Math. Struct. in Comp. Science (2014), vol. 24, iss. 2, e240201, 49 pages. © Cambridge University Press 2013
doi:10.1017/S0960129513000182

Computability structures, simulations and realizability

JOHN LONGLEY

Laboratory for Foundations of Computer Science, School of Informatics,
University of Edinburgh, 10 Crichton Street,
Edinburgh, EH8 9AB, United Kingdom
Email: jrl@staffmail.ed.ac.uk

Received 12 May 2011; revised 10 January 2013

We generalise the standard construction of realizability models (specifically, of categories of assemblies) to a wide class of *computability structures*, which is broad enough to embrace models of computation such as labelled transition systems and process algebras. We consider a general notion of *simulation* between such computability structures, and show how these simulations correspond precisely to certain functors between the realizability models. Furthermore, we show that our class of computability structures has good closure properties – in particular, it is 'cartesian closed' in a slightly relaxed sense. Finally, we investigate some important subclasses of computability structures and of simulations between them. We suggest that our 2-category of computability structures and simulations may offer a useful framework for investigating questions of computational power, abstraction and simulability for a wide range of models.

1. Introduction

The purpose of this paper is to present a rather broad generalisation of some ideas and results from the theory of realizability models: in particular, the construction of a category of *assemblies* over a given model, and a notion of *simulation* of one model in another. Our theory is general enough to be applicable to a wide range of models of computation from many areas of computer science, and we suggest that this may provide a useful framework for studying very general questions of computational power, abstraction and simulability.

1.1. *Background*

We will begin with a brief review of the ideas that we wish to generalise. In the standard account of realizability models (as presented, for example, in van Oosten (2008)), we begin with a *partial combinatory algebra* (or PCA) A – a structure that may be loosely regarded as some kind of 'abstract machine' or 'untyped model of computation'. From this, we build a category, such as the *category of assemblies* $\mathbf{Asm}(A)$, which can be thought of as a 'universe of computable functions' arising from A: its objects can be regarded as 'datatypes' that can be represented or implemented on A, and its morphisms can be viewed as 'A-computable maps' between such datatypes. (We shall review the precise definition

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

of **Asm**(A) in Section 2 below.) Categories of this form possess a very rich mathematical structure and have proved interesting from several points of view – for instance:

— as categorical models corresponding to Kleene-style realizability interpretations of intuitionistic logic (Hyland 1982);
— as models for powerful polymorphic type systems (Longo and Moggi 1991); or
— as universes within which certain 'sets' carry an intrinsic domain-like structure (Hyland 1990; Taylor 1991).

Longley (1995) introduced a notion of morphism between PCAs that interacts well with the **Asm** construction. Informally, an *applicative morphism* $A \longrightarrow\!\!\!\!\triangleright\, B$ is a way of simulating or implementing the abstract machine A on the machine B, and in many cases it makes sense to think of A here as modelling computation at a 'more abstract' level than B. Such a simulation induces a functor **Asm**(A) $\rightarrow$ **Asm**(B). Moreover, we can give a precise categorical characterisation of those functors that arise in this way up to natural isomorphism. In this way, we obtain a tight correspondence between the (2-)category of PCAs and applicative morphisms (which we call $\mathcal{PCA}$) and a certain (2-)category of categories (again, this material will be reviewed in Section 2). From now on, we shall refer to this result as the *correspondence theorem.*

Our 2-category $\mathcal{PCA}$ gives rise to a notion of *equivalence* between PCAs. Informally, two PCAs A, B are equivalent if there are simulations $A \longrightarrow\!\!\!\!\triangleright\, B$ and $B \longrightarrow\!\!\!\!\triangleright\, A$ that are 'mutually inverse up to computable translation'. This notion highlights some interesting differences in 'computational strength' between well-known models of computation: for example, the PCA consisting of the natural numbers with Kleene application turns out to be *inequivalent* to the PCA of untyped λ-terms modulo β-equality, even though these models have the same computational power as measured by the functions $\mathbb{N} \rightarrow \mathbb{N}$ that they can compute[†].

A consequence of the above theory is that A and B are equivalent if and only if **Asm**(A) and **Asm**(B) are equivalent as categories. This suggests that we may in some sense view equivalent PCAs as just alternative presentations of the same underlying 'notion of computability', and that the category of assemblies gives us a kind of presentation-invariant embodiment of what it is that they have in common (however, Section 2.1 provides an important caveat on this idea). Indeed, if our main interest is in the PCAs themselves and their interrelationships, the correspondence theorem can be viewed as offering some sort of vindication of the definition of the 2-category $\mathcal{PCA}$ (and of the resulting notion of PCA equivalence).

All this is a pleasing enough story as far as it goes, and the correspondence theorem we have mentioned is mathematically satisfying. However, this story might be felt to be of limited interest on at least two counts:

— The vast majority of 'models of computation' currently studied in computer science do not naturally take the form of PCAs. However, the general concepts of 'computational

[†] This example is explained in detail in Longley (1995, Chapter 3). Briefly, each of these models may be simulated in the other, but the problem is that the simulations cannot be 'essentially mutually inverse' in the required sense because the λ-calculus is in some way 'more abstract' than the Kleene model.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

power', 'simulation' and 'levels of abstraction' would intuitively seem to be of much more general significance, so it is natural to wonder whether our mathematical framework for discussing these concepts could be broadened to encompass other kinds of models of computation, such as labelled transition systems or process calculi.

— The 2-category $\mathcal{PCA}$ turns out to be relatively poor in its 2-categorical structure. In the interests of coming up with a mathematically fruitful theory, it is natural to ask whether we can do better.

As regards the first of these points, some progress was made in Longley (1999a), Longley (1999b) and Lietz and Streicher (2002), where it was shown that the basic theory can be straightforwardly generalised to a setting of *typed PCAs*, of which ordinary (untyped) PCAs form a special case. This greatly extends the scope of the theory: for instance, the typed PCA framework embraces all the cartesian closed categories used in denotational semantics, as well as a wealth of syntactic models for languages such as PCF and extensions thereof. This version of the theory was presented more fully in Longley (2007), where the typed PCA framework played a crucial role in an investigation of higher-order computability notions. A further generalisation to 'linear' (in contrast to 'intuitionistic') structures was considered in Hoshino (2007).

Even so, the typed PCA framework comes nowhere near to embracing the enormous variety of 'models of computation' currently studied in computer science as a whole. Moreover, from the point of view of its categorical structure, the 2-category of typed PCAs is scarcely better than our original 2-category of PCAs. We may therefore still ask whether it is possible to do better on both these scores.

1.2. *Content of the paper*

Our purpose in this paper is to offer a significantly more general framework for the study of models of computation and the 'notions of computability' they embody. Specifically, we shall define a rather wide[†] 2-category $\mathcal{CSTRUCT}$ of *computability structures* and *simulations* between them in such a way that:

— a version of the **Asm** construction still makes sense, and an appropriate generalisation of the correspondence theorem goes through;
— a far wider range of models from across computer science is admitted – for example, labelled transition systems and process calculi are included;
— the class of computability structures has better closure properties than that of (untyped or typed) PCAs.

Our work is motivated by both mathematical and computer science considerations. From a purely mathematical point of view, it can be seen as a deeper investigation into the correspondence theorem first established in the PCA setting: in particular, we ask just how far does this phenomenon extend, and what mathematical machinery is needed to make it work? Moreover, the identification of a class of models with richer closure properties is of mathematical interest in its own right, and perhaps an indication that we are ploughing

[†] It is much broader than the 2-category of typed PCAs.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

fertile ground. From a computer science point of view, our hope is to offer a general framework for a study of 'computability' and 'simulations' that does justice to the diversity of computation models currently in use. Much of the richness of computer science, after all, comes from the use of a wide variety of modelling styles to study computational systems at many different levels of description or abstraction, so a general framework for talking about these modelling styles and their interrelationships holds out the hope of bringing some degree of unification to hitherto disparate parts of the subject.

The key idea behind our generalisation is a switch from a 'higher-order' to a 'first-order' style of modelling. Typed PCAs naturally model higher-order flavours of computation, in the sense that a 'computable operation' from a type σ to a type τ is a value of some type $(\sigma \to \tau)$, which may itself serve as input to other computable operations. In the first-order style of modelling, data values are rigidly separated from the computable operations that act on them, and only later do we show how the higher-order situation may be conveniently recovered as a special case. Indeed, one of the main goals of this paper is to show how a surprisingly large amount of the basic machinery of realizability models can be smoothly adapted to this 'flattened' first-order setting.

Two major omissions should be noted at the outset:

(1) Our focus in this paper is mostly on setting up the general framework, and there will be relatively little investigation into particular models. Although in Section 3 we shall briefly indicate some of the main sources of examples we have in mind, and present one non-trivial example of a simulation in detail, it would require a far deeper study of specific models from diverse areas of computer science to substantiate our claims regarding the unifying potential of our framework.

(2) Whilst we shall pay some attention to the *categorical* structure of $\mathbf{Asm}(\mathsf{C})$ for various kinds of model C, we shall barely touch on the kinds of *logical* constructs that such categories allow us to interpret. Insofar as the interpretation of logic was a key motivation for the study of realizability in the first place, this might appear as a serious deficiency. On the other hand, anyone experienced in categorical logic will have little difficulty in seeing broadly what kinds of logical structure our categories support.

1.3. *Structure of the paper*

In Section 2, we give a brief technical summary of the relevant parts of the existing theory of untyped and typed PCAs. In Section 3, we define our main category of interest: the 2-category $\mathcal{CSTRUCT}$ of *computability structures* and *simulations*. As a non-trivial example, we show how Milner's well-known translation from the lazy λ-calculus to the π-calculus (Milner 1992) can be viewed as a simulation between the relevant computability structures. We also show how any computability structure C gives rise to a 'category of assemblies' $\mathbf{Asm}(\mathsf{C})$, and establish a tight correspondence between simulations $\mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$ and certain functors $\mathbf{Asm}(\mathsf{C}) \to \mathbf{Asm}(\mathsf{D})$, thereby generalising the existing correspondence theorem for PCAs. In Section 4, we investigate the closure properties of $\mathcal{CSTRUCT}$, showing, in particular, that it is 'almost cartesian closed', in a sense that is good enough to ensure that for any structures C and D, the structure D^{C} is uniquely determined up to

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

equivalence. In Section 5, we mention some possible variations on our theory and some interesting subcategories of $\mathcal{CSTRUCT}$: in particular, we show how the theory of typed PCAs fits into this more general framework. We conclude in Section 6, with a general discussion and some suggestions for further work.

1.4. *Related work*

Our work overlaps significantly with recent and essentially independent work by Cockett and Hofstra (Cockett and Hofstra 2008; Cockett and Hofstra 2010). In both their work and ours, the key idea is to generalise the definition of PCA to something with a typed, first-order flavour, and to show how a general notion of 'simulation' may be defined in this setting. Another point in common is the idea that models of computation give rise to categories of certain kinds, in such a way that simulation equivalence corresponds precisely to some well-behaved kind of categorical equivalence. In both cases, then, the basic philosophy is that the category serves as a kind of presentation-invariant embodiment of the underlying 'notion of computability'.

Various differences between Cockett and Hofstra (2010) and our work should also be noted:

— There is a general difference in spirit: whereas Cockett and Hofstra adopt a thoroughgoing 'categorical' approach that provides a detailed and precise axiomatisation of the mathematical structure that the theory presupposes, we have opted for a relatively simple-minded 'set theoretic' treatment of models of computation as they concretely appear to the working computer scientist.
— At a technical level, there are various differences in the precise level of generality in which the phenomena in question are studied, even after the categorical framework of Cockett and Hofstra (2010) has been specialised to a familiar set-theoretic setting. These differences will be discussed as we proceed (see Sections 2.1 and 3.1). Nevertheless, there is a very significant area of overlap in which we are talking about essentially the same things – in particular, it is striking that our notion of 'equivalence of models' turns out to coincide exactly (for models within the overlap) with the *lax simulation equivalence* of Cockett and Hofstra (2010).
— When it comes to the categories derived from the computation models, the focus of the two approaches is somewhat different. Whereas our aim is to describe concretely a construction that directly generalises that of the 'classical' category of assemblies, the aim in Cockett and Hofstra (2008) and Cockett and Hofstra (2010) is to give a categorical analysis of the construction in terms of several smaller steps. Indeed, most of the focus in Cockett and Hofstra (2008) and Cockett and Hofstra (2010) is not on the classical category of assemblies itself, but on an intermediate structure known as a *Turing category*. One of the main results of Cockett and Hofstra (2010) is a correspondence theorem for Turing categories, which is closely analogous to ours for categories of assemblies. We shall resume discussion of this in Section 2.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Aside from these differences, the following new technical contributions are specific to the current work:

(1) We identify a precise categorical characterisation of those functors between assembly categories that arise from simulations between computability structures – this enables us to state and prove our version of the correspondence theorem. The key notion here is that of a *quasi-regular* functor, which is a generalisation of the notion of a regular functor to a setting where finite products are not assumed.
(2) Our identification of the 'almost cartesian closed' structure in our 2-category of models is novel.
(3) There is an explicit emphasis in our work on the generalisation to *non-deterministic* models of computation, such as those arising from concurrency theory, and in some respects our definitions are tuned so that they admit a large class of examples from process algebra and related areas (see Example 3.3(iii) and the subsequent discussion).

Also similar in spirit to our work is the general framework for realizability developed by Hofstra in Hofstra (2006). He starts with a broad 2-category of *basic combinatorial objects*, whose definition resembles that of our computability structures in some respects. On one axis, Hofstra's framework is more restricted than ours since his main interest is in what is needed to build a realizability *topos* (via a tripos), so he restricts attention to untyped structures rather than general typed ones. Along another axis, however, his framework is broader in that it embraces the notion of *ordered PCAs*, which are a generalisation of the ordinary PCAs introduced in van Oosten (1997). As shown in Hofstra (2006), this extra generality allows the unification 'computational' examples with geometric or 'localic' ones. We would expect that most if not all of our theory extends readily to the ordered setting. However, we have chosen to eschew this additional generality for the purposes of the current paper, both because we are not aware of any *computationally* motivated examples that require it, and because the extra generality of the ordered setting is purchased at the cost of a modest technical overhead: our interest here is not so much in maximising generality as in minimising the machinery required for our theory to work.

Finally, Abramsky's work on 'process realizability' (Abramsky 2000) should be mentioned as another attempt to broaden the scope of realizability to embrace concurrent and non-deterministic flavours of computation, in addition to the usual functional or applicative kinds. Abramsky (2000) developed the ideas in the setting of (a mild extension of) Milner's CCS. In place of a single 'application' operation as in the theory of PCAs, he considers both a 'left' and a 'right' application, which are dual to each other and allow the same process term to act as a 'function' in two entirely symmetrical ways. When transposed to our setting here, an effect of the flattening from 'higher order' to 'first order' is that both these kinds of application are subsumed as instances of the more general concept of a computable operation. However, the realizability construction Abramsky considers is interestingly different from the standard one, and exploits the duality inherent in his setup to give a model of Classical Linear Logic. We have not yet investigated whether this idea can be fruitfully applied within our present first-order framework.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

2. Untyped and typed PCAs

In this section we give a brief review of the existing definitions and results for untyped and typed PCAs – for fuller details, see Longley (1995) or van Oosten (2008). We shall recover this material in Section 5 as a special case of our more general theory.

If e, e' are mathematical expressions, we write:

— $e \downarrow$ to mean 'the value of e is defined';
— $e \simeq e'$ to mean 'if either e or e' is defined, then so is the other, and their values are equal';
— $e \succeq e'$ to mean 'if e' is defined, then so is e, and their values are equal'.

2.1. *Untyped PCAs*

The following definition (with minor differences) appeared in Feferman (1975); it generalises the notion of a total combinatory algebra introduced in Schönfinkel (1924).

Definition 2.1. An *(untyped) partial combinatory algebra*, or *PCA*, is a set A equipped with a (left-associative) partial binary operation $\cdot : A \times A \rightharpoonup A$ and containing elements k, s such that

$$k \cdot x \cdot y = x$$
$$s \cdot x \cdot y \downarrow$$
$$s \cdot x \cdot y \cdot z \succeq x \cdot z \cdot (y \cdot z).$$

Here, and throughout the rest of the paper, we adopt the convention that variables in displayed formulae that are not explicitly introduced elsewhere (in this case, x, y, z) are regarded as implicitly universally quantified at the start of the displayed formula.

Variations on the above definition are possible. For instance, the definition is often given with '$\simeq$' in place of '$\succeq$' (we may call a PCA *strict* if this stronger condition is satisfied), but the theory we shall present works naturally with our slightly weaker definition[†]. Furthermore, the condition $s \cdot x \cdot y \downarrow$ is unnecessary from a certain point of view (see van Oosten (2008, Theorem 1.2.3)), although it renders the development of our theory somewhat smoother.

The following definition first appeared in Longley (1995).

Definition 2.2. Let A, B be PCAs. Then:

(i) An *applicative morphism* $\gamma : A \longrightarrow\!\!\!\!\triangleright\, B$ is a total relation γ from A to B such that for some $r \in B$, we have

$$\gamma(a, b) \wedge \gamma(a', b') \wedge a \cdot a' \downarrow \quad \Rightarrow \quad \gamma(a \cdot a',\ r \cdot b \cdot b').$$

(ii) Given $\gamma, \delta : A \longrightarrow\!\!\!\!\triangleright\, B$, we write $\gamma \preceq \delta$, and say there is an *applicative transformation* from γ to δ, if for some $t \in B$, we have

$$\gamma(a, b) \quad \Rightarrow \quad \delta(a,\ t \cdot b).$$

[†] van Oosten (2008) refers to our 'strict PCAs' simply as 'PCAs', and to our 'PCAs' as 'lax PCAs'.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

An applicative morphism $A \longrightarrow\!\!\!\!\triangleright\, B$ can be seen as a way of simulating A in B – a convenient slogan is 'application in A is effective in B'. The condition $\gamma \preceq \delta$ says that γ-representations for elements of A may be transformed into δ-representations (effectively within B). It is easy to check that PCAs, applicative morphisms and applicative transformations constitute a preorder-enriched category, which we call $\mathcal{PCA}$.

We will now show how to construct a realizability model over a PCA.

Definition 2.3. For any PCA A, the *category of assemblies* $\mathbf{Asm}(A)$ is defined as follows:

— Objects are pairs $(X, \Vdash)$ where X is a set and $\Vdash \subseteq A \times X$ is a relation such that for every $x \in X$ there is some $a \Vdash x$.

— Morphisms $(X, \Vdash) \to (Y, \Vdash')$ are set-theoretic functions $f : X \to Y$ such that for some $r \in A$ we have

$$a \Vdash x \quad \Rightarrow \quad r \cdot a \Vdash f(x)$$

— Identities and composition are as for ordinary functions.

