Runners for Interleaving Algebraic Effects

Niels F.W. Voorneveld*[0000-0001-6650-3493]

Tallinn University of Technology, niels@cs.ioc.ee

Abstract. We study a model of interleaving concurrency for higher-order functional programs with algebraic effects, which can function as a basis for notions of behavioural equivalence of effectful programs within concurrent processes. Using the category of relations to describe nondeterministic functions, we model runs of programs using trace semantics. These traces have actions describing possible program-environment interactions. The functor of traces forms both a monad and a comonad in the category of relations, allowing us to describe programs as both active computations and passive background processes.
We adapt traditional concurrent interleaving semantics for traces as an operation in the category of relations, merging two traces into a set of interwoven traces. These semantics give rise to a runner for the monad, and in some cases form a monad-comonad interaction law. With this, we can simulate an environment of concurrent background processes, and describe the behaviour of a program within such an environment. This runner for interleaving concurrency is readily composable with runners for algebraic effects, allowing us to describe a wide variety of different concurrent effectful scenarios.

Keywords: Algebraic Effects · Interleaving Concurrency · Stateful Runners · Trace Semantics · Program Equivalence.

1 Introduction

Programs are rarely run in isolation. They are executed in a specific environment, using its resources and communicating to it. In the mean time, other programs may be executed in the same environment. How do programs behave in such circumstances where any line of communication with its environment may be interfered with by other background processes? In this paper, we aim to develop a model for describing the behaviour of higher-order functional effectful languages with such interfering background processes.

Interleaving Algebraic Effects. We study in particular programs with *algebraic effects* [23]. These express interactions with the environment in terms of algebraic operations. Each operation is a line of communication with the environment, awaiting a response from the environment and containing a continuation

* Supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

for each possible response. Programs are denoted by combinations of nested operations, which form elements in a *monad* of inductively generated trees [22].

The evaluation of such programs can be modelled by *stateful runners* [29], which consult a global state for *handling* the operations [24]. This gives us a way to find the result of a program, dependent on the initial state of the environment. These runners induce an algebraic theory of equations, relating trees when they are resolved in the same way in each possible state of the environment.

Complications arise when multiple effectful programs run in parallel, since the state of the environment may be changed in unpredictable ways. We investigate such situations using *interleaving concurrency*, which has been widely studied in different areas of computer science, including, but not limited to, labelled transition systems [14], bisimulation [21], and Petri nets [6].

We can apply traditional interleaving semantics, as studied in *communication processes* [5,4] and *actor semantics* [7], to our denotational trees. We work in the category of relations **Rel** (e.g. see [17]) and its subcategory of *total relations* $\mathbf{Rel}_+$, which allow us to express nondeterministic functions. In both categories, we can transform our trees into collections of *traces* describing the branches of the tree, following ideas from *trace semantics* [12,28]. These traces will consist of a list of actions followed by either a result, or an exception. These exceptions are only included if there are nullary operations in the signature of effect operations, for instance an exception labelled $\perp$ for describing divergence.

The interleaving semantics gives a *merge operator* [25] on the monad of traces $\mathtt{T}$, merging two traces into a set of "merged" traces. This merge function, called the *parallel operator*, satisfies a variety of properties, like symmetry and associativity. It gives a binatural transformation if either there are no errors, or we limit ourselves to $\mathbf{Rel}_+$. The parallel operator is however not preserved over program composition, which is necessary for the specification of a *congruent program equivalence* for functional languages. In order to establish preservation over program composition, we need to investigate a different perspective.

Congruent Program Equivalences. Program equivalences for programming languages are studied to answer the question: When can a program safely be replaced by another, without it affecting the behaviour of the entire system? A program equivalence is a relation on programs, which should provide a sufficient condition for guaranteeing this "safe replacement". Program equivalences with this feature are commonly called *congruences*.

Relators [16,27] are a common tool for defining notions of program equivalence. They lift relations on return values to relations on denotations of programs returning those values, following certain considerations of behaviour. They can be used to define a congruent notion of *applicative bisimilarity* [1] for effectful programs, as seen in paper [15]. Motivated by these applications, we use stateful runners to define relators with the right properties. Particularly, we need such relators to be preserved over program composition.

In order to recover preservation over program composition, we use a variant semantics of the parallel operator using *strongly focussed parallel operators*. This

variant parallel operator nondeterministically chooses which program to focus on, whereafter that chosen program *has* to perform the next step. If that chosen program is done, the parallel operator is done, regardless of the state of the program we are currently not focussing on.

This models the situation where one program takes the lead, and the second program is considered as a background process, as part of the environment. To accommodate this shift in perspective, we use the interesting fact that the monad of traces `T` also forms a *comonad* in the category of relations, and comonads are commonly used to model notions of environment [30]. This construction is similar to the treatment of the exponential modality in linear logic as used in [8,9,20].

Contributions. One of the main contributions of this paper is the discovery that the focussed parallel operator satisfies the equations required by `T`-*residual monad-comonad interaction laws* [13] between `T` as a monad, and `T` as a comonad. If we leave out exceptions, this operation forms an interaction law in **Rel**. Regardless of the inclusion of exceptions, the operation induces a stateful runner which we call the *parallel runner* in the category of total relations $\mathbf{Rel}_+$, where the denotation of the second program is part of the state of the environment.

Combining the parallel runner with runners for modelling algebraic effects, we get a runner for modelling effectful programs within environments of other effectful programs running in the background. The parallel runner implements a *program context* with a background process, e.g. `Run` P `during` Q where Q is considered as being run in the background. Importantly, Q is not necessarily evaluated completely, as it may continue to be evaluated after P is done.

The big motivation for this perspective is that runners readily specify relators with the right properties for inducing congruent notions of program equivalence for functional languages such as the untyped lambda calculus, following results from [15]. The final contribution is the development of state-dependent instances of such relators, which gives rise to a notion of equivalence dependent on the initial state of the environment, and which moreover allows for fine-tuning what aspects of the environment state are externally observable. Hence they are a powerful tool for verifying safety of effectful programs that are evaluated in parallel with other programs.

Paper Overview. We assume the reader is familiar with basic concepts from category theory, including but not limited to; natural transformations, monads and monoidal products. In Section 2 we look at the category of relations, and properties of tree monads used for modelling programs. In Section 3, we consider stateful runners on trees, and how to describe trees as collections of traces. Then, in Section 4, we study a variation on the interleaving concurrent operation, as an operation in the category of relations. Combining the above, we can construct runners for a variety of algebraic effects in concurrent environments, and we study how they induce state-dependent relators in Section 5 suitable for formulating program equivalences. We make some final observations in Section 6.

2 The Category of Total Relations

We will give a brief overview of some of the categorical concepts used in this paper. We use the category of relations **Rel** as a basis, which has as objects sets, and as morphisms relations between them. The space of morphisms from X to Y is denoted by $X \multimap Y$, and contains relations $\mathcal{R}$ relating elements from X to elements from Y. Such a relation can be given multiple representations:

1. a relation is a subset of $X \times Y$. This is used mainly in relational reasoning.
2. a relation is a set-theoretic function from X to the powerset $\mathcal{P}(Y)$, called a *nondeterministic function*. This is a common model for nondeterminism.

We write $x \mathcal{R} y$ to say $\mathcal{R}$ relates x to y, and $\mathcal{R}(x)$ for the set $\{y \in Y \mid x \mathcal{R} y\} \in \mathcal{P}(Y)$. The identity relation on X is denoted as $\mathcal{I}_X$, where $x \mathcal{I}_X x'$ if $x = x'$, and relation composition as $\mathcal{R};\mathcal{S}$, where $x \mathcal{R};\mathcal{S} z$ if there is a y such that $x \mathcal{R} y$ and $y \mathcal{S} z$. We write ι for the identity natural transformation, where $\iota_X = \mathcal{I}_X$.

We focus on two important properties a relation can have, which are satisfied by the identity relation, and preserved under composition.

- $\mathcal{R} : X \multimap Y$ is *total* if for any $x \in X$, there is a $y \in Y$ such that $x \mathcal{R} y$. In other words, for any $x \in X$, $\mathcal{R}(x)$ is non-empty.
- $\mathcal{R} : X \multimap Y$ is *thin* if for any $x \in X$ and $y, y' \in Y$, if $x \mathcal{R} y$ and $x \mathcal{R} y'$, then $y = y'$. In other words, for any $x \in X$, $\mathcal{R}(x)$ has at most on element.

