Bisimulation as Path Type for Guarded Recursive Types

RASMUS EJLERS MØGELBERG, IT University of Copenhagen, Denmark
NICCOLÒ VELTRI, IT University of Copenhagen, Denmark

In type theory, coinductive types are used to represent processes, and are thus crucial for the formal verification of non-terminating reactive programs in proof assistants based on type theory, such as Coq and Agda. Currently, programming and reasoning about coinductive types is difficult for two reasons: The need for recursive definitions to be productive, and the lack of coincidence of the built-in identity types and the important notion of bisimilarity.

Guarded recursion in the sense of Nakano has recently been suggested as a possible approach to dealing with the problem of productivity, allowing this to be encoded in types. Indeed, coinductive types can be encoded using a combination of guarded recursion and universal quantification over clocks. This paper studies the notion of bisimilarity for guarded recursive types in *Ticked Cubical Type Theory*, an extension of Cubical Type Theory with guarded recursion. We prove that, for any functor, an abstract, category theoretic notion of bisimilarity for the final guarded coalgebra is equivalent (in the sense of homotopy type theory) to path equality (the primitive notion of equality in cubical type theory). As a worked example we study a guarded notion of labelled transition systems, and show that, as a special case of the general theorem, path equality coincides with an adaptation of the usual notion of bisimulation for processes. In particular, this implies that guarded recursion can be used to give simple equational reasoning proofs of bisimilarity. This work should be seen as a step towards obtaining bisimilarity as path equality for coinductive types using the encodings mentioned above.

CCS Concepts: • **Theory of computation → Process calculi**; **Type theory**; **Denotational semantics**;

Additional Key Words and Phrases: Dependent types, coinductive types, cubical type theory, homotopy type theory, guarded recursion, bisimulation, labelled transition systems, CCS

ACM Reference Format:
Rasmus Ejlers Møgelberg and Niccolò Veltri. 2019. Bisimulation as Path Type for Guarded Recursive Types. *Proc. ACM Program. Lang.* 3, POPL, Article 4 (January 2019), 29 pages. https://doi.org/10.1145/3290317

1 INTRODUCTION

Programming languages with dependent types such as Coq, Agda, F* and Idris are attracting increasing attention these years. The main use of dependency is for expressing predicates on types as used in formal verification. Large scale projects in this area include the CompCert C compiler, fully verified in Coq [Leroy 2006], and the ongoing Everest project [The Project Everest Team 2018] constructing a fully verified HTTPS stack using F*. Even when full formal verification is not the goal, dependent types can be used for software development [Brady 2016], pushing the known advantages of types in programming further.

For these applications, coinductive types are of particular importance, because they describe processes, i.e., non-terminating and reactive programs. However, programming and reasoning

Authors' addresses: Rasmus Ejlers Møgelberg, Department of Computer Science, IT University of Copenhagen, Rued Langgaards Vej 7, Copenhagen, 2300, Denmark, mogel@itu.dk; Niccolò Veltri, Department of Computer Science, IT University of Copenhagen, Rued Langgaards Vej 7, Copenhagen, 2300, Denmark, nive@itu.dk.

2475-1421/2019/1-ART4
https://doi.org/10.1145/3290317

about coinductive types is difficult in existing systems with dependent types for two reasons. The first is the need for totality in type theory, as required for soundness of the logical interpretation of types. To ensure this, recursively defined data of coinductive type must be *productive* [Coquand 1993] in the sense that all finite unfoldings can be computed in finite time.

Most proof assistants in use today use syntactic checks to ensure productivity of recursive definitions of coinductive data, but these checks are not modular, and therefore a certain overhead is associated with convincing the productivity checker. Guarded recursion in the style of Nakano [2000] has been suggested as a solution to this problem, encoding productivity in types. The idea is to use a modal type operator $\triangleright$ to encode a time delay in types. This allows productive recursive definitions to be encoded as maps of type $\triangleright A \to A$, and thus a fixed point operator taking input of that type allows for productive recursive programming. These fixed points can be used for programming with *guarded recursive types*, i.e., recursive types where the recursion variable only occurs guarded by $\triangleright$ modalities. Consider for example, the type of guarded streams satisfying the type equivalence $\mathsf{Str}^{\mathsf{g}} \simeq \mathbb{N} \times \triangleright \mathsf{Str}^{\mathsf{g}}$, expressing that the head of the stream is immediately available, but the tail takes one time step to compute. In this type, the stream of 0s can be defined using the productive recursive definition $\lambda x.\mathsf{fold}\,(0, x) : \triangleright \mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$. Guarded recursive types can be used to encode coinductive types [Atkey and McBride 2013], if one allows for the delay modality to be indexed by a notion of clocks which can be universally quantified. If, e.g., κ is a clock variable and $\mathsf{Str}^{\mathsf{g}}_{\kappa} \simeq \mathbb{N} \times \triangleright^{\kappa} \mathsf{Str}^{\mathsf{g}}_{\kappa}$ is a guarded recursive type, then $\forall\kappa.\mathsf{Str}^{\mathsf{g}}_{\kappa}$ is the coinductive type of streams.

The second problem for coinductive types in dependent type theory is the notion of equality. The natural notion of equality for coinductive types is bisimilarity, but this is not known to coincide with the build in notion of identity types, and indeed, many authors define bisimilarity as a separate notion when working with coinductive types [Abel et al. 2017; Coquand 1993]. For simple programming with coinductive types it should be the case that these notions coincide, indeed, it should be the case that the identity type is *equivalent* to the bisimilarity type. Such a result would be in the spirit of homotopy type theory, providing an extensionality principle to coinductive types, similar to function extensionality and univalence, which can be viewed as an extensionality principle for universes.

In this paper we take a step towards such an extensionality principle for coinductive types, by proving a similar statement for guarded recursive types. Rather than working with identity types in the traditional form, we work with path types as used in cubical type theory [Cohen et al. 2018], a recent type theory based on the cubical model of univalence [Bezem et al. 2013]. Precisely, we present *Ticked Cubical Type Theory* (**TCTT**), an extension of cubical type theory with guarded recursion and ticks, a tool deriving from Clocked Type Theory [Bahr et al. 2017] to be used for reasoning about guarded recursive types. The guarded fixed point operator satisfies the fixed point unfolding equality up to path equality, and using this, one can encode guarded recursive types up to type equivalence, as fixed points of endomaps on the universe.

We study a notion of guarded coalgebra, i.e., coalgebra for a functor of the form $F(\triangleright(-))$, and prove that for the final coalgebra the path equality type is equivalent to a general category theoretic notion of coalgebraic bisimulation for $F(\triangleright(-))$. This notion of bisimulation is an adaptation to type theory of a notion defined by Hermida and Jacobs [1998]. As a running example, we study *guarded labelled transition systems* (GLTSs), i.e., coalgebras for the functor $\mathsf{P}_{\mathsf{fin}}(A \times \triangleright(-))$. Here $\mathsf{P}_{\mathsf{fin}}$ is the finite powerset functor, which can be defined [Frumin et al. 2018] as a higher inductive type (HIT) [Univalent Foundations Program 2013], i.e., an inductive type with constructors for elements as well as for equalities. We show that path equality in the final coalgebra for $\mathsf{P}_{\mathsf{fin}}(A \times \triangleright(-))$ coincides with a natural adaptation of bisimilarity for finitely branching labelled transition systems to their guarded variant. This can be proved either directly using guarded recursion, or as a consequence of the general theorem mentioned above. However, there is a small difference between the abstract category theoretic notion of bisimulation used in the proof of the general theorem and the concrete one for GLTSs: The concrete formulation is a propositionally truncated [Univalent Foundations Program 2013] version of the abstract one. The truncated version is more convenient to work with in the special case, due to the set-truncation used in the finite powerset functor. On the other hand, using truncation in the general case would break equivalence of bisimilarity and path types.

As a consequence of the coincidence of bisimilarity and path equality, bisimilarity of processes can be proved using simple equational reasoning and guarded recursion. We give a few examples of that. Moreover, we show how to represent Milner's Calculus of Communicating Systems [Milner 1980] as well as Hennesy-Milner logic in our type theory.

The use of the *finite* powerset functor is motivated by the desire to extend this work from guarded recursive types to coinductive types in future work. It is well known from the set theoretic setting, that the unrestricted powerset functor does not have a final coalgebra, but restrictions to subsets of bounded cardinality do [Adámek et al. 2015; Schwencke 2010]. It is therefore to be expected that a similar restriction is needed in type theory to model processes with non-determinism as a coinductive type. We believe that the results presented here can be proved also for other cardinalities than finite, such as the countable powerset functor. This allows more general notions of processes to be modelled. See Section 9 for a discussion of this point.

We prove consistency of **TCTT** by constructing a denotational model using the category of presheaves over $C \times \omega$. The model combines the constructions of the cubical model (presheaves over the category of cubes C) and the topos of trees model of guarded recursion (presheaves over ω), and builds on a similar construction used to model Guarded Cubical Type Theory [Birkedal et al. 2016], but extends this to ticks using techniques of Mannaa and Møgelberg [2018]. Higher inductive types, such as the finite powerset type used in the running example, can be modelled using the techniques of Coquand et al. [2018], which adapt easily to the model used here.

1.1 Related Work

Guarded recursion in the form used here originates with Nakano [2000]. Aside from the applications to coinduction mentioned above, guarded recursion has also been used to model advanced features of programming languages [Birkedal et al. 2012; Bizjak et al. 2014], using an abstract form of step-indexing [Appel and McAllester 2001]. For this reason, type theory with guarded recursion has been proposed as a metalanguage for reasoning about programming languages, and indeed guarded recursion is one of the features of IRIS [Jung et al. 2018], a framework for higher-order separation logic implemented in Coq.

