigma}$, written as an infix operator, such that:

(i) If $\lambda_1 \bowtie \lambda_2$ then $\lambda_2 \bowtie \lambda_1$.

(ii) $\lambda \bowtie abort$ holds for all $\lambda \in \hat{\Sigma}$.

Obviously there is always a "minimal" conflict relation for a given choice of resource model and enabling relation, characterized as the least subset $\bowtie$ satisfying (i), and (ii). But our definition allows more generality, as shown in the following examples.

Example 4.2 The *trivial* conflict relation (for the trivial resource model and the trivial enabling relation) is $\bowtie = \{(abort, \lambda), (\lambda, abort) \mid \lambda \in \hat{\Sigma}\}$.

Example 4.3 Let $\Sigma = \Delta \cup \{acq(r), rel(r)\}$ as in Example 3.6 and let $\Delta \subseteq \Sigma$ include store and heap actions, as in [8]. We can define a conflict relation $\bowtie$ in which a write conflicts with all actions involving the same variable or heap cell, exactly as in [8], so that a concurrent write to a variable or heap cell being either read or written is regarded as a conflict. We can also define a larger conflict relation $\bowtie' \supseteq \bowtie$ by throwing in all pairs of actions that read the same variable or heap cell, if we want to regard concurrent reads as conflicting.

We can now extend the definition of RFM_k to deal with conflict.

Definition 4.4 Let Σ be a set of actions, let M be a resource model, and let $\bowtie$ be a conflict relation for M and Σ. The conflict-sensitive fair merge functions $\widehat{RFM}_k : M^k \to \mathcal{P}((\hat{\Sigma}^\infty)^k \times \hat{\Sigma}^\infty)$ are given by:

$$((\alpha_1, \ldots, \alpha_k), \beta) \in \widehat{RFM}_k(m_1, \ldots, m_k)$$

iff either $\beta \in \Sigma^\infty$ and $((\alpha_1, \ldots, \alpha_k), \beta) \in RFM_k(m_1, \ldots, m_k)$, or $\beta = \beta'\, abort$ and there are finite prefixes $\alpha'_1 \in pre(\alpha_1), \ldots, \alpha'_k \in pre(\alpha_k)$, indices $i \neq j$, and actions λ_i, λ_j, such that

(a) $((\alpha'_1, \ldots, \alpha'_k), \beta') \in RFM_k(m_1, \ldots, m_k)$

(b) $\alpha'_i\lambda_i \in pre(\alpha_i)$, $\alpha'_j\lambda_j \in pre(\alpha_j)$, and $\lambda_i \bowtie \lambda_j$.

It is possible to augment the earlier proofs to show that the family of $\widehat{RFM}_k$ functions enjoy the same prefix/suffix and concatenation properties as the RFM_k functions, although we omit the details for space reasons.

Theorem 4.5 *For the trivial resource model, trivial enabling relation, and trivial conflict relation, the restriction of $\widehat{RFM}_k(0, \ldots, 0)$ to $(\Sigma^\infty)^k$ coincides with $RFM_k(0, \ldots, 0)$ and hence with FM_k.*

Proof. the actions λ_i and λ_j in (b) are drawn from traces appearing in RFM_k, so belong to Σ. Hence $\lambda_i \bowtie \lambda_j$ is always false here. □

As promised in 3.10 we recover the race-detecting, resource-sensitive model of concurrent separation logic [8] as an instance of our framework.

Theorem 4.6 *For the resource sets model (Example 4.3), when $A \cap B = \{\}$, $((\alpha, \beta), \gamma) \in \widehat{RFM}_k(A, B)$ iff $\gamma \in \alpha_A \|_B \beta$, where $\cdot_A\|_B\cdot$ is the race-detecting resource-sensitive fair merge used in the semantics of concurrent separation logic [8]. In particular, $((\alpha, \beta), \gamma) \in \widehat{RFM}_2(\{\}, \{\})$ iff $\gamma \in \alpha_{\{\}} \|_{\{\}} \beta$.*

Proof. Adapt the proof of Theorem 3.10. The key point is that when $\lambda_1 \bowtie \lambda_2$, $\alpha'\lambda_1 \leq \alpha, \beta'\lambda_2 \leq \beta$, and $\gamma' \in \alpha'_A \|_B \beta'$, the fairmerge from [8] has $\gamma' abort \in \alpha_A \|_B \beta$, matching the race-detection clause of Definition 4.4. □

Again we obtain general associativity.