The category $\mathbf{Asm}(A)$ is equipped with an evident forgetful functor $\Gamma_A : \mathbf{Asm}(A) \to \mathbf{Set}$, and a functor $\nabla_A : \mathbf{Set} \to \mathbf{Asm}(A)$ defined as follows:

$$\nabla_A(X) = (X, \Vdash) \qquad (\text{where } a \Vdash x \text{ for all } a \text{ and } x)$$
$$\nabla_A(f : X \to Y) = f.$$

It turns out that $\mathbf{Asm}(A)$ is, among other things, a *regular* category: that is, it has finite limits and coequalisers of kernel-pairs, and the pullback of a regular epi along any morphism is a regular epi. (An in-depth understanding of this definition is not needed for this paper.) Furthermore, both Γ_A and ∇_A are *regular functors* (also known as exact functors): that is to say, they preserve finite limits and coequalisers of kernel-pairs. Note also that Γ_A is left adjoint to ∇_A and that $\Gamma_A \circ \nabla_A = \mathrm{id}$.

An applicative morphism $\gamma : A \longrightarrow\!\!\!\!\triangleright\, B$ induces a functor

$$\mathbf{Asm}(\gamma) : \mathbf{Asm}(A) \to \mathbf{Asm}(B)$$

as follows:

$$\mathbf{Asm}(\gamma)(X, \Vdash) = (X, \Vdash') \qquad \text{where } b \Vdash' x \Leftrightarrow \exists a \Vdash x.\, \gamma(a, b))$$
$$\mathbf{Asm}(\gamma)(f) = f.$$

It is easy to show that $\mathbf{Asm}(\gamma)$ is a regular functor and that

$$\Gamma_B \circ \mathbf{Asm}(\gamma) \cong \Gamma_A$$
$$\mathbf{Asm}(\gamma) \circ \nabla_A \cong \nabla_B.$$

Moreover, an applicative transformation $\gamma \preceq \delta$ induces a natural transformation

$$\mathbf{Asm}(\gamma) \to \mathbf{Asm}(\delta).$$

Abstracting from this situation, we arrive at the following definition (where we write $*$ for horizontal composition).

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Definition 2.4.

(i) A $\Gamma\nabla$-*regular category* is any regular category $\mathcal{C}$ equipped with regular faithful functors $\Gamma : \mathcal{C} \to \mathbf{Set}$ and $\nabla : \mathbf{Set} \to \mathcal{C}$ such that

$$\Gamma \dashv \nabla$$
$$\Gamma \circ \nabla \cong \mathrm{id}.$$

(ii) If $\mathcal{C}, \mathcal{D}$ are $\Gamma\nabla$-regular categories, a $\Gamma\nabla$-*regular functor* $F : \mathcal{C} \to \mathcal{D}$ is a regular functor such that

$$\Gamma_{\mathcal{D}} \circ F \cong \Gamma_{\mathcal{C}}$$
$$F \circ \nabla_{\mathcal{C}} \cong \nabla_{\mathcal{D}}.$$

(These isomorphisms are unique when they exist.)

(iii) If $F, G : \mathcal{C} \to \mathcal{D}$ are $\Gamma\nabla$-regular functors, a $\Gamma\nabla$-*natural transformation* $\eta : F \to G$ is a natural transformation such that

$$\Gamma_{\mathcal{D}} * \eta = \mathrm{id}_{\Gamma_{\mathcal{C}}}$$
$$\eta * \nabla_{\mathcal{C}} = \mathrm{id}_{\nabla_{\mathcal{D}}}.$$

(Both these conditions are automatic in the case $\mathcal{C} = \mathbf{Asm}(A)$, $\mathcal{D} = \mathbf{Asm}(B)$.)

(iv) We write $\Gamma\nabla\mathcal{REG}$ for the (large) 2-category of $\Gamma\nabla$-regular categories, $\Gamma\nabla$-regular functors and natural transformations between them.

In fact, the ∇ condition in part (ii) is not strictly necessary: a simple cardinality argument (appearing as van Oosten (2008, Lemma 1.6.3)) shows that for any functor

$$F : \mathbf{Asm}(A) \to \mathbf{Asm}(B),$$

the condition

$$\Gamma_B \circ F \cong \Gamma_A$$

automatically implies

$$F \circ \nabla_A \cong \nabla_B.$$

The tight correspondence between applicative morphisms and regular functors is encapsulated in the following theorem from Longley (1995) (see also van Oosten (2008)).

Theorem 2.5 (Correspondence theorem). The **Asm** construction yields a 2-functor $\mathcal{PCA} \to \Gamma\nabla\mathcal{REG}$ that is *locally an equivalence*: that is, for any A, B, the ordinary functor

$$\mathbf{Asm} : \mathcal{PCA}[A, B] \to \Gamma\nabla\mathcal{REG}[\mathbf{Asm}(A), \mathbf{Asm}(B)]$$

is part of an equivalence of preorders.

Corollary 2.6. $\mathbf{Asm}(A)$ and $\mathbf{Asm}(B)$ are equivalent as categories if and only if A and B are *applicatively equivalent* (that is, $A \simeq B$ in the 2-category $\mathcal{PCA}$).

The corollary follows since it is easy to see that any equivalence of categories $\mathbf{Asm}(A) \simeq \mathbf{Asm}(B)$ respects the Γ and ∇ functors.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Before proceeding to typed PCAs, we will pause here to discuss a significant choice involved in our definition of applicative morphism: in the terminology of Cockett and Hofstra (2010), the choice between *lax* and *strict* simulations. Our present Definition 2.2(i) corresponds to the lax notion: if r realises $\gamma : A \longrightarrow\!\!\!\!\triangleright\, B$, we do not require that the degree of partiality of r exactly matches that of application in A. To obtain the stricter notion of simulation, we would need to add a condition such as

$$\gamma(a, a') \wedge \gamma(b, b') \wedge r \cdot b \cdot b' \downarrow \quad \Rightarrow \quad a \cdot a' \downarrow .$$

(The notion of applicative transformation given by Definition 2.2(ii) remains the appropriate one to use in connection with strict simulations – *cf.* Cockett and Hofstra (2010, Section 2)). It is known that the equivalence of (strict) PCAs through strict applicative morphisms is a stronger condition than the equivalence through lax ones (Cockett and Hofstra 2008, Section 5)[†].

An analogue of our correspondence theorem can also be obtained for the stricter equivalence: whereas lax equivalence coincides with equivalence of the associated *categories of assemblies*, Cockett and Hofstra (2008) showed that strict equivalence coincides with the Morita equivalence of *Turing categories*.

Cockett and Hofstra (2008) and Cockett and Hofstra (2010) argue compellingly that strict equivalence is more appropriate where questions of *computability* are concerned, while the lax notion is better adapted to questions of *realizability*. This is because in realizability it is never a problem if a realiser does 'more' than the job it is designed for, whereas in computability we are often interested in the precise domain of a computable operation. Certainly, if the idea is that equivalent structures should share 'the same' computability theory in some sense, then there will be significant aspects of our structures (such as their theory of m-reducibility) that are stable under strict equivalences but not under lax ones. So, if we consider models modulo lax equivalence, we are at best dealing with 'computability notions' in a somewhat coarse-grained sense.

Our main reason for concentrating here on the lax notion of simulation is that this appears to us (at present) to be the mathematically more fruitful choice once we have generalised to *first-order* structures as in Section 3. Indeed, we shall see that the **Asm** construction and its associated theory generalise well to this broader setting, whereas it seems there is rather little to say by way of a corresponding generalisation of Turing categories: most of the interest in these categories (as presented in Cockett and Hofstra (2008)) seems tied to their higher-order nature. We would also add that even our more coarse-grained view would seem to be adequate in practice for mapping out the computability landscape: we do not know of any 'computationally natural' examples of models that are laxly not strictly equivalent.

[†] In the context of Cockett and Hofstra (2008), simulations are also required to be *single-valued*. (*cf.* Remark 3.6(iii)). However, this does not affect the present discussion since equivalences in both the lax and strict settings may be taken without loss of generality to consist of single-valued simulations (see Corollary 5.18 and the subsequent remark).

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

2.2. *Typed PCAs*

In this section we give a brief sketch of the generalisation of the above theory to *typed PCAs* as in Longley (1999a) and Longley (1999b)[†]. The essential difference compared with untyped PCAs is that in place of a single carrier set A, we now allow a whole family of carrier sets indexed by *types*.

Definition 2.7.

(i) A *type world* is simply an inhabited set T of *type names* (which we shall just call *types* from now on) endowed with right-associative binary operations $*$ and $\Rightarrow$.

(ii) A *typed PCA* (or TPCA) over a type world T is a family $(A_\sigma \mid \sigma \in T)$ of inhabited sets equipped with partial 'application' operations $\cdot_{\sigma\tau} : A_{\sigma\Rightarrow\tau} \times A_\sigma \rightharpoonup A_\tau$ for each $\sigma, \tau \in T$, such that for every $\rho, \sigma, \tau \in T$, there exist elements

$$\begin{aligned} k_{\sigma\tau} &\in A_{\sigma\Rightarrow\tau\Rightarrow\sigma} \\ s_{\rho\sigma\tau} &\in A_{(\rho\Rightarrow\sigma\Rightarrow\tau)\Rightarrow(\rho\Rightarrow\sigma)\Rightarrow\rho\Rightarrow\tau} \\ pair_{\sigma\tau} &\in A_{\sigma\Rightarrow\tau\Rightarrow(\sigma*\tau)} \\ fst_{\sigma\tau} &\in A_{(\sigma*\tau)\Rightarrow\sigma} \\ snd_{\sigma\tau} &\in A_{(\sigma*\tau)\Rightarrow\tau} \end{aligned}$$

satisfying the following for all appropriately typed x, y, z:

$$\begin{aligned} k \cdot x \cdot y &= x \\ s \cdot x \cdot y &\downarrow \\ s \cdot x \cdot y \cdot z &\succeq x \cdot z \cdot (y \cdot z) \\ fst \cdot (pair \cdot x \cdot y) &= x \\ snd \cdot (pair \cdot x \cdot y) &= y. \end{aligned}$$

Note that TPCAs are inherently higher order: an 'operation' that maps values of type σ to values of type τ is itself a value of type $\sigma \Rightarrow \tau$. In the study of higher-order computability, we are often interested in the type world freely generated from some set of basic type names by means of $*$ and $\Rightarrow$; a TPCA over such a type world may be called a *simply typed* PCA. Note also that an untyped PCA is just a TPCA over the singleton type world $\mathbf{1}$.

The remaining definitions from Section 2.1 can be straightforwardly adapted to the TPCA setting by means of suitable type decorations. For instance, if A and B are TPCAs over type worlds T and U, respectively, an applicative morphism $\gamma : A \longrightarrow\!\!\!\!\triangleright\, B$ consists of a function $\gamma : T \to U$ together with a family of total relations γ_σ from A_σ to $B_{\gamma\sigma}$ such that each application operation $\cdot_{\sigma\tau}$ in A is 'tracked' by some $r \in B_{\gamma(\sigma\Rightarrow\tau)\Rightarrow\gamma\sigma\Rightarrow\gamma\tau}$ in the sense of Definition 2.2(i). With the evident definition of an applicative transformation $\gamma \preceq \delta$, this yields a preorder-enriched category $\mathcal{TPCA}$, of which $\mathcal{PCA}$ is a full sub-2-category.

[†] Note that typed PCAs were called 'partial combinatory type structures' in Longley (1999b).

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

An assembly over a TPCA A consists of a triple $(X, \sigma, \Vdash)$ where X is a set, σ is a type and $\Vdash \subseteq A_\sigma \times X$ is a relation such that for every $x \in X$, there is some $a \Vdash X$. A morphism $(X, \sigma, \Vdash) \to (Y, \tau, \Vdash')$ of such assemblies is just a function $X \to Y$ that is tracked by some $r \in A_{\sigma \Rightarrow \tau}$ in the obvious way. Once again, the category **Asm**(A) of assemblies over a TPCA A turns out to be a $\Gamma\nabla$-regular category, and the **Asm** construction yields a 2-functor $\mathcal{TPCA} \to \Gamma\nabla\mathcal{REG}$, which is locally an equivalence and extends the 2-functor $\mathcal{PCA} \to \Gamma\nabla\mathcal{REG}$ described earlier. As a corollary, the categories **Asm**(A) and **Asm**(B) are equivalent if $A \simeq B$ in the 2-category $\mathcal{TPCA}$. Note that equivalences are possible between TPCAs over different type worlds – indeed, a major theme of Longley (1999a) was the identification of equivalences between particular untyped PCAs and particular simply typed ones.

3. Computability structures and realizability

We are now ready to introduce our primary object of study: the 2-category of *computability structures* (which we will refer to as *C-structures* from now on) and *simulations* between them. We offer this as a relatively broad framework within which a version of the above theory naturally goes through.

3.1. *Computability structures*

In essence, the notion of a computability structure can be seen as a 'first-order flattening' of the notion of a TPCA, in which data values are sharply distinguished from the computable operations that act on them. Furthermore, our computability structures allow computable operations to be multi-valued or 'non-deterministic'.

Definition 3.1. A *C-structure* C consists of the following:

— a family $|\mathsf{C}|$ of inhabited sets (regarded as 'datatypes')
— for each $A, B \in |\mathsf{C}|$, a set $\mathsf{C}[A, B]$ of relations from A to B (regarded as 'computable operations', possibly partial and possibly multi-valued)

such that:

— for each $A \in |\mathsf{C}|$, there exists $i \in \mathsf{C}[A, A]$ such that $i(a, a)$ for all $a \in A$ (we call such an i a *superidentity* relation on A)
— if $r \in \mathsf{C}[A, B]$ and $s \in \mathsf{C}[B, C]$, there exists $t \in \mathsf{C}[A, C]$ such that if $r(a, b)$ and $s(b, c)$, then $t(a, c)$ (we call such a relation t a *supercomposite* of r and s).

A few technical remarks on this definition are in order here.

Remarks 3.2.

(i) Arguably, it would be more principled to draw a distinction between *names* of types and their sets of values – that is, in the spirit of Definition 2.7, to define a C-structure over a set T of type names to be a family of sets $(A_\tau \mid \tau \in T)$ together with certain relations between them. This would, for example, allow distinct type names to be assigned the same set of values. However, no interesting extra generality

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

is at stake here since we may simply relabel elements of datatypes if necessary, and we prefer to avoid the extra notational clutter that such an approach would entail.

(ii) Related to this is our decision to require our datatypes to be inhabited. This is more a matter of stylistic preference than mathematical necessity: all of our main results can be made to go through in the presence of empty types. However, if *one* empty type were admitted, we would also have to allow C-structures with several unrelated empty types (for example, to uphold the product construction of Proposition 4.1); and in the absence of type names, our framework cannot support these. As it is, we prefer to sidestep these annoying complications by outlawing empty types altogether. Note, however, that the empty C-structure $\mathbf{0}$ (with no datatypes at all) *is* admitted by Definition 3.1.

(iii) The fact that we only require 'superidentity' and 'supercomposite' relations is a feature shared with Hofstra (2006), though not with Cockett and Hofstra (2010), where the ambient categorical framework imposes the existence of genuine identities and composites. In the case of superidentities, the extra generality is probably a mere curiosity since we do not know of any natural examples where proper identities are not present. In the case of composition, however, the additional generality is certainly significant: as we shall shortly see, there are natural examples in which supercomposites are present but ordinary relational composites are not.

The decision to work with these 'super' notions can also be seen as consonant with the decision to use $\geq$ rather than $\simeq$ in the definition of a PCA. Following our terminology for PCAs, C-structures with proper identities and composition may be referred to as *strict*.

(iv) For anyone familiar with TPCAs, it might appear surprising that our 'computable operations' are essentially *extensional* objects: there would seem to be no provision for distinct 'operations' that induce the same relation on data values. This is because our 'computable operations' generalise the notion of TPCA elements only in their role as the *left* argument in an application $a \cdot b$ – and for this, only their extension is relevant. We shall see later (Definition 5.22) that when computable operations may themselves be represented by data values, intensional distinctions are indeed allowed for as in TPCAs.

(v) Note how the above definition relates to the approach taken in Cockett and Hofstra (2010). There, the basic notion of a 'model of computation' is that of a category $\mathcal{D}$ equipped with a functor $F : \mathcal{D} \to \mathcal{C}$ where $\mathcal{C}$ is a *restriction category* (a 'category of partial maps' in a certain abstract sense). No real generality is lost by assuming F is faithful. The leading example of a restriction category is the category of sets and partial functions, and in this case such models amount to strict C-structures in which all computable operations are single-valued. We shall consider this important class of structures in Section 5.2 under the name of *deterministic* C-structures. If $\mathcal{C}$ is taken to be the category **Rel** of sets and relations, we obtain, in essence, the class of all strict C-structures.

One may also take $\mathcal{C}$ to be the category of sets and *total* functions (with a trivial restriction structure). Indeed, given any category $\mathcal{C}$ with a terminal object 1, we

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

may form the C-structure whose datatypes are the sets $\mathrm{Hom}(1, X)$ for $X \in \mathcal{C}$ with computable operations induced by morphisms in $\mathcal{C}$ through composition.

(vi) It is natural to ask whether we can 'categorify' our notion of a (lax) C-structure by considering a category $\mathcal{D}$ endowed with a *lax* functor $F : \mathcal{D} \to \mathbf{Rel}$:

$$\begin{aligned} \mathrm{id}_{F(X)} &\subseteq F(\mathrm{id}_X) \\ F(g) \circ F(f) &\subseteq F(gf). \end{aligned}$$

Indeed, the image in **Rel** of any such F is in essence a C-structure. However, there are also C-structures that do not arise in this way. For instance, consider the C-structure C with four datatypes

$$\begin{aligned} A &= \{a\} \\ B &= \{b\} \\ C &= \{c\} \\ D &= \{d, d'\}, \end{aligned}$$

and with operations as follows (alongside the four identity operations):

$$\begin{aligned} \{a \mapsto b\} &\in \mathsf{C}[A, B] \\ \varnothing &\in \mathsf{C}[B, C] \\ \{c \mapsto d\} &\in \mathsf{C}[C, D] \\ \{a \mapsto c\} &\in \mathsf{C}[A, C] \\ \{b \mapsto d'\} &\in \mathsf{C}[B, D] \\ \{a \mapsto d\}, \{a \mapsto d'\} &\in \mathsf{C}[A, D]. \end{aligned}$$

Suppose C were the image of a lax functor $F : \mathcal{D} \to \mathbf{Rel}$, where

$$\begin{aligned} F(f) &= \{a \mapsto b\} \\ F(g) &= \varnothing \\ F(h) &= \{c \mapsto d\}. \end{aligned}$$

Then we must have

$$F(gf) = \{a \mapsto c\},$$

so

$$F(hgf) = \{a \mapsto d\}.$$

On the other hand,

$$F(hg) = \{b \mapsto d'\},$$

so

$$F(hgf) = \{a \mapsto d'\}.$$

As the above remarks show, the notion of a C-structure is mathematically very general indeed. However, our primary interest is in C-structures that model computational

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

processes or operations of some sort. The following selection of examples gives a broad indication of the kinds of examples we have in mind.

Examples 3.3.

(i) Any TPCA A over a type world T gives rise to a C-structure A as follows. Let $|\mathsf{A}|$ be the family of sets A_σ where $\sigma \in T$ (relabelling if necessary as per Remark 3.2(i)), and let $\mathsf{A}[A_\sigma, A_\tau]$ be the set of partial functions $A_\sigma \rightharpoonup A_\tau$ induced by elements of $A_{\sigma \Rightarrow \tau}$.