Previous work on applications of guarded recursion to programming and reasoning with coinductive types has mainly studied the examples of streams [Bizjak et al. 2016] and the lifting monad [Møgelberg and Paviotti 2016]. To our knowledge, bisimulation and examples involving higher inductive types have not previously been studied. The type theory closest to **TCTT** studied previously is Guarded Cubical Type Theory (**GCTT**) [Birkedal et al. 2016] which introduced the idea of fixed point unfoldings as paths. The only difference between the two languages is that **GCTT** uses delayed substitutions for reasoning about guarded recursive types, and **TCTT** replaces these by ticks. Ticks can be used to encode delayed substitutions, but have better operational behaviour [Bahr et al. 2017]. Birkedal et al. [2016] also proved that bisimilarity for guarded streams coincides with path equality.

Sized types [Hughes et al. 1996] is a different approach to the problem of encoding productivity in types. The idea is to associate abstract ordinals to types indicating the number of possible unfoldings of data of the type. The coinductive type is the sized type associated with the size ∞. Unlike guarded recursion, sized types have been implemented in the proof assistant Agda [The Agda Team 2018], and this has been used, e.g., for object oriented GUI programming [Abel et al.

2017]. On the other hand, the denotational semantics of guarded recursion is better understood than that of sized types, and this has the benefit of making it easier to combine with cubical type theory, and in particular to prove soundness of this combination.

Danielsson [2018] studies (also weak) bisimulation for coinductive types using sized types in Agda, proving soundness of up-to-techniques [Milner 1983], for a wide range of systems. In all these cases bisimulation is a separate type, and does not imply equality as in this work.

Ahrens et al. [2015] encode M-types (types of coinductive trees) in homotopy type theory and prove that for these bisimilarity logically implies equality. Our result is stronger in two ways: We consider more general coinductive types (although guarded versions of these), and we prove equivalence of types. Vezzosi [2017] proves that bisimilarity for streams implies path equality in Cubical Type Theory extended with streams. The proof can most likely be generalised to M-types.

1.2 Overview

Section 2 gives a brief introduction to Cubical Type Theory and the notion of higher inductive types. Cubical type theory is extended to Ticked Cubical Type Theory (**TCTT**) in Section 3. Section 4 introduces the finite powerset functor, which is used to define the notion of guarded labelled transition systems studied in Section 5. Among the results proved there are the fact that bisimilarity and path equality coincide in the final coalgebra, and examples of bisimilarity of processes. Section 6 presents the general theory of guarded coalgebras, proving the general statement of equivalence of path equality and bisimilarity, and Section 7 relates the abstract notion of bisimilarity of Section 6 to the concrete one for guarded labelled transition systems of Section 5. Finally, Section 8 constructs the denotational semantics of **TCTT**, and Section 9 concludes and discusses future work.

2 CUBICAL TYPE THEORY

Cubical type theory (**CTT**) [Cohen et al. 2018] is an extension of Martin-Löf type theory with concepts spawning from the cubical interpretation of homotopy type theory (**HoTT**) [Bezem et al. 2013; Cohen et al. 2018]. **CTT** is a dependent type theory with Π types, Σ types, sum types, natural numbers and a universe U. We write $A \longrightarrow B$ and $A \times B$ for non-dependent Π and Σ types respectively. We also write $a = b$ for judgemental equality of terms.

In **CTT**, the identity types of Martin-Löf type theory are replaced by a notion of *path* types. They correspond to an internalization of the homotopical interpretation of equalities as paths. Given $x, y : A$, we write $\mathsf{Path}_A\, x\, y$ for the type of paths between x and y in A, i.e., functions from an interval $\mathbb{I}$ to A with endpoints x and y. The interval is not a type, but still name assumptions of the form $i : \mathbb{I}$ can appear in contexts, and there is a judgement $\Gamma \vdash r : \mathbb{I}$ which means that r is formed from the names appearing in Γ using the grammar

$$r, s ::= 0 \mid 1 \mid i \mid 1 - r \mid r \wedge s \mid r \vee s$$

Two such *dimensions* or *names* are considered equal, if they can be proved equal using the laws of De Morgan algebra. Formally, the interval is thus the free De Morgan algebra and it can be thought of as the real interval $[0, 1]$ with $\wedge$ and $\vee$ as the minimum and maximum operations.

A type in a context Γ containing the names $i_1, \ldots, i_n : \mathbb{I}$ should be thought of as a n-dimensional cube. For example, the type $i : \mathbb{I}, j : \mathbb{I} \vdash A$ corresponds to a square:

$$\begin{tikzcd}
A(i/0)(j/1) \arrow[r, "A(j/1)"] \arrow[d, "A(i/0)"'] \arrow[dr, phantom, "A"] & A(i/1)(j/1) \arrow[d, "A(i/1)"] \\
A(i/0)(j/0) \arrow[r, "A(j/0)"'] & A(i/1)(j/0)
\end{tikzcd}$$

Here we have written $A(i/0)$ for the result of substituting 0 for i in A. Similarly to function spaces, path types have abstraction and application, denoted $\lambda i.\, p$ and $p\, r$ respectively, satisfying β and η equality: $(\lambda i.\, p)\, r = p(i/r)$ and $\lambda i.\, p\, i = p$. The typing rules for these two operations are given as follows:

$$\frac{\Gamma \vdash A \quad \Gamma, i : \mathbb{I} \vdash p : A}{\Gamma \vdash \lambda i.\, p : \mathsf{Path}_A\, (p(i/0))\, (p(i/1))} \qquad \frac{\Gamma \vdash p : \mathsf{Path}_A\, x\, y \quad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash p\, r : A}$$

satisfying, for $p : \mathsf{Path}_A\, x\, y$, the judgemental equalities $p\, 0 = x$ and $p\, 1 = y$. These equalities state that p is a path with endpoints x and y. In what follows, we write $\mathsf{Path}\, x\, y$ instead of $\mathsf{Path}_A\, x\, y$ if the type A is clear from context. Sometimes, especially in equational proofs, we also write $x \equiv y$ instead of $\mathsf{Path}\, x\, y$.

With the rules given so far, it is possible to prove certain fundamental properties of path equality. The identity path on a term $x : A$ is given by: $\mathsf{refl}\, x = \lambda i.\, x : \mathsf{Path}_A\, x\, x$. The inverse of a path $p : \mathsf{Path}_A\, x\, y$ is given by: $p^{-1} = \lambda i.\, p\, (1 - i) : \mathsf{Path}_A\, y\, x$. One can prove that functions respect path equality: Given $f : A \to B$ and a path $p : \mathsf{Path}_A\, x\, y$ we have $\mathsf{ap}_f\, p = \lambda i.\, f\, (p\, i) : \mathsf{Path}_B\, (f\, x)\, (f\, y)$. Moreover, one can prove extensionality principles which are generally not provable in standard Martin-Löf type theory, like function extensionality: Given two functions $f, g : \Pi x : A.\, B\, x$ and a family of paths $p : \Pi x : A.\, \mathsf{Path}_{B\, x}\, (f\, x)\, (g\, x)$ one can define $\mathsf{funext}\, f\, g\, p = \lambda i.\, \lambda x.\, p\, x\, i : \mathsf{Path}_{\Pi x:A.\, B\, x}\, f\, g$. But other properties characterizing path equality, such as transitivity or substitutivity, are not provable in the type theory we have described so far. In order to prove these properties, **CTT** admits a specific operation called *composition*.

In order to define this operation, we first introduce the *face lattice* $\mathbb{F}$. Formally, $\mathbb{F}$ is the free distributive lattice generated by the symbols $(i = 0)$ and $(i = 1)$, and the relation $(i = 0) \wedge (i = 1) = 0_{\mathbb{F}}$. This means that its elements are generated by the grammar:

$$\varphi, \psi ::= 0_{\mathbb{F}} \mid 1_{\mathbb{F}} \mid (i = 0) \mid (i = 1) \mid \varphi \wedge \psi \mid \varphi \vee \psi$$

The judgement $\Gamma \vdash \varphi : \mathbb{F}$ states that φ contains only names declared in the context Γ. There is an operation Γ, φ restricting the context Γ, if $\Gamma \vdash \varphi : \mathbb{F}$. Types and terms in context Γ, φ are called *partial*. Intuitively, a type in a restricted context should be thought of as a collection of faces of a higher dimensional cube. For example, the type $i : \mathbb{I}, j : \mathbb{I}, (i = 0) \vee (j = 1) \vdash A$ corresponds to the left and top faces of a square:

$$\begin{tikzcd} A(i/0)(j/1) \arrow[r, "A(j/1)"] \arrow[d, "A(i/0)"'] & A(i/1)(j/1) \\ A(i/0)(j/0) & A \end{tikzcd}$$

Given a partial term $\Gamma, \varphi \vdash u : A$, we write $\Gamma \vdash t : A[\varphi \mapsto u]$ for the conjunction of the two judgements $\Gamma \vdash t : A$ and $\Gamma, \varphi \vdash t = u : A$. In this case we say that the term t extends the partial term u on the extent φ. This notation can be extended to judgements of the form $t : A[\varphi_1 \mapsto u_1, \ldots, \varphi_n \mapsto u_n]$, abbreviating one typing and n equality judgements.

The composition operation is defined by the following typing rule:

$$\frac{\Gamma \vdash \varphi : \mathbb{F} \quad \Gamma, i : \mathbb{I} \vdash A \quad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \quad \Gamma \vdash u_0 : A(i/0)[\varphi \mapsto u(i/0)]}{\Gamma \vdash \mathsf{comp}^i_A\, [\varphi \mapsto u]\, u_0 : A(i/1)[\varphi \mapsto u(i/1)]}$$

Intuitively, the terms u and u_0 in the hypothesis specify an open box in the type A: u_0 corresponds to the base of the box, while u corresponds to the sides. The comp operation constructs a lid for this open box. Composition can be used to construct, e.g., the composition of paths $p : \mathsf{Path}_A\, x\, y$ and $q : \mathsf{Path}_A\, y\, z$:

$$p;q = \lambda i.\,\mathsf{comp}^j_A\,[(i=0)\mapsto x,(i=1)\mapsto q\,j]\,(p\,i) : \mathsf{Path}_A\,x\,z$$

We also refer to [Cohen et al. 2018] for the remaining constructions of **CTT**, which include systems and a glueing construction. The latter is a fundamental ingredient in the proof of the univalence axiom and the construction of composition for the universe. We will not be needing gluing and systems directly in this paper, and so omit them from the brief overview.