Theorem 4.7 *The functions $\widehat{RFM}_n$ satisfy:*

(i) If $(\overline{\alpha_1}, \beta_1) \in \widehat{RFM}_{k_1}(\overline{m_1}), \ldots, (\overline{\alpha_n}, \beta_n) \in \widehat{RFM}_{k_n}(\overline{m_n})$, and $((\beta_1, \ldots, \beta_n), \gamma) \in \widehat{RFM}_n(\oplus\overline{m_1}, \ldots, \oplus\overline{m_n})$, then $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in \widehat{RFM}_{k_1+\cdots+k_n}(\overline{m_1 \ldots m_n})$.

(ii) If $(\overline{\alpha_1 \ldots \alpha_n}, \gamma) \in \widehat{RFM}_{k_1+\cdots+k_n}(\overline{m_1 \ldots m_n},)$ and $len(\overline{\alpha_i}) = len(\overline{m_i}) = k_i$ for $1 \leq i \leq n$, then there are traces $\beta_1, \ldots, \beta_n$ such that $(\overline{\alpha_1}, \beta_1) \in \widehat{RFM}_{k_1}(\overline{m_1},), \ldots, (\overline{\alpha_n}, \beta_n) \in \widehat{RFM}_{k_n}(\overline{m_n},)$ and $((\beta_1, \ldots, \beta_n), \gamma) \in \widehat{RFM}_n(\oplus\overline{m_1}, \ldots, \oplus\overline{m_n})$.

Proof. Follows from the general associativity of the RFM_k family, and the assumed properties of $\bowtie$. The prefix/suffix property of RFM_k is also relevant in the error case. □

5 Related work

We developed a framework for defining fairmerge relations, parameterized by a choice of resource model, enabling, and conflict relation. General theorems applicable to all instances of this framework yield uniform proofs of natural properties of some prior semantic models. Our fairness framework is deliberately agnostic as to the choice of "alphabet" Σ, and is sufficiently general to encompass traditional Park-style traces whose steps are pairs of states [6,21], as well as traces built from symbolic "actions" whose effect on the state is handled implicitly [7,8]. Indeed, by keeping the alphabet abstract and working with resource algebras we come closer in spirit to the general approach of [11], which also avoids explicit choices as to the concrete representation of "state".

There is an extensive literature concerning various alternative notions of fair interleaving, characterized in different settings, notably earlier definitions based on automata and oracles [14,15,16,19,22] and a significant collection of work using traces of some kind [10,12,24,26]. A major aim in much of this prior work was to generalize the Kahn Principle [13] to dataflow networks containing non-deterministic processes [1]. A number of strikingly original and surprising results have been obtained concerning the relative expressive power of various variant forms of fair merge in dataflow networks [19,20]. Since fairness is a key factor in the semantics of concurrent programs, it is not surprising that the concepts, and some of the technical details, made explicit in our approach may have implicit or explicit roots in prior literature. However, our development shows how to spell out the details carefully in a general setting, avoiding the use of book-keeping machinery such as oracles, and avoiding the need to tag and untag actions with channel labels that typically appears in dataflow semantics. Moreover, unlike most of the earlier work, our formulation permits straightforward derivation of natural algebraic properties such as associativity.

Perhaps the most closely related work to ours is a technical report by Shanbhogue [25], which uses a trace semantics for dataflow networks based on events of form (h, v), where h is a channel name and v is a data value, and defines the trace set of a *multiway fair merge process* with input channels $i_1, \ldots, i_m$ and output o to consist of all sequences $\alpha \in (\{i_1, \ldots, i_m, o\} \times V)^\infty$ such that there is a function $f : \mathbb{N} \to \{1, \ldots, m\}$ mapping the positional indices of the output events in α to input channel indices, satisfying certain natural constraints. Intuitively, for all n the data value of the n^{th} output in α is obtained from an earlier input in α from channel $i_{f(n)}$, and the relative order of data along each input channel is preserved in the output subsequence. Let $F(i_1, \ldots, i_m; o)$ denote this trace set, augmenting Shanbhogue's notation with explicit indication of the channel names involved in the construction.