Set-theoretic functions $f : X \to Y$ can be represented by total and thin relations between X and Y. This makes the category of sets **Set** the wide subcategory of **Rel** of total and thin relations. We denote the set of total relations from X to Y as $X \rightarrowtail Y$, and consider the wide subcategory of **Rel** of total relations $\mathbf{Rel}_+$. Moreover, **Rel**, $\mathbf{Rel}_+$ are the Kleisli categories over respectively the powerset monad $\mathcal{P}$, and the non-empty powerset monad $\mathcal{P}_+$ in **Set**.

The Cartesian product $\times$ in **Set** can be lifted to a symmetric monoidal product in **Rel**, which sends sets X and Y to their Cartesian product $X \times Y$, and relations $\mathcal{R} : X \multimap X'$ and $\mathcal{S} : Y \multimap Y'$ to $(\mathcal{R} \times \mathcal{S}) : X \times Y \multimap X' \times Y'$, where $(x, y)\,(\mathcal{R} \times \mathcal{S})\,(x', y')$ if $x \mathcal{R} x'$ and $y \mathcal{S} y'$. Since it preserves totality, it also forms a symmetric monoidal product in $\mathbf{Rel}_+$. We write γ for symmetry $\gamma_{X,Y} : X \times Y \multimap Y \times X$, and α for associativity $\alpha_{X,Y,Z} : (X \times Y) \times Z \multimap X \times (Y \times X)$.

The category **Rel** has a *dagger operation* $(-)^\dagger$ which sends morphisms $\mathcal{R} : X \multimap Y$ to $\mathcal{R}^\dagger : Y \multimap X$ defined as: $y \mathcal{R}^\dagger x$ if and only if $x \mathcal{R} y$. This satisfies the properties $(\mathcal{I}_X)^\dagger = \mathcal{I}_X$, $(\mathcal{R};\mathcal{S})^\dagger = \mathcal{S}^\dagger;\mathcal{R}^\dagger$, and $((\mathcal{R})^\dagger)^\dagger = \mathcal{R}$. The dagger operation does not preserve totality or thinness, hence does not exist in **Set** and $\mathbf{Rel}_+$.

We work in **Rel** and $\mathbf{Rel}_+$, since it internalises the nondeterminism and we do not need keep track of the powerset monad $\mathcal{P}$ when composing functions. Instead, we need to check that the operations we define are *natural*.

Let $\mathbf{1}$ be the singleton set $\{*\}$. Consider two families of relations, $d_X : X \multimap \mathbf{1}$ and $c_X : X \multimap X \times X$ ranging over a set X, where $x\, d_X\, *$ and $x\, c_X\, (y, z)$ if $x = y = z$. Considered as functions, these respectively *delete* and *copy* elements. However, neither of the two are natural transformations in **Rel**. Specifically, d_X is only

natural on total relations, hence natural in **Set** and $\mathbf{Rel}_+$, but not **Rel**, and c_X is only natural on thin relations, hence natural in **Set** but not **Rel** and $\mathbf{Rel}_+$.

So **Rel** is particularly restrictive in what it allows as natural transformations, since we can neither delete nor copy. $\mathbf{Rel}_+$ however does allow us to delete and trim data. Though we will do most constructions in **Rel**, we shall keep track of those structures which are natural in $\mathbf{Rel}_+$. We call a transformation *positively natural* if it is natural with respect to total relations. If a transformation is both positively natural and total, it is a natural transformation in $\mathbf{Rel}_+$.

2.1 Trees as Monads and Comonads

We can define inductive data structures using *containers* to describe *signatures* of algebraic effect operations. A container S is a pair (O, ar) given by a set O of operations together with a function $ar : O \to \mathbf{Set}$ associating an *arity* to each operation. We consider several examples of operations.

- We can model printing a message m on screen with an operation $\texttt{output}(m)$ of arity $\mathbf{1} = \{*\}$. Each separate message has a separate operation.
- We can model reading an input from a user with an operation `input` whose arity has all possible responses the user can make.
- We can raise an exception or error using an operation $\texttt{error}(m)$ of arity $\mathbf{0}$.
- We can model a choice (e.g. nondeterministic) with an operation of arity $\mathbf{2}$.

Given a container S, denote programs using such programs with S-trees.

Definition 1. *Given a container $S = (O, ar)$, the endofunctor in **Rel** of S-*trees *sends a set X to the set of trees $\mathtt{M}_S X$ inductively defined by:*

- $\mathit{leaf}(x) \in \mathtt{M}_S X$ *for any* $x \in X$.
- $\mathit{node}(o)(c)$ *for any* $o \in O$ *and* $c : ar(o) \to \mathtt{M}_S X$.

and it sends a relation $\mathcal{R} : X \multimap Y$ to $\mathtt{M}_S(\mathcal{R}) : \mathtt{M}_S X \multimap \mathtt{M}_S Y$, defined as:

- $\mathit{leaf}(x)\, \mathtt{M}_S(\mathcal{R})\, \mathit{leaf}(y)$ *for any* $x \in X$ *and* $y \in Y$ *such that* $x \mathcal{R} y$.
- $\mathit{node}(o)(c)\, \mathtt{M}_S(\mathcal{R})\, \mathit{node}(o)(d)$ *for any* $o \in O$, $c : ar(o) \to \mathtt{M}_S X$ *and* $d : ar(o) \to \mathtt{M}_S Y$, *such that* $\forall i \in ar(o).\, c(i)\, \mathtt{M}_S(\mathcal{R})\, d(i)$.

$\mathtt{M}_S X$ as an endofunctor in **Set** is actually the *free monad* over the functor $\mathtt{F}_S X = \Sigma_{o \in O}(X^{ar(o)})$. Hence $\mathtt{M}_S$ in **Rel** can be seen as a lifting of that free monad to the Kleisli category of $\mathcal{P}$, using a distributivity law of the free monad over $\mathcal{P}$.

$\mathtt{M}_S$ is a monad in **Rel**, since we can lift the monad structure from **Set**. The unit transformation $\eta^S_X : X \multimap \mathtt{M}_S X$ is defined as $\eta^S_X(x) = \{\mathit{leaf}(x)\}$, and the multiplication relation $\mu^S_X : \mathtt{M}_S \mathtt{M}_S X \multimap \mathtt{M}_S X$ is inductively defined as $\mu^S_X(\mathit{leaf}(t)) = \{t\}$ and $\mu^S_X(\mathit{node}(o)(c)) = \{\mathit{node}(o)(e) \in \mathtt{M}_S X \mid \forall i \in ar(o),\, e(i) \in \mu^S_X(c(i))\}$. Both are natural transformations consisting solely of total and thin relations, which reflects the fact that they were lifted from the monad structure in **Set**.

Lemma 1. *Suppose (M, η, μ) is a monad in **Rel** with the additional property that for any relation $\mathcal{R} : X \multimap Y$, $(M(\mathcal{R}))^\dagger = M(\mathcal{R}^\dagger)$, then $(M, \eta^\dagger, \mu^\dagger)$ is a comonad in **Rel**.*

Indeed, $\mathtt{M}_S$ satisfies the property required in Lemma 1, so $(\mathtt{M}_S, (\eta^S)^\dagger, (\mu^S)^\dagger)$ is a comonad. We give an alternative inductive definition for the comonadic operations. Let $\varepsilon^S_X : \mathtt{M}_S X \multimap X$ and $\delta^S_X : \mathtt{M}_S X \multimap \mathtt{M}_S\mathtt{M}_S X$ be inductively defined as:

- $\varepsilon^S_X(\mathit{leaf}(x)) = \{x\}$, and $\varepsilon^S_X(\mathit{node}(o)(c)) = \varnothing$.
- $\delta^S_X(\mathit{leaf}(x)) = \{\mathit{leaf}(\mathit{leaf}(x))\}$, and $\delta^S_X(\mathit{node}(o)(c)) = \{\mathit{leaf}(\mathit{node}(o)(c))\} \cup \{\mathit{node}(o)(d) \mid \forall i \in ar(o).\, c(i)\, \delta^S_X\, d(i)\}$.