(ii) Given a labelled transition system with set of states S, set of labels L and transition relation $\rightarrow \subseteq S \times L \times S$ (with transitive closure $\twoheadrightarrow$), we may define a C-structure C as follows. Take $|\mathsf{C}| = \{S\}$. For $w \in L^*$ any finite sequence of labels, let r_w be the relation $\{(x, y) \mid x \overset{w}{\twoheadrightarrow} y\}$ on S, and let $\mathsf{C}[S, S]$ be the set $\{r_w \mid w \in L^*\}$. This is clearly a C-structure.

(iii) Let $\mathcal{L}$ be some language or calculus comprising the following ingredients:

— A set T of *types*.

— For each type $\sigma \in T$, an inhabited set $\mathcal{L}_\sigma$ of *terms* of type σ (we may think of them as 'closed terms').

— For each $\sigma \in T$, a reflexive, transitive *many-step reduction* relation $\twoheadrightarrow \subseteq \mathcal{L}_\sigma \times \mathcal{L}_\sigma$.

— A small category $\mathcal{K}$ whose set of objects is T and whose morphisms are regarded as syntactic *evaluation contexts*, denoted by $K[-]$. This is equipped with a functor $\Theta : \mathcal{K} \to \mathbf{Set}$ such that $\Theta(\sigma) = \mathcal{L}_\sigma$ for each $\sigma \in T$; we write $\Theta(K[-])(M)$ as $K[M]$. We also impose the 'evaluation context condition' that $M \twoheadrightarrow M'$ implies $K[M] \twoheadrightarrow K[M']$ for each $K[-]$ in $\mathcal{K}$.

— For each $\sigma \in T$, an inhabited subset $\mathcal{V}_\sigma \subseteq \mathcal{L}_\sigma$ of terms designated as *values*.

Given such a language $\mathcal{L}$, we may define a C-structure $\mathsf{C}_\mathcal{L}$ as follows. Let $|\mathsf{C}_\mathcal{L}|$ be the family of all sets $\mathcal{V}_\sigma$ for $\sigma \in T$ (again relabelling if necessary as per Remark 3.2(i)). For any $K[-] : \sigma \to \tau$ in $\mathcal{K}$, let $r_K \subseteq \mathcal{V}_\sigma \times \mathcal{V}_\tau$ be the relation $\{(M, N) \mid K[M] \twoheadrightarrow N\}$, and take

$$\mathsf{C}_\mathcal{L}[\mathcal{V}_\sigma, \mathcal{V}_\tau] = \{r_K \mid K[-] : \sigma \to \tau\}.$$

It is easy to check that $\mathsf{C}_\mathcal{L}$ is endowed with identities and supercomposition of relations.

From example (iii), we may now see the prototypical reason why a naturally arising C-structure need not be closed under ordinary relational composition. If $K[M] \twoheadrightarrow N$ and $L[N] \twoheadrightarrow P$, then $L[K[M]] \twoheadrightarrow P$, but the term $L[K[M]]$ might also admit reductions to values that do not proceed via any such $L[N]$. For instance, in a process algebra with parallel composition, if $K[-] = Q|-$ and $L[-] = R|-$, then $L[K[M]] = R|Q|M$ may admit reductions that depend on R interacting directly with M[†].

To give further insight into how the machinery of Example 3.3(iii) works in practice, we will now briefly outline two concrete specialisations, which will also feature later in Example 3.8.

[†] A further example of some historical interest is Kleene's class of (S1)–(S9) computable partial functionals of higher type, which is not closed under ordinary composition but admits supercomposites in the present sense.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Examples 3.4.

(i) Abramsky's *lazy λ-calculus* (Abramsky 1989) gives rise to a C-structure as follows. Consider the language with a single type whose terms are just the closed terms of untyped λ-calculus. Let $\to_l$ be the *lazy reduction* relation generated by

$$(\lambda x.M)N \ \to_l \ M[x \mapsto N]$$
$$M \ \to_l \ M' \ \Rightarrow \ MN \ \to_l \ M'N,$$

and let $\twoheadrightarrow_l$ be its reflexive-transitive closure. As evaluation contexts, we take all syntactic contexts of the form $-N_1 \dots N_k$ where $k \geqslant 0$ and the N_i are closed terms; these clearly satisfy the evaluation context condition. (Note that contexts $M-$ will not satisfy this condition unless we choose to add a further reduction rule.) Finally, we may take as values all closed terms of the form $\lambda x.M$. We write L for the resulting C-structure defined as in Example 3.3(iii).

(ii) Likewise, Milner's *π-calculus* gives rise to a C-structure. We assume familiarity here with the presentation in Milner (1992). To facilitate the construction of complex process terms, we shall employ a formulation that allows for *meta-terms* $\Lambda x.P$ in which a single name x has been abstracted. Specifically, terms and meta-terms are generated by the following grammar, where x and y range over names:

$$\begin{array}{lrcl} \text{Terms} & P, Q & ::= & \bar{x}y.P \ \mid \ x(y).P \ \mid \ \mathbf{0} \ \mid \ P|Q \ \mid \ !P \ \mid \ (y)P \ \mid \ Uy \\ \text{Meta-terms} & U & ::= & \Lambda x.P \end{array}$$

Terms and meta-terms are considered modulo a *structural congruence* relation $\equiv$, which is defined as in Milner (1992, Definition 3.1), but with the following additional clauses:

$$\Lambda x.P \ \equiv \ \Lambda y.P[x \mapsto y]$$
$$(\Lambda x.P)y \ \equiv \ P[x \mapsto y]$$
$$P \ \equiv \ P' \ \Rightarrow \Lambda x.P \ \equiv \ \Lambda x.P'.$$

Thus, every term P is congruent to a Λ-free one.

A reduction relation $\to_p$ is now defined on terms and meta-terms modulo $\equiv$ as in Milner (1992, Definition 3.2), but with the additional clauses:

$$P \ \to_p \ P' \ \Rightarrow \ \Lambda x.P \ \to_p \ \Lambda x.P'$$
$$U \ \to_p \ U' \ \Rightarrow \ Uy \ \Rightarrow_p \ U'y$$

For Λ-free terms, this clearly coincides with the reduction relation defined in Milner (1992).

We may now consider the language whose terms are closed *meta-terms* U, taking $\twoheadrightarrow_p$ to be the reflexive-transitive closure of $\to_p$ restricted to such meta-terms. As evaluation contexts, we may take the class of *unguarded* contexts over meta-terms, namely those generated by composition from syntactic contexts of the form

$$-|Q \qquad (y)- \qquad -\,y \qquad \Lambda x.-$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

The definition of $\twoheadrightarrow_p$ ensures that these satisfy the evaluation context condition. Finally, as values, we take those meta-terms that cannot be further reduced via $\rightarrow_p$. We write P for the resulting C-structure.

3.2. *Simulations*

In this section we introduce the notion of a *simulation* of one C-structure in another, which generalises the notion of an applicative morphism introduced in Section 2.1.

Definition 3.5. Let C, D be C-structures. Then:

(i) A *simulation* $\gamma : \mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$ consists of

— a function $A \mapsto \gamma A : |\mathsf{C}| \to |\mathsf{D}|$

— for each $A \in |\mathsf{C}|$, a total relation γ_A from A to γA

such that every relation $r \in \mathsf{C}[A, B]$ is *tracked* by some $r' \in \mathsf{D}[\gamma A, \gamma B]$: that is,

$$r(a, b) \wedge \gamma_A(a, a') \quad \Rightarrow \quad \exists b'.\, r'(a', b') \wedge \gamma_B(b, b').$$

(ii) If γ, δ are simulations $\mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$, we say γ is *transformable* to δ, and write $\gamma \preceq \delta$, if for each $A \in |\mathsf{C}|$ there exists $t \in \mathsf{D}[\gamma A, \delta A]$ such that

$$\gamma_A(a, a') \quad \Rightarrow \quad \exists a''.\, t(a', a'') \wedge \delta_A(a, a'').$$

Remarks 3.6.

(i) In accordance with the discussion of Section 2.1, we have opted for a 'lax' rather than a 'strict' notion of simulation. This is also coherent with our decision to give primacy to lax C-structures rather than strict ones. The strict definition would incorporate the condition

$$\gamma_A(a, a') \wedge r'(a', b') \;\Rightarrow\; \exists b.\, r(a, b) \wedge \gamma_B(b, b').$$

(ii) The definition of a simulation also presents us with a choice regarding the treatment of non-deterministic (that is, multi-valued) relations. In the above definition, we have taken the 'liberal' approach of requiring only that every possible behaviour of r on an input a can be mirrored by r' on any corresponding element a', though there may be additional behaviours of r' on a' with no counterpart in r. In Section 5.1, we shall consider a stricter notion of tracking in which the range of possible behaviours of r' on a' precisely matches that of r on a: this yields the notion of a *tight* simulation. Note that all single-valued simulations are tight, although not all are strict. Note also that the tightness condition is vacuous if every relation in D is single-valued.

(iii) Our definition of simulation is closely related to what is termed a *lax total simulation* in Cockett and Hofstra (2010, Section 2.2). Indeed, if we take $\mathcal{C}$ to be the base category of sets and partial functions, then lax total simulations over $\mathcal{C}$ are basically *single-valued* simulations between the corresponding C-structures, and the *refinement* relation between such simulations coincides with our $\preceq$. Although from our point of view the restriction to single-valued simulations is rather severe and excludes

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

many of our leading examples even in the untyped PCA setting (see, for example, Longley (1995, Chapter 3)), it is at least the case that *equivalence* via single-valued simulations coincides with equivalence via arbitrary ones: see Section 5.3.

If $\mathcal{C}$ is taken to be the category of sets and relations, the connection is less clear-cut: every lax total simulation is a simulation in our sense, but not conversely.

Proposition 3.7. C-structures and simulations (ordered by $\preceq$) form a preorder-enriched category, which we denote by $\mathcal{CSTRUCT}$.

Proof. The identity simulation on a C-structure C is as expected – note that every relation in C tracks itself. Composition of simulations is given by 'typewise' relational composition: given $\gamma : \mathsf{C} \to \mathsf{D}$ and $\delta : \mathsf{D} \to \mathsf{E}$, the action of $\delta \circ \gamma$ on $|\mathsf{C}|$ is just the composition of δ and γ, and for each $A \in |\mathsf{C}|$, the relation $(\delta \circ \gamma)_A$ is simply the (forwards) relational composition $\gamma_A ; \delta_{\gamma A}$. To see that $\delta \circ \gamma$ is indeed a simulation, suppose $r \in \mathsf{C}[A, B]$ and take $r' \in \mathsf{D}[\gamma A, \gamma B]$ tracking r with respect to γ, and $r'' \in \mathsf{E}[\delta\gamma A, \delta\gamma B]$ tracking r' with respect to δ. It is then clear that r'' tracks r with respect to $\delta \circ \gamma$. The unit and associativity laws for simulations are now immediate from the corresponding facts for relations.

Next we check that for any C and D, the simulations $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ are preordered by $\preceq$. For any such simulation γ, the transformation $\gamma \preceq \gamma$ is witnessed at each $A \in \mathsf{C}$ by any superidentity relation $i \in \mathsf{D}[\gamma A, \gamma A]$. Now suppose $\gamma \preceq \delta \preceq \epsilon$. For any $A \in |\mathsf{C}|$, take $t \in \mathsf{D}[\gamma A, \delta A]$ and $u \in \mathsf{D}[\delta A, \epsilon A]$ witnessing $\gamma \preceq \delta$ and $\delta \preceq \epsilon$, respectively, at A, and let $v \in \mathsf{D}[\gamma A, \epsilon A]$ be any supercomposite of t and u. Then v witnesses $\gamma \preceq \epsilon$ at A.

Finally, we show that composition of simulations is monotone in both arguments with respect to $\preceq$. First, given $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ and $\delta \preceq \delta' : \mathsf{D} \longrightarrow\!\!\triangleright \mathsf{E}$, if t witnesses $\delta \preceq \delta'$ at γA, it is clear that t also witnesses $\delta \circ \gamma \preceq \delta' \circ \gamma$ at A. Second, given $\gamma \preceq \gamma' : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ and $\delta : \mathsf{D} \longrightarrow\!\!\triangleright \mathsf{E}$, if $t \in \mathsf{D}[\gamma A, \gamma' A]$ witnesses $\gamma \preceq \gamma'$ at A, we may take $t' \in \mathsf{E}[\delta\gamma A, \delta\gamma' A]$ tracking t with respect to δ. We then claim t' witnesses $\delta \circ \gamma \preceq \delta \circ \gamma'$ at A. To see this, suppose $(\delta \circ \gamma)_A(a, a')$. Then there exists a^* with $\gamma_A(a, a^*)$ and $\delta_{\gamma A}(a^*, a')$, so there exists a^{**} with $t(a^*, a^{**})$ and $\gamma'(a, a^{**})$. Hence, there exists a'' with $t'(a', a'')$ and $\delta(a^{**}, a'')$, so $(\delta \circ \gamma')_A(a, a'')$ as required. □

We will conclude this section with some examples of simulations.

Examples 3.8.

(i) Suppose A, B are TPCAs and $\gamma : A \longrightarrow\!\!\triangleright B$ is an applicative morphism in the sense of Section 2.2. Let A, B be the C-structures arising from A, B as in Example 3.3(i). Then γ may clearly be viewed as a simulation $\mathsf{A} \longrightarrow\!\!\triangleright \mathsf{B}$. Since it is clear that the preorder $\preceq$ on applicative morphisms agrees with that on simulations, we obtain an inclusion 2-functor $\mathcal{TPCA} \to \mathcal{CSTRUCT}$.

The question of which simulations $\mathsf{A} \longrightarrow\!\!\triangleright \mathsf{B}$ arise from applicative morphisms in this way will be addressed in Section 5.5.

(ii) Suppose $(S, L, \to)$ and $(S', L, \to')$ are labelled transition systems over the same set L of labels, and let C, C' be the corresponding C-structures as in Example 3.3(ii). A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{C}'$ is then just a certain kind of relation $\gamma \subseteq S \times S'$. We

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

say γ *respects labels* if $r'_{w'}$ tracks r_w implies $w' = w$. A label-respecting simulation is thus precisely 'half a bisimulation' in the sense of concurrency theory, and γ is a bisimulation if and only if both γ and γ^{op} are label-respecting simulations.

(iii) As a more substantial example, we will indicate how Milner's well-known translation from the lazy λ-calculus to the π-calculus (Milner 1992) can be viewed as a simulation $\mathsf{L} \longrightarrow\!\triangleright \mathsf{P}$. In effect, Milner defines a compositional translation $[\![-]\!]$ from untyped λ-terms to π-calculus meta-terms:

$$\begin{aligned}
[\![\lambda x.M]\!] &= \Lambda u.\, u(x).u(v).[\![M]\!]v \\
[\![x]\!] &= \Lambda u.\, \bar{x}u \\
[\![MN]\!] &= \Lambda u.\, (v)([\![M]\!]v \mid (x)\bar{v}x.\bar{v}u.[\![x := N]\!]) \quad (x \notin FV(N)) \\
[\![x := N]\!] &= !x(w).[\![M]\!]w.
\end{aligned}$$

Note that if M is a λ-abstraction, then $[\![M]\!]$ is a value (that is, it admits no $\rightarrow_p$-reductions).

The following 'simulation relation' is introduced in Milner (1992, Definition 4.4). If L is a closed untyped λ-term and U is a closed π-calculus meta-term, we write $\rho(L, U)$ if there exist λ-terms $M, N_1, \ldots, N_k$ and distinct variables $x_1, \ldots, x_k$ $(k \geqslant 0)$ with

$$\begin{aligned}
FV(M) &\subseteq x_1, \ldots, x_k \\
FV(N_i) &\subseteq x_{i+1}, \ldots, x_k
\end{aligned}$$

such that

$$\begin{aligned}
L &\equiv M[x_1 \mapsto N_1] \cdots [x_k \mapsto N_k] \\
P &\equiv \Lambda u.\, (x_1) \cdots (x_k)([\![M]\!]u \mid [\![x_1 := N_1]\!] \mid \cdots \mid [\![x_k := N_k]\!]).
\end{aligned}$$

Restricting ρ to values, we obtain a total relation ρ from elements of L to elements of P (note that for any closed λ-abstraction L, we have $\rho(L, [\![L]\!])$).

To see that ρ is a simulation $\mathsf{L} \longrightarrow\!\triangleright \mathsf{P}$, it will suffice to show that an L-context $-N$ is tracked by the P-context

$$\Pi_N[-] \;=\; \Lambda w.\, (v)(-v \mid (x)\bar{v}x.\bar{v}w.[\![x := N]\!])$$

where x is fresh. From this it follows easily that a general L-context $-N_1 \ldots N_k$ is tracked by the corresponding composition of P-contexts. First, a simple calculation shows that if L is a value $\lambda z.L'$ and $\rho(L, P)$, then for any closed N, we have $\Pi_N(P) \twoheadrightarrow_p P'$ for some P' with $\rho(L'[z \mapsto N], P')$. Second, Milner (1992, Lemma 4.5) shows that:

— If $L'[z \mapsto N] \rightarrow_l L_1$ and $\rho(L'[z \mapsto N], P')$, then for some P_1 we have $P' \twoheadrightarrow_p P_1$ and $\rho(L_1, P_1)$.

— If $\rho(L_1, P_1)$ and L_1 is an abstraction, then for some value P'_1 we have $P_1 \twoheadrightarrow_p P'_1$ and $\rho(L_1, P'_1)$.

Putting all this together gives us that if LN reduces to a value L_1 and $\rho(L, P)$, then $\Pi_N(P)$ reduces to a value P'_1 where $\rho(L_1, P'_1)$ as required.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

3.3. *The* **Asm** *construction on* $\mathcal{CSTRUCT}$

The 'category of assemblies' construction can be naturally generalised to C-structures.

Definition 3.9. Given a C-structure C, we define a category $\mathbf{Asm}(\mathsf{C})$ and a functor $\Gamma_{\mathsf{C}} : \mathbf{Asm}(\mathsf{C}) \to \mathbf{Set}$ as follows:

— An object X in $\mathbf{Asm}(\mathsf{C})$ is a triple $(|X|, A_X, \Vdash_X)$ where $|X|$ is a set, $A_X \in |\mathsf{C}|$ and $\Vdash_X \subseteq A_X \times |X|$ is a relation such that for every $x \in |X|$ there exists some $a \Vdash_X x$.

— A morphism $f : X \to Y$ in $\mathbf{Asm}(\mathsf{C})$ is a function $f : |X| \to |Y|$ that is *tracked* by some $r \in \mathsf{C}[A_X, A_Y]$: that is,

$$a \Vdash_X x \quad \Rightarrow \quad \exists b.\, r(a,b) \wedge b \Vdash_Y f(x).$$

Identities and composition are defined as in **Set**.

— Γ_{C} is the evident forgetful operation $X \mapsto |X|$, $f \mapsto f$.