Note that Shanbhogue regards fair merge (with respect to a given list of channel names) as a *process* and accordingly describes it as a trace set, whereas we treat fair merge as a semantic operation to be used to construct the trace set of a parallel program from the traces of its constituent processes. Moreover, the indexing function f in Shanbhogue's formulation operates in positional indices and serves a very different purpose from the scheduling functions that appear in the definition of FM. Our formulation, based on prefixes, avoids tags and index manipulation. Shanbhogue's construction is limited to the dataflow setting, and it is not obvious how to generalize it to incorporate resources and race detection. Although his paper is not concerned with algebraic properties such as prefix/suffix and concatenation (Theorem 2.8), he did prove [25] that in any dataflow network built from deterministic processes and multiway fair merge processes, every subnetwork built entirely from two-way merges can be replaced by a network containing a single multiway merge (together with some deterministic processes designed to perform some simple tagging and untagging of data); and that a multiway merge can be replaced by a cascade of two-way merge processes. Ignoring the dataflow trappings, these results look very similar in form to our general associativity theorem (Theorem 2.11), but again it is not so simple to adapt the dataflow proof from [25] to a more general setting. Indeed, our Theorem 2.11 cannot easily be derived in its full generality from Shanbhogue's results.

A concrete connection between our work and Shanbhogue's can be made in the following manner. Suppose $\alpha \in F(i_1, \ldots, i_m; o)$, using index function $f : \mathbb{N} \to \{1, \ldots, m\}$, according to Shanbhogue's construction. For each j such that $1 \leq j \leq m$, let α_j be the subsequence of α containing the input events on channel i_j and the output events whose indices map via f to j. Then $((\alpha_1, \ldots, \alpha_m), \alpha) \in FM_m$ according to our construction, the choice of relevant scheduling functions being obvious. Conversely, suppose for each j we have traces $\alpha_j \in (\{i_j, o\} \times V)^\infty$, such that $data(\alpha_j \lceil o) = data(\alpha_j \lceil i_j)$, and for every prefix β of α_j, $data(\beta \lceil o)$ is a prefix of $data(\beta \lceil i_j)$. (We write $data(\alpha)$ for the sequence of data values contained in the events of α.) If $((\alpha_1, \ldots, \alpha_m), \alpha) \in FM_m$ according to our definition, using scheduling functions $(f_1, \ldots, f_m)$, we will also have $\alpha \in F(i_1, \ldots, i_m; o)$ according to Shanbhogue's definition, the index function f being easy to derive from the f_i.

To illustrate this connection, let α be the trace $((i_1, 0)(i_2, 1)(o, 0)(o, 1))^\omega$ and let β be the trace $((i_1, 0)(i_1, 0)(i_2, 1)(o, 0)(o, 1))^\omega$. Each trace belongs to $F(i_1, i_2; o)$, both relying on the same indexing function $f(n) = n \,\mathbf{mod}\, 2 + 1$, using 0 as the index for the first event in a trace. Corresponding to (α, f) we have that $(((i_1, 0)(o, 0))^\omega, ((i_2, 1)(o, 1))^\omega), \alpha) \in FM_2$, and corresponding to β we have that $(((i_1, 0)(i_1, 0)(o, 0))^\omega, ((i_1, 1)(o, 1))^\omega), \beta) \in FM_2$.

In fact, we can express the dataflow process $F(i_1, \ldots, i_m; o)$ very simply in terms of FM as a fair parallel combination of "unbounded buffer" processes, defined as follows. When i and o are distinct channel names, let $buf(i, o)$ be the trace set given

by

$$buf(i,o) = \{\alpha \in (\{i,o\} \times V)^\infty \mid data(\alpha\lceil o) = data(\alpha\lceil i) \;\&\; \forall \alpha' \in pre(\alpha).\ data(\alpha'\lceil o) \leq data(\alpha\lceil i)\}$$

It follows from the analysis above that, assuming the channels $i_1, \ldots, i_m, o$ are all distinct, for all traces α, we have $\alpha \in F(i_1, \ldots, i_m; o)$ if and only if there are traces $\alpha_1 \in buf(i_1, o), \ldots, \alpha_m \in buf(i_m, o)$ such that $((\alpha_1, \ldots, \alpha_m), \alpha) \in FM_m$.