Lemma 2. *$\varepsilon^S_X = (\eta^S_X)^\dagger$ and $\delta^S_X = (\mu^S_X)^\dagger$, hence $(\mathtt{M}_S, \varepsilon^S, \delta^S)$ is a comonad.*

Now, ε is not total, hence it is not an operation in $\mathbf{Rel}_+$ and **Set**, though it is a natural transformation in **Rel**. On the other hand, δ is not thin, hence it is not an operation in **Set**, though it is a natural transformation in both **Rel** and $\mathbf{Rel}_+$. Let us look at some additional properties which are satisfied:

Lemma 3. *The following four equations hold:*

- $\eta^S; \varepsilon^S = \iota$, $\quad \delta^S; \mu^S = \iota(\mathtt{M}_S)$, $\quad \mu^S; \varepsilon^S = \varepsilon^S(\varepsilon^S)$, $\quad$ *and* $\eta^S; \delta^S = \eta^S(\eta^S)$.

2.2 The Occasional Strength of Trees

Strength is an important property for monads when modelling programming languages. For example, it allows us to schedule the order of evaluation of programs. To define strength, consider the family of relations $\sigma^S_{X,Y} : X \times \mathtt{M}_S Y \multimap \mathtt{M}_S(X \times Y)$ over two set parameters X and Y, defined inductively as:

- $\sigma^S_{X,Y}(x, \mathit{leaf}(y)) = \{\mathit{leaf}(x, y)\}$.
- $\sigma^S_{X,Y}(x, \mathit{node}(o)(c)) = \{\mathit{node}(o)(d) \mid \forall i \in ar(o).\, d(i) \in \sigma_{X,Y}(x, c(i))\}$.

This is lifting the natural operation for strength of $\mathtt{M}_S$ as an endofunctor in **Set**, to a family of total and thin relations in **Rel**. This family is natural in Y, but not necessarily natural in X, due to previous observations related to copying and deleting. If the signature $S = (O, ar)$ has an operation $o \in O$ such that the set $ar(o)$ has more than one element, then σ^S is able to copy X. If the signature $S = (O, ar)$ has an operation $o \in O$ such that the set $ar(o) = \varnothing$, then σ^S is able to delete X. As a consequence, we have the following consequences:

1. σ^S is natural in **Rel** if and only if for each $o \in O$, $|ar(o)| = 1$.
2. σ^S is natural in $\mathbf{Rel}_+$ if and only if for each $o \in O$, $|ar(o)| \leq 1$.
3. σ^S is natural in **Set**, regardless of the signature.

Because of the above, we consider the following subclass of tree monads:

Definition 2. *Given sets A and E, which we respectively call actions and exceptions, we define the monad of (A,E)-traces as the monad of trees over the signature $(A+E, ar)$, where $ar(inl(a)) = \{*\}$ for any $a \in A$ (unary) and $ar(inr(e)) = \varnothing$ for any $e \in E$ (nullary). We denote this monad as $(\mathtt{T}_{A,E}, \eta^{A,E}, \mu^{A,E})$.*

We may write $(a)t$ or $@t$ for $node(inl(a))(t)$, and $\langle e\rangle$ or $\diamondsuit e$ for $node(inr(e))()$. Lastly, we may write $[x]$ for $leaf(x)$, hence using a different bracket type for each of the three possible constructors of the monad. Given a set X, the set $\mathtt{T}_{A,E}X$ is isomorphic to $A^* \times (X + E)$, where A^* is the set of lists over A. For example, the maybe monad can be given by $\mathtt{T}_{\varnothing,\{\bot\}}$. For any A and E, $\mathtt{T}_{A,E}$ is a strong monad in $\mathbf{Rel}_+$, and for any A, $\mathtt{T}_{A,\varnothing}$ is a strong monad and a comonad in $\mathbf{Rel}$.

Lemma 4. *For any signature S, and sets X and Y,*

- $(\eta^S_X \times \iota_Y); \sigma^S_{X,Y} = \eta^S_{X\times Y}$ *and* $(\mu^S_X \times \iota_Y); \sigma^S_{X,Y} = \sigma^S_{\mathtt{M}_S X,Y}; \mathtt{M}_S(\sigma_{X,Y}); \mu_{X\times Y}$.
- $\sigma^S_{X,Y}; \varepsilon^S_{X\times Y} = \varepsilon^S_X \times \iota_Y$ *and* $(\delta^S_X \times \iota^S_Y); \sigma^S_{\mathtt{M}_S X,Y}; \mathtt{M}_S(\sigma^S_{X,Y}) = \sigma^S_{X,Y}; \delta^S_{X\times Y}$.

We leave out subscripts and superscripts when they are obvious from context.

3 Nondeterministic Stateful Trace Runners

We model the behaviour of effectful programs by resolving algebraic effect operations by consulting some environmental state. This is done by stateful runners in the category of total relations $\mathbf{Rel}_+$. We shall first look at runners for trace monads, whereafter we shall see how they can be extended to runners on tree monads as well. Firstly, we look at runners in general, as they appear in [29,13].

Definition 3. *Given a monoidal closed category with monads M and N, an N-*residual runner *on M is given by an object K and a natural transformation $\theta_X : MX \times K \to N(X \times K)$ such that:*

$$\begin{array}{ccc} X\times K & & \\ {\scriptstyle \eta^M_X\times K}\downarrow & \searrow^{\eta^N_{X\times K}} & \\ MX\times K & \xrightarrow[\theta_X]{} & N(X\times K) \end{array} \qquad \begin{array}{ccccc} MMX\times K & \xrightarrow{\theta_{MX}} & N(MX\times K) & \xrightarrow{N(\theta_X)} & NN(X\times K) \\ {\scriptstyle \mu^M_X\times K}\downarrow & & & & \downarrow{\scriptstyle \mu^N_{X\times K}} \\ MX\times K & & \xrightarrow[\theta_X]{} & & N(X\times K) \end{array}$$

For instance, if M is a strong monad, then for each object K the strength transformation σ (with symmetry) forms an M-residual runner on M.

Definition 4. *Given sets A, E, A', E', a trace runner from (A, E) to (A', E') is a $\mathtt{T}_{A',E'}$-residual runner on $\mathtt{T}_{A,E}$ in $\mathbf{Rel}_+$.*

Lemma 5. *Trace runners from (A, E) to (A', E') are in 1-to-1 correspondence with triples consisting of a set K and two morphisms:*

- *a morphism $A \times K \nrightarrow \mathtt{T}_{A',E'}K$ (or $A \times K \nrightarrow A'^* \times (K + E')$),*
- *a morphism $E \times K \nrightarrow \mathtt{T}_{A',E'}\mathbf{0}$ (or $E \times K \nrightarrow A'^* \times E'$).*

Proof. Given $f : A \times K \nrightarrow \mathtt{T}_{A',E'}K$ and $g : E \times K \nrightarrow \mathtt{T}_{A',E'}\mathbf{0}$ we inductively define $\theta_X : \mathtt{T}_{A,E}X \times K \nrightarrow \mathtt{T}_{A',E'}(X \times K)$ as:

$\theta_X([x], k) = [x, k]$, $\quad \theta_X(@(t), k) = (\mathtt{T}_{A',E'}(\lambda v.\theta_X(t, v)); \mu^{A',E'})(f(a, k))$,
$\theta_X(\diamondsuit e, k) = \mathtt{T}_{A',E'}()(g(e, k))$.

Vice versa, given $\theta_X : \mathtt{T}_{A,E}X \times K \nrightarrow \mathtt{T}_{A',E'}(X \times K)$, we define $f : A \times K \nrightarrow \mathtt{T}_{A',E'}K$ as $\lambda(a, k).\theta_{\mathbf{1}}(@[*], k)$ using the isomorphism $\mathbf{1} \times K \simeq K$, and we define $g : E \times K \nrightarrow \mathtt{T}_{A',E'}\mathbf{0}$ as $\lambda(e, k).\theta_{\mathbf{0}}(\diamondsuit e, k)$ using the isomorphism $\mathbf{0} \times K \simeq \mathbf{0}$.

We call the morphisms from Lemma 5 the *local functions* for the runner. In **Rel**, a runner can never raise an error when resolving an action. This severely restricts the examples we can model, and hence we focus on runners in $\mathbf{Rel}_+$.

Following the theory on runners [29], we can compose them in the following way. Given a trace runner θ from (A_1, E_1) to (A_2, E_2) on state space K_1, and a trace runner ϕ from (A_2, E_2) from (A_3, E_3) on state space K_2, we can compose them into a trace runner $\theta \bullet \phi$ from (A_1, E_1) to (A_3, E_3) on state space $K_1 \times K_2$, given by $(\theta \bullet \phi)_X = (\theta_X \times \iota_{K_2}); \phi_{X \times K_1} : \mathtt{T}_{A_1,E_1} X \times K_1 \times K_2 \nrightarrow \mathtt{T}_{A_3,E_3}(X \times K_1 \times K_2)$, using associativity of $\times$.