2.1 Higher Inductive Types

Recently, Coquand et al. [2018] have introduced an extension of **CTT** with higher inductive types (HITs). HITs are an important concept in **HoTT** which generalize the notion of inductive type. A HIT A can be thought of as an inductive type in which the introduction rules not only specify the generators of A, but can also specify the generators of the higher equality types of A. Using HITs, one can define topological spaces such as the circle, the torus and suspensions in **HoTT** and develop homotopy theory in a synthetic way [Univalent Foundations Program 2013, Ch. 8]. HITs are also used for implementing free algebras of signatures specified by operations and equations, for example the free group on a type, or to construct quotients of types by equivalence relations.

We now recall how to extend **CTT** with HITs by showing how to add propositional truncation. We refer to [Coquand et al. 2018] for the definition of other HITs and the description of a common pattern for introducing HITs in **CTT**. In Section 4, we will also present the finite powerset construction as a HIT.

Given a type A, we write $\|A\|$ for the *propositional truncation* of A. The type $\|A\|$ can have at most one inhabitant up to path equality, informally, an "uninformative" proof of A. In other words, the existence of a term $t : \|A\|$ tells us that the type A is inhabited, but it does not provide any explicit inhabitant of A. Moreover, any other term $u : \|A\|$ is path equal to t. The introduction rules of $\|A\|$ are the following:

$$\frac{x : A}{|x| : \|A\|} \qquad \frac{u, v : \|A\| \quad r : \mathbb{I}}{\mathsf{sq}\,u\,v\,r : \|A\|}$$

with the judgemental equalities $\mathsf{sq}\,u\,v\,0 = u$ and $\mathsf{sq}\,u\,v\,1 = v$. Notice that the higher constructor sq, which when applied to terms u and v specifies an element in $\mathsf{Path}\,u\,v$, is treated in **CTT** as a point constructor which depends on a name $r : \mathbb{I}$.

Remember that in **CTT** every type is endowed with a composition operation. Coquand et al. showed that it is possible to define composition for a HIT by adding a *homogeneous composition* as an additional constructor for the type. In the case of propositional truncation, the latter is introduced by the following rule:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \varphi : \mathbb{F} \quad \Gamma, \varphi, i : \mathbb{I} \vdash u : \|A\| \quad \Gamma \vdash u_0 : \|A\|[\varphi \mapsto u(i/0)]}{\Gamma \vdash \mathsf{hcomp}^i_{\|A\|}\,[\varphi \mapsto u]\,u_0 : \|A\|[\varphi \mapsto u(i/1)]}$$

Notice that hcomp differs from comp, since in the latter the type A depends also on the name $i : \mathbb{I}$. We refer to [Coquand et al. 2018] for the details on the derivation of a composition operation for $\|A\|$ and for other HITs.

We now describe the elimination principle of propositional truncation in **CTT**. Given $x : \|A\| \vdash P\,x$, a family of terms $x : A \vdash t\,x : P\,|x|$ and a family of paths

$$u, v : \|A\|, x : P\,u, y : P\,v, i : \mathbb{I} \vdash sq\,u\,v\,x\,y\,i : P\,(\mathsf{sq}\,u\,v\,i)[(i=0)\mapsto x,(i=1)\mapsto y],$$

we can define $f : \Pi x : \|A\|.\,P\,x$ by cases:

$$f\,|x| = t\,x$$
$$f\,(\mathsf{sq}\,u\,v\,r) = sq\,u\,v\,(f\,u)\,(f\,v)\,r$$

plus a case for the hcomp constructor, see [Coquand et al. 2018].

The addition of propositional truncation and other HITs to **CTT** is justified by the existence of these types in the cubical set model of **CTT** [Coquand et al. 2018].

We conclude this section by briefly describing an auxiliary HIT S. The type S is generated by two points $\mathsf{b}_0, \mathsf{b}_1 : \mathsf{S}$ and two paths $\mathsf{p}_0, \mathsf{p}_1$ between b_0 and b_1. Similarly to other HITs, we also have to add an hcomp constructor. The induction principle of S states that in order to give a map of type $\Pi s : \mathsf{S}.\, P\, s$ one has to provide, for $j = 0, 1$, a point $b_j : P\, \mathsf{b}_j$ and a path

$$i : \mathbb{I} \vdash p_j\, i : P\, (\mathsf{p}_j\, i)[(i = 0) \mapsto b_0, (i = 1) \mapsto b_1].$$

The type S is used in the HoTT book for specifying the 0-truncation operation as a HIT [Univalent Foundations Program 2013, Ch. 6.9]. In Section 4, we will employ the type S in the trunc constructor of the finite powerset, that will force the finite powerset type to be a set.

2.2 Some Basic Notions and Results

This section recalls some basic notions and results from homotopy type theory. All results mentioned are proved in [Univalent Foundations Program 2013] unless otherwise stated.

We say that a type is *contractible* if it has exactly one inhabitant. We define a type $\mathsf{isContr}\, A = \Sigma x : A.\, \Pi y : A.\, \mathsf{Path}\, x\, y$, which is inhabited if and only if A is contractible. The unit type 1 is contractible.

We say that a type is a *proposition*, or it is *(-1)-truncated*, if there exists a path between any two of its inhabitants. We define a type $\mathsf{isProp}\, A = \Pi x\, y : A.\, \mathsf{Path}\, x\, y$, which is inhabited if and only if A is a proposition. We also define a type Prop consisting of all types which are propositions. Every contractible type is a proposition. The empty type 0 is a proposition. The propositional truncation of a type is a proposition by construction. The dependent function type $\Pi x : A.\, B\, x$ is a proposition if and only if $B\, x$ is a proposition for all $x : A$. Given two types A and B, we define $A \vee B = \|A + B\|$. Moreover, given a type A and a type family P on A, we define $\exists x : A.\, P\, x = \|\Sigma x : A.\, P\, x\|$. When the latter type is inhabited, we say that there *merely exists* a term $x : A$ such that $P\, x$ holds.

We say that a type is a *set*, or it is *0-truncated*, if there exists at most one path between any two of its inhabitants. We define a type $\mathsf{isSet}\, A = \Pi x\, y : A.\, \Pi p\, q : \mathsf{Path}\, x\, y.\, \mathsf{Path}\, p\, q$, which is inhabited if and only if A is a set. We also define a type Set consisting of all types which are sets. Every proposition is a set. The type $\mathbb{N}$ of natural numbers is a set. Another example of a set is the finite powerset introduced in Section 4.

The *fiber* of a function $f : A \to B$ over a term $y : B$ is given by the type $\mathsf{fib}_f\, y = \Sigma x : A.\, \mathsf{Path}\, (f\, x)\, y$. We say that a function $f : A \to B$ is an *equivalence* if all the fibers of f are contractible. We define a type $\mathsf{isEquiv}\, f = \Pi y : B.\, \mathsf{isContr}\, (\mathsf{fib}_f\, y)$, which is inhabited if and only if f is an equivalence. We say that two types are *equivalent* if there exists an equivalence between them. We also define a type $\mathsf{Equiv}\, A\, B$ consisting of all functions between A and B which are equivalences. In order to prove that a function f is an equivalence, it is enough to construct a function $g : B \to A$ such that $g \circ f$ is path equal to the identity function on A and $f \circ g$ is path equal to the identity function on B.

A characteristic feature of **HoTT** is the *univalence axiom*, stating that the canonical map from $\mathsf{Path}\, A\, B$ to $\mathsf{Equiv}\, A\, B$ is an equivalence, for all types A and B. The univalence axiom is provable in **CTT** [Cohen et al. 2018]. In the rest of the paper, when we refer to the univalence axiom, we mean direction $\mathsf{Equiv}\, A\, B \to \mathsf{Path}\, A\, B$ of the equivalence. In other words, when we need to prove that two types are path equal, we will invoke the univalence axiom and prove that the two types are equivalent instead. When A and B are propositions, the univalence axiom implies that the type $\mathsf{Path}\, A\, B$ is equivalent to the type $A \leftrightarrow B$ of logical equivalences between A and B, where $A \leftrightarrow B = (A \to B) \times (B \to A)$.

We conclude this section with a couple of lemmata that we will employ later on. These are standard results of **HoTT** that we state without proofs.

Lemma 2.1. *Given a type X and elements $x, y, z : X$, if there exists a path $p : \mathsf{Path}_X\, x\, y$, then the types $\mathsf{Path}_X\, x\, z$ and $\mathsf{Path}_X\, y\, z$ are equivalent.*

Lemma 2.2. *Let $X, X' : \mathsf{U}$ and $Y : X' \to \mathsf{U}$. If there exists an equivalence $f : X \to X'$, then the function $h : (\Sigma x : X.\, Y(f\,x)) \to \Sigma x' : X'.\, Y x'$ given by $h\,(x, y) = (f\,x\,,\ y)$ is also an equivalence.*

3 TICKED CUBICAL TYPE THEORY

We now extend **CTT** with guarded recursion. The resulting type theory, called *ticked cubical type theory* (**TCTT**), differs from guarded cubical type theory (**GCTT**) [Birkedal et al. 2016] only by featuring ticks. Ticks are an invention deriving from Clocked Type Theory [Bahr et al. 2017] and can be used for reasoning about delayed data. In particular, ticks can be used to encode the delayed substitutions of **GCTT**, which served the same purpose, but are simpler and can be given confluent, strongly normalising reduction semantics satisfying canonicity. Indeed, these results have been proved for Clocked Type Theory which includes guarded fixed points [Bahr et al. 2017].