The relationship between our fair merge relations, which manage to avoid tagging and untagging, and prior work in the dataflow setting, in which tags are prominent and participate crucially in key definitions and results, may be clarified a little by the following observation, which recasts our fair merge construction in an equivalent way that makes more explicit mention of tags. Let $pre(\alpha_1) + \cdots + pre(\alpha_k)$ denote the "coalesced sum" of the ordered sets $(pre(\alpha_i), <)$, which we render concretely as the set

$$\{\epsilon\} \;\cup\; \{(i, \alpha') \mid 1 \leq i \leq k \;\&\; \alpha' \in pre(\alpha_i) \backslash \{\epsilon\}\},$$

with an ordering $\sqsubset$ such that ϵ is the least element, and $(i, \alpha') \sqsubset (j, \beta')$ iff $i = j \;\&\; \alpha' < \beta'$. Then $((\alpha_1, \ldots, \alpha_k), \beta) \in FM_k$ iff there is a strict, monotone injective function F from $(\sum_{i=1}^k pre(\alpha_i), \sqsubset)$ onto $pre(\beta)$, such that for all i and all $\alpha' \in pre(\alpha_i)$, $last(F(i, \alpha')) = last(\alpha')$. Indeed, when such a function F exists, we can obtain scheduling functions $f_i : pre(\alpha_i) \to pre(\beta)$ by letting $f_i(\epsilon) = \epsilon$ and $f_i(\alpha') = F(i, \alpha')$ for all i and all $\alpha' \in pre(\alpha_i)$. And conversely, given scheduling functions $(f_1, \ldots, f_k)$ we can define a suitable function F from $\sum_{i=1}^k pre(\alpha_i)$ onto $pre(\beta)$ by setting $F(\epsilon) = \epsilon$ and $F(i, \alpha') = f_i(\alpha')$ for all i and all $\alpha' \in pre(\alpha_i)$.

We prefer to work directly with the scheduling functions f_i, as in the technical development given above, although it would of course be possible to repeat the entire development based on this alternative description of the *FM* family. Furthermore, we emphasize that tags play an *inessential* rôle in our framework, despite the existence of this alternative characterization that does involve tagging, and our tag-free formulation makes it easier to tackle fairness in a more general setting.

6 Future work

We plan to make further extensions to our framework, starting with a notion of fairmerge parameterized by a family of constraints, to allow expression of refinements such as fairness with respect to channels or semaphores, together with a theorem that characterizes necessary and sufficient conditions on these constraints that ensure that parallel composition is still associative. So far we have dealt with variants of *weak process fairness*, and we plan to augment our formalism to include various forms of weakly fair shared resource (such as a channel or a semaphore), which can be implemented by equipping a shared resource with a priority queue that keeps track of the processes currently blocked and waiting to access the resource.

We will also investigate the interaction between fairmerge and notions of *trace equivalence* or simplification operations on trace sets, such as those that occur in attempts to produce fully abstract models of concurrent languages [6]. For example,

it is common to assume that the alphabet includes an "idle" action δ that behaves as a unit for concatenation, inducing an equivalence relation on traces based on "stuttering" and "mumbling" with respect to idle steps. We would like to identify sufficient conditions on a trace equivalence $\equiv$ that allow the fairmerge relations to be quotiented in the obvious way and ensure that the resulting family $\langle FM_k/_{\equiv}\rangle$ enjoys standard algebraic properties, including generalized associativity, prefix/suffix, and concatenation properties.

We also plan to tackle a wider range of semantics, including models such as [3,4,5,9] based on *permissions*, and to incorporate fairness notions relevant for models of message-passing processes [7]. It would also be interesting to investigate fairness, resources, and race detection in a "true concurrency" setting [23], such as the *pomset* semantics described in [7]. It should also be possible to formulate a general notion of *ownership transfer* [8,17] within our framework, aiming for a streamlined and more general soundness proof for concurrent separation logic (again inspired by [11]). This would involve an enabling relation in which resource acquisition and release combine with transfer of precisely designated pieces of state as in the "local enabling relation" of [8].

Acknowledgement

Thanks to the referees for suggesting a number of improvements and drawing my attention to Shanbhogue's paper [25].

References

[1] S. Abramsky. *A generalized Kahn principle for abstract asynchronous networks.* Proc. 5^{th} MFPS Conference, Springer LNCS 442, pp. 1–21, 1989.

[2] H. Bekič. *Definable Operations in General Algebras, and the Theory of Automata and Flowcharts.* IBM Technical Report, 1969. In: **Programming Languages and Their Definition**, edited by C. B. Jones, Springer LNCS 177, 1984.