3.1 The Monad Morphism of Branches

Let us consider runners on trees as well.

Definition 5. *Given a container S and sets A and E, a* tree runner *from S to (A, E) is a $\mathtt{T}_{A,E}$-residual runner on $\mathtt{M}_S$ in $\mathbf{Rel}_+$.*

We can use trace runners to define runners on trees. This is done by defining an operation which takes a tree $\mathtt{M}_S X$, and collects all the branches of the tree. We define the set of S-actions as the set $A(S) = \{(o, i) \mid o \in O, i \in ar(o)\}$, and the set of S-errors as $E(S) = \{o \in O \mid ar(o) = \varnothing\}$. The action (o, i) signifies the operation o being called, and the response i being given to the operation. Consider the family of operations $\beta^S_X : \mathtt{M}_S X \multimap \mathtt{T}_{A(S),E(S)} X$, defined by

- $\beta^S_X(\mathit{leaf}(x)) = \{\mathit{leaf}(x)\}$,
- $\beta^S_X(\mathit{node}(o)(c)) = \{(o, i)(t) \mid i \in ar(o), t \in \beta^S_X(c(i))\}$ if $ar(o)$ is non-empty,
- $\beta^S_X(\mathit{node}(o)(c)) = \{\langle o \rangle\}$ if $ar(o)$ is empty.

If S has any arity with more than one element, β^S_X is not natural in **Rel**. However, β^S_X is both total and natural in $\mathbf{Rel}_+$.

Lemma 6. *In $\mathbf{Rel}_+$, β^S forms a monad morphism from $\mathtt{M}_S$ to $\mathtt{T}_{A(S),E(S)}$.*

As a direct consequence, we can transform a trace runner from $(A(S), E(S))$ to (A', E'), to a tree runner from S to (A', E').

3.2 Examples

We consider some examples using the maybe monad $\mathtt{T}_{\varnothing,\{\bot\}} = (-)_\bot$ as residual. In each case, we specify the signature S, and define the runner on $\mathtt{T}_{A(S),E(S)}$ by specifying its local functions (see Lemma 5).

Example 1. Consider a set of messages M, and for each $m \in M$ an output operation $\mathtt{output}(m)$ of arity $\mathbf{1}$. In this case, $A(S) = \{(\mathtt{output}(m), *) \mid m \in M\} \simeq M$ and $E(S) = \varnothing$. A simple trace runner to consider is one that records all the outputs in a single list. We take as state space $K = M^*$, and define the runner with the local function $f : M \times M^* \nrightarrow (M^*)_\bot$ using append $f(m, \tau) = \{m : \tau\}$.

Example 2. Consider a signature with a single input operation $\texttt{input}$ of arity M, giving us $A(S) = \{(\texttt{input}, m) \mid m \in M\} \simeq M$ and $E(S) = \varnothing$. The runner consults some oracle of inputs, and verifies that the right response is made. As state space, we use $M^{\mathbb{N}}$, and the runner is given by the local function $f : M \times M^{\mathbb{N}} \nrightarrow (M^{\mathbb{N}})_\perp$ where: $f(m, s) = \{\lambda n.s(n+1)\}$ if $s(0) = m$, and $\{\perp\}$ otherwise.

Example 3. We model global store over some global state M by taking as operations: for each $m \in M$ an operation $\texttt{update}(m)$ of arity $\mathbf{1}$, and an operation $\texttt{lookup}$ of arity M. We simplify $A(S) = \{m!, m? \mid m \in M\}$ where $m! = (\texttt{update}(m), *)$ and $m? = (\texttt{lookup}, m)$. We define a runner over state space M with the local function $f : A(S) \times M \nrightarrow M_\perp$ where: $f(m!, n) = \{m\}$, and $f(m?, n) = \{n\}$ if $m = n$, otherwise $\{\perp\}$.

Example 4. Consider a single choice operation $\texttt{or}$ of arity $\mathbf{2} = \{0, 1\}$, hence our actions $A(S)$ are in bijection to $\mathbf{2}$. Suppose $\texttt{or}$ models some unbalanced nondeterministic choice, where the right choice only happens when some fee is paid, whereas the left choice is free. As state space we take $\mathbb{N}$, which tells us how many times we can pay the fee, and define the runner with the local function $f : \mathbf{2} \times \mathbb{N} \nrightarrow \mathbb{N}_\perp$ where $f(0, n) = \{n\}$, $f(1, 0) = \{\perp\}$ and $f(1, n+1) = \{n\}$.

4 Interleaving Concurrency

Having established our main way of modelling effect behaviour in terms of runners, we now start looking at interleaving concurrency. We study a variation on the standard interleaving semantics, and use it to formulate a trace runners. This variation is necessary to ensure that the right properties hold, especially the unit equation for trace runners.

Definition 6. *The interleaving concurrency operations are three mutually inductive families of operations* $\mathbb{P}, \mathbb{L}, \mathbb{R} : TX \times TY \multimap T(X \times Y)$ *where:*

1. $\mathbb{P}(l, r) = \mathbb{L}(l, r) \cup \mathbb{R}(l, r)$.
2. $\mathbb{L}([x], r) = [x, \varepsilon(r)]$.
3. $\mathbb{L}(\textcircled{@} l, r) = \textcircled{@}\mathbb{P}(l, r)$.
4. $\mathbb{L}(\diamondsuit e, r) = \diamondsuit e$.
5. $\mathbb{R}(l, r) = T(\gamma)(\mathbb{L}(r, l))$.

We call $\mathbb{L}$ the *left-focussed parallel operator*, and $\mathbb{R}$ the *right-focussed parallel operator*. The main difference between the above semantics and traditional interleaving semantics (as e.g. in process algebra [4]) is in the treatment of the termination case. Here, $\mathbb{L}([x], r)$ will only give a result if r immediately terminates as well (is a leaf). If not, $\mathbb{L}([x], r) = \varnothing$. More traditionally, $\mathbb{L}([x], r)$ is taken to be $\sigma_{X,Y}(x, r)$. Despite this, we do have the following result:

Lemma 7. *The following equation holds:* $\mathbb{P}([x], r) = \sigma(x, r)$.

As a direct consequence, the map $\mathbb{P}$ does not change if we change the definition of $\mathbb{L}$ to give $\mathbb{L}([x], r) = \sigma(x, r)$ instead of $\mathbb{L}([x], r) = [x, \varepsilon(r)]$. We keep the formulation from Definition 7 for the following two reasons: 1) The left parallel operator

$\mathbb{L}$ has a richer structure which allow us to formulate trace runners, and 2) this gives less duplicate results which makes it less cumbersome in formalisation efforts (less cases in set equality proofs). We make the following observations.

- $\mathbb{P}$ is total, $\mathbb{L}$ and $\mathbb{R}$ are not total if there is at least one action or exception.
- $\mathbb{P}$, $\mathbb{L}$ and $\mathbb{R}$ are positively natural over X and Y, hence $\mathbb{P}$ is natural in $\mathbf{Rel}_+$.
- If E is empty, then $\mathbb{P}$, $\mathbb{L}$ and $\mathbb{R}$ are natural in $\mathbf{Rel}$.

Lemma 8. *$\mathbb{P}$ is associative and symmetric, as it satisfies the equations:*

- $(\mathbb{P} \times \iota); \mathbb{P}; T(\alpha) = \alpha; (\iota \times \mathbb{P}); \mathbb{P}$,
- $(\mathbb{L} \times \iota); \mathbb{L}; T(\alpha) = \alpha; (\iota \times \mathbb{P}); \mathbb{L}$,
- $(\mathbb{R} \times \iota); \mathbb{L}; T(\alpha) = \alpha; (\iota \times \mathbb{L}); \mathbb{R}$,
- $(\mathbb{P} \times \iota); \mathbb{R}; T(\alpha) = \alpha; (\iota \times \mathbb{R}); \mathbb{R}$.
- $\mathbb{P}; T(\gamma) = \gamma; \mathbb{P}$.
- $\mathbb{L}; T(\gamma) = \gamma; \mathbb{R}$.
- $\mathbb{R}; T(\gamma) = \gamma; \mathbb{L}$.

4.1 Monadic and Comonadic Properties

We will study how the interleaving transformations interact with both the monad and comonad structure of our monad $\mathtt{T}$. First of all, consider the following.