It is easy to see that $\mathbf{Asm}(\mathsf{C})$ is indeed a category and Γ_{C} is a faithful functor. We shall always think of $\mathbf{Asm}(\mathsf{C})$ as coming equipped with the functor Γ_{C}; in contrast to the situation for PCAs, we cannot in general recover Γ_{C} from $\mathbf{Asm}(\mathsf{C})$ as the functor $\mathrm{Hom}(1, -)$.

Once again, our notion of 'tracking' here involves a choice regarding the treatment of non-determinism. Our choice of definition here is consonant with Definition 3.5, though a more restricted notion of assembly morphism will be appropriate when we consider a stricter notion of simulation in Section 5.1.

We now investigate what structure is present in $(\mathbf{Asm}(\mathsf{C}), \Gamma_{\mathsf{C}})$. To this end, we introduce the following general notions.

Definition 3.10. Let $\mathcal{C}$ be a category and Γ be a faithful functor $\mathcal{C} \to \mathbf{Set}$.

(i) $(\mathcal{C}, \Gamma)$ *has subobjects* if for any object X in $\mathcal{C}$ and any mono $s : S \rightarrowtail \Gamma(X)$ in **Set**, there exists a morphism $\bar{s} : Y \to X$ in $\mathcal{C}$ (necessarily mono) such that $\Gamma(\bar{s}) = s$, and for any morphism $f : Z \to X$ such that $\Gamma(f)$ factors through s, there is a unique $g : Z \to Y$ such that $f = \bar{s} \circ g$.

(ii) $(\mathcal{C}, \Gamma)$ *has quotients* if for any object X in $\mathcal{C}$ and any epi $q : \Gamma(X) \twoheadrightarrow Q$ in **Set**, there exists a morphism $\bar{q} : X \to V$ in $\mathcal{C}$ (necessarily epi) such that $\Gamma(\bar{q}) = q$, and for any morphism $f : X \to W$ such that $\Gamma(f)$ factors through q, there is a unique $g : V \to W$ such that $f = g \circ \bar{q}$.

(iii) $(\mathcal{C}, \Gamma)$ *has copies* if for any $X \in \mathcal{C}$ and $S \in \mathbf{Set}$, there is an object $X \propto S$ in $\mathcal{C}$ equipped with morphisms

$$\pi : X \propto S \to X$$
$$\theta : \Gamma(X \propto S) \to S$$

such that for any $f : Z \to X$ and $\phi : \Gamma(Z) \to S$ there is a unique $g : Z \to X \propto S$ with $f = \pi \circ g$ and $\phi = \theta \circ \Gamma(g)$.

(iv) We say $(\mathcal{C}, \Gamma)$ is a *quasi-regular category over* **Set** if it has subobjects, quotients and copies.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Note that the universal structures given by this definition, when they exist, are unique up to unique isomorphism. In the context of part (i), any morphism $\bar{s}$ with the required properties is called a *lifting* of s to $\mathcal{C}$. Similarly, in part (ii), we say $\bar{q}$ is a lifting of q to $\mathcal{C}$. In part (iii), an object $X \propto S$ with the required properties is called an *S-fold copy of X*.

The name 'quasi-regular' reflects the idea that this is as much of the structure of a regular category as survives in this general setting (where we do not have finite products, for instance). The existence of copies does duty for the functor $\nabla : \mathbf{Set} \to \mathbf{Asm}(A)$ in the PCA case.

Proposition 3.11. $(\mathbf{Asm}(\mathsf{C}), \Gamma_{\mathsf{C}})$ is a quasi-regular category over **Set**.

Proof. To see that $(\mathbf{Asm}(\mathsf{C}), \Gamma_{\mathsf{C}})$ has subobjects, suppose $X \in \mathbf{Asm}(\mathsf{C})$ and $s : S \rightarrowtail |X|$. Let Y be the assembly defined by

$$|Y| = S$$
$$A_Y = A_X$$

and

$$a \Vdash_Y y \quad \text{iff} \quad a \Vdash_X s(y),$$

and let $\bar{s} = s$ (this is tracked by any superidentity for A_X). The universal property is then easy. (In fact, it is clear that such a lifting $\bar{s}$ exists even when s is not mono.)

For quotients, suppose $X \in \mathbf{Asm}(\mathsf{C})$ and $q : |X| \twoheadrightarrow Q$. Let V be the assembly defined by

$$|V| = Q$$
$$A_V = A_X,$$

and

$$a \Vdash_V v$$

if and only if there exists $x \in |X|$ with $a \Vdash_X x$ and $q(x) = v$, and let $\bar{q} = q$ (this is tracked by any superidentity for A_X). The universal property is easy.

For copies, suppose $X \in \mathbf{Asm}(\mathsf{C})$ and $S \in \mathbf{Set}$. Define the assembly $X \propto S$ by

$$|X \propto S| = |X| \times S$$
$$A_{X \propto S} = A_X$$

and

$$a \Vdash_{X \propto S} (x, s) \quad \text{iff} \quad a \Vdash_X x.$$

Let $\pi : X \propto S \to X$ be the first projection (this is tracked by any superidentity), and let $\theta : |X \propto S| \to S$ be the second projection. Once again, the universal property is easy. □

We will now show how the **Asm** construction may be extended to simulations.

Definition 3.12. Given a simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$, define

$$\gamma_* = \mathbf{Asm}(\gamma) : \mathbf{Asm}(\mathsf{C}) \to \mathbf{Asm}(\mathsf{D})$$

as follows:

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

— On objects X, we define $\gamma_*(X)$ by

$$|\gamma_*(X)| = |X|$$
$$A_{\gamma_*(X)} = \gamma A_X,$$

and

$$b \Vdash_{\gamma_*(X)} x$$

if and only if

$$\exists a \in A_X . \gamma_{A_X}(a, b) \wedge a \Vdash_X x.$$

— On morphisms $f : X \to Y$, we define $\gamma_*(f) = f$. Note that if f is tracked by $r \in \mathsf{C}[A_X, A_Y]$, and r is itself tracked (with respect to γ) by $r' \in \mathsf{D}[\gamma A_X, \gamma A_Y]$, then $f : \gamma_*(X) \to \gamma_*(Y)$ is tracked by r'.

Proposition 3.13.

(i) $\mathbf{Asm}(\gamma)$ is a functor $\mathbf{Asm}(\mathsf{C}) \to \mathbf{Asm}(\mathsf{D})$, and

$$\Gamma_{\mathsf{D}} \circ \mathbf{Asm}(\gamma) = \Gamma_{\mathsf{C}}.$$

(ii) $\mathbf{Asm}(\gamma)$ preserves subobjects, quotients, and copies.

Proof. All parts are trivial. Indeed, the structures described in the proof of Proposition 3.11 are preserved on the nose by $\mathbf{Asm}(\gamma)$. □

Finally, we show how **Asm** acts on transformations between simulations.

Definition 3.14. Given simulations $\gamma \preceq \delta : \mathsf{C} \longrightarrow\!\!\!\!\triangleright \mathsf{D}$, let $\mathbf{Asm}(\gamma \preceq \delta)$ denote the natural transformation $\xi : \mathbf{Asm}(\gamma) \to \mathbf{Asm}(\delta)$ defined by

$$\xi_X = \mathrm{id}_{|X|} : \gamma_*(X) \to \delta_*(X).$$

(Note that if $t \in \mathsf{D}[\gamma A_X, \delta A_X]$ witnesses $\gamma \preceq \delta$ at A_X, then t tracks ξ_X.)

Since $\mathbf{Asm}(\gamma \preceq \delta)$ is simply the identity at the level of sets, it is clear that it is indeed a natural transformation.

We may now put all this together.

Definition 3.15. Let $\Gamma\mathcal{QRE}\mathcal{G}$ be the (large) 2-category defined as follows:

— Objects are quasi-regular categories over **Set**.

— Morphisms $(\mathcal{C}, \Gamma_{\mathcal{C}}) \to (\mathcal{D}, \Gamma_{\mathcal{D}})$ are functors $F : \mathcal{C} \to \mathcal{D}$ equipped with a natural isomorphism $\iota : \Gamma_{\mathcal{C}} \cong \Gamma_{\mathcal{D}} \circ F$ such that:

- F preserves subobjects modulo ι: that is, if $\bar{s} : Y \to X$ is a subobject lifting of $s : S \rightarrowtail \Gamma_{\mathcal{C}}(X)$, then $F(\bar{s}) : F(Y) \to F(X)$ is a subobject lifting of $\iota_X \circ s \circ \iota_Y^{-1}$.
- F preserves quotients modulo ι (the definition is similar).
- F preserves copies modulo ι: that is, if $(X \propto S, \pi, \theta)$ is an S-fold copy of X in C, then $(F(X \propto S), F(\pi), \theta \circ \iota^{-1}_{X \propto S})$ is an S-fold copy of $F(X)$ in D.

We call such a pair (F, ι) a *quasi-regular* functor.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

— 2-cells $(F, \iota) \to (G, \jmath)$ are natural transformations $\alpha : F \to G$.

The following proposition addresses a couple of fine details. Again, we write $*$ for horizontal composition.

Proposition 3.16.

(i) For a quasi-regular functor (F, ι), the isomorphism ι is uniquely determined by F (so ι may be dropped from the data for a quasi-regular functor).

(ii) If α is any natural transformation $F \to G$ where (F, ι) and $(G, \jmath)$ are quasi-regular functors $(\mathcal{C}, \Gamma_{\mathcal{C}}) \to (\mathcal{D}, \Gamma_{\mathcal{D}})$, then automatically $(\Gamma_{\mathcal{D}} * \alpha) \circ \iota = \jmath$.

Proof.

(i) Suppose $X \in \mathsf{C}$ and $x \in \Gamma_{\mathsf{C}}(X)$. We wish to show that $\iota_X(x)$ is uniquely determined. Let 1 be a singleton set and regard x as a morphism $1 \rightarrowtail \Gamma_{\mathsf{C}}(X)$, and lift this to a morphism $\bar{x} : Y \to X$ in C. Since F preserves subobjects modulo ι, we have that $F(\bar{x}) : F(Y) \to F(X)$ is a subobject lifting of $\iota_X \circ x \circ \iota_Y$. But the latter is a mapping from a singleton set to $\Gamma_{\mathsf{D}}(F(X))$ that picks out $\iota_X(x)$. Hence $\iota_X(x)$ must be the element picked out by $\Gamma_{\mathsf{D}}(F(\bar{x}))$.

(ii) Suppose $X \in \mathsf{C}$ and $x \in \Gamma_{\mathsf{C}}(X)$. We wish to show that

$$\Gamma_D(\alpha_X)(\iota_X(x)) = \jmath_X(x).$$

Again, we regard x as a morphism $1 \rightarrowtail \Gamma_{\mathsf{C}}(X)$ and lift it to a morphism $\bar{x} : Y \to X$ in C. As in the proof of (i), we have that $\Gamma_{\mathsf{D}}(F(\bar{x}))$ picks out $\iota_X(x)$ and $\Gamma_{\mathsf{D}}(G(\bar{x}))$ picks out $\jmath_X(x)$. Now applying Γ_D to the naturality square for α at $\bar{x}$, we have

$$\Gamma_D(\alpha_X) \circ \Gamma_{\mathsf{D}}(F(\bar{x})) = \Gamma_{\mathsf{D}}(G(\bar{x})) \circ \Gamma_{\mathsf{D}}(\alpha_Y).$$

Since $\Gamma_{\mathsf{D}}(\alpha_Y)$ is just a function between singleton sets, this says that

$$\Gamma_D(\alpha_X)(\iota_X(x)) = \jmath_X(x),$$

as required. □

Theorem 3.17. The constructions of Definitions 3.9, 3.12 and 3.14 constitute a 2-functor

$$\mathbf{Asm} \ : \ \mathcal{CSTRUCT} \to \Gamma\mathcal{QREG}.$$

Proof. In addition to the properties already established, it is trivial that **Asm** respects identity simulations, composition of simulations, identity transformations and vertical and horizontal composition of transformations. □

It is clear that the **Asm** construction on C-structures agrees up to isomorphism with that on TPCAs (as in Section 2.2) modulo the inclusion 2-functor $\mathcal{TPCA} \to \mathcal{CSTRUCT}$ noted under Example 3.8(i).

We conclude this section with a brief aside on how our notions of subobject and quotient relate to regular monos and epis in **Asm**(C) (we leave the verifications as exercises). On the one hand, all equalisers exist in **Asm**(C) and our subobjects are precisely the regular monos. On the other hand, all coequalisers exist in **Asm**(C) and every regular epi is a

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

quotient morphism, though the converse does not seem to be true in general. However, if C is endowed with *cartesian products* (as in Section 5.4), the notions of quotient and regular epi coincide.

Furthermore, for any simulation γ, the functor $\mathbf{Asm}(\gamma)$ preserves all equalisers and coequalisers. In the light of Theorem 3.20 below, this means that any quasi-regular functor $\mathbf{Asm}(\mathsf{C}) \to \mathbf{Asm}(\mathsf{D})$ preserves equalisers and coequalisers.

3.4. *The correspondence theorem*

In this section we work towards showing that the 2-functor **Asm** is *locally an equivalence*: that is, for any C and D, the ordinary functor

$$\mathbf{Asm} \;:\; \mathcal{CSTRUCT}[\mathsf{C},\mathsf{D}] \to \Gamma\mathcal{QREG}[\mathbf{Asm}(\mathsf{C}),\mathbf{Asm}(\mathsf{D})]$$

is part of an equivalence of preorders. The following proposition shows that essentially every quasi-regular functor $\mathbf{Asm}(\mathsf{C}) \to \mathbf{Asm}(\mathsf{D})$ arises from some simulation $\mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$.

Proposition 3.18. Let $F : (\mathbf{Asm}(\mathsf{C}), \Gamma_{\mathsf{C}}) \to (\mathbf{Asm}(\mathsf{D}), \Gamma_{\mathsf{D}})$ be a morphism in $\Gamma\mathcal{QREG}$. Then there is a simulation $\gamma : \mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$ such that there is a unique natural isomorphism $\mathbf{Asm}(\gamma) \cong F$.

Proof. Given F, we define γ as follows. For each $A \in |\mathsf{C}|$, let Z_A be the 'object of realisers' in $\mathbf{Asm}(\mathsf{C})$ given by

$$|Z_A| = A$$
$$A_{Z_A} = A$$
$$\Vdash_{Z_A} = \mathrm{id}_A.$$

Let $W_A = F(Z_A)$, and define $\gamma A = A_{W_A}$ and $\gamma_A(a,b)$ if and only if $b \Vdash_{W_A} a$.

To see that γ is a simulation, suppose $r \in \mathsf{C}[A,B]$. We wish to show that r is tracked by some $r' \in \mathsf{D}[\gamma A, \gamma B]$. Let R be the relation r viewed as a subset of

$$|Z_A \propto B| = A \times B,$$

and lift this to a subobject $Z_R \rightarrowtail Z_A \propto B$. Likewise, let S be the opposite of r viewed as a subset of

$$|Z_B \propto A| = B \times A,$$

and lift this to a subobject

$$Z_S \rightarrowtail Z_B \propto A.$$

By the universal property of the latter, the canonical bijection $t : R \to S$ (the twist map) lifts to a morphism $\tilde{t} : Z_R \to Z_S$, which is tracked by r itself:

$$\begin{array}{ccc} Z_R & \xrightarrow{\tilde{t}} & Z_S \\ \downarrow & & \downarrow \\ Z_A \propto B & & Z_B \propto A \end{array}$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

We now apply F to this whole diagram, and then replace the relevant copies and subobjects by the isomorphic 'canonical' ones in **Asm**(D) (that is, those described in the proof of Proposition 3.11):

$$\begin{array}{ccc} Z'_R & \xrightarrow{\hat{t}} & Z'_S \\ \downarrow & & \downarrow \\ F(Z_A) \propto B & & F(Z_B) \propto A \end{array}$$

Note that the type of realisers for Z'_R is that of $F(Z_A)$, that is γA, and, similarly, the type of realisers for Z'_S is γB. So we let $r' \in \mathsf{D}[\gamma A, \gamma B]$ be any relation tracking $\hat{t}$ and claim that r' tracks r with respect to γ. Indeed, suppose $r(a,b)$ and $\gamma_A(a,a')$. Then $a' \Vdash_{Z'_R} (a,b)$ since γ_A is by definition extracted from $F(Z_A)$. So there exists $b' \Vdash_{Z'_S} (b,a) = \hat{t}(a,b)$ such that $r'(a',b')$. But now $\gamma_B(b,b')$ by the definition of $\Vdash_{Z'_S}$ and γ_B (both are induced by $F(Z_B)$) and we are done.

It remains to show that $\gamma_* = \mathbf{Asm}(\gamma) \cong F$. Consider an arbitrary $X \in \mathbf{Asm}(\mathsf{C})$. We may express X as a quotient of a subobject of $Z_A \propto |X|$, where $A = A_X$. Specifically, let R be the subset of $|Z_A \propto |X||$ corresponding to $\Vdash_X$, and let $q : R \twoheadrightarrow |X|$ be the surjection given by second projection. These lift to subobject and quotient maps in **Asm**(C), giving the diagram

$$\begin{array}{ccc} Z_R & \rightarrowtail & Z_A \propto |X| \\ \downarrow & & \\ X & & \end{array}$$

Applying F to this diagram and replacing copies, subobjects and quotients in turn by isomorphic canonical ones, we obtain a diagram

$$\begin{array}{ccc} Z'_R & \rightarrowtail & F(Z_A) \propto |X| \\ \downarrow & & \\ \gamma_*(X) & & \end{array}$$

So we obtain an isomorphism $\gamma_*(X) \cong F(X)$ that agrees on underlying sets with the given isomorphism $\Gamma_{\mathsf{C}} \cong \Gamma_{\mathsf{D}} \circ F$.

To see that this isomorphism is natural in X, it now suffices to note that if $f : X \to Y$ in **Asm**(C), then

$$\Gamma_{\mathsf{D}}(\gamma_*(f)) = f,$$

and

$$\Gamma_{\mathsf{D}}(F(f)) \cong \Gamma_{\mathsf{C}}(f) = f$$

through the given isomorphism $\Gamma_{\mathsf{D}} \circ F \cong \Gamma_{\mathsf{C}}$. □

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

We can now show how the correspondence works at the level of 2-cells.

Proposition 3.19. Suppose $\gamma, \delta : \mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$ are simulations, and suppose $\eta : \gamma_* \to \delta_*$ is a natural transformation. Then $\gamma \preceq \delta$ and $\eta = \mathbf{Asm}(\gamma \preceq \delta)$.

Proof. We first note that by Proposition 3.16(ii), it is automatic that $\Gamma_{\mathsf{D}} * \eta = \mathrm{id}_{\Gamma_{\mathsf{C}}}$.