[3] R. Bornat, C. Calcagno, P. W. O'Hearn, and M. Parkinson. *Permission accounting in separation logic.* POPL 2005, SIGPLAN Notices, 40(1), 259-270.

[4] R. Bornat, C. Calcagno, and H. Yang. *Variables as Resource in Separation Logic.* MFPS XXI, Birmingham, 2005. ENTCS, Volume 155, 247-276. May 2006.

[5] R. Bornat, M. Parkinson, and C. Calcagno. *Variables as Resource in Hoare Logic.* LICS 2006.

[6] S. Brookes. *Full abstraction for a shared-variable parallel language.* Proc. 8th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press (1993), 98–109. Journal version in: *Inf. Comp.*, vol 127(2):145-163, June 1996.

[7] S. Brookes. *Traces, pomsets, fairness and full abstraction for communicating processes.* Proc. CONCUR 2002, Springer LNCS 2421, 466-482. August 2002.

[8] S. Brookes. *A semantics for concurrent separation logic.* Invited paper, CONCUR 2004, London. Springer LNCS 3170, August 2004. Journal version in: *Theoretical Computer Science*, 375(1-3), May 2007.

[9] S. Brookes. *Variables as Resource for Shared-Memory Programs: Semantics and Soundness.* MFPS 2006, Genova, Italy. ENTCS, Volume 158, 123-150. May 2006

[10] M. Broy. *Semantics of finite and infinite networks of communicating agents.* Distributed Computing, 2:13–31, 1987.

[11] C. Calcagno, P. W. O'Hearn, and H. Yang. *Local Action and Abstract Separation Logic*. LICS 2007.

[12] B. Jonsson. *A fully abstract trace model for dataflow networks*. Proc. 16^{th} POPL Conference, ACM Press, pp. 155-165, 1989.

[13] G. Kahn. *The semantics of a simple language for parallel programming*. In *Information Processing 74*, pp. 993 – 998. North Holland, 1974.

[14] R. Keller. *Denotational models for parallel programs with indeterminate operators*. IFIP Working Conference on the Formal Description of Programming Concepts, St Andrews, New Brunswick, Canada, August 1977.

[15] R. Milner. *An approach to the semantics of parallel programs*. Proceedings of Convegno di Informatica Teorica, Pisa, 1973.

[16] Y. Moschovakis. *A model of concurrency with fair merge and full recursion. Inf. Comp.*, vol. 93:114-171, 1991.

[17] P. W. O'Hearn. *Resources, Concurrency, and Local Reasoning*. Invited paper, CONCUR 2004, London. Springer LNCS 3170, pp. 49-67, August 2004. Journal version in: *Theoretical Computer Science*, 375(1-3), 271-307, May 2007.

[18] S. Owicki and L. Lamport. *Proving liveness properties of concurrent programs*. ACM TOPLAS, 4(3): 455-495, July 1982.

[19] P. Panangaden and E. W. Stark. *Computations, Residuals, and the Power of Indeterminacy*. Proc. 15^{th} ICALP, Springer LNCS 317, 439–454, 1988.

[20] P. Panangaden and V. Shanbhogue. *The Expressive Power of Indeterminate Dataflow Primitives. Inf. Comp.*, vol. 98(1): 99-131, 1992.

[21] D. Park. *On the semantics of fair parallelism*. In: **Abstract Software Specifications**, Springer-Verlag LNCS vol. 86, 504–526, 1979.

[22] D. Park. *Concurrency and Automata on Infinite Sequences*. Proc. Conference on Theoretical Computer Science, Karlsruhe. Springer LNCS 104, pp. 167-183, March 1981.

[23] V. Pratt. *Modeling concurrency with partial orders*. International Journal of Parallel Programming, 15(1):33–71, 1986.

[24] J. Russell. *Full Abstraction and Fixed-Point Principles for Indeterminate Computation*. Ph. D. thesis, Cornell University, Department of Computer Science, April 1990.

[25] V. Shanbhogue. *The Relationship Between Multiway and Two-way Fair Merges*. Technical Report TR 90-1146, Cornell University, Department of Computer Science, August 1990.

[26] J. Staples and V. L. Nguyen. *A Fixpoint Semantics for Nondeterministic Data Flow*. J. ACM, 32(2): 411–444, April 1985.