Lemma 9. *The following equation and point-wise inclusion hold:*

$$\begin{array}{ccc} X \times Y & = & X \times Y \\ {\scriptstyle \eta\times\eta}\downarrow & & \downarrow{\scriptstyle \eta} \\ TX \times TY & \multimap_{\mathbb{P}} & T(X \times Y) \end{array} \qquad \begin{array}{ccccc} TTX \times TTY & \xrightarrow{\mathbb{P}} & T(TX \times TY) & \xrightarrow{T\mathbb{P}} & TT(X \times Y) \\ {\scriptstyle \mu\times\mu}\downarrow & & \cap & & \downarrow{\scriptstyle \mu} \\ TX \times TY & & \xrightarrow[\mathbb{P}]{} & & T(X \times Y) \end{array}$$

This shows that $\mathtt{T}$ is a monoidal monad in $\mathbf{Rel}_+$ in a *lax* sense. In relation to the comonad structure however, there is a stronger result.

Lemma 10. *The following two equations hold:*

$$\begin{array}{ccc} TX \times TY & \xrightarrow{\mathbb{P}} & T(X \times Y) \\ {\scriptstyle \varepsilon\times\varepsilon}\downarrow & & \downarrow{\scriptstyle \varepsilon} \\ X \times Y & = & X \times Y \end{array} \qquad \begin{array}{ccccc} TX \times TY & & \xrightarrow{\mathbb{P}} & & T(X \times Y) \\ {\scriptstyle \delta\times\delta}\downarrow & & & & \downarrow{\scriptstyle \delta} \\ TTX \times TTY & \xrightarrow[\mathbb{P}]{} & T(TX \times TY) & \xrightarrow[T\mathbb{P}]{} & TT(X \times Y) \end{array}$$

We could call $\mathtt{T}$ a monoidal comonad, though note that ε is not total. Hence $\mathtt{T}$ is only a monoidal comonad in $\mathbf{Rel}$ and only when $E = \varnothing$. Last but not least, we present equations which mix the monad and comonad structures.

Lemma 11. *The following two equations hold:*

$$\begin{array}{ccc} & {\scriptstyle \iota\times\varepsilon} & X\times Y = X\times Y \\ X \times TY & \swarrow & \downarrow{\scriptstyle \eta} \\ & {\scriptstyle \eta\times\iota} & TX \times TY \xrightarrow[\mathbb{L}]{} T(X\times Y) \end{array} \qquad \begin{array}{ccccc} & {\scriptstyle \iota\times\delta} & TTX\times TTY & \xrightarrow{\mathbb{L}} & T(TX\times TY) & \xrightarrow{T\mathbb{L}} & TT(X\times Y) \\ TTX \times TY & \swarrow & & & & & \downarrow{\scriptstyle \mu} \\ & {\scriptstyle \mu\times\iota} & TX \times TY & & \xrightarrow[\mathbb{L}]{} & & T(X\times Y) \end{array}$$

Proof (Notes). The second equation can be shown by mutual induction with properties: 3. $\mathbb{P}(\mu(d), r) = \mu(\mathtt{T}\mathbb{L}(\mathbb{P}(d, \delta(r))))$, 4a. $\mathbb{R}(\mu(d), r) \subseteq \mu(\mathtt{T}\mathbb{L}(\mathbb{P}(d, \delta(r))))$, and 4b. $\mathbb{P}(\mu(d), r) \supseteq \mu(\mathtt{T}\mathbb{L}(\mathbb{R}(d, \delta(r))))$. Property 3 can be directly shown using the other three properties, whereas those other properties can be proven by case analysis on the trace they focus on, using the induction hypothesis on property 3.

The above result shows a strict preservation over program composition, and as a direct consequence, we can observe a connection to interaction laws [13].

Corollary 1. *If $E = \varnothing$, then in the category of relations **Rel**, $\mathbb{L}$ forms a T-residual monad-comonad interaction law between T and T.*

4.2 The Parallel Runner

Following Lemma 11 and the theory developed in [13], we know that the following construction gives us a trace runner.

Definition 7. *For a set U and a trace monad T, the U-parallel runner is the trace runner $\rho^U_X : TX \times TU \nrightarrow T(X \times TU)$ defined by $\rho^U_X = (\iota_{TX} \times \delta_U); \mathbb{L}$.*

This runner simulates running a program modelled by a set of traces from $\mathtt{T}X$ in parallel with some background process modelled by a set of traces from $\mathtt{T}U$. This background process need not finish when the program finishes, and a remainder of its trace may carry over to continuations of the program.

The parallel runner goes hand in hand with the left-focussed parallel transformation $\mathbb{L}$, since the former is defined by the latter, and the latter can be retrieved from the former: $\mathbb{L}(l, r) = \mathtt{T}(\iota \times \varepsilon)(\rho(l, r))$. Through Lemma 5, we can find the local definition of the parallel runner, given as follows:

- The morphism $A \times K \nrightarrow \mathtt{T}K$, where $K = \mathtt{T}Y$, sends (a, r) to $(a)\delta_U(r)$.
- The morphism $E \times K \nrightarrow \mathtt{T}\mathbf{0}$ sends (e, r) to $\langle e \rangle$.

As a culmination of previous results, we can observe some extra properties.

Lemma 12. *The following two equations hold, modulo associativity:*

- $(\rho^Y_X \times \iota_{TZ}); \rho^Z_{X \times TY}; T(\iota_X \times \gamma_{TY,TZ}) = (\iota_{TX} \times \gamma_{TY,TZ}); (\rho^Z_X \times \iota_{TY}); \rho^Y_{X \times TZ}$,
- $(\rho^Y_X \times \iota_{TZ}); \rho^Z_{X \times TY}; T(\iota_X \times \mathbb{P}_{Y,Z}) = (\iota_{TX} \times \mathbb{P}_{Y,Z}); \rho^{(Y \times Z)}_X$.

Definition 8. *The U-*concurrent completion *of a trace runner θ from (A, E) to (A', E') on state space K is the trace runner $\theta^{|U}$ from (A, E) to (A', E') on state space $T_{A,E}U \times K$ given by the composition $(\rho^U \bullet \theta)$.*

We commonly take the **1**-concurrent completion of a trace runner, since our focus is on the program and not on the result of the background process. If we want to study the background process instead, we would shift focus and use the runner the other way around. Note that by Lemma 12, taking two concurrent completions is similar to taking one. In other words, running in parallel with two processes separately is, from the perspective of the program, the same as running in parallel with the merger of the two processes together.

5 Stateful Relational Reasoning

Consider a tree runner from S to (A, E). This runner induces a relation on trees over the signature S. In this section, we shall formulate such relations dependent on the state of the environment, and look at examples of such relations in scenarios of algebraic effects with concurrency.

5.1 Relators

When one wants to study the behaviour of programs denoted by a monad M, one may need tools to relate them. Given some notion of relatedness on values as a relation between two objects X and Y, we want to define a notion of relatedness between programs that produce these values; a relation between MX and MY. This can be done using a *relator* [16,27].

Definition 9. *Given a monad M in **Set**, a* relator Γ *for M is a family of functions $\{\Gamma_{X,Y}\}_{X,Y}$ sending $\mathcal{R} \subseteq X \times Y$ to $\Gamma_{X,Y}(\mathcal{R}) \subseteq MX \times MY$ such that:*

Identity: $\mathcal{I}_{MX} \subseteq \Gamma_{X,X}(\mathcal{I}_X)$.
Composability: $\Gamma(\mathcal{S}); \Gamma(\mathcal{R}) \subseteq \Gamma(\mathcal{S}; \mathcal{R})$.
Order preservation: *If $\mathcal{R} \subseteq \mathcal{S}$, then $\Gamma(\mathcal{R}) \subseteq \Gamma(\mathcal{S})$.*
Naturality: *For $f : X \to Z$ and $g : Y \to W$, $M(f)(a)\,\Gamma(\mathcal{R})\,M(g)(b)$ holds if and only if $a\,\Gamma(\{(x,y) \mid f(x)\,\mathcal{R}\,g(y)\})\,b$.*

The lifting of M to **Rel** is an example of a relator, though the main examples we study in this paper are not of that form. There are two properties we would like a relator to satisfy in order for it to induce a congruent notion of equivalence.

Definition 10. *A relator Γ on a monad M is* monadic *if:*

Unit: *If $x\,\mathcal{R}\,y$ then $\eta^M_X(x)\,\Gamma(\mathcal{R})\,\eta^M_Y(y)$.*
Multiplication: *If $d\,\Gamma(\Gamma(\mathcal{R}))\,e$ then $\mu^M_X(d)\,\Gamma(\mathcal{R})\,\mu^M_Y(e)$.*

We shall focus on relators on tree monads (which include trace monads). In particular, the examples we have studied were formulated by runners to the exception monad $\mathtt{T}_{\varnothing,E}X \simeq X + E$. As such, we shall first look at relators there.