Formally, **TCTT** extends **CTT** with tick assumptions $\alpha : \mathbb{T}$ in the context, along with abstraction and application to ticks following these rules.

$$\frac{\Gamma \vdash}{\Gamma, \alpha : \mathbb{T} \vdash} \qquad \frac{\Gamma, \alpha : \mathbb{T} \vdash A}{\Gamma \vdash \triangleright (\alpha : \mathbb{T}).A} \qquad \frac{\Gamma, \alpha : \mathbb{T} \vdash t : A}{\Gamma \vdash \lambda(\alpha : \mathbb{T}).t : \triangleright (\alpha : \mathbb{T}).A}$$

$$\frac{\Gamma \vdash t : \triangleright (\alpha : \mathbb{T}).A}{\Gamma, \beta : \mathbb{T}, \Gamma' \vdash t\,[\beta] : A(\alpha/\beta)} \qquad \frac{\Gamma, \alpha : \mathbb{T} \vdash A : \mathsf{U}}{\Gamma \vdash \triangleright (\alpha : \mathbb{T}).A : \mathsf{U}}$$

together with β and η rules:

$$(\lambda(\alpha : \mathbb{T}).t)[\beta] = t(\alpha/\beta) \qquad \lambda(\alpha : \mathbb{T}).t\,[\alpha] = t$$

The sort $\mathbb{T}$ of ticks enjoys the same status as the interval $\mathbb{I}$. In particular it is not a type. The type $\triangleright (\alpha : \mathbb{T}).A$ should be thought of as classifying data of type A that is only available one time step from now. Similarly, ticks should be thought of as evidence that time has passed. For example, in a context of the form $\Gamma, \alpha : \mathbb{T}, \Gamma'$, the assumptions in Γ are available for one more time step than those of Γ'. In the application rule, the assumption states that t is a promise of data of type A available one time-step after the variables in Γ have arrived, and thus the tick β, can be used to open t to an element of type A. We write $\triangleright A$ for $\triangleright (\alpha : \mathbb{T}).A$ where α does not occur free in A. Note in particular that the rule for tick application prevents terms like $\lambda x.\lambda(\alpha : \mathbb{T}).x\,[\alpha]\,[\alpha] : \triangleright \triangleright A \to \triangleright A$ being well typed. Such a term in combination with the fixed point operator to be introduced below would make any type of the form $\triangleright A$ inhabited, and logically trivialise a large part of the theory.

The abstraction of the tick α in type $\triangleright (\alpha : \mathbb{T}).A$ makes the type behave like a dependent function space between ticks and the type A, similarly to the path type. This can be used, e.g., to type the terms of a dependent form of the applicative functor law:

$$\begin{aligned} \mathsf{next} &= \lambda x.\, \lambda(\alpha : \mathbb{T}).x : A \to \triangleright A \\ \odot &= \lambda f.\, \lambda y.\, \lambda(\alpha : \mathbb{T}).f[\alpha]\,(y[\alpha]) : \triangleright(\Pi x : A.\, B\,x) \to \Pi y : \triangleright A.\, \triangleright (\alpha : \mathbb{T}).B(x/y[\alpha]) \end{aligned} \tag{1}$$

As part of their special status, names and faces are independent of time, in the sense that they can always be commuted with ticks as expressed in the invertible rules

$$\frac{\Gamma, \alpha : \mathbb{T}, i : \mathbb{I} \vdash A}{\Gamma, i : \mathbb{I}, \alpha : \mathbb{T} \vdash A} \qquad \frac{\Gamma, \alpha : \mathbb{T}, i : \mathbb{I} \vdash t : A}{\Gamma, i : \mathbb{I}, \alpha : \mathbb{T} \vdash t : A} \qquad \frac{\Gamma, \alpha : \mathbb{T}, \varphi \vdash A}{\Gamma, \varphi, \alpha : \mathbb{T} \vdash A} \qquad \frac{\Gamma, \alpha : \mathbb{T}, \varphi \vdash t : A}{\Gamma, \varphi, \alpha : \mathbb{T} \vdash t : A} \tag{2}$$

This effect could have similarly been obtained by not erasing names and faces from the assumption of the tick application rule above, in which case the above rules could have been derived. One consequence of these rules is an extensionality principle for $\triangleright$ stating the equivalence of types

$$\mathsf{Path}_{\triangleright(\alpha:\mathbb{T}).A}\, x\, y \simeq \triangleright(\alpha : \mathbb{T}).(\mathsf{Path}_A\,(x\,[\alpha])\,(y\,[\alpha])) \tag{3}$$

as witnessed by the terms

$$\lambda p.\, \lambda\alpha.\, \lambda i.\, (p\, i)[\alpha] : \mathsf{Path}_{\triangleright(\alpha:\mathbb{T}).A}\, x\, y \to \triangleright(\alpha : \mathbb{T}).(\mathsf{Path}_A\,(x\,[\alpha])\,(y\,[\alpha]))$$

and

$$\lambda p.\, \lambda i.\, \lambda\alpha.\, p[\alpha]\, i : \triangleright(\alpha : \mathbb{T}).(\mathsf{Path}_A\,(x\,[\alpha])\,(y\,[\alpha])) \to \mathsf{Path}_{\triangleright(\alpha:\mathbb{T}).A}\, x\, y$$

In particular, if $x, y : A$ then

$$\mathsf{Path}_{\triangleright A}\,(\mathsf{next}\, x)\,(\mathsf{next}\, y) \simeq \triangleright(\mathsf{Path}_A\, x\, y) \tag{4}$$

Note that as a consequence of this, the type of propositions is closed under $\triangleright$:

Lemma 3.1. *Let* $A : \triangleright\mathsf{U}$. *If* $\triangleright(\alpha : \mathbb{T}).\mathsf{isProp}(A\,[\alpha])$ *also* $\mathsf{isProp}(\triangleright(\alpha : \mathbb{T}).A\,[\alpha])$.

Proof. Suppose $x, y : \triangleright(\alpha : \mathbb{T}).A\,[\alpha]$, we must show that $x \equiv y$. By extensionality it suffices to show $\triangleright(\alpha : \mathbb{T}).(x\,[\alpha] \equiv y\,[\alpha])$. By assumption there is a $p : \triangleright(\alpha : \mathbb{T}).\mathsf{isProp}(A\,[\alpha])$ and the term $\lambda(\alpha : \mathbb{T}).p\,[\alpha]\,(x\,[\alpha])\,(y\,[\alpha])$ inhabits the desired type. □

One additional benefit of the ticks is that the composition operator for $\triangleright$ can be defined in type theory. To see this, assume

$$\Gamma \vdash \varphi : \mathbb{F} \qquad \Gamma, i : \mathbb{I} \vdash \triangleright(\alpha : \mathbb{T}).A$$
$$\Gamma, \varphi, i : \mathbb{I} \vdash u : \triangleright(\alpha : \mathbb{T}).A \qquad \Gamma \vdash u_0 : (\triangleright(\alpha : \mathbb{T}).A)(i/0)[\varphi \mapsto u(i/0)].$$

Since these imply assumptions for the composition operator on A:

$$\Gamma, \alpha : \mathbb{T} \vdash \varphi : \mathbb{F} \qquad \Gamma, \alpha : \mathbb{T}, i : \mathbb{I} \vdash A$$
$$\Gamma, \alpha : \mathbb{T}, \varphi, i : \mathbb{I} \vdash u\,[\alpha] : A \qquad \Gamma, \alpha : \mathbb{T} \vdash u_0\,[\alpha] : A(i/0)[\varphi \mapsto u\,[\alpha](i/0)].$$

we can define

$$\Gamma \vdash \mathsf{comp}^i_{\triangleright(\alpha:\mathbb{T}).A}\,[\varphi \mapsto u]\, u_0 = \lambda(\alpha : \mathbb{T}).\mathsf{comp}^i_A\,[\varphi \mapsto u\,[\alpha]]\,(u_0\,[\alpha]) : (\triangleright(\alpha : \mathbb{T}).A)(i/1)$$

Note that this uses the rules in (2) for exchange of ticks and names and faces. It is easy to see that the term $\mathsf{comp}^i_{\triangleright(\alpha:\mathbb{T}).A}\,[\varphi \mapsto u]\, u_0$ extends $u(i/1)$ on φ.

3.1 Fixed Points

TCTT comes with a primitive fixed point operator mapping terms f of type $\triangleright A \to A$ to fixed points of the composition $f \circ \mathsf{next}$. As in **GCTT**, the fixed point equality holds only up to path, not judgemental equality. This is to ensure termination of the reduction semantics, but is not necessary for logical consistency, as verified by the model, in which the fixed point equality holds definitionally. The typing rule gives the path, with the end points definitionally equal to the two sides of the fixed point equality.

$$\frac{\Gamma, x : \triangleright A \vdash t : A \quad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash \mathsf{dfix}^r x.t : \triangleright A} \qquad \mathsf{dfix}^1 x.t = \mathsf{next}\, t(x/\mathsf{dfix}^0 x.t)$$

Note that the above *delayed* fixed point is of type $\triangleright A$ rather than A. This is to maintain canonicity, but one can define a fixed point $\mathsf{fix}^r x.t$ as $t(x/\mathsf{dfix}^r x.t)$, which gives the derived rule

$$\frac{\Gamma, x : \triangleright A \vdash t : A}{\Gamma \vdash \mathsf{pfix}\, x.t : \mathsf{Path}_A(\mathsf{fix}\, x.t)(t(x/\mathsf{next}(\mathsf{fix}\, x.t)))}$$

by defining $\mathsf{pfix}\, x.t = \lambda i.\mathsf{fix}^i x.t$. We write simply $\mathsf{fix}\, x.t$ for $\mathsf{fix}^0\, x.t$

Fixed points are unique in the following sense. Let $\Gamma, x : \triangleright A \vdash t : A$ and consider a term $a : A$ such that $p : \mathsf{Path}_A\, a\, (t(x/\mathsf{next}\, a))$. Then a is path equal to $\mathsf{fix}\, x.t$ as can be proved by guarded recursion: If $e : \triangleright \mathsf{Path}_A\, a\, (\mathsf{fix}\, x.t)$ then

$$\lambda i.t(x/\lambda(\alpha : \mathbb{T}).e[\alpha]\, i) : \mathsf{Path}_A\, (t(x/\mathsf{next}\, a))\, (t(x/\mathsf{next}(\mathsf{fix}\, x.t)))$$

and thus we can prove $\mathsf{Path}_A\, a\, (\mathsf{fix}\, x.t)$ by composing this with p and $(\mathsf{pfix}\, x.t)^{-1}$.