For each $A \in |\mathsf{C}|$, let Z_A be the object of realisers as in the previous proof, and consider

$$\eta_{Z_A} : \gamma_*(Z_A) \to \delta_*(Z_A).$$

Suppose this is tracked by $t \in \mathsf{D}[\gamma A, \delta A]$. Since η_{Z_A} is just the identity on A, we have

$$a' \Vdash_{\gamma_*(Z_A)} a \;\Rightarrow\; \exists a''.\, t(a', a'') \wedge a'' \Vdash_{\delta_*(Z_A)} a.$$

That is, $\gamma(a, a')$ implies $\exists a''.\, t(a', a'') \wedge \delta(a, a'')$. In other words, t witnesses $\gamma \preceq \delta$ at A.

Again by Proposition 3.16(ii), it is automatic that

$$\Gamma_{\mathsf{D}} * \mathbf{Asm}(\gamma \preceq \delta) = \mathrm{id}_{\Gamma_{\mathsf{C}}}.$$

So

$$\Gamma_{\mathsf{D}} * \eta = \Gamma_{\mathsf{D}} * \mathbf{Asm}(\gamma \preceq \delta),$$

and because Γ_{D} is faithful, this means $\eta = \mathbf{Asm}(\gamma \preceq \delta)$. □

We now have everything needed for our generalised correspondence theorem, which to some extent validates our definition of $\mathcal{CSTRUCT}$.

Theorem 3.20. The 2-functor $\mathbf{Asm} : \mathcal{CSTRUCT} \to \Gamma\mathcal{QREG}$ is locally an equivalence.

Corollary 3.21. The C-structures C and D are equivalent in $\mathcal{CSTRUCT}$ if and only if there is an equivalence of categories

$$(F, G) : \mathbf{Asm}(\mathsf{C}) \simeq \mathbf{Asm}(\mathsf{D})$$

with

$$\Gamma_{\mathsf{D}} \circ F \cong \Gamma_{\mathsf{C}}$$
$$\Gamma_{\mathsf{C}} \circ G \cong \Gamma_{\mathsf{D}}.$$

Proof. The forwards implication is immediate from Theorem 3.17 (indeed, this yields an equivalence that commutes with Γ on the nose). For the reverse, it is easy to see that if F and G are as above, then F and G are quasi-regular functors and thus constitute an equivalence in $\Gamma\mathcal{QREG}$. It now follows from Theorem 3.20 that $\mathsf{C} \simeq \mathsf{D}$ in $\mathcal{CSTRUCT}$. □

4. Categorical structure in $\mathcal{CSTRUCT}$

In this section we investigate some of the categorical structure available in $\mathcal{CSTRUCT}$. Although our investigation here is far from exhaustive, the picture that emerges is that $\mathcal{CSTRUCT}$ is considerably richer in structure than either $\mathcal{PCA}$ or $\mathcal{TPCA}$.

If X and Y are sets of sets, we write $X \boxtimes Y$ for the set $\{A \times B \mid A \in X, B \in Y\}$.

Proposition 4.1. $\mathcal{CSTRUCT}$ has finite products enriched with respect to $\preceq$.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Proof. The terminal object $\mathbf{1}$ is the unique C-structure with $|\mathbf{1}| = \{\{*\}\}$. If C, D are C-structures, we define $\mathsf{C} \times \mathsf{D}$ by

$$|\mathsf{C} \times \mathsf{D}| = |\mathsf{C}| \boxtimes |\mathsf{D}|$$
$$(\mathsf{C} \times \mathsf{D})[A \times B, C \times D] = \mathsf{C}[A, C] \boxtimes \mathsf{D}[B, D].$$

It is clear that $\mathsf{C} \times \mathsf{D}$ is a C-structure, and there are evident projection simulations

$$\pi_{\mathsf{C}} : \mathsf{C} \times \mathsf{D} \longrightarrow\!\!\triangleright\, \mathsf{C}$$
$$\pi_{\mathsf{D}} : \mathsf{C} \times \mathsf{D} \longrightarrow\!\!\triangleright\, \mathsf{C}.$$

Given simulations

$$\gamma : \mathsf{E} \longrightarrow\!\!\triangleright\, \mathsf{C}$$
$$\delta : \mathsf{E} \longrightarrow\!\!\triangleright\, \mathsf{D},$$

we have a simulation

$$\langle \gamma, \delta \rangle : \mathsf{E} \longrightarrow\!\!\triangleright\, \mathsf{C} \times \mathsf{D}$$

given by

$$\langle \gamma, \delta \rangle A = \gamma A \times \delta A$$
$$\langle \gamma, \delta \rangle_A (e, (c, d)) \Leftrightarrow \gamma_A (e, c) \wedge \delta_A (e, d).$$

It is clear that this is the unique simulation such that

$$\pi_{\mathsf{C}} \circ \langle \gamma, \delta \rangle = \gamma$$
$$\pi_{\mathsf{D}} \circ \langle \gamma, \delta \rangle = \delta,$$

and $\langle \gamma, \delta \rangle \preceq \langle \gamma', \delta' \rangle$ if and only if $\gamma \preceq \gamma'$ and $\delta \preceq \delta'$. □

Proposition 4.2. $\mathcal{CSTRUCT}$ has finite sums, enriched with respect to $\preceq$.

Proof. The initial object $\mathbf{0}$ is the empty C-structure. As temporary notation, for any set A, we write A_i for $\{i\} \times A$. If C and D are C-structures, we define $\mathsf{C} + \mathsf{D}$ by

$$|\mathsf{C} + \mathsf{D}| = \{A_0 \mid A \in |\mathsf{C}|\} \cup \{B_1 \mid B \in |\mathsf{D}|\}$$
$$(\mathsf{C} + \mathsf{D})[A_0, B_0] \cong \mathsf{C}[A, B]$$
$$(\mathsf{C} + \mathsf{D})[A_1, B_1] \cong \mathsf{D}[A, B]$$
$$(\mathsf{C} + \mathsf{D})[A_i, B_j] = \varnothing \quad \text{if } i \neq j.$$

$\mathsf{C} + \mathsf{D}$ is clearly a C-structure, and there are evident inclusion simulations

$$\iota_{\mathsf{C}} : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{C} + \mathsf{D}$$
$$\iota_{\mathsf{D}} : \mathsf{D} \longrightarrow\!\!\triangleright\, \mathsf{C} + \mathsf{D}.$$

Given simulations

$$\gamma : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{E}$$
$$\delta : \mathsf{D} \longrightarrow\!\!\triangleright\, \mathsf{E},$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

there is evidently a unique simulation

$$[\gamma, \delta] : \mathsf{C} + \mathsf{D} \longrightarrow\!\triangleright \mathsf{E}$$

such that

$$[\gamma, \delta] \circ \iota_{\mathsf{C}} = \gamma$$
$$[\gamma, \delta] \circ \iota_{\mathsf{D}} = \delta,$$

and it is clear that $[\gamma, \delta] \preceq [\gamma', \delta']$ if and only if $\gamma \preceq \gamma'$ and $\delta \preceq \delta'$. □

Although both these results are trivial, Proposition 4.2 already gives us something not available in $\mathcal{PCA}$ or $\mathcal{TPCA}$. If A, B were TPCAs over some T, for instance, the evident disjoint union $A + B$ would not (canonically) be a TPCA since it would lack a type of operations mapping A_σ to B_τ (where $\sigma, \tau \in T$). Most probably, $\mathcal{PCA}$ and $\mathcal{TPCA}$ do not have categorical binary sums at all.

More interestingly, it turns out that $\mathcal{CSTRUCT}$ is very nearly cartesian closed: indeed, for any C-structures C, D, an 'exponential' C-structure D^{C} exists in a slightly weak sense, which is, nonetheless, sufficient to characterise D^{C} uniquely up to equivalence in $\mathcal{CSTRUCT}$. Let us begin by making the characteristic property of D^{C} precise.

Definition 4.3. A simulation $\gamma : \mathsf{C} \longrightarrow\!\triangleright \mathsf{D}$ is *single-valued* if for any $A \in |\mathsf{C}|$ and $a \in A$, there is a unique $a' \in \gamma A$ with $\gamma_A(a, a')$.

Definition 4.4. Suppose C, D are C-structures, and we are given a C-structure F along with a simulation $eval : \mathsf{F} \times \mathsf{C} \longrightarrow\!\triangleright \mathsf{D}$. We call $(\mathsf{F}, eval)$ a *near-exponential* for (C, D) if, for any C-structure E and simulation $\alpha : \mathsf{E} \times \mathsf{C} \longrightarrow\!\triangleright \mathsf{D}$, there is a *single-valued* simulation $\bar{\alpha} : \mathsf{E} \longrightarrow\!\triangleright \mathsf{F}$ such that $eval \circ (\bar{\alpha} \times \mathrm{id}_{\mathsf{C}}) = \alpha$, and, moreover, $\bar{\alpha}$ is, up to $\preceq\succeq$, the unique single-valued simulation with this property.

Because single-valued simulations compose and identity simulations are single-valued, it is easy to see that a near-exponential for (C, D) is uniquely determined up to equivalence in $\mathcal{CSTRUCT}$.

The rest of this section will be devoted to the proof of the following theorem.

Theorem 4.5. $\mathcal{CSTRUCT}$ possesses a near-exponential $(\mathsf{D}^{\mathsf{C}}, eval)$ for every (C, D).

The conceptual significance of this result is far from clear to us: if C and D represent 'models of computation', what on earth does it mean to consider D^{C} as a 'model of computation'? At present, the existence of near-exponentials stands as something of a curiosity, albeit a rather striking one that offers evidence for the mathematical richness of our framework.

From now on, we let C and D be fixed C-structures. We start with the definition of the C-structure D^{C}.

Definition 4.6.

(i) Given any function $U : |\mathsf{C}| \to |\mathsf{D}|$, we write $[\mathsf{C} \longrightarrow\!\triangleright_U \mathsf{D}]$ for the set of simulations $\gamma : \mathsf{C} \longrightarrow\!\triangleright \mathsf{D}$ such that $\gamma A = U(A)$ for all $A \in |\mathsf{C}|$.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

(ii) A family $F \subseteq [\mathsf{C} \longrightarrow\!\!\triangleright_U \mathsf{D}]$ of simulations is *uniformly tracked* if for all $A, B \in |\mathsf{C}|$ and $r \in \mathsf{C}[A, B]$, there exists $r' \in \mathsf{D}[U(A), U(B)]$ such that r' tracks r with respect to every $\gamma \in F$. That is,

$$\forall \gamma \in F.\ \ r(a,b) \wedge \gamma_A(a,a') \ \Rightarrow\ \exists b'.\, r'(a',b') \wedge \gamma_B(b,b').$$

Note that if F is uniformly tracked and inhabited, we can recover U from F; we write it as U_F and call it the *underlying type operation* of F.

(iii) If

$$F \subseteq [\mathsf{C} \longrightarrow\!\!\triangleright_U \mathsf{D}]$$
$$G \subseteq [\mathsf{C} \longrightarrow\!\!\triangleright_V \mathsf{D}]$$

are each uniformly tracked and inhabited, we say $R \subseteq F \times G$ is a *uniform transformation* if for all $A \in |\mathsf{C}|$, there exists $t \in \mathsf{D}[U(A), V(A)]$ such that for every $(\gamma, \delta) \in R$ we have that t witnesses $\gamma \preceq \delta$ at A. That is,

$$\forall (\gamma, \delta) \in R.\ \ \gamma_A(a,b) \ \Rightarrow\ \exists c.\, t(b,c) \wedge \delta_A(a,c).$$

(iv) We now define a C-structure D^{C} as follows:

— $|\mathsf{D}^{\mathsf{C}}|$ is the set of all uniformly tracked and inhabited families F of simulations $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$.

— For any $F, G \in |\mathsf{D}^{\mathsf{C}}|$, we have $\mathsf{D}^{\mathsf{C}}[F, G]$ is the set of uniform transformations $R \subseteq F \times G$.

Proposition 4.7. D^{C} is indeed a C-structure.

Proof. To see that D^{C} has (super)identities, given any $F \in |\mathsf{D}^{\mathsf{C}}|$, take $I \subseteq F \times F$ to be the identity relation. It is clear that I is a uniform transformation: for any $A \in |\mathsf{C}|$, let i be any superidentity on $U_F(A)$; then for any $\gamma \in F$, i witnesses $\gamma \preceq \gamma$ at A. So $I \in \mathsf{D}^{\mathsf{C}}[F, F]$.

To see that D^{C} has (super)composites, suppose

$$R \in \mathsf{D}^{\mathsf{C}}[F, G]$$
$$S \in \mathsf{D}^{\mathsf{C}}[G, H],$$

and let T be the relational composition $R; S$. We claim T is a uniform transformation. Given $A \in |\mathsf{C}|$, take:

— $t \in \mathsf{D}[U_F(A), U_G(A)]$ uniformly witnessing $\gamma \preceq \delta$ at A for all $(\gamma, \delta) \in R$;

— $u \in \mathsf{D}[U_G(A), U_H(A)]$ uniformly witnessing $\delta \preceq \epsilon$ at A for all $(\delta, \epsilon) \in S$.

Let $v \in \mathsf{D}[U_F(A), U_H(A)]$ be any supercomposite of t and u. It is then clear that v uniformly witnesses $\gamma \preceq \epsilon$ at A for all $(\gamma, \epsilon) \in T$. □

We must now equip D^{C} with a suitable evaluation morphism.

Definition 4.8. Let $eval : \mathsf{D}^{\mathsf{C}} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ be the simulation given as follows:

— The action on types is given by

$$eval(F \times A) = U_F(A).$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

— For each F and A, we define

$$eval_{F\times A} \subseteq (F \times A) \times U_F(A)$$

by

$$eval_{F\times A}((\gamma, a), b) \;\Leftrightarrow\; \gamma_A(a, b).$$

(This is a total relation since each γ_A is total.)

Proposition 4.9. *eval* : $\mathsf{D}^{\mathsf{C}} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is indeed a simulation.

Proof. We wish to show that every relation $R \times r$ in

$$(\mathsf{D}^{\mathsf{C}} \times \mathsf{C})[F \times A, G \times B] = \mathsf{D}^{\mathsf{C}}[F, G] \boxtimes \mathsf{C}[A, B]$$

is tracked by some

$$u \in \mathsf{D}[U_F(A), U_G(B)]$$

with respect to *eval*. Let

$$s \in \mathsf{D}[U_F(A), U_F(B)]$$

track r uniformly with respect to all $\gamma \in F$, and let

$$t \in \mathsf{D}[U_F(B), U_G(B)]$$

witness $\gamma \preceq \delta$ at B uniformly for all $(\gamma, \delta) \in R$. We now let u be a supercomposite of s and t.

To see that u tracks $R \times r$ with respect to *eval*, we suppose $((\gamma, a), (\delta, b)) \in (R \times r)$ and $eval((\gamma, a), a')$ – in other words, $(\gamma, \delta) \in R$, $(a, b) \in r$ and $\gamma_A(a, a')$. We require $b' \in \delta B$ such that $u(a', b')$ and $eval((\delta, b), b')$ – that is, $\delta_B(b, b')$. Since s tracks r with respect to γ, we can find b'' such that $s(a, b'')$ and $\gamma_B(b, b'')$. And since t witnesses $\gamma \preceq \delta$ at B, we can find $t(b'', b')$ and $\delta_B(b, b')$. This gives $u(a', b')$ as required. □

It remains to verify that $(\mathsf{D}^{\mathsf{C}}, eval)$ is a near-exponential for (C, D). From now on, we let E be some fixed C-structure and $\alpha : \mathsf{E} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ some fixed simulation.

For any $E \in |\mathsf{E}|$ and any $e \in E$, we write α_e for the simulation $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ defined by

$$\alpha_e A = \alpha(E \times A)$$
$$\alpha_{eA}(c, d) \;\Leftrightarrow\; \alpha_{E\times A}((e, c), d)$$

To see that α_e is a simulation, suppose $r \in \mathsf{C}[A, B]$ and let i be a superidentity in $\mathsf{E}[E, E]$. Then any $t \in \mathsf{D}[\alpha(E \times A), \alpha(E \times B)]$ that tracks $i \times r$ with respect to α also tracks r with respect to α_e. Since such a t may be chosen independently of e, the family $\{\alpha_e \mid e \in E\}$ is uniformly tracked. This justifies the following definition.

Definition 4.10. Let $\bar{\alpha} : \mathsf{E} \longrightarrow\!\!\triangleright \mathsf{D}^{\mathsf{C}}$ be defined as follows (we call $\bar{\alpha}$ the *transpose* of α):

— For each $E \in |\mathsf{E}|$, let $\bar{\alpha}E = \{\alpha_e \mid e \in E\} \in |\mathsf{D}^{\mathsf{C}}|$.
— For $E \in |\mathsf{E}|$ and $e \in E$, take $\bar{\alpha}_E(e, \gamma)$ if and only if $\gamma = \alpha_e$.

Proposition 4.11. $\bar{\alpha}$ is a single-valued simulation, and $eval \circ (\bar{\alpha} \times \mathrm{id}_{\mathsf{C}}) = \alpha$ on the nose.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Proof. That $\bar{\alpha}$ is single-valued is immediate from the definition. To see that $\bar{\alpha}$ is a simulation, suppose $r \in \mathsf{E}[E, F]$, and let $R = \{(\alpha_e, \alpha_f) \mid (e, f) \in r\}$. We first show that R is a uniform transformation. Given any $A \in |\mathsf{C}|$, let $i \in \mathsf{C}[A, A]$ be a superidentity, and take $r' \in \mathsf{D}[\alpha(E \times A), \alpha(F \times A)]$ tracking $r \times i$ with respect to α. We claim that for any $(e, f) \in r$, we have r' tracks $\alpha_e \preceq \alpha_f$ at A. To see this, suppose $\alpha_e(c, d)$. Then $\alpha((e, c), d)$, so there exists g with $r(d, g)$ and $\alpha((f, c), g)$, and thus $\alpha_f(c, g)$, so $R \in \mathsf{D}^{\mathsf{C}}[\bar{\alpha}E, \bar{\alpha}F]$.

Next we claim that R tracks r with respect to $\bar{\alpha}$. To see this, suppose $r(e, f)$ and $\bar{\alpha}(e, \gamma)$. Then $\gamma = \alpha_e$, so, taking $\delta = \alpha_f$, we have $\bar{\alpha}(f, \delta)$ and $R(\gamma, \delta)$.

To see that $eval \circ (\bar{\alpha} \times \mathrm{id}_{\mathsf{C}}) = \alpha$, note that if $E \in |\mathsf{E}|$, $C \in |\mathsf{C}|$ and $D = \alpha(E \times C)$, then

$$\begin{aligned}(eval \circ (\bar{\alpha} \times \mathrm{id}_{\mathsf{C}}))_{E \times C}((e, c), d) &\Leftrightarrow eval_{\bar{\alpha}E \times C}((\alpha_e, c), d) \\ &\Leftrightarrow \alpha_{eC}(c, d) \\ &\Leftrightarrow \alpha_{E \times C}((e, c), d).\end{aligned}$$

□

This already establishes that $\mathcal{CSTRUCT}$ is at least *weakly cartesian closed*. To conclude our proof that $\mathcal{CSTRUCT}$ has near-exponentials, we need the following proposition.