Example 5. Specifying a subset $V \subseteq E$ of detectable errors, we can define a relator Γ_V on $\mathtt{T}_{\varnothing,E}$ given by:

- $[x]\,\Gamma_V(\mathcal{R})\,[y]$ if $x\,\mathcal{R}\,y$, $\quad \diamondsuit\!\!\!\!\!\!\diamondsuit\, \Gamma_V(\mathcal{R})\, \diamondsuit\!\!\!\!\!\!\diamondsuit$, $\quad \diamondsuit\!\!\!\!\!\!\diamondsuit\, \Gamma_V(\mathcal{R})\, t$ for any t if $e \notin V$.

A common sub-example is to take $E = \{\bot\}$ and $V = \varnothing$, in which case we get the standard relator $\Gamma_{\{\bot\}}$ on the maybe monad. This makes the $\bot$ exception not observable by the relator, which is useful in such cases where we model programs with undecidable termination, and non-termination is marked by $\bot$.

To deal with nondeterminism, we use a relator from the literature (e.g. [15]).

Definition 11. *We define a monadic relator $\overline{\mathcal{P}_+}$ on the monad $\mathcal{P}_+(-)$, where: $V\,\overline{\mathcal{P}_+}(\mathcal{R})\,W$ if for any $x \in V$, there is a $y \in W$ such that $x\,\mathcal{R}\,y$.*

A tree runner from S to (A,E) gives an $\mathcal{P}_+\mathtt{T}_{A,E}$-residual runner on $\mathtt{M}_S$ in **Set**. Hence it gives a natural transformation $\theta_X : \mathtt{M}_S X \times K \to \mathcal{P}_+\mathtt{T}_{A,E}(X \times K)$. We shall use this runner to define a relator on $\mathtt{M}_S$. As base, we shall specify a relator Γ on $\mathtt{T}_{A,E}$, and compose it with the relator $\overline{\mathcal{P}_+}$ on $\mathcal{P}_+$, creating a relator on $\mathcal{P}_+\mathtt{T}_{A,E}$ which we denote by $\Gamma^{\mathcal{P}_+}$. In order for $\Gamma^{\mathcal{P}_+}$ to be monadic, we need the following distributivity property on Γ using a *distributivity law*[1] $d : \mathtt{T}\mathcal{P}_+ \to \mathcal{P}_+\mathtt{T}$:

[1] A distributivity law satisfies some equations with respect to the monad structure, as specified in [3].

$\mathcal{P}_+$**-Distributivity:** If $t\,\Gamma(\overline{\mathcal{P}_+}(\mathcal{R}))\,r$ then $d_X(t)\,\overline{\mathcal{P}_+}(\Gamma(\mathcal{R}))\,d_Y(r)$.

Example 5 satisfies the distributivity property, and the proof of this necessarily uses the fact that $\mathcal{P}_+X$ does not include the empty set. The following relator is defined using runners, and is monadic as we shall see in Lemma 13.

Definition 12. *Suppose given a tree runner θ from S to (A, E) and a relator Γ on $\mathtt{T}_{A,E}$. The* global θ-relator Γ^θ *is the relator on $\mathtt{M}_S$ given by:*

$$a\,\Gamma^\theta(\mathcal{R})\,b \quad\Longleftrightarrow\quad \forall k \in K.\, \theta_X(a,k)\,\Gamma^{\mathcal{P}_+}(\mathcal{R}\times\mathcal{I}_K)\,\theta_Y(b,k)$$

Example 6. In case of $A = \varnothing$ and $V \subseteq E$, $a\,\Gamma^\theta_V(\mathcal{R})\,b$ holds if for any $k \in K$,

- If $[x,u] \in \theta_X(a,k)$, then there is a result $[y,u] \in \theta_Y(b,k)$ such that $x\,\mathcal{R}\,y$.
- If $\diamondsuit e \in \theta_X(a,k)$ is a detectable exception $e \in V$, then $\diamondsuit e \in \theta_Y(b,k)$.

5.2 State Initialisations and Distinctions

The global relator is quite fine-grained, testing for any possible initial state, and distinguishing between any two different final states. We can improve on this, formulating what constitutes as a good starting set of states, and a good relation on final states, such that we still get a relator. We do this in two steps: firstly we need to establish what features of the state we deem observable. This entails formulating a base relation on state which is preserved by runs over a program. Secondly, we need to establish which states can occur in a particular situation. In this subsection, we consider fixed a tree runner θ from S to (A, E) with state space K and a monadic relator Γ on $\mathtt{T}_{A,E}$.

Distinguishing Final States. We study *preorders* on the state space K (reflexive and transitive relations). Given a preorder $\mathcal{R} \subseteq K\times K$, we define $\theta(\mathcal{R}) \subseteq K\times K$:

$$a\,\theta(\mathcal{R})\,b \quad\Longleftrightarrow\quad \forall p \in \mathtt{T}X.\, \theta_X(p,a)\,\Gamma^{\mathcal{P}_+}(\iota_X\times\mathcal{R})\,\theta_X(p,b)$$

The result $\theta(\mathcal{R})$ is also a preorder, and we say $\mathcal{R}$ is *preserved by runs* if $\mathcal{R} \subseteq \theta(\mathcal{R})$. In general, $\theta(\mathcal{R})$ is included in $\mathcal{R}$, as can be observed by taking $p = \mathit{leaf}(x)$ in the definition. If $\mathcal{R} \subseteq \mathcal{U}$, then $\theta(\mathcal{R}) \subseteq \theta(\mathcal{U})$.

Definition 13. *The θ-*closure *of a preorder $\mathcal{R} \subseteq K \times K$ is the preorder $\mathcal{R}^\theta := \bigcup\{\mathcal{S} \subseteq \mathcal{R} \mid \mathcal{S} \subseteq \theta(\mathcal{S})\}$. This is the largest subrelation of $\mathcal{R}$ preserved by runs.*

The canonical choice would be to take $(K \times K)^\theta$, the θ-closure of the maximal relation on K. More generally though, one would start with $K \times K$, remove any pairs you would like to distinguish making sure the result stays a preorder, and take the θ-closure of whatever you have left. The identity relation is the smallest preorder preserved by runs, and is the one used in Definition 12.

In some cases, the θ-closure can be more easily constructed.

- If θ gives at most one result for each input (it is thin), then $\mathcal{R}^\theta = \theta(\mathcal{R})$.
- If θ gives a finite number of results for each input, then $\mathcal{R}^\theta = \bigcap_{n\in\mathbb{N}} \theta^n(\mathcal{R})$.

For establishing stateful relations, we fix one preorder $\prec$ preserved by runs. If we have no extra requirements, we take $(K \times K)^\theta$, the maximal option.

Limiting Initial States. It is a strong requirement to demand programs to be related for any possible initial state, especially when considering concurrent programs. For instance, the behavioural equivalence for global store programs will become as fine-grained as input/output programs if one can test it concurrently with any possible other global store program (this second program can simulate tests with its updates and lookups). Hence, for the verification of concurrent programs, it is important to limit the possible states that can occur.

We introduce the notion of *state world*. Given a trace $t \in \mathtt{T}_{A,E}X$ and an element $x \in X$, we write $t \trianglerighteq x$ if x is a leaf of t. In other words, $[x] \trianglerighteq x$, $\diamondsuit e \not\trianglerighteq x$ and $@t \trianglerighteq x$ if and only if $t \trianglerighteq x$. Let $\rightsquigarrow \subseteq K \times K$ be the relation such that $v \rightsquigarrow w$ when $\exists p \in \mathtt{M}X, x \in X. \theta_X(p, v) \trianglerighteq (x, w)$. This is a preorder due to the unit and multiplication properties of the runner. A state world is a subset $W \subseteq K$ such that $v \in W \wedge v \rightsquigarrow w \implies w \in W$. In other words, it is a set closed under runs of programs. For each $s \in K$, we define the world $[s] = \{z \in K \mid s \rightsquigarrow z\}$.

Example 7. Considering Example 4, with as global state $\mathbb{N}$ the number of fees that can be paid, which determines how many possible results we may get. In this case, the θ-closure of $\mathbb{N} \times \mathbb{N}$ is the standard ordering $\leq$ on $\mathbb{N}$, since a higher number gives more possible results. The relation $\rightsquigarrow$ is the relation $\geq$ since the number can only go down, hence state worlds are down-closed subsets of $\mathbb{N}$.