One application of fixed points is to define *guarded recursive types* as fixed points of endomaps on the universe. For example, one can define a type of guarded streams of natural numbers as $\mathsf{Str}^{\mathsf{g}} = \mathsf{fix}\, X.\mathbb{N} \times \triangleright (\alpha : \mathbb{T}).X[\alpha] : \mathsf{U}$. Since path equality at the universe is equivalence of types, we obtain an equivalence

$$\mathsf{Str}^{\mathsf{g}} \simeq \mathbb{N} \times \triangleright \mathsf{Str}^{\mathsf{g}}$$

witnessed by terms $\mathsf{fold} : \mathbb{N} \times \triangleright \mathsf{Str}^{\mathsf{g}} \to \mathsf{Str}^{\mathsf{g}}$ and $\mathsf{unfold} : \mathsf{Str}^{\mathsf{g}} \to \mathbb{N} \times \triangleright \mathsf{Str}^{\mathsf{g}}$. One can then use the fixed point operator for programming with streams and, e.g., define a constant stream of zeros as $\mathsf{fix}\, x.\mathsf{fold}\, (0, x)$. We refer to [Bizjak et al. 2016] for more examples of programming with guarded recursive types.

Guarded recursive types are simultaneously initial algebras and final coalgebras. This is a standard result [Birkedal et al. 2012], but formulated here in terms of path equality, so we repeat it. Recall that a small functor is a term $F : \mathsf{U} \to \mathsf{U}$ together with another term (also called F) of type $\Pi X, Y : U.(X \to Y) \to FX \to FY$ satisfying the functor laws up to path equality, i.e., with witnesses of types

$$\Pi X : U.\mathsf{Path}_{FX \to FX}(\lambda x : FX.x)\, (F(\lambda x : X.x))$$
$$\Pi X, Y, Z : U, f : X \to Y, g : Y \to Z.\mathsf{Path}_{FX \to FZ}(Fg \circ Ff)\, (F(g \circ f))\,.$$

For example, $\triangleright : \mathsf{U} \to \mathsf{U}$ is a functor, with functorial action given by $\triangleright f = \lambda x.\lambda(\alpha : \mathbb{T}).f(x\, [\alpha]) : \triangleright A \to \triangleright B$ for $f : A \to B$.

Proposition 3.2. *Let F be a small functor. The fixed point $\nu F^{\mathsf{g}} = \mathsf{fix}\, X.F(\triangleright (\alpha : \mathbb{T}).X[\alpha])$ is an initial algebra and a final coalgebra for the functor $F \circ \triangleright$. We spell out the final coalgebra statement: For any $A : \mathsf{U}$ and $f : A \to F(\triangleright A)$, there exists a unique (up to path equality) term $h : A \to \nu F^{\mathsf{g}}$ such that*

$$\mathsf{unfold} \circ h \equiv F(\triangleright h) \circ f$$

Proof. The equality to be satisfied by h is equivalent to

$$h \equiv \mathsf{fold} \circ F(\triangleright h) \circ f$$

and this is satisfied by defining h to be $\mathsf{fix}\, k.\mathsf{fold} \circ F(\lambda x.k \odot x) \circ f$ using the applicative action (1) to apply $k : \triangleright(A \to \nu F^{\mathsf{g}})$ to $x : \triangleright A$. Uniqueness follows from uniqueness of fixed points. The initial algebra property can be proved similarly. □

4 THE FINITE POWERSET FUNCTOR

In order to define finitely branching labelled transition systems, we need to represent finite subsets of a given type. There are several different ways to describe finite sets and finite subsets in type theory [Spiwack and Coquand 2010]. Recently, Frumin et al. [2018] have presented an implementation of the finite powerset functor in **HoTT** as a higher inductive type. Given a type A, they construct the type $\mathsf{P}_{\mathsf{fin}}A$ of finite subsets of A as the free join semilattice over A. We define the type $\mathsf{P}_{\mathsf{fin}}A$ in **CTT** following the pattern for HITs given by Coquand et al. [2018] that we summarized in Section 2.1.

Notice that the same definition can be read in the extended type theory **TCTT**. Given a type A, the type $\mathsf{P}_{\mathsf{fin}}A$ is introduced by the following constructors:

$$\frac{}{\varnothing : \mathsf{P}_{\mathsf{fin}}A} \qquad \frac{a : A}{\{a\} : \mathsf{P}_{\mathsf{fin}}A} \qquad \frac{x, y : \mathsf{P}_{\mathsf{fin}}A}{x \cup y : \mathsf{P}_{\mathsf{fin}}A}$$

$$\frac{x : \mathsf{P}_{\mathsf{fin}}A \quad r : \mathbb{I}}{\mathsf{nl}\, x\, r : \mathsf{P}_{\mathsf{fin}}A} \quad \frac{x, y, z : \mathsf{P}_{\mathsf{fin}}A \quad r : \mathbb{I}}{\mathsf{assoc}\, x\, y\, z\, r : \mathsf{P}_{\mathsf{fin}}A} \quad \frac{a : A \quad r : \mathbb{I}}{\mathsf{idem}\, a\, r : \mathsf{P}_{\mathsf{fin}}A} \quad \frac{x, y : \mathsf{P}_{\mathsf{fin}}A \quad r : \mathbb{I}}{\mathsf{com}\, x\, y\, r : \mathsf{P}_{\mathsf{fin}}A}$$

$$\frac{\Gamma \vdash f : \mathsf{S} \to \mathsf{P}_{\mathsf{fin}}A \quad r, s : \mathbb{I}}{\mathsf{trunc}\, f\, r\, s : \mathsf{P}_{\mathsf{fin}}A}$$

with the judgemental equalities:

$$\begin{aligned}
\mathsf{nl}\, x\, 0 &= \varnothing \cup x & \mathsf{nl}\, x\, 1 &= x \\
\mathsf{assoc}\, x\, y\, z\, 0 &= (x \cup y) \cup z & \mathsf{assoc}\, x\, y\, z\, 1 &= x \cup (y \cup z) \\
\mathsf{idem}\, a\, 0 &= \{a\} \cup \{a\} & \mathsf{idem}\, a\, 1 &= \{a\} \\
\mathsf{com}\, x\, y\, 0 &= x \cup y & \mathsf{com}\, x\, y\, 1 &= y \cup x \\
\mathsf{trunc}\, f\, 0\, j &= f\, (\mathsf{p}_0\, j) & \mathsf{trunc}\, f\, 1\, j &= f\, (\mathsf{p}_1\, j) \\
\mathsf{trunc}\, f\, i\, 0 &= f\, (\mathsf{b}_0) & \mathsf{trunc}\, f\, i\, 1 &= f\, (\mathsf{b}_1)
\end{aligned}$$

As for the higher constructor sq of propositional truncation given in Section 2.1, the higher constructors of $\mathsf{P}_{\mathsf{fin}}A$ are introduced as point constructors depending on names in $\mathbb{I}$. For example, the constructor nl states that the empty set $\varnothing$ is a left unit for the union operation $\cup$. Given $x : \mathsf{P}_{\mathsf{fin}}A$, we have that $\mathsf{nl}\, x$ is a path between $\varnothing \cup x$ and x.

The constructor trunc refers to the HIT S introduced in the end of Section 2.1, and forces $\mathsf{P}_{\mathsf{fin}}A$ to be a set. To see this, suppose $x, y : \mathsf{P}_{\mathsf{fin}}A$ and $p, q : x \equiv y$. To show $p \equiv q$, define $f : \mathsf{S} \to \mathsf{P}_{\mathsf{fin}}A$ by recursion on S:

$$f\, \mathsf{b}_0 = x \qquad f\, \mathsf{b}_1 = y \qquad j : \mathbb{I} \vdash f\, (\mathsf{p}_0\, j) = p\, j \qquad j : \mathbb{I} \vdash f\, (\mathsf{p}_1\, j) = q\, j.$$

Then $\mathsf{trunc}\, f : p \equiv q$.

Following the pattern for higher inductive types in **CTT**, we also introduce a constructor hcomp which imposes a homogenous composition structure on $\mathsf{P}_{\mathsf{fin}}A$ and allows us to define composition for $\mathsf{P}_{\mathsf{fin}}A$. Assuming $\Gamma \vdash A$, the hcomp operation is given as:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash \varphi : \mathbb{F} \quad \Gamma, \varphi, i : \mathbb{I} \vdash u : \mathsf{P}_{\mathsf{fin}}A \quad \Gamma \vdash u_0 : \mathsf{P}_{\mathsf{fin}}A\, [\varphi \mapsto u(i/0)]}{\Gamma \vdash \mathsf{hcomp}^i_{\mathsf{P}_{\mathsf{fin}}A}\, [\varphi \mapsto u]\, u_0 : \mathsf{P}_{\mathsf{fin}}A\, [\varphi \mapsto u(i/1)]}$$

Finally, we will assume that the universe is closed under higher inductive type formers such as finite powersets in the sense that if $A : \mathsf{U}$ also $\mathsf{P}_{\mathsf{fin}}A : \mathsf{U}$. Consistency of the extension of **TCTT** with the HITs used in this paper is justified by the denotational model presented in Section 8.

The type $\mathsf{P}_{\mathsf{fin}}A$ comes with a rather complex induction principle, which is similar to the one described by Frumin et al. [2018]. We spell it out for the case in which the type we are eliminating into is a proposition. Let $Q : \mathsf{P}_{\mathsf{fin}}A \to \mathsf{Prop}$. Given $e : Q\, \varnothing$, $s : \Pi a : A.\, Q\, \{a\}$ and $u : \Pi x\, y : \mathsf{P}_{\mathsf{fin}}A.\, Q\, x \to Q\, y \to Q\, (x \cup y)$, then there exists a term $g : \Pi x : \mathsf{P}_{\mathsf{fin}}A.\, Q\, x$ such that:

$$g\, \varnothing = e \qquad g\, \{a\} = s\, a \qquad g\, (x \cup y) = u\, x\, y\, (g\, x)\, (g\, y)$$

The assumption that $Q\, x$ is a proposition means that we do not have to specify where the higher constructors should be mapped.