Proposition 4.12. Suppose $\bar{\alpha}' : \mathsf{E} \longrightarrow\!\!\triangleright \mathsf{D}^{\mathsf{C}}$ is any single-valued simulation such that $eval \circ (\bar{\alpha}' \times \mathrm{id}_{\mathsf{C}}) = \alpha$. Then $\bar{\alpha} \preceq \bar{\alpha}' \preceq \bar{\alpha}$.

Proof. Suppose $\bar{\alpha}'$ is as above. For any $E \in |\mathsf{E}|$ and $e \in E$, we write α'_e for the unique simulation $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ such that $\bar{\alpha}'_E(e, \alpha'_e)$. Then, as in the proof of Proposition 4.11, we have $\alpha'_{eC}(c, d)$ if and only if $\alpha_{eC}(c, d)$. So $\alpha'_e = \alpha_e$, though $\bar{\alpha}E$ and $\bar{\alpha}'E$ may be different. However, we have shown that $\bar{\alpha}'(e, \alpha_e)$ for each e, so we know that

$$\bar{\alpha}E = \{\alpha_e \mid e \in E\} \subseteq \bar{\alpha}'E.$$

So we may consider $\bar{\alpha}E \times \bar{\alpha}E$ as a subset of $\bar{\alpha}E \times \bar{\alpha}'E$, or of $\bar{\alpha}'E \times \bar{\alpha}E$. In either case, $\bar{\alpha}E \times \bar{\alpha}E$ is a uniform transformation, being witnessed by superidentities everywhere, and thus it is an element of both $\mathsf{D}^{\mathsf{C}}[\bar{\alpha}E, \bar{\alpha}'E]$ and $\mathsf{D}^{\mathsf{C}}[\bar{\alpha}'E, \bar{\alpha}E]$.

The first of these witnesses $\bar{\alpha} \preceq \bar{\alpha}'$ at E, and the second witnesses $\bar{\alpha}' \preceq \bar{\alpha}$ at E. So, since $E \in |\mathsf{E}|$ was arbitrary, we have $\bar{\alpha} \preceq \bar{\alpha}' \preceq \bar{\alpha}$. □

This completes the proof of Theorem 4.5. However, it is worth noting that a little more may be said about the morphism $\bar{\alpha}$.

Proposition 4.13. Suppose $\bar{\alpha}' : \mathsf{E} \longrightarrow\!\!\triangleright \mathsf{D}^{\mathsf{C}}$ is any simulation (not necessarily single-valued) such that $eval \circ (\bar{\alpha}' \times \mathrm{id}_{\mathsf{C}}) = \alpha$. Then $\bar{\alpha}' \preceq \bar{\alpha}$.

Proof. Suppose $\bar{\alpha}'$ is as above. Given $E \in |\mathsf{E}|$ and $e \in \mathsf{E}$, let ϵ be any simulation $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ such that $\bar{\alpha}'_E(e, \epsilon)$. Then we have the following sequence of implications:

$$\begin{aligned}\epsilon_C(c, d) &\Rightarrow eval_{\bar{\alpha}'E \times C}((\epsilon, c), d) \\ &\Rightarrow (eval \circ (\bar{\alpha}' \times \mathrm{id}_{\mathsf{C}}))_{E \times C}((e, c), d) \\ &\Rightarrow \alpha_{E \times C}((e, c), d) \\ &\Rightarrow \alpha_{eC}(c, d).\end{aligned}$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

So at any $C \in |\mathsf{C}|$, we have $\epsilon \preceq \alpha_e$ is witnessed at C by a superidentity on $\alpha(E \times C)$, independently of e. So let $R \subseteq \alpha' E \times \alpha E$ be the relation

$$\{(\epsilon, \alpha_e) \mid e \in E, (e, \epsilon) \in \bar{\alpha}'_E\}.$$

Then R is a uniform transformation, and it witnesses $\bar{\alpha}' \preceq \bar{\alpha}$ at E. Since E is arbitrary, we have $\bar{\alpha}' \preceq \bar{\alpha}$. □

We remark in passing that even for relatively simple C and D, the structure of D^{C} may be wildly intractable. For instance, if K_1 denotes Kleene's first model (the PCA of natural numbers with Kleene application), then $K_1^{K_1}$ would appear to be at least as complicated as the lattice of Turing degrees, and probably much worse. A possibility for a radically cut-down version of our near-exponentials will be briefly mentioned in Section 6.2.

Finally, it is natural to ask whether our near-exponential construction can be adapted to yield genuine exponentials in some suitable subcategory. Indeed (as suggested by the anonymous referee), one can achieve something of this kind by working in the 2-category of C-structures and *single-valued* simulations. In this case, one should of course define $|\mathsf{D}^{\mathsf{C}}|$ to consist only of uniformly tracked and inhabited families of single-valued simulations. Once this is done, the evaluation morphism is indeed single-valued, as is the transpose $\bar{\alpha}$ of a single-valued morphism α. However, this still only makes D^{C} a 'pseudo-exponential' since the uniqueness property of $\bar{\alpha}$ holds only up to $\preceq\succeq$. (Of course, one can then get a category with genuine exponentials by quotienting modulo $\preceq\succeq$ if so desired.) Moreover, the possible conceptual significance of this 'single-valued exponential' is at present no clearer to us than that of the full one.

5. Some subcategories of $\mathcal{CSTRUCT}$

There are many natural subcategories of $\mathcal{CSTRUCT}$ one might consider. Longley (1995, Chapter 2) showed how certain specific properties of PCAs and applicative morphisms were reflected by categorical properties of the corresponding categories and functors. In this section we will show how a similar, and somewhat richer, story may be told for C-structures and simulations.

Sections 5.1–5.3 explore some simple properties of C-structures and simulations that do not involve any additional structure on the class of datatypes. In Section 5.1, we investigate the alternative treatment of non-determinism alluded to in Remark 3.6(ii) – this leads us to the 2-category of *tight* C-structures and tight simulations between them. In Section 5.2, we consider the important subclass of *deterministic* C-structures (those in which all operations are partial functions) – here the tight/non-tight distinction vanishes completely. In Section 5.3, we revisit two properties of simulations that played an important role in Longley (1995, Chapter 2), namely *discreteness* and *projectivity* (here called *univalence*). As the culmination of this section, we obtain a useful property of *equivalences* between C-structures, *viz*. that they can always be taken to consist of discrete, univalent simulations.

In Sections 5.4 and 5.5, we discuss some C-structures whose class of datatypes carries some additional structure. In Section 5.4 we briefly consider C-structures equipped with

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

product types. This is a necessary preliminary to Section 5.5, where we develop the theory of *higher-order* C-structures. It is here that we make good on our earlier promise and show how the theory of TPCAs may be recovered as a special case of our more general theory: roughly speaking, deterministic higher-order C-structures correspond closely to *relative* TPCAs.

5.1. *Tightness*

We begin by exploring the alternative approach to non-determinism mentioned earlier. The basic idea here is that if a relation r' is meant to 'simulate' some relation r, it is reasonable to require that r' admits only as much non-determinism as r does – a simulation with this property is said to be *tight*. As we shall see, a satisfactory analogue of our entire theory may be obtained by consistently 'tightening' all our definitions, and whilst this involves a little more technical detail, in some respects, this alternative theory enjoys more pleasant properties than the original. (Our use of the term 'tight' is similar in spirit to, though technically different from, its use in Cockett and Hofstra (2010).)

Definition 5.1.

(i) A C-structure C is said to be *tight* if:

— for any $A \in |\mathsf{C}|$, the identity relation id_A is present in $\mathsf{C}[A, A]$;

— for any $r \in \mathsf{C}[A, B]$ and $s \in \mathsf{C}[B, C]$, there exists $t \in \mathsf{C}[A, C]$ such that

$$a \in \mathrm{dom}(r; s) \;\Rightarrow\; (t(a, c) \;\Leftrightarrow\; \exists b.\, r(a, b) \wedge s(b, c)).$$

(ii) A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$ is *tight* if every $r \in \mathsf{C}[A, B]$ is *tightly tracked* by some $r' \in \mathsf{D}[\gamma A, \gamma B]$: that is, r' tracks r in the usual sense and

$$r(a, b) \wedge \gamma(a, a') \wedge r'(a', b') \;\Rightarrow\; \gamma(b, b').$$

(iii) Given $\gamma, \delta : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$, we say γ is *tightly transformable* to δ, and write $\gamma \preceq_t \delta$, if for each $A \in |\mathsf{C}|$ there exists $t \in \mathsf{D}[\gamma A, \delta A]$ such that

$$\gamma_A(a, a') \;\Rightarrow\; (\exists a''.\, t(a', a'')) \wedge (\forall a''.\, t(a', a'') \Rightarrow \delta_A(a, a'')).$$

A tight C-structure still need not be closed under relational composition since in part (i) of the above definition the domain of t may be larger than that of $r; s$. Whilst in principle we could consider tight simulations between non-tight C-structures, there would appear to be little merit in doing so. It is routine to check that tight C-structures, tight simulations and tight transformations between them constitute a preorder-enriched category, which we denote by $\mathcal{CSTRUCT}_t$. We say C, D are *tightly equivalent*, and write $\mathsf{C} \simeq_t \mathsf{D}$, if they are equivalent in $\mathcal{CSTRUCT}_t$.

We may also apply the concept of tightness to morphisms of assemblies.

Definition 5.2. A morphism $f : X \to Y$ in $\mathbf{Asm}(\mathsf{C})$ is said to be *tight* if it is *tightly tracked* by some $r \in \mathsf{C}[A_X, A_Y]$: that is, r tracks f in the usual sense, and

$$a \Vdash_X x \wedge r(a, b) \;\Rightarrow\; b \Vdash_Y f(x).$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

If C is tight, it is easy to check that assemblies and tight morphisms constitute a wide subcategory of $\mathbf{Asm}(\mathsf{C})$, which we denote by $\mathbf{Asm}_t(\mathsf{C})$. The relationship with tight simulations is given by the following proposition.

Proposition 5.3.

(i) Suppose C, D are tight C-structures. A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$ is tight if and only if $\gamma_* = \mathbf{Asm}(\gamma)$ maps tight morphisms to tight morphisms.

(ii) Suppose $\gamma, \delta : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$ are tight simulations between tight C-structures, and $\gamma \preceq \delta$. Then $\gamma \preceq_t \delta$ if and only if all components of $\mathbf{Asm}(\gamma \preceq \delta)$ are tight morphisms.

Proof.

(i) Suppose γ is tight. If r tightly tracks $f : X \to Y$ in $\mathbf{Asm}(\mathsf{C})$, and r' tightly tracks r with respect to γ, it is easy to check that r' tightly tracks $\gamma_*(f)$ in $\mathbf{Asm}(\mathsf{D})$. Conversely, suppose γ_* preserves tight morphisms, and suppose $r \in \mathsf{C}[A, B]$.
We define assemblies X, Y as follows:

$$\begin{aligned} |X| &= |Y| = \operatorname{dom} r \\ A_X &= A \\ A_Y &= B \\ a' \Vdash_X a &\Leftrightarrow a' = a \\ b \Vdash_Y a &\Leftrightarrow r(a, b). \end{aligned}$$

Let $f : X \to Y$ be the identity function on $\operatorname{dom} r$. This is clearly tightly tracked by r itself. So $\gamma_*(f)$ is tightly tracked by some r', and it is easy to see that this r' tightly tracks r with respect to γ.

(ii) Let $\xi = \mathbf{Asm}(\gamma \preceq \delta)$. We first suppose $\gamma \preceq_t \delta$ and consider the morphism ξ_X for an arbitrary assembly $X \in \mathbf{Asm}(\mathsf{C})$. Take $t \in \mathsf{D}[\gamma A_X, \delta A_X]$ that tightly witnesses $\gamma \preceq \delta$ at A_X. Then t tightly tracks ξ_X. Conversely, suppose every ξ_X is tight, and consider an arbitrary $A \in |\mathsf{C}|$. Let Z_A be the object of realisers given by

$$\begin{aligned} |Z_A| &= A \\ A_{Z_A} &= A \\ \Vdash_{Z_A} &= \mathrm{id}_A, \end{aligned}$$

and suppose r tightly tracks ξ_{Z_A}. Then r tightly witnesses $\gamma \preceq \delta$ at A. □

In fact, by considering $\mathbf{Asm}_t(\mathsf{C})$ as a category in its own right, we may obtain a 'tight' analogue of our theory that makes no reference to $\mathbf{Asm}(\mathsf{C})$. If C is tight, then it is clear that all the subobject, quotient and copy projection morphisms appearing in the proof of Proposition 3.11 are tight, and it is easy to see that the relevant universal properties still hold within $\mathbf{Asm}_t(\mathsf{C})$. So $(\mathbf{Asm}_t(\mathsf{C}), \Gamma_{\mathsf{C}})$ is a quasi-regular category. Moreover, the above proposition shows that if γ is a tight simulation between tight C-structures, then $\mathbf{Asm}(\gamma)$ restricts to a functor

$$\mathbf{Asm}_t(\gamma) : \mathbf{Asm}_t(\mathsf{C}) \to \mathbf{Asm}_t(\mathsf{D}),$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

and that if $\gamma \preceq_t \delta$, then $\mathbf{Asm}(\gamma \preceq \delta)$ constitutes a natural transformation between such functors. We thus have a 2-functor

$$\mathbf{Asm}_t \ : \ \mathcal{CSTRUCT}_t \to \Gamma\mathcal{QREG}.$$

Furthermore, the following analogue of Theorem 3.20 holds.

Theorem 5.4. The 2-functor $\mathbf{Asm}_t$ is locally an equivalence. Hence $\mathsf{C} \simeq_t \mathsf{D}$ if and only if $\mathbf{Asm}_t(\mathsf{C})$ and $\mathbf{Asm}_t(\mathsf{D})$ are equivalent in $\Gamma\mathcal{QREG}$.

Proof. The proof of Theorem 3.20 goes through in the tight setting with some minor adjustments. □

The categorical structure investigated in Section 4 also carries over to $\mathcal{CSTRUCT}_t$. First, $\mathcal{CSTRUCT}_t$ clearly inherits the finite products and sums described in Propositions 4.1 and 4.2, so the inclusion $\mathcal{CSTRUCT}_t \to \mathcal{CSTRUCT}$ preserves this structure. Second, $\mathcal{CSTRUCT}_t$ possesses near-exponentials analogous to, but different from, those of $\mathcal{CSTRUCT}$, so these are not preserved by the inclusion. Specifically, to construct D^C within $\mathcal{CSTRUCT}_t$, we should take $|\mathsf{D}^\mathsf{C}|$ to be the set of all inhabited and uniformly *tightly* tracked families of simulations $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$, and $\mathsf{D}^\mathsf{C}[F, G]$ to be the set of uniform *tight* transformations $R \subseteq F \times G$. The proof of Theorem 4.5 then goes through *mutatis mutandis*.

A few minor gaps in this story should be noted. There does not seem to be a simple categorical characterisation of the tight assembly morphisms among those of $\mathbf{Asm}(\mathsf{C})$, nor of those functors that preserve tight morphisms. Similarly, the property that C is tight does not seem to be mirrored by any particular categorical property of $\mathbf{Asm}(\mathsf{C})$. Indeed, we conjecture that a tight C-structure may be equivalent in $\mathcal{CSTRUCT}$ to a non-tight one.

5.2. *Deterministic C-structures*

A particular class of C-structures that it is very natural to consider consists of those that model deterministic flavours of computation. We here show that this is a mathematically well-behaved class from the point of view of our theory.

Definition 5.5. A C-structure C is *deterministic* if for any $r \in \mathsf{C}[A, B]$, we have

$$r(a, b) \wedge r(a, b') \ \Rightarrow \ b = b'.$$

Thus, deterministic C-structures are those in which the 'relations' are partial functions. We write $r \cdot a$ for the unique b such that $r(a, b)$ if one exists. For such structures, the tight/non-tight distinction evaporates completely. For example, if D is deterministic, then not only is D tight, but every simulation $\mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is tight, as is every transformation $\gamma \preceq \delta$ where $\gamma, \delta : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$. Furthermore, we have the following result.

Proposition 5.6. A C-structure D is deterministic if and only if $\mathbf{Asm}(\mathsf{D}) = \mathbf{Asm}_t(\mathsf{D})$.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Proof. If D is deterministic and r tracks $f : X \to Y$ in $\mathbf{Asm}(\mathsf{D})$, then for any $a \Vdash_X x$, there is exactly one b with $r(a,b)$, so r tracks f tightly and $f \in \mathbf{Asm}_t(\mathsf{D})$. Conversely, if D is not deterministic, take $r \in \mathsf{D}[A,B]$ and $a \in A$, $b, b' \in B$ with $r(a,b)$, $r(a,b')$ and $b \neq b'$, and consider the assemblies X, Y defined by

$$\begin{aligned} |X| \;=\; |Y| \;&=\; \{b, b'\} \\ A_X &= A \\ A_Y &= B \\ a' \Vdash_X x \;&\Leftrightarrow\; a' = a \\ b'' \Vdash_Y y \;&\Leftrightarrow\; b'' = y. \end{aligned}$$

Then the identity map $X \to Y$ is tracked by r, but cannot be tightly tracked by any $r' \in \mathsf{D}[A,B]$. □

Deterministic C-structures form a full sub-2-category $\mathcal{CSTRUCT}_d$ of both $\mathcal{CSTRUCT}$ and $\mathcal{CSTRUCT}_t$. Moreover, the class of deterministic C-structures is closed under equivalences in $\mathcal{CSTRUCT}_t$.

Proposition 5.7. If D is deterministic and $\mathsf{C} \simeq \mathsf{D}$ in $\mathcal{CSTRUCT}_t$, then C is deterministic.

Proof. Clearly, an equivalence in $\mathcal{CSTRUCT}_t$ is also an equivalence in $\mathcal{CSTRUCT}$, so by the 2-functoriality of $\mathbf{Asm}_t$ and $\mathbf{Asm}$, if $\mathsf{C} \simeq \mathsf{D}$ in $\mathcal{CSTRUCT}_t$, then both

$$\mathbf{Asm}_t(\mathsf{C}) \simeq \mathbf{Asm}_t(\mathsf{D})$$

and

$$\mathbf{Asm}(\mathsf{C}) \simeq \mathbf{Asm}(\mathsf{D}).$$

Moreover, the equivalences are easily seen to commute with the inclusions

$$\mathbf{Asm}_t(-) \to \mathbf{Asm}(-).$$

But if D is deterministic, $\mathbf{Asm}_t(\mathsf{D})$ is the whole of $\mathbf{Asm}(\mathsf{D})$, so

$$\mathbf{Asm}_t(\mathsf{C}) = \mathbf{Asm}(\mathsf{C}),$$

and C is deterministic by Proposition 5.6. □

This says, in effect, that 'determinism' is indeed an invariant property of a computability notion with respect to tight equivalences. We do not know whether this is the case for equivalences in $\mathcal{CSTRUCT}$.