We fix a tree runner θ and a preorder $\prec$ preserved by runs.

Definition 14. *We define the following relators on $\mathtt{M}_S$:*

- *for each $s \in K$ a relator Γ^s where: $a \,\Gamma^s(\mathcal{R})\, b$ if $\theta(a, s)\, \Gamma^{\mathcal{P}_+}(\mathcal{R} \times \prec)\, \theta(b, s)$.*
- *for each subset $W \subseteq K$, a relator Γ^W where: $a \,\Gamma^W(\mathcal{R})\, b$ if $\forall s \in W$, $a \,\Gamma^s(\mathcal{R})\, b$.*

Lemma 13. *Given a state world $W \subseteq K$ and a state $s \in W$:*

- *$x \,\mathcal{R}\, y$ implies $\eta(x)\, \Gamma^W(\mathcal{R})\, \eta(y)$, which implies $\eta(x)\, \Gamma^s(\mathcal{R})\, \eta(y)$.*
- *$d\, \Gamma^s(\Gamma^W(\mathcal{R}))\, e \Rightarrow \mu(d)\, \Gamma^s(\mathcal{R})\, \mu(e)$, $d\, \Gamma^W(\Gamma^W(\mathcal{R}))\, e \Rightarrow \mu(d)\, \Gamma^W(\mathcal{R})\, \mu(e)$.*

Proof. Given $d\, \Gamma^s(\Gamma^W(\mathcal{R}))\, e$, we know that for any $t \in \theta(d, s)$ there is an $r \in \theta(e, s)$ such that $t\, \Gamma(\Gamma^W(\mathcal{R}) \times \prec)\, r$. Consider the sub-relation $\prec_W := (\prec) \cap (W \times W)$ of $\prec$, then $t\, \Gamma(\Gamma^W(\mathcal{R}) \times \prec_W)\, r$ since any state in a leaf of t and r are in W.

We prove: If $(a, u)\, \Gamma^W(\mathcal{R}) \times \prec_W\ (b, v)$, then $\theta(a, u)\, \Gamma^{\mathcal{P}_+}(\mathcal{R} \times \prec)\, \theta(b, v)$. Suppose $(a, u)\, \Gamma^W(\mathcal{R}) \times \prec_W\ (b, v)$, then $a \,\Gamma^W(\mathcal{R})\, b$, and $u \prec_W v$, hence 1) $u \prec v$ and 2) $v \in W$. By the former, $\theta(a, u)\, \Gamma^{\mathcal{P}_+}(\mathcal{I}_X \times \prec)\, \theta(a, v)$, since $\prec$ is preserved by runs. By the latter, $\theta(a, v)\, \Gamma^{\mathcal{P}_+}(\mathcal{R} \times \prec)\, \theta(b, v)$, since $a \,\Gamma^W(\mathcal{R})\, b$. So by the relator properties on $\Gamma^{\mathcal{P}_+}$, and transitivity on $\prec$, we get that $\theta(a, u)\, \Gamma^{\mathcal{P}_+}(\mathcal{R} \times \prec)\, \theta(b, v)$.

By naturality, $\mathcal{P}_+\mathtt{T}(\theta)(\theta(d, s))\, \Gamma^{\mathcal{P}_+}(\Gamma^{\mathcal{P}_+}(\mathcal{R} \times \prec))\, \mathcal{P}_+\mathtt{T}(\theta)(\theta(e, s))$, hence due to the multiplication property on $\overline{\mathcal{P}_+}\Gamma = \Gamma^{\mathcal{P}_+}$ and the multiplication property for runners, $\theta(\mu d, s)\, \overline{\mathcal{P}_+}(\Gamma(\mathcal{R} \times \prec))\, \theta(\mu e, s)$. Hence $\mu d\, \Gamma^s(\mathcal{R})\, \mu e$.

We conclude that for any state world $W \subseteq S$, Γ^W is monadic.

5.3 Algebraic Concurrent Theories

Taking a set of variables X, we can define for each state world $W \subseteq S$ the *stateful approximation* $\leq_W \subseteq \mathtt{M}X \times \mathtt{M}X$ as $\Gamma^W(\iota_X)$, and *stateful equivalence* $\equiv_W \subseteq \mathtt{M}X \times \mathtt{M}X$ as $a \equiv_W b \iff a \leq_W b \wedge b \leq_W a$. Both these relations are preorders. The fact that the relator is monadic consequently means these relations are *substitutive*:

Corollary 2. *Given a state world $W \subseteq K$, $a \leq_W b$ and $f, g : X \to \mathtt{T}X$ s.t. $\forall x \in X.\, f(x) \leq_W g(x)$, then $\mu^S_X(\mathtt{M}(f)(a)) \leq_W \mu^S_X(\mathtt{M}(g)(b))$. Similarly for $\equiv_W$.*

Hence, the relators give rise to proper algebraic theories. The above results hold for any tree runner, including those implementing concurrency. Hence the relators give a notion of equivalence for effectful programs within concurrent environments containing other programs. We look at some concrete examples of such situations, and how this concurrency affects the stateful equivalences.

Example 8 (Printing with interference). Consider Example 1 of programs printing messages to an environment. Suppose some background process is printing messages at unpredictable times. We describe this with the runner ϕ, defining it to be the **1**-concurrent completion $\theta^{|\mathbf{1}}$ of the runner θ from Example 1. This has as state space $\mathtt{T1} \times M^* \simeq M^* \times M^*$, containing the background process together with a history of printed messages. We want to distinguish between different histories, but not necessarily between different states of the background process. Hence we take as relation on state $\prec := ((\mathtt{T1})^2 \times \iota_{M^*})^\phi$, the largest relation preserved by runs which distinguishes different histories of printed messages.

Running a program in such an environment will intersperse the messages of the program with messages from the background process. Suppose the background process keeps printing $a \in M$. We can model this with an inductively defined set $U \subseteq \mathtt{T1}$, such that $[*] \in U$ and $t \in U$ implies $@(t) \in U$ (approximations of an infinite sequence of prints). Note that $U \times M^*$ is a state world, since it contains its own continuations. The notion of equivalence for running programs with this background process can be specified using the relator $\Gamma^{U \times M^*}$.

The algebraic theory resulting from this environment reflects the fact that, since $\mathtt{output}_a$ could be scheduled after any output action, any $\mathtt{output}_a$ produced by the program after the initial output can be ignored. So we have the equation: $\mathtt{output}_b(\mathtt{output}_a(\mathit{leaf}(x))) \equiv_{U \times M^*} \mathtt{output}_b(\mathit{leaf}(x))$.

Example 9 (Global Store). Consider Example 3 of the global store effect, where for simplicity we take as memory space $\{0, 1\}$, hence the `lookup` operation has arity $\{0, 1\}$, and we have two update operations. Let $t \in \mathtt{M}_S X$ be the tree $\mathtt{update}_0(\mathtt{lookup}(0 \mapsto \bot, 1 \mapsto \mathit{leaf}(x)))$, which gives the set of branches $\beta(t) = \{(0!)(0?)\langle\bot\rangle, (0!)(1?)[x]\} \subseteq \mathtt{T}X$. If t is run on its own with any initial state $i \in \{0, 1\}$, it must eventually yield the error message $\bot$, since 0 is saved to and subsequently read from the global memory. Hence, under standard global store semantics (implemented with the runner from Example 3), it holds that $t \equiv \bot$.

However, suppose t is run in an environment with a backup process which updates the global store to 1, e.g. $r = (1!)(r') \in \mathtt{T}Y$. Then $(0!)(1!)(1?)[x, r'] \in \rho(t, r)$, which under global store semantics will yield, for any initial state $i \in \{0, 1\}$, a result (x, r') with final state 1. Hence, in this environment, $t \not\equiv \bot$.

6 Conclusions

We have formulated an operation for interleaving concurrency satisfying the unit and multiplication equations for residual monad-comonad interaction laws in the sense of [13]. This operation can be used in at least two ways. Firstly, it can evaluate two programs concurrently, producing a process which returns a pair of return values. Secondly, we can run one program concurrently with another program, where this other program need only be evaluated partially. This forms a runner [29], which produces a return value of the former program and remainder of the latter program. Both together allow us to precisely schedule concurrent interleavings of higher-order programs.