Using this induction principle of $\mathsf{P}_{\mathsf{fin}}A$, one can define a membership predicate $\in\, : A \to \mathsf{P}_{\mathsf{fin}}A \to \mathsf{Prop}$:

$$a \in \varnothing = \bot \qquad a \in \{b\} = \|a \equiv b\| \qquad a \in x \cup y = a \in x \vee a \in y.$$

The cases for the other constructors are dealt with in a straightforward way using the univalence axiom.

Given $x, y : \mathsf{P_{fin}}A$, we write $x \subseteq y$ for $\Pi a : A.\, a \in x \to a \in y$. Using the induction principle of the finite powerset construction, it is also possible to prove an extensionality principle for finite subsets: two subsets $x, y : \mathsf{P_{fin}}A$ are path equal if and only if they contain the same elements, i.e. if the type $\Pi a : A.\, a \in x \leftrightarrow a \in y$ is inhabited [Frumin et al. 2018].

Using the non-dependent version of the induction principle, it is possible to prove that $\mathsf{P_{fin}}$ is a functor. Given $f : A \to B$, we write $\mathsf{P_{fin}}f : \mathsf{P_{fin}}A \to \mathsf{P_{fin}}B$ for the action of $\mathsf{P_{fin}}$ on the function f. We have:

$$\mathsf{P_{fin}}f\ \varnothing = \varnothing \qquad \mathsf{P_{fin}}f\ \{a\} = \{f\ a\} \qquad \mathsf{P_{fin}}f\ (x \cup y) = \mathsf{P_{fin}}f\ x \cup \mathsf{P_{fin}}f\ y$$

The cases for the higher constructors are straightforward. For example, $\mathsf{P_{fin}}f\ (\mathsf{nl}\, x\, r) = \mathsf{nl}\, (\mathsf{P_{fin}}f\ x)\, r$.

We conclude this section by mentioning an auxiliary result describing the image of $\mathsf{P_{fin}}f$.

Lemma 4.1. *Let $f : A \to B$ and $x : \mathsf{P_{fin}}A$. If $a \in x$ then also $f(a) \in \mathsf{P_{fin}}f\ x$. If $b : B$ is in $\mathsf{P_{fin}}f\ x$, then there merely exists an $a : A$ such that $a \in x$ and $b \equiv f(a)$.*

Proof. The first statement can be proved by induction on x, and we omit the simple verification, focusing on the second statement, which is also proved by induction on x.

The case of $x = \varnothing$ implies $\mathsf{P_{fin}}f\ x = \varnothing$ and so the assumption $b \in \mathsf{P_{fin}}f\ x$ implies absurdity. If $x = \{a\}$, then $\mathsf{P_{fin}}f\ x = \{f(a)\}$ and so the assumption implies $\|b \equiv f(a)\|$. Since we are proving a proposition, we can apply induction to the latter and obtain a proof $p : b \equiv f(a)$. Then $|\,(a, (|\mathsf{refl}\ a|, p))\,|$ proves the case.

If $x = x_1 \cup x_2$, then $\mathsf{P_{fin}}f\ x = \mathsf{P_{fin}}f\ x_1 \cup \mathsf{P_{fin}}f\ x_2$ and so $b \in \mathsf{P_{fin}}f\ x_1 \vee b \in \mathsf{P_{fin}}f\ x_2$. By induction we may thus assume that either $b \in \mathsf{P_{fin}}f\ x_1$ or $b \in \mathsf{P_{fin}}f\ x_2$, the proof of two cases are symmetric. In the first case, by induction, there merely exists an a such that $a \in x_1$ and $b \equiv f(a)$. Since the former of these implies $a \in x$, this implies the proof of the case. □

5 GUARDED LABELLED TRANSITION SYSTEMS

In this section, we show how to represent a guarded version of finitely branching labelled transition systems in **TCTT**. From now on, we omit the attribute "finitely branching" since this will be the only kind of system we will consider in this paper. As we will see in Section 6, bisimulation for final guarded coalgebras coincides with path equality, and labelled transition systems provide an interesting test case of this. In particular, we shall see that proving bisimilarity of processes can be done using simple equational reasoning in combination with guarded recursion.

A *guarded labelled transition system*, or *GLTS* for short, consists of a type X of states, a type A of actions (which we will assume small in the sense that $A : \mathsf{U}$) and a function $f : X \to \mathsf{P_{fin}}(A \times \triangleright X)$. Given a state $x : X$, a later state $y : \triangleright X$ and an action $a : A$, we write $x \xrightarrow{a}_f y$ for $(a, y) \in f\ x$.

Example 5.1. Consider the GLTS described pictorially as the following labelled directed graph:

$$\begin{tikzcd}
x_0 \arrow[dr, bend left, "\mathsf{ff}"] \arrow[dd, bend left, "\mathsf{ff}"] & & & y_0 \arrow[dr, "\mathsf{ff}"] & \\
& x_1 \arrow[ul, bend left, "\mathsf{tt}"] \arrow[dl, "\mathsf{ff}"] & & & y_1 \arrow[loop right, "\mathsf{ff}"] \arrow[dl, bend left, "\mathsf{tt}"] \\
x_2 \arrow[uu, bend left, "\mathsf{tt}"] \arrow[loop below, "\mathsf{ff}"] & & & y_2 \arrow[ur, bend left, "\mathsf{ff}"] &
\end{tikzcd}$$

This can be implemented as the GLTS with X being the inductive type with six constructors $x_0, x_1, x_2, y_0, y_1, y_2 : X$, with A being the inductive type with constructors $\mathsf{ff}, \mathsf{tt} : A$, and

$$\begin{aligned}
&f : X \to \mathsf{P_{fin}}(A \times \triangleright X)\\
&f\, x_0 = \{(\mathsf{ff}, \mathsf{next}\, x_1)\} \cup \{(\mathsf{ff}, \mathsf{next}\, x_2)\}\\
&f\, x_1 = \{(\mathsf{tt}, \mathsf{next}\, x_0)\} \cup \{(\mathsf{ff}, \mathsf{next}\, x_2)\}\\
&f\, x_2 = \{(\mathsf{tt}, \mathsf{next}\, x_0)\} \cup \{(\mathsf{ff}, \mathsf{next}\, x_2)\}\\
&f\, y_0 = \{(\mathsf{ff}, \mathsf{next}\, y_1)\}\\
&f\, y_1 = \{(\mathsf{ff}, \mathsf{next}\, y_1)\} \cup \{(\mathsf{tt}, \mathsf{next}\, y_2)\}\\
&f\, y_2 = \{(\mathsf{ff}, \mathsf{next}\, y_1)\}
\end{aligned}$$

One is typically interested in *final semantics*, i.e. all possible runs, or *processes*, of a certain GLTS. These are obtained by unwinding a GLTS starting from a particular state. In categorical terms, the type of processes is given by the final coalgebra of the functor $\mathsf{P_{fin}}\,(A \times \triangleright(-))$, which can be defined in **TCTT** as a guarded recursive type using the fixpoint operation:

$$\mathsf{Proc} = \mathsf{fix}\ X.\, \mathsf{P_{fin}}(A \times \triangleright (\alpha : \mathbb{T}).X\,[\alpha]).$$

Note that the mapping of X to $\mathsf{P_{fin}}(A \times \triangleright (\alpha : \mathbb{T}).X\,[\alpha])$ has type $\triangleright \mathsf{U} \to \mathsf{U}$ because the universe is closed under $\triangleright$ and finite powersets as assumed above. As was the case for the guarded streams example of Section 3, the fixed point path induces an equivalence of types witnessed in one direction by $\mathsf{unfold} : \mathsf{Proc} \to \mathsf{P_{fin}}(A \times \triangleright \mathsf{Proc})$, and thus $(\mathsf{Proc}, A, \mathsf{unfold})$ is a GLTS.

Given a GLTS (X, A, f), the process associated to a state is defined as the map $[\![-]\!] : X \to \mathsf{Proc}$ defined using the final coalgebra property of Proposition 3.2. In the case of Example 5.1 it is also possible to define the processes associated to the GLTS (X, A, f) directly via mutual guarded recursion, without using the evaluation function $[\![-]\!]$. We can do this since the type X is inductively defined.

Example 5.1 (continued). Let ps and qs be the elements of type $\mathsf{Proc} \times \mathsf{Proc} \times \mathsf{Proc}$ given as follows:

$$\begin{aligned}
ps = {}& \mathsf{fix}\, zs.\\
&(\mathsf{fold}\,(\{(\mathsf{ff}, \triangleright\pi_1\, zs)\} \cup \{(\mathsf{ff}, \triangleright\pi_2\, zs)\}),\\
&\ \ \mathsf{fold}\,(\{(\mathsf{tt}, \triangleright\pi_0\, zs)\} \cup \{(\mathsf{ff}, \triangleright\pi_2\, zs)\}),\\
&\ \ \mathsf{fold}\,(\{(\mathsf{tt}, \triangleright\pi_0\, zs)\} \cup \{(\mathsf{ff}, \triangleright\pi_2\, zs)\}))\\
qs = {}& \mathsf{fix}\, zs.\\
&(\mathsf{fold}\,\{(\mathsf{ff}, \triangleright\pi_1\, zs)\},\\
&\ \ \mathsf{fold}\,(\{(\mathsf{ff}, \triangleright\pi_1\, zs)\} \cup \{(\mathsf{tt}, \triangleright\pi_2\, zs)\}),\\
&\ \ \mathsf{fold}\,\{(\mathsf{ff}, \triangleright\pi_1\, zs)\})
\end{aligned}$$

We define $p_i = \pi_i\, ps$ and $q_i = \pi_i\, qs$, for $i = 0, 1, 2$. Using the fixed point operator, one can prove that the processes p_i and $[\![x_i]\!]$ are path equal, and similarly the process q_i and $[\![y_i]\!]$ are path equal. For this, let $D = \mathsf{Path}\, p_0\, [\![x_0]\!] \times \mathsf{Path}\, p_1\, [\![x_1]\!] \times \mathsf{Path}\, p_2\, [\![x_2]\!]$, and assume $e : \triangleright D$. Then

$$p_0 \equiv \mathsf{fold}\,(\{(\mathsf{ff}, \mathsf{next}\, p_1)\} \cup \{(\mathsf{ff}, \mathsf{next}\, p_2)\}) \equiv \mathsf{fold}\,(\{(\mathsf{ff}, \mathsf{next}[\![x_1]\!])\} \cup \{(\mathsf{ff}, \mathsf{next}[\![x_2]\!])\}) \equiv [\![x_0]\!]$$

where the middle path is obtained by first applying $\triangleright\pi_1$ to e to produce a term in $\triangleright(\mathsf{Path}\, p_1\, [\![x_1]\!])$, then using the extensionality principle (4) for $\triangleright$ to give an element in $\mathsf{Path}\,(\mathsf{next}\, p_1)\,(\mathsf{next}\, [\![x_1]\!])$. Analogously, one constructs a path between $\mathsf{next}\, p_2$ and $\mathsf{next}\, [\![x_2]\!]$.