Deterministic C-structures are generally pleasant to work with since they allow many of the basic definitions to be significantly simplified. For instance, treating relations as partial functions, the notion 'r' tracks r with respect to γ' may be expressed as

$$\gamma_A(a,a') \wedge r(a)\downarrow \;\Rightarrow\; \gamma_B(r \cdot a, r' \cdot a'),$$

and 't witnesses $\gamma \preceq \delta$ at A' may be expressed as

$$\gamma_A(a,a') \;\Rightarrow \gamma_A(a, t \cdot a').$$

The definition of morphism in $\mathbf{Asm}(\mathsf{C})$ may be simplified similarly.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

The class of deterministic C-structures is clearly closed under finite products and sums. Unfortunately, they are not closed under the near-exponentials of Definition 4.6 (or their tight counterparts). However, it turns out that $\mathcal{CSTRUCT}_d$ is endowed with its own 'near-exponentials' in a somewhat weaker sense. Let us here say a simulation $\mathsf{E} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is *left-injective* if for any $E \in |\mathsf{E}|$, the simulations α_e for $e \in E$ are all distinct.

Theorem 5.8. If C, D are deterministic C-structures, there exist a deterministic C-structure F and a left-injective simulation $eval : \mathsf{F} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ with the property that for any *left-injective* $\alpha : \mathsf{E} \times \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$, there is a single-valued simulation $\bar{\alpha} : \mathsf{E} \longrightarrow\!\!\triangleright \mathsf{F}$ such that $eval \circ (\bar{\alpha} \times \mathrm{id}_{\mathsf{C}}) = \alpha$, and, moreover, $\bar{\alpha}$ is unique up to $\preceq\succeq$ among the simulations with this property.

Proof sketch. Take $|\mathsf{F}|$ to be the set of all inhabited, uniformly tracked families of simulations as before, and for $F, G \in |\mathsf{F}|$, define $\mathsf{F}[F, G]$ to be the set of *single-valued* uniform transformations $R \subseteq F \times G$. The rest of the argument then proceeds essentially as before – the significance of left-injectivity is that in the proof of Proposition 4.11, we need to know that if $r \in \mathsf{F}[E, F]$ is single-valued, then so is $R = \{(\alpha_e, \alpha_f) \mid (e, f) \in r\}$. □

Note that even this weaker universal property is still sufficient to characterise the C-structure F up to equivalence, so we may speak of the near-exponentiation of deterministic C-structures as a well-defined operation on deterministic notions of computability.

5.3. *Discrete and univalent morphisms*

Among the properties of simulations considered in Longley (1995) in the PCA setting were two properties called *discreteness* and *projectivity*. We here consider the analogues of both these properties, changing the name of the latter to *univalence*, for reasons to be explained. Each of these properties of simulations is naturally correlated with a property of assemblies.

Definition 5.9.

(i) A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is said to be *discrete* if $\gamma_A(a, b)$ and $\gamma_A(a', b)$ imply $a = a'$.
(ii) An assembly X over C is said to be *discrete* if $a \Vdash_X x$ and $a \Vdash_X x'$ imply $x = x'$. Such assemblies are also known as *modest sets*.

The following result is straightforward.

Proposition 5.10. A simulation γ is discrete if and only if $\mathbf{Asm}(\gamma)$ preserves discrete objects.

When C is tight, discrete objects may be characterised categorically within $\mathbf{Asm}_t(\mathsf{C})$ (whence the functors $\mathbf{Asm}_t(\gamma)$ arising from discrete simulations may also be characterised categorically). By a *singleton assembly*, we mean any assembly U such that $\Gamma(U)$ is a singleton set.

Proposition 5.11. Suppose C is tight. An assembly X over C is discrete if and only if for every singleton object U and set S, every morphism $U \propto S \to X$ in $\mathbf{Asm}_t(\mathsf{C})$ is constant.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Proof. The forwards implication is straightforward. For the converse, if X is not discrete, take $x \neq x'$ and a such that $a \Vdash_X x$ and $a \Vdash_X x'$. Let U be the singleton assembly with just the realiser a, let S be a two-element set and let $f : U \propto S \to X$ map the two elements of $|U \propto S|$ to x and x', respectively. Since C is tight, f is tightly tracked by the appropriate identity relation, and hence is a non-constant morphism of $\mathbf{Asm}_t(\mathsf{C})$. □

There does not appear to be a correspondingly simple characterisation of the discrete objects within $\mathbf{Asm}(\mathsf{C})$. It is also worth noting that the tight setting preserves a familiar feature from the classical theory of realizability: namely, that a morphism into a discrete assembly is uniquely determined by any tracker for it.

Two further facts about discrete simulations in the tight setting are worth recording. The proofs are straightforward.

Proposition 5.12.

(i) If $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ and $\delta : \mathsf{D} \longrightarrow\!\!\triangleright \mathsf{C}$ are tight simulations with $\delta \circ \gamma \preceq \mathrm{id}_{\mathsf{C}}$, then γ is discrete.
(ii) If $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is discrete and D is deterministic, then C is deterministic.

Combining the two halves of this yields a useful strengthening of Proposition 5.7.

Next we consider a property of simulations we shall call *univalence*. Although the name is perhaps not ideal, the intuition is that these are simulations that are 'essentially single-valued'. This notion comes in both non-tight and tight flavours. The notion of a single-valued simulation was defined in Definition 4.3. Similarly, we say an assembly X is single-valued if for each $x \in |X|$, there is a unique a with $a \Vdash_X x$.

Definition 5.13.

(i) A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is *univalent* (respectively, *tightly univalent*) if for some single-valued simulation γ' we have $\gamma \preceq\succeq \gamma'$ (respectively, $\gamma \preceq_t\succeq_t \gamma'$).
(ii) An assembly X over C is *univalent* (respectively, *tightly univalent*) if for some single-valued assembly X' we have $X \cong X'$ in $\mathbf{Asm}(\mathsf{C})$ (respectively, in $\mathbf{Asm}_t(\mathsf{C})$).

Proposition 5.14.

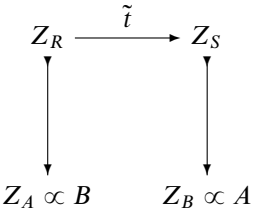
(i) $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is univalent if and only if $\mathbf{Asm}(\gamma)$ preserves univalent assemblies.
(ii) γ is tightly univalent if and only if $\mathbf{Asm}_t(\gamma)$ preserves tightly univalent assemblies.

Proof.

(i) The forwards implication is easy. For the reverse, suppose $\gamma_* = \mathbf{Asm}(\gamma)$ preserves univalent assemblies. For each $A \in |\mathsf{C}|$, let $Z_A \in \mathbf{Asm}(\mathsf{C})$ be the corresponding object of realisers, and choose $Y_A \in \mathbf{Asm}(\mathsf{D})$ to be single-valued such that $|Y_A| = A$ and $Y_A \cong \gamma_*(Z_A)$ through the identity on underlying sets. Defining $\gamma' : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ by $\gamma'_A(a, b) \Leftrightarrow b \Vdash_{Y_A} a$, we need to show that γ' is a simulation, and it will then be clear that γ' is single-valued and $\gamma \preceq\succeq \gamma'$.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Given $r \in \mathsf{C}[A, B]$, we construct the following diagram in **Asm**(C) as in the proof of Proposition 3.18:



We now apply γ_* to obtain a diagram Δ in **Asm**(D) and construct a diagram Δ' in **Asm**(D) isomorphic to Δ by replacing $\gamma_*(Z_A)$ and $\gamma_*(Z_B)$ by Y_A and Y_B, respectively, and then replacing the copies and subobjects in turn by the corresponding canonical ones in **Asm**(D). Let f be the morphism in Δ' corresponding to $\gamma_*(\tilde{t})$ in Δ, and take r' tracking f. It is easy to see that r' tracks r with respect to γ'.

(ii) The proof of part (ii) is similar. □

In contrast to the situation for discrete assemblies, it is the non-tight setting that seems more friendly to univalent assemblies. This is illustrated by the following categorical characterisation of univalent objects within **Asm**(C) (the proof involves the Axiom of Choice).

Proposition 5.15. An assembly X over C is univalent if and only if (in **Asm**(C)) for every morphism $f : X \to Z$ and every quotient morphism $h : Y \twoheadrightarrow X$ there exists $\bar{f} : X \to Y$ such that $h \circ \bar{f} = f$.

Proof. We first suppose X is univalent – we may as well also assume it is single-valued. Suppose f and h are as above with f tracked by r. We define $\bar{f}$ as follows. For each $x \in |X|$, choose some $a \in A_X$ and $b \in A_Z$ such that

$$a \Vdash_X x$$
$$b \Vdash_Z f(x)$$
$$r(a, b).$$

We then choose $y \in |Y|$ such that $h(y) = f(x)$ and $b \Vdash_Y y$, and set $\bar{f}(x) = y$. It is then clear that $\bar{f}$ is tracked by r and $h \circ \bar{f} = f$.

For the converse, we suppose X has the given quotient property and consider the assembly Y where $|Y|$ is the relation $\Vdash_X$ considered as a set of pairs (a, x), and $a' \Vdash_Y (a, x)$ if and only if $a' = a$. Let $h : Y \twoheadrightarrow X$ be the quotient map given by second projection (this is tracked by a superidentity), and let $f : X \to X$ be the identity morphism. We now take $\bar{f} : X \to Y$ such that $h \circ \bar{f} = f$. We consider the image of $\bar{f}$ as a subset of $|Y|$, and lift it to a subobject $s : X' \rightarrowtail Y$. So $\bar{f}$ factors through s, and we have an isomorphism $X \cong X'$ with X' single-valued. □

Remark 5.16. In the PCA setting, the quotient maps are precisely the regular epis, so the quotient property featuring in the above proposition is precisely the property of being a *(regular) projective*. For this reason, univalent simulations and assemblies were referred

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

to as *projective* in Longley (1995). However, we hesitate to adopt this terminology in our present setting, not only because, in general, not all quotients need be regular epis, but also because there does not appear to be an analogue of the above proposition for $\mathbf{Asm}_t(\mathsf{C})$.

The following useful fact generalises Longley (1995, Theorem 2.5.3(ii)).

Theorem 5.17. If D is deterministic and $\gamma : \mathsf{C} \longrightarrow\!\!\!\!\triangleright\ \mathsf{D}$ and $\delta : \mathsf{D} \longrightarrow\!\!\!\!\triangleright\ \mathsf{C}$ are simulations with $\delta \circ \gamma \preceq \mathrm{id}_{\mathsf{C}}$ and $\mathrm{id}_{\mathsf{D}} \preceq \gamma \circ \delta$, then δ is univalent.

Proof. For each $B \in |\mathsf{D}|$, we take $u_B \in \mathsf{D}[B, \gamma\delta B]$ tracking $\mathrm{id}_{\mathsf{D}} \preceq \gamma \circ \delta$ at B. Since D is deterministic, for each $b \in B$, there is a unique d_b with $u_B(b, d_b)$, and it satisfies $(\gamma\delta)_B(b, d_b)$. Moreover, there is a unique $c_b \in \delta_B$ with $\delta_B(b, c_b)$ and $\gamma_{\delta B}(c_b, d_b)$ since γ is discrete by Proposition 5.12. We now define $\delta' : \mathsf{D} \longrightarrow\!\!\!\!\triangleright\ \mathsf{C}$ by $\delta'_B(b, c) \Leftrightarrow c = c_b$ and show that δ, δ' are 'intertransformable' as relations, and hence that δ' is a simulation.

Clearly, $\delta'_B(b, c)$ implies $\delta_B(b, c)$. Conversely, if $\delta_B(b, c)$, we take $u'_B \in \mathsf{C}[\delta B, \delta\gamma\delta B]$ tracking u_B with respect to δ. Then for some $c' \in \delta\gamma\delta B$, we have $u'_B(c, c')$ and $\delta_{\gamma\delta B}(d_b, c')$, so $(\delta\gamma)_{\delta B}(c_b, c')$. We now take t_B tracking $\delta \circ \gamma \preceq \mathrm{id}_{\mathsf{C}}$ at δ_B, so we have $t_B(c', c_b)$. Thus, taking $v_B \in \mathsf{C}[\delta B, \delta' B]$ a supercomposite of u'_B and t_B, we have $v_B(c, c_b)$. It is now easy to see that δ' is a simulation since for any $r \in \mathsf{D}[A, B]$, we can take $r' \in \mathsf{C}[\delta A, \delta B]$ tracking r with respect to δ, and take $r'' \in \mathsf{C}[\delta' A, \delta' B]$ a supercomposite of r' and v_B (noting that $\delta' A = \delta A$) and it is then clear that r'' tracks r with respect to δ'. Moreover, $\delta' \preceq \delta$ is tracked at B by a superidentity, and $\delta \preceq \delta'$ is tracked at B by v_B. □

Corollary 5.18. If $\mathsf{C} \simeq \mathsf{D}$ in $\mathcal{CSTRUCT}_d$, both halves of the equivalence are discrete and univalent, so there is an equivalence $\mathsf{C} \simeq \mathsf{D}$ consisting of single-valued simulations.

In the light of Remark 3.6(ii), this shows that, for deterministic C-structures, our notion of equivalence agrees precisely with the *lax simulation equivalence* of Cockett and Hofstra (2010) (taking sets and partial functions as the base category). As shown in Longley (1995, Chapter 3), knowing that equivalences, when they exist, can be assumed to be single-valued is often helpful for proving the *inequivalence* of two models.

Note also that if C and D are *strict* deterministic C-structures, the above argument can be adapted to show that an equivalence consisting of strict simulations can be replaced by one consisting of strict single-valued simulations. The resulting notion thus agrees with the *strict simulation equivalence* of Cockett and Hofstra (2010).

5.4. *C-structures with products*

We now turn to consider C-structures in which the class of datatypes carries some additional structure.

We regard it as a matter of some pride that our theory so far has not required our C-structures to be endowed with *products* of any kind. Nevertheless, most naturally arising C-structures do have products of some kind, so it is natural to consider various possible notions of product structure. This is also a necessary prerequisite for the notion of a *higher-order* C-structure, which we shall investigate in Section 5.5.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

We opt here for a simple notion of product types with a relatively 'strict' flavour.

Definition 5.19. Suppose C is a C-structure.Then:

(i) We say C is *strict monoidal* if:

— $|\mathsf{C}|$ contains the singleton set $I = \{*\}$ and is closed under set-theoretic binary products.

— For any $r \in \mathsf{C}[A, B]$ and $s \in \mathsf{C}[C, D]$, the relation '$r \times s$' is present in $\mathsf{C}[A \times C, B \times D]$.

— The canonical bijections

$$(A \times B) \times C \cong A \times (B \times C)$$
$$I \times A \cong A$$
$$A \times I \cong A$$

are all present as relations in C in both directions.

(ii) We say C has *strict cartesian* if, in addition to the above, the canonical projection and diagonal maps

$$A \times B \to A$$
$$A \times B \to B$$
$$A \to A \times A$$

are present as relations in C.

Other more relaxed notions of monoidal or cartesian C-structure may be devised, but it is typically the case that any such C-structure will be equivalent to a strict one (we refrain from going into the details here). For example, all TPCAs (including untyped PCAs) are equivalent to strict cartesian C-structures, although they may not be such structures themselves.

Clearly, if C is strict monoidal (respectively, strict cartesian), then $\mathbf{Asm}(\mathsf{C})$ is a monoidal (respectively, cartesian) category (and similarly for $\mathbf{Asm}_t(\mathsf{C})$ when C is tight).

The relevant structure-respecting simulations for strict monoidal C-structures are given by the following definition.

Definition 5.20. Suppose C, D are strict monoidal. A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright\, \mathsf{D}$ is *monoidal* if:

— For any $A, B \in |\mathsf{C}|$ there is a relation $t \in \mathsf{D}[\gamma A \times \gamma B, \gamma(A \times B)]$ such that

$$\gamma_A(a, a') \wedge \gamma_B(b, b') \;\Rightarrow\; \exists c'.\, t((a', b'), c') \wedge \gamma_{A \times B}((a, b), c').$$

— There exists $u \in \mathsf{D}[I, \gamma I]$ such that $u(*, a)$ for some $a \Vdash_I *$.

Recall that if $(\mathcal{C}, \otimes, I)$ and $(\mathcal{D}, \otimes', I')$ are monoidal categories, a *lax monoidal functor* between them is a functor $F : \mathcal{C} \to \mathcal{D}$ together with a family of morphisms

$$\phi_{A,B} : F(A) \otimes' F(B) \to F(A \otimes B)$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

that are natural in A and B, and a morphism $\psi : I' \to F(I)$, such that certain coherence conditions involving the associativity and unit morphisms are satisfied. We record the following easy fact without proof.

Proposition 5.21. Suppose C and D are strict monoidal. A simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is *monoidal* if and only if $\mathbf{Asm}(\gamma)$ is lax monoidal.

Note also that if the products in C and D are cartesian, then a lax monoidal functor $\mathbf{Asm}(\gamma)$ is automatically cartesian (that is, preserves finite products).

We may write $\mathcal{CSTRUCT}_m$ for the preorder-enriched category of strict monoidal C-structures and monoidal simulations. We say C and D are *monoidally equivalent* ($\mathsf{C} \simeq_m \mathsf{D}$) if they are equivalent in $\mathcal{CSTRUCT}_m$ (it does not seem that $\mathsf{C} \simeq \mathsf{D}$ implies $\mathsf{C} \simeq_m \mathsf{D}$). The 2-category $\mathcal{CSTRUCT}_m$ has finite products as in $\mathcal{CSTRUCT}$, and it is routine to verify that a monoidal analogue of the construction of near-exponentials goes through. Note, however, that our simple construction of finite sums is not available in $\mathcal{CSTRUCT}_m$ since $\mathsf{C} + \mathsf{D}$ will not typically contain products $A \times B$ where $A \in |\mathsf{C}|$ and $B \in |\mathsf{D}|$.

Finally, note that analogous results hold in the 'tight' setting. If C and D are tight and strict monoidal, we may say a tight simulation $\gamma : \mathsf{C} \longrightarrow\!\!\triangleright \mathsf{D}$ is *tightly monoidal* if it satisfies the obvious tightened version of Definition 5.20. We then have that γ is tightly monoidal if and only if $\mathbf{Asm}_t(\gamma)$ is lax monoidal; such a functor is automatically cartesian if C and D are. The corresponding 2-category $\mathcal{CSTRUCT}_{mt}$ enjoys properties similar to $\mathcal{CSTRUCT}_m$.

5.5. *Higher-order C-structures*

With the machinery of products now in place, we may now identify a class of C-structures that model *higher-order* notions of computation, and thus retrieve the theory of TPCAs as a special case of our theory. The key idea here is that a C-structure is higher order if its computable operations can themselves be represented as values of an appropriate type.

Definition 5.22. A *higher-order C-structure* is a strict monoidal C-structure C equipped with the following: for each $A, B \in |\mathsf{C}|$, a datatype $(A \Rightarrow B) \in |\mathsf{C}|$ and an operation $app_{AB} \in \mathsf{C}[(A \Rightarrow B) \times A, B]$ such that:

— For any $r \in \mathsf{C}[C \times A, B]$ there exists a total relation $\bar{r} \in \mathsf{C}[C, (A \Rightarrow B)]$ with $r \subseteq (\bar{r} \times \mathrm{id}_A); app_{AB}$.

It is crucial here that $\bar{r}$ is not required to be unique, hence we may say app_{AB} enjoys a 'weak universal property'.