The sequences of communications between a program and its environment are given by traces, which can be given a behavioural interpretation using stateful runners; a categorical formulation of how to handle and resolve effect operations in a state dependent manner. This can be implemented in practise using *Handlers* for algebraic effects [24,2]. One line of work for the future, is formulating a congruent notion of applicative bisimilarity for concrete higher-order concurrent languages, for example extending results from [15,26] for lambda-calculus style languages, or from [18,19] for a continuation-passing style languages.

It is difficult to generalise the interleaving operation to the monad of trees due to the lack of naturality of the strength operation for trees in any wide subcategory of **Rel** containing multi-valued (non-thin) relations. This is a problem, since the interleaving concurrency operation itself is multi-valued. By using the category of total relations $\mathbf{Rel}_+$, we are still able to include nullary operations in our monad of traces. This is important, since many situations necessitate the use of exceptions; some effect examples need to exclude impossible sequences of events like reading the wrong state, and in case of infinite recursive processes we may have to mark the end of an approximation.

Models for alternative forms of concurrency could potentially be implemented by adapting the formalism of this paper. Firstly, we might want to run certain sequences of operations without being interrupted by other programs, for instance by only giving one program access to the environment at a time. This could be modelled by using lists of actions as our atomic actions. Secondly, we could look at models closer to *true concurrency* [11]. For instance, some operations may be executed at the exact same time, producing traces over some monoid of actions. Such a denotation would need to be endowed with appropriate notions of behaviour for simultaneously executed operations, which could be implemented by a runner. Lastly, concurrency itself could be described with algebraic effects, following [10], which may be combined with the approach of this paper.

Acknowledgement: This version of the contribution has been accepted for publication, after peer review (when applicable) but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: https://doi.org/10.1007/978-3-031-17715-6_26. Use of this Accepted Version is subject to the publisher's Accepted Manuscript

References

1. Abramsky, S.: The Lazy Lambda Calculus, p. 65–116. Addison-Wesley Longman Publishing Co., Inc., USA (1990)
2. Ahman, D., Bauer, A.: Runners in action. In: Müller, P. (ed.) Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Dublin, Ireland, April 25-30. Lecture Notes in Computer Science, vol. 12075, pp. 29–55. Springer (2020). https://doi.org/10.1007/978-3-030-44914-8_2
3. Beck, J.: Distributive laws. In: Eckmann, B. (ed.) Seminar on Triples and Categorical Homology Theory. pp. 119–140. Springer Berlin Heidelberg, Berlin, Heidelberg (1969). https://doi.org/10.1007/BFb0083084
4. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theoretical Computer Science **37**, 77–121 (1985). https://doi.org/10.1016/0304-3975(85)90088-X
5. Bergstra, J.A., Klop, J.W., Tucker, J.V.: Process algebra with asynchronous communication mechanisms. In: Brookes, S.D., Roscoe, A.W., Winskel, G. (eds.) Seminar on Concurrency. pp. 76–95. Springer Berlin Heidelberg, Berlin, Heidelberg (1985). https://doi.org/10.1007/3-540-15670-4_4
6. Busi, N., Gorrieri, R.: A Survey on Non-interference with Petri Nets, pp. 328–344. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27755-2_8
7. Clinger, W.D.: Foundations of actor semantics. Tech. rep., Massachusetts Institute of Technology, USA (1981), http://hdl.handle.net/1721.1/6935
8. Fiore, M., Gambino, N., Hyland, M., Winskel, G.: The cartesian closed bicategory of generalised species of structures. Journal of the London Mathematical Society **77**, 203–220 (2008). https://doi.org/10.1112/jlms/jdm096
9. Fiore, M., Gambino, N., Hyland, M., Winskel, G.: Relative pseudomonads, Kleisli bicategories, and substitution monoidal structures. Selecta Mathematica **24**(3), 2791–2830 (Nov 2017). https://doi.org/10.1007/s00029-017-0361-3
10. Glabbeek, R.v., Plotkin, G.: On CSP and the Algebraic Theory of Effects, pp. 333–369. Springer London, London (2010). https://doi.org/10.1007/978-1-84882-912-1_15
11. Gorrieri, R.: Interleaving vs true concurrency: Some instructive security examples. In: Janicki, R., Sidorova, N., Chatain, T. (eds.) Application and Theory of Petri Nets and Concurrency. pp. 131–152. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-51831-8_7
12. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace semantics via coinduction. Logical Methods in Computer Science **3** (11 2007). https://doi.org/10.2168/LMCS-3(4:11)2007
13. Katsumata, S.y., Rivas, E., Uustalu, T.: Interaction laws of monads and comonads. In: Proc. of the 35th Ann. ACM/IEEE Symp. on Logic in Computer Science. p. 604–618. LICS '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3373718.3394808
14. Keller, R.M.: Formal verification of parallel programs. Commun. ACM **19**(7), 371–384 (1976). https://doi.org/10.1145/360248.360251

15. Lago, U.D., Gavazzo, F., Levy, P.B.: Effectful applicative bisimilarity: Monads, relators, and Howe's method. In: Proc. of 32nd Ann. ACM/IEEE Symp. on Logic in Computer Science, LICS'17. pp. 1–12. IEEE Computer Society (2017). https://doi.org/10.1109/LICS.2017.8005117
16. Levy, P.B.: Similarity quotients as final coalgebras. In: Hofmann, M. (ed.) Proc. of 14th Int. Conf. on Foundations of Software Science and Computational Structures, FoSSaCS'11. Lecture Notes in Computer Science, vol. 6604, pp. 27–41. Springer (2011). https://doi.org/10.1007/978-3-642-19805-2_3
17. MacLane, S.: Categories for the Working Mathematician. Springer-Verlag, New York (1971). https://doi.org/10.1007/978-1-4612-9839-7, graduate Texts in Mathematics, Vol. 5
18. Matache, C.: Program Equivalence for Algebraic Effects via Modalities. Master's thesis, University of Oxford (2019)
19. Matache, C., Staton, S.: A sound and complete logic for algebraic effects. In: Proc. of 22nd Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS'19. Lecture Notes in Computer Science, vol. 11425, pp. 382–399. Springer (2019). https://doi.org/10.1007/978-3-030-17127-8_22
20. Melliès, P.A.: Template games and differential linear logic. In: 2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS). pp. 1–13 (2019). https://doi.org/10.1109/LICS.2019.8785830
21. Milner, R.: Communication and Concurrency. Prentice-Hall, Inc., USA (1989)
22. Moggi, E.: Notions of computation and monads. Inf. Comput. **93**(1), 55–92 (Jul 1991). https://doi.org/10.1016/0890-5401(91)90052-4
23. Plotkin, G.D., Power, J.: Adequacy for algebraic effects. In: Honsell, F., Miculan, M. (eds.) Proc. of 4th Int. Conf. on Foundations of Software Science and Computation Structures, FoSSaCS'01. Lecture Notes in Computer Science, vol. 2030, pp. 1–24 (2001). https://doi.org/10.1007/3-540-45315-6_1
24. Plotkin, G.D., Pretnar, M.: Handling algebraic effects. Log. Methods Comput. Sci. **9**(4, article 23), 1–36 (2013). https://doi.org/10.2168/lmcs-9(4:23)2013
25. Rivas, E., Jaskelioff, M.: Monads with merging (Jun 2019), https://hal.inria.fr/hal-02150199, working paper or preprint
26. Simpson, A., Voorneveld, N.: Behavioural equivalence via modalities for algebraic effects. ACM Transactions on Programming Languages and Systems **42**(1), 4:1–4:45 (2020). https://doi.org/10.1145/3363518
27. Thijs, A.M.: Simulation and fixpoint semantics. Ph.D. thesis, University of Groningen (1996), https://research.rug.nl/en/publications/simulation-and-fixpoint-semantics
28. Turi, D., Plotkin, G.: Towards a mathematical operational semantics. In: Proceedings of Twelfth Annual IEEE Symposium on Logic in Computer Science. pp. 280–291 (1997). https://doi.org/10.1109/LICS.1997.614955
29. Uustalu, T.: Stateful runners of effectful computations. Electronic Notes in Theoretical Computer Science **319**, 403–421 (2015). https://doi.org/10.1016/j.entcs.2015.12.024, the 31st Conference on the Mathematical Foundations of Programming Semantics
30. Uustalu, T., Vene, V.: Comonadic notions of computation. Electronic Notes in Theoretical Computer Science **203**(5), 263–284 (2008). https://doi.org/10.1016/j.entcs.2008.05.029, proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science (CMCS 2008)