Similarly, one shows that p_1 and p_2 are path equal to $[\![x_1]\!]$ and $[\![x_2]\!]$. Piecing these proofs together we get an inhabitant of $t : D$, and thus $\mathsf{fix}\, e.t$ is an element in D proving the desired path equalities.

5.1 Bisimulation for GLTSs

The natural notion of equality for two processes in a labelled transition system is bisimilarity. We now state a notion of bisimilarity for the notion of GLTS used in this paper. Let (X, A, f) be a GLTS and let $R : X \to X \to \mathsf{U}$ be a relation. Then R is a *guarded bisimulation* iff the following type is inhabited:

$$\begin{aligned}&\mathsf{isGLTSBisim}_f\, R = \Pi\, x\, y : X.\, R\, x\, y \to \\ &\quad (\Pi\, x' : \triangleright X.\, \Pi\, a : A.\, (a, x') \in f\, x \to \exists\, y' : \triangleright X.\, (a, y') \in f\, y \times \triangleright (\alpha : \mathbb{T}).R\, (x'\, [\alpha])\, (y'\, [\alpha])) \\ &\qquad\qquad\times \\ &\quad (\Pi\, y' : \triangleright X.\, \Pi\, a : A.\, (a, y') \in f\, y \to \exists\, x' : \triangleright X.\, (a, x') \in f\, x \times \triangleright (\alpha : \mathbb{T}).R\, (x'\, [\alpha])\, (y'\, [\alpha]))\end{aligned}$$

Notice that what we call guarded bisimulation is just the guarded recursive variant of the usual notion of bisimulation for labelled transition systems. In words, a relation R is a bisimulation if whenever two states x and y are related by R, two conditions hold: for all transitions $x \xrightarrow{a}_f x'$ there exists a transition $y \xrightarrow{a}_f y'$ such that the later states x' and y' are later related by R; for all transitions $y \xrightarrow{a}_f y'$ there exists a transition $x \xrightarrow{a}_f x'$ such that the later states x' and y' are later related by R . Notice the use of the existential quantifier $\exists$ instead of Σ here. This is necessary for the proofs of Proposition 5.2 and Theorem 5.3 below.

Proposition 5.2. *Let (X, A, f) be a GLTS. Then the relation R defined as $R\, x\, y = \mathsf{Path}_{\mathsf{Proc}} [\![x]\!] [\![y]\!]$ is a bisimulation.*

Proof. Suppose $R\, x\, y$ and that $(a, x') \in f(x)$. We show that there merely exists a y' such that $(a, y') \in f(y)$ and $\triangleright (\alpha : \mathbb{T}).R\, (x'\, [\alpha])\, (y'\, [\alpha]))$. The other direction is proved similarly. By Lemma 4.1 $(A \times \triangleright[\![-]\!])(a, x')$ is in the finite set $\mathsf{P}_{\mathsf{fin}}(A \times \triangleright[\![-]\!])(f(x))$. By definition of $[\![-]\!]$ as the unique map of coalgebras, and the assumption that $[\![x]\!] \equiv [\![y]\!]$ we get

$$\begin{aligned}\mathsf{P}_{\mathsf{fin}}(A \times \triangleright[\![-]\!])(f(x)) &\equiv \mathsf{unfold}([\![x]\!]) \\ &\equiv \mathsf{unfold}([\![y]\!]) \\ &\equiv \mathsf{P}_{\mathsf{fin}}(A \times \triangleright[\![-]\!])(f(y))\end{aligned}$$

By Lemma 4.1 the property $(A \times \triangleright[\![-]\!])(a, x') \in \mathsf{P}_{\mathsf{fin}}(A \times \triangleright[\![-]\!])(f(y))$ implies the mere existence of a z such that $(A \times \triangleright[\![-]\!])z \equiv (A \times \triangleright[\![-]\!])(a, x')$ and $z \in f(y)$. Thus z must be of the form (a, y'), such that

$$\begin{aligned}\lambda(\alpha : \mathbb{T}).[\![y'\, [\alpha]]\!] &= \triangleright[\![-]\!](y') \\ &\equiv \triangleright[\![-]\!](x') \\ &= \lambda(\alpha : \mathbb{T}).[\![x'\, [\alpha]]\!]\, .\end{aligned}$$

By the extensionality principle (3) for $\triangleright$ this implies $\triangleright (\alpha : \mathbb{T}).([\![x'\, [\alpha]]\!] \equiv [\![y'\, [\alpha]]\!])$, which by definition is $\triangleright (\alpha : \mathbb{T}).R\, (x'\, [\alpha])\, (y'\, [\alpha]))$ as desired. □

We can also define the greatest guarded bisimulation on a GLTS (X, A, f) by guarded recursion.

$$\begin{aligned}&\sim_f = \mathsf{fix}\, R.\, \lambda x\, y : X. \\ &\quad (\Pi\, x' : \triangleright X.\, \Pi\, a : A.\, (a, x') \in f\, x \to \exists\, y' : \triangleright X.\, (a, y') \in f\, y \times \triangleright (\alpha : \mathbb{T}).R\, [\alpha]\, (x'\, [\alpha])\, (y'\, [\alpha])) \\ &\qquad\qquad\times \\ &\quad (\Pi\, y' : \triangleright X.\, \Pi\, a : A.\, (a, y') \in f\, y \to \exists\, x' : \triangleright X.\, (a, x') \in f\, x \times \triangleright (\alpha : \mathbb{T}).R\, [\alpha]\, (x'\, [\alpha])\, (y'\, [\alpha]))\end{aligned}$$

This is indeed the greatest bisimulation in the sense that if $R\, x\, y$ holds for some f-bisimulation R, then also $x \sim_f y$ holds as can be proved by guarded recursion. We call this relation *guarded* *bisimilarity.* When considering the GLTS $(\mathsf{Proc}, A, \mathsf{unfold})$, we write $\sim$ for the guarded bisimilarity relation $\sim_{\mathsf{unfold}}$. Notice that the type $x \sim_f y$ is a proposition, for all $x, y : X$ and $f : X \to \mathsf{P}_{\mathsf{fin}}(A \times \triangleright X)$.

In **TCTT**, the coinduction proof principle can be stated as follows. Given a guarded bisimulation R for the GLTS $(\mathsf{Proc}, A, \mathsf{unfold})$, if two processes are related by R then they are path equal, i.e. for all $p, q : \mathsf{Proc}$ we have an implication $R\, p\, q \to \mathsf{Path}_{\mathsf{Proc}}\, p\, q$. The coinduction proof principle is derivable from the following theorem, which is a well-known result in the theory of coalgebras developed in set theory [Rutten 2000].

Theorem 5.3. *Let (X, A, f) be a GLTS. For all $x, y : X$, the types $x \sim_f y$ and $\mathsf{Path}_{\mathsf{Proc}} [\![x]\!] [\![y]\!]$ are path equal.*

Proof. By univalence, it sufficies to show that the two propositions are logically equivalent. The right-to-left implication follows from Proposition 5.2 and the fact that $\sim_f$ is the greatest guarded bisimulation on (X, A, f). For the left-to-right implication, we proceed by guarded recursion. Suppose $e : \triangleright(\Pi x, y : X. x \sim_f y \to [\![x]\!] \equiv [\![y]\!])$, and let $x, y : X$ such that $x \sim_f y$ holds. We construct a proof of $[\![x]\!] \equiv [\![y]\!]$. Note that the type $x \sim_f y$ is equivalent to the following type:

$$\begin{gathered}(\Pi\, x' : \triangleright X. \Pi\, a : A. (a, x') \in f\, x \to \exists\, y' : \triangleright X. (a, y') \in f\, y \times \triangleright (\alpha : \mathbb{T}).(x' [\alpha] \sim_f y' [\alpha])) \\ \times \\ (\Pi\, y' : \triangleright X. \Pi\, a : A. (a, y') \in f\, y \to \exists\, x' : \triangleright X. (a, x') \in f\, x \times \triangleright (\alpha : \mathbb{T}).(x' [\alpha] \sim_f y' [\alpha]))\end{gathered}$$

By the assumption e, there is an implication:

$$\triangleright (\alpha : \mathbb{T}).(x' [\alpha] \sim_f y' [\alpha]) \to \triangleright (\alpha : \mathbb{T}).([\![x' [\alpha]]\!] \equiv [\![y' [\alpha]]\!])$$

which, by extensionality for $\triangleright$, is equal to the type:

$$\triangleright (\alpha : \mathbb{T}).(x' [\alpha] \sim_f y' [\alpha]) \to \triangleright [\![-]\!](x') \equiv \triangleright [\![-]\!](y')$$

So the first line of the unfolding of $x \sim_f y$ implies

$$(\Pi\, x' : \triangleright X. \Pi\, a : A. (a, x') \in f\, x \to \exists\, y' : \triangleright X. (a, y') \in f\, y \times \triangleright [\![-]\!](x') \equiv \triangleright [\![-]\!](y')). \tag{5}$$

We show that this in turn implies

$$\mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, x) \subseteq \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y).$$

In fact, suppose given an inhabitant h of type (5) above. Let $a : A$ and $p : \triangleright\mathsf{Proc}$ such that $(a, p) \in \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, x)$. By Lemma 4.1 this implies the mere existence of a z such that $z \in f\, x$ and $(A \times \triangleright [\![-]\!]) z \equiv (a, p)$. Thus z must be of the form (a, x') such that $p \equiv \triangleright [\![-]\!] x'$. By the assumption h, there merely exists $y' : \triangleright X$ such that $(a, y') \in f\, y$ and $\triangleright [\![-]\!] x' \equiv \triangleright [\![-]\!] y'$. By Lemma 4.1, this implies $(A \times \triangleright [\![-]\!])(a, y') \in \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y)$. Since $(A \times \triangleright [\![-]\!])(a, y') \equiv (a, p)$ we conclude that $(a, p) \in \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y)$.