It is not hard to show that if C is a C-structure (respectively, cartesian C-structure), then $\mathbf{Asm}(\mathsf{C})$ is monoidal closed (respectively, cartesian closed). Thus, as is often the case in realizability, the **Asm** construction turns a weak universal structure into a strong one.

Any TPCA C may be turned into a deterministic, higher-order cartesian C-structure $\mathbf{CStr}(C)$ as follows. First, it is easy to construct some C' equivalent to C (within $\mathcal{TPCA}$) in which product types are genuine set-theoretic products, and in which a unit type exists.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Such a C' may then be viewed as a C-structure $\mathbf{CStr}(C)$ as in Example 3.3(i). Clearly, $\mathbf{CStr}(C)$ is deterministic and cartesian, and for any types A, B, the element

$$\lambda^* p.\,(fst\ p)(snd\ p)\ \in\ (((A \Rightarrow B) \times A) \Rightarrow B)$$

induces a computable operation

$$app_{AB} : (A \Rightarrow B) \times A \rightharpoonup B$$

with the required weak universal property. (We assume familiarity here with the λ^* notation for combinators – for details, see, for example, Longley (1995, Chapter 1).)

In one important respect, however, deterministic higher-order cartesian C-structures are significantly more general than TPCAs. In C-structures of the form $\mathbf{CStr}(C)$, every element a of any $A \in |\mathsf{C}|$ is picked out by some operation in $\mathsf{C}[I, A]$ (we may call such elements *computable* since the intuition is that mappings in $\mathsf{C}[I, A]$ are 'computable operations'). In general, however, this need not be the case – rather, the elements picked out by such operations will constitute a sub-C-structure of C in an evident sense. This amounts to the observation that deterministic higher-order cartesian C-structures naturally embrace the concept of *relative realizability* (see, for example, Birkedal and van Oosten (2002)). Informally, the idea behind relative realizability is to consider (for instance) a TPCA A equipped with a sub-TPCA $A^\sharp$ of elements we regard as 'computable'. Under certain strictness assumptions[†], it can be shown that deterministic higher-order cartesian C-structures correspond exactly to TPCAs equipped with such a choice of sub-TPCA.

In any case, this convergence between TPCAs and certain C-structures suggests how the notion of an applicative morphism for TPCAs (as in Section 2.2) might be generalised to arbitrary higher-order C-structures as follows.

Definition 5.23. Suppose C and D are higher-order C-structures. An *applicative morphism* $\gamma : \mathsf{C} \to \mathsf{D}$ consists of a mapping $A \mapsto \gamma A : |\mathsf{C}| \to |\mathsf{D}|$ and a family of total relations $\gamma_A \subseteq A \times \gamma A$ (for $A \in |\mathsf{C}|$) satisfying the following conditions:

(1) For every computable element $a \in A$ (where $A \in |\mathsf{C}|$), there is some computable $a' \in \gamma A$ such that $\gamma(a, a')$.

(2) 'Application in C is tracked in D'. That is, for any $A, B \in |\mathsf{C}|$, we may choose

$$app_{AB} \in \mathsf{C}[(A \Rightarrow B) \times A, B]$$

as in Definition 5.22 such that for some

$$app'_{AB} \in \mathsf{D}[\gamma(A \Rightarrow B) \times \gamma A, \gamma B],$$

we have

$$\gamma_{(A \Rightarrow B)}(f, f') \wedge \gamma_A(a, a') \wedge app_{AB}((f, a), b)\ \Rightarrow\ \exists b'.\,\gamma_B(b, b') \wedge app'_{AB}((f', a'), b').$$

[†] Specifically, *strict* TPCAs endowed with such a sub-TPCA correspond precisely to deterministic higher-order C-structures in which the operation $\bar{r}$ satisfies $r = (\bar{r} \times \mathrm{id}_A); app_{AB}$.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

If C and D are TPCAs, it is easy to see that applicative morphisms $\mathbf{CStr}(C) \to \mathbf{CStr}(D)$ in the above sense correspond exactly to applicative morphisms $C \longrightarrow\!\!\!\!\triangleright\, D$ in the sense of Section 2.2. Indeed, if $r \in D_{\gamma(A \Rightarrow B) \Rightarrow \gamma A \Rightarrow \gamma B}$ tracks $\cdot_{AB}$ in the TPCA sense, then

$$\lambda^* p.\, r(fst\ p)(snd\ p)$$

induces a suitable operation

$$app'_{AB} \in \mathbf{CStr}(D)[(A \Rightarrow B) \times A, B].$$

Conversely, any such operation app'_{AB} is induced by some $r' \in D_{\gamma(A \Rightarrow B) \times \gamma A \Rightarrow \gamma B}$, and then $\lambda^* f a.\, r'(pair\ f\ a)$ tracks $\cdot_{AB}$ in the TPCA sense. So Definition 5.23 is a natural and direct generalisation of the TPCA notion of applicative morphism.

The connection with simulations is now given by the following proposition.

Proposition 5.24. If C and D are higher-order C-structures, then the applicative morphisms from C to D are exactly the monoidal simulations $\mathsf{C} \longrightarrow\!\!\!\!\triangleright\, \mathsf{D}$.

Proof. We first suppose that γ is a monoidal simulation. To see that condition (1) of Definition 5.23 is satisfied, note that if $a \in A$ is picked out by $r \in \mathsf{C}[I, A]$ and r is tracked by $r' \in \mathsf{D}[\gamma I, \gamma A]$, then, by supercomposing r with the operation $u \in \mathsf{D}[I, \gamma I]$ from Definition 5.20, we get an operation in $\mathsf{D}[I, \gamma A]$ that picks out a γ-realiser for a. For condition (2), given any $A, B \in |\mathsf{C}|$, we have an operation

$$t \in \mathsf{D}[\gamma(A \Rightarrow B) \times \gamma A,\ \gamma((A \Rightarrow B) \times A)]$$

as in Definition 5.20, and an operation

$$u \in \mathsf{D}[\gamma((A \Rightarrow B) \times A), \gamma B]$$

tracking the operation app_{AB} in C. Supercomposing these gives an operation

$$app'_{AB} \in \mathsf{D}[\gamma(A \Rightarrow B) \times \gamma A, \gamma B]$$

with the required properties.

Conversely, suppose γ is an applicative morphism. Given any $r \in \mathsf{C}[A, B]$, we supercompose with the bijection in $\mathsf{C}[I \times A, A]$ to obtain $r' \in \mathsf{C}[I \times A, B]$, then choose $\bar{r} \in \mathsf{C}[I, (A \Rightarrow B)]$ as in Definition 5.22. Let $f \in (A \Rightarrow B)$ be any computable element picked out by $\bar{r}$. Then, by condition (1) of Definition 5.23, we may find $f' \in \gamma(A \Rightarrow B)$ picked out by some $t \in \mathsf{D}[I, \gamma(A \Rightarrow B)]$ such that $\gamma_{(A \Rightarrow B)}(f, f')$. By supercomposing the canonical operation in $\mathsf{D}[A, I \times A]$ with

$$t \times i \in \mathsf{D}[I \times A, (A \Rightarrow B) \times A],$$

and then with app'_{AB}, we obtain an operation in $\mathsf{D}[A, B]$ that tracks r. □

The basic idea here – *viz.* that if the application operations are tracked then so are all operations – is one that also plays an important role in Cockett and Hofstra (2008).

Under the above correspondence, it is easy to see that the preorder $\preceq$ on monoidal simulations agrees precisely with the one on applicative morphisms as in Section 2.2. Thus, the **CStr** construction outlined above extends to a 2-functor

$$\mathbf{CStr} : \mathcal{TPCA} \to \mathcal{CSTRUCT}_m$$

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

that is locally an equivalence. (Hence two TPCAs A, B are equivalent if and only if $\mathbf{CStr}(A), \mathbf{CStr}(B)$ are equivalent as monoidal C-structures.) Moreover, as noted earlier, the **Asm** construction on TPCAs agrees with that on C-structures modulo this inclusion.

The observation that the notion of an applicative morphism is in essence a 'first-order' one highlights a significant philosophical point. One might have supposed that two models of higher-order computation could be equivalent as 'flattened' first-order models but differ non-trivially on account of the different higher-order structures they carried. Proposition 5.24 shows that this is not the case, at least within our present framework: the equivalence of higher-order models is simply their equivalence *qua* first-order models.

Note, however, that the class of higher-order C-structures appears to have poor closure properties by comparison with that of all C-structures. Not only are higher-order C-structures not closed under finite sums, but they do not appear to admit an analogue of our near-exponential construction. Curiously, then, it is the shift from higher-order to first-order structures that allows 'higher-order' phenomena to appear at the framework level.

Finally, a brief remark on the 'tight' counterpart of the above results is in order here. The above definition of a higher-order C-structure is still appropriate in the tight setting, but there is an obvious tight analogue of Definition 5.23 in the case where C and D are tight higher-order C-structures. The analogue of Proposition 5.24 then goes through: the tight applicative morphisms from C to D are exactly the tightly monoidal simulations as discussed at the end of Section 5.4. Of course, the ordinary and tight versions of these results are equally good as generalisations of the theory of TPCAs since the latter fall within the realm of deterministic C-structures.

6. Conclusions and further work

6.1. *Summary of achievements*

In this paper we have presented a broad framework for the study of models of computation and of simulations between them, and shown that the 'category of assemblies' construction and associated correspondence theorem generalise smoothly to this setting. We have also shown that our 2-category of models possesses some non-trivial mathematical structure (most notably a slightly relaxed form of exponentiation), as well as a number of well-behaved subcategories corresponding to various properties of computational interest. Whilst our investigation of this structure has been far from exhaustive, we believe that our results sufficiently demonstrate the richness and fertility of our proposed framework.

We have also made a tentative start on showing how a wide range of existing computation models not previously studied in a realizability context might fit naturally within our framework. In particular, in Section 3 we showed how a very general class of 'syntactic models' can be regarded as C-structures, and presented one concrete and substantial example of a simulation embodying Milner's translation from lazy λ-calculus to π-calculus. However, a much broader and deeper investigation would be required to clarify the precise extent of applicability of our framework.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Along the way, we have compared our framework with other possible candidates for a general framework for computability models and simulations, such as those proposed by Cockett and Hofstra (Cockett and Hofstra 2008; Cockett and Hofstra 2010). Some of the differences are rather technical in nature, and we have not arrived at any definitive arguments in favour of one approach over another: indeed, it may be that we are still far from the end of the story with regard to finding the 'right' level of generality for these ideas. Nevertheless, the broad picture that emerges is that our category $\mathcal{CSTRUCT}$ offers a more general setting than other current contenders (which can be naturally viewed as subcategories of it) – with the notable exception of its 'ordered' counterpart (*cf.* van Oosten (1997) and Hofstra (2006)), which would appear to be a straightforward structural enrichment of our framework. It remains open whether this ordered version admits any additional examples of computational interest.

Finally, we have shown how the higher-order framework represented by the theory of TPCAs can be naturally subsumed within a first-order framework. Indeed, it now appears that higher-order structure plays a less vital role than we previously supposed, since the same notion of (monoidal) simulation, and hence of equivalence, is equally appropriate in both the first-order and higher-order settings. Combined with the observation that the class of higher-order C-structures does not enjoy particularly good closure properties, we are led to the view that higher-order models should really be regarded as nothing more than first-order models that happen to satisfy a certain completeness condition – this provides a moral of some significance for our general programme of mapping out computability notions.

6.2. *Further work*

It is likely that there is further categorical structure to be uncovered within $\mathcal{CSTRUCT}$. There are certainly other subcategories of interest that we have not discussed here. For example, it is natural to look at C-structures that contain a pair of elements playing the role of the booleans. The relevant boolean-respecting simulations between such structures correspond to what are called *decidable* morphisms in Longley (1995)[†]. The subcategory of such simulations would itself appear to have reasonable structure. Moreover, it admits a cut-down version of the near-exponential construction, which in typical cases yields a much more tractable structure than the one in $\mathcal{CSTRUCT}$ (for instance, the near-exponential $K_1^{K_1}$ in the category of boolean-respecting simulations is the one-element C-structure). Similarly, we could restrict to C-structures that contain a good representation of the natural numbers, and simulations that respect them (these play an important role in Longley (2007)).

Beyond this, the main outstanding task is to populate our abstract framework with some interesting examples (outside the domain of higher-order C-structures, which has already received significant attention), and to investigate more fully the extent of its applicability. It seems to us that the realm of *process calculi* offers a suitable domain in

[†] van Oosten (2008) observed that the name 'decidable' was not very well chosen, which is a point we are happy to concede; 'boolean-respecting' might be more appropriate.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

which to begin such an investigation, and our example involving the π-calculus makes a modest start on this. In the first instance, we would expect that Milner's translation from *call-by-value* λ-calculus to π-calculus, as well as other translations in a similar spirit (see, for example, Sangiorgi (1993)), should furnish further examples of simulations. However, one should also note that it is at present far from clear whether the construction given in Example 3.4(ii) represents the 'best' way of building a C-structure based on the π-calculus. Other possible constructions, and the relationships between them, certainly deserve investigation.

One might also look at other process calculi, such as Milner's CCS or Hoare's CSP, and possible translations between them. This would lead naturally to an investigation of the *equivalence* or otherwise of different models of concurrent computation. In our view, either of the following would represent an interesting advance:

— An example of two known process calculi or models that are not *prima facie* equivalent but can be shown to yield equivalent C-structures.
— A proof of the inequivalence of two calculi or models that sheds some interesting light on the essential difference in computational power between them.

If the study of process calculi from the point of view of $\mathcal{CSTRUCT}$ proves a success, one might then broaden the investigation to embrace other paradigms such as quantum computation or membrane computation.

Of course, one ulterior motive for this entire project is the search for natural 'computability notions' underpinning existing models of computation. For instance, if several superficially different computation models turned out to yield equivalent C-structures, this might be seen as evidence that the underlying notion of computability that these represented was somehow a fundamental one. There is thus the hope that our general programme of identifying and classifying natural computability notions (as exemplified in Longley (2000), Longley (1999a) and Longley (2007)) might eventually be extended to a much wider field.

Whether or not this ambitious goal turns out to be achievable, we are hopeful that our theory might play some role in bringing together a wide range of current approaches to modelling computation under a single roof, thus enabling us to grasp how they all fit together. In particular, our notion of simulation would seem to offer a promising way of talking about how the views of a computational system at different 'levels of abstraction' are related to one another. Already, the typed PCA framework has proved itself capable of offering such a unifying perspective over a large tract of territory (see Longley (2007)). In the long run, we are hopeful that this may be even more true for C-structures. In this way, we can envisage our framework, or something like it, as playing a useful organisational role within theoretical computer science, perhaps somewhat akin to that of category theory itself.

Acknowledgements

I am very grateful to Robin Cockett and Pieter Hofstra for their invitation to present this material at the PCARC workshop, for illuminating technical discussions and for

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

encouraging me to write this paper. I am also grateful to the other PCARC participants for their valuable feedback, and to the anonymous referee for several helpful comments and suggestions.

I have benefited from the use of Paul Taylor's LaTeX package for category theory diagrams in the production of this paper.

References

Abramsky, S. (1989) The lazy lambda calculus. In: Turner, D. (ed.) *Research Topics in Functional Programming*, Addison-Wesley 65–116.

Abramsky, S. (2000) Process realizability. In: Bauer, F. L. and Steinbrüggen, R. (eds.) *Foundations of Secure Computation*, IOS Press 167–180.

Birkedal, L. and van Oosten, J. (2002) Relative and modified relative realizability. *Annals of Pure and Applied Logic* **118** 115–132.

Cockett, J. R. B. and Hofstra, P. J. W. (2008) Introduction to Turing categories. *Annals of Pure and Applied Logic* **156** (2-3) 183–209.

Cockett, J. R. B. and Hofstra, P. J. W. (2010) Categorical simulations. *Journal of Pure and Applied Algebra* **214** (10) 1835–1853.

Feferman S. (1975) A language and axioms for explicit mathematics. In: Crossley, J. (ed.) Algebra and Logic. *Springer-Verlag Lecture Notes in Mathematics* **450** 87–139.

Hofstra, P. J. W. (2006) All realizability is relative. *Mathematical Proceedings of the Cambridge Philosophical Society* **141** 239–264.

Hoshino, N. (2007) Linear realizability. In: Duparc, J. and Henzinger, T. A. (eds.) Proceedings, Computer Science Logic 2007. *Springer-Verlag Lecture Notes in Mathematics* **4646** 420–434.

Hyland, J. M. E. (1982) The effective topos. In: Troelstra, A. and van Dalen, D. (eds.) *L. E. J. Brouwer Centenary Symposium*, North-Holland 165–216.

Hyland, J. M. E. (1990) First steps in synthetic domain theory. In: Carboni, A., Pedicchio, M. and Rosolini, G. (eds.) Proceedings, Category Theory, Como. *Springer-Verlag Lecture Notes in Mathematics* **1488** 131–156.

Lietz, P. and Streicher, T. (2002) Impredicativity entails untypedness. *Mathematical Structures in Computer Science* **12** 335–347.

Longley, J. R. (1995) *Realizability Toposes and Language Semantics*, Ph.D. thesis CST-120-95, Department of Computer Science, University of Edinburgh.

Longley, J. R. (1999a) Matching typed and untyped realizability. In: Birkedal, L., van Oosten, J., Rosolini, G. and Scott, D. (eds.) Proceedings, Workshop on Realizability, Trento. *Electronic Notes in Theoretical Computer Science* **23** (1).

Longley, J. R. (1999b) Unifying typed and untyped realizability. Unpublished electronic note, available from `http://homepages.inf.ed.ac.uk/jrl/Research`.

Longley, J. R. (2000) Notions of computability at higher types I. In: Cori, R., Razborov, A., Todorčević, S. and Wood, C. (eds.) Proceedings, Logic Colloquium 2000. *Lecture Notes in Logic* **19**, ASL 32–142.

Longley, J. R. (2007) On the ubiquity of certain total type structures. *Mathematical Structures in Computer Science* **17** (5) 841–953.

Longo, G. and Moggi, E. (1991) Constructive natural deduction and its 'ω-set' interpretation. *Mathematical Structures in Computer Science* **1** 215–254.

Milner, R. (1992) Functions as processes. *Mathematical Structures in Computer Science* **2** 119–141.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press

Sangiorgi, D. (1993) *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*, Ph.D. thesis CST-99-93, Department of Computer Science, University of Edinburgh.

Schönfinkel, M. (1924) Über die Bausteine der Mathematischen Logik. *Mathematische Annalen* **92** 305–316. (Translation in van Heijenoort, J. (ed.) (1967) *A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press 344–366.)

Taylor, P. (1991) The fixed point property in synthetic domain theory. In: *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science* 152–160.

van Oosten, J. (1997) Extensional realizability. *Annals of Pure and Applied Logic* **84** 317–349.

van Oosten, J. (2008) *Realizability: An Introduction to its Categorical Side*, Studies in Logic and the Foundations of Mathematics **152**, Elsevier.

https://doi.org/10.1017/S0960129513000182 Published online by Cambridge University Press