Similarly the second line of the unfolding of $x \sim_f y$ implies $\mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y) \subseteq \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, x)$. Therefore, by extensionality for finite powersets, $x \sim_f y$ implies $\mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, x) \equiv \mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y)$, which in turn implies

$$[\![x]\!] \equiv \mathsf{fold}\,(\mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, x)) \equiv \mathsf{fold}\,(\mathsf{P}_{\mathsf{fin}}(A \times \triangleright [\![-]\!])(f\, y)) \equiv [\![y]\!]. \qquad \square$$

Corollary 5.4. *For all $p, q : \mathsf{Proc}$, the types $p \sim q$ and $p \equiv q$ are path equal.*

Proof. This follows from Theorem 5.3 and the fact that the unique map of coalgebras $[\![-]\!] : \mathsf{Proc} \to \mathsf{Proc}$ is path equal to the identity function. □

Example 5.1 (continued). It is not difficult to see that the processes p_0 and q_0 (or alternatively, $[\![x_0]\!]$ and $[\![y_0]\!]$) are bisimilar, and therefore equal by the coinduction proof principle. But it is simpler to prove them equal directly by guarded recursion. Note that by Proposition 5.2 this implies x_0

and y_0 bisimilar. To do this, let $D = \mathsf{Path}\,p_0\,q_0 \times \mathsf{Path}\,p_1\,q_1 \times \mathsf{Path}\,p_0\,q_2 \times \mathsf{Path}\,p_2\,q_1$. Assuming $e : \triangleright D$, we must construct four proofs, one for each path type in D. We just construct a term e_0 of type $\mathsf{Path}\,p_0\,q_0$, the other proofs are given in a similar manner. We have the following sequence of equalities:

$$\begin{aligned} p_0 &\equiv \mathsf{fold}\,(\{(\mathsf{ff}, \mathsf{next}\,p_1)\} \cup \{(\mathsf{ff}, \mathsf{next}\,p_2)\}) \\ &\equiv \mathsf{fold}\,(\{(\mathsf{ff}, \mathsf{next}\,q_1)\} \cup \{(\mathsf{ff}, \mathsf{next}\,q_1)\}) \\ &\equiv \mathsf{fold}\,\{(\mathsf{ff}, \mathsf{next}\,q_1)\} \\ &\equiv q_0 \end{aligned}$$

where the second equality follows from the assumption $\triangleright D$ which by (4) implies $\mathsf{next}\,p_1 \equiv \mathsf{next}\,q_1$ and $\mathsf{next}\,p_2 \equiv \mathsf{next}\,q_1$.

5.2 CCS

As an extended example of a GLTS we now show how to represent the syntax of Milner's Calculus of Communicating Systems (CCS) [Milner 1980]. We consider a version with guarded recursion, i.e., processes can be defined recursively with the restriction that the recursive variable may only occur under an action. More precisely, we consider the grammar

$$P ::= 0 \mid a.P \mid P + P \mid P \| P \mid \nu a.P \mid X \mid \mu X.P$$

with the restriction that in $\mu X.P$, the variable X may only occur under an action $a.(-)$. For example, $\mu X.a.X$ is a well formed process, but $\mu X.(P \| X)$ (which would correspond to replication $!P$ as in the π-calculus [Milner 1999]) is not.

For simplicity, we use a De Bruijn representation of processes. Names will be simply numbers and we will define, for each $n : \mathbb{N}$ a type $\mathsf{CCS}(n) : \mathsf{U}$ of closed CCS terms whose freely occurring names are among the first n numbers. For this we use the inductive family $\mathsf{Fin} : \mathbb{N} \to \mathsf{U}$, where $\mathsf{Fin}(n)$ contains all natural numbers strictly smaller than n. The type of actions is then $\mathsf{Act}(n) = \mathsf{Fin}(n) + \mathsf{Fin}(n) + 1$. Following standard conventions, we write simply m for $\mathsf{in}_1(m)$, $\bar{m}$ for $\mathsf{in}_2(m)$ and τ for $\mathsf{in}_3(\star)$.

The inductive family of *closed* CCS terms should satisfy the type equivalence

$$\mathsf{CCS}(n) \simeq 1 + \mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n) + \mathsf{CCS}(n) \times \mathsf{CCS}(n) + \mathsf{CCS}(n) \times \mathsf{CCS}(n) + \mathsf{CCS}(n+1) \qquad (6)$$

stating that a closed CCS term can either be 0, an action, a binary sum, a parallel composition or a name abstraction. The use of $\triangleright$ in the case of actions allows for the definition of guarded recursive processes, e.g., $\mu X.a.X$ can be represented as $\mathsf{fix}\,x.\mathsf{in}_2(a, x)$. In the case of name abstraction, the abstracted process can have one more free name than the result.

To define CCS, first consider

$$F : \triangleright(\mathbb{N} \to \mathsf{U}) \to (\mathbb{N} \to \mathsf{U}) \to \mathbb{N} \to \mathsf{U}$$

defined as

$$F\,X\,Y\,n = 1 + \mathsf{Act}(n) \times \triangleright(\alpha : \mathbb{T}).(X\,[\alpha](n)) + Y(n) \times Y(n) + Y(n) \times Y(n) + Y(n+1)$$

and define

$$\mathsf{CCS} = \mathsf{fix}\,X.\mu Y.F\,X\,Y$$

where μ refers to the inductive family type former. Inductive types and families are special cases of HITs, which the universe is closed under by assumption. Equation (6) is then satisfied by unfolding the fixed point and inductive family once.

By definition of fix, we have

$$\mathsf{CCS} = \mu Y.F\,(\mathsf{CCS}')\,Y$$

where $\mathsf{CCS}' = \mathsf{dfix}\, X.\mu Y.F\,X\,Y$. Using the path from $\mathsf{dfix}\, X.\mu Y.F\,X\,Y$ to $\mathsf{next}(\mathsf{fix}\, X.\mu Y.F\,X\,Y)$ we obtain a type equivalence $\triangleright(\alpha : \mathbb{T}).\mathsf{CCS}'\,[\alpha](n) \simeq \triangleright\mathsf{CCS}(n)$. Let $g : \triangleright(\alpha : \mathbb{T}).\mathsf{CCS}'\,[\alpha](n) \to \triangleright\mathsf{CCS}(n)$ be the map underlying this equivalence.

The set of CCS actions is recursively defined as a GLTS $\mathsf{act} : \mathsf{CCS}(n) \to \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n))$

$$\begin{aligned}
\mathsf{act}(0) &= \varnothing \\
\mathsf{act}(a.P) &= \{(a, g\,P)\} \\
\mathsf{act}(P_1 + P_2) &= \mathsf{act}(P_1) \cup \mathsf{act}(P_2) \\
\mathsf{act}(P_1 \| P_2) &= \mathsf{act_L}(\mathsf{act}(P_1), P_2) \cup \mathsf{act_R}(P_1, \mathsf{act}(P_2)) \cup \mathsf{synch}(\mathsf{act}(P_1), \mathsf{act}(P_2)) \\
\mathsf{act}(\nu a.P) &= \mathsf{act}_\nu(\mathsf{act}(P))
\end{aligned}$$

The auxiliary functions

$$\begin{aligned}
\mathsf{act_L} &: \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n)) \times \mathsf{CCS}(n) \to \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n)) \\
\mathsf{act_R} &: \mathsf{CCS}(n) \times \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n)) \to \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n))
\end{aligned}$$

are given by

$$\mathsf{act_L}(u, P) = \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright(-\|P))(u) \qquad \mathsf{act_R}(P, u) = \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright(P\|-))(u).$$

The auxiliary function

$$\mathsf{synch} : \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n))^2 \to \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n))$$

is recursively defined as

$$\begin{aligned}
\mathsf{synch}(\varnothing, v) &= \varnothing \\
\mathsf{synch}(\{(a, P)\}, \varnothing) &= \varnothing \\
\mathsf{synch}(\{(m, P)\}, \{(\bar{m}, Q)\}) &= \{(\tau, \lambda(\alpha : \mathbb{T}).P\,[\alpha] \| Q\,[\alpha])\} \\
\mathsf{synch}(\{(\bar{m}, P)\}, \{(m, Q)\}) &= \{(\tau, \lambda(\alpha : \mathbb{T}).P\,[\alpha] \| Q\,[\alpha]\} \\
\mathsf{synch}(\{(a, P)\}, \{(b, Q)\}) &= \varnothing \qquad \text{(if } a \text{ and } b \text{ do not fit the previous two cases)} \\
\mathsf{synch}(\{(a, P)\}, v_1 \cup v_2) &= \mathsf{synch}(\{(a, P)\}, v_1) \cup \mathsf{synch}(\{(a, P)\}, v_2) \\
\mathsf{synch}(u_1 \cup u_2, v) &= \mathsf{synch}(u_1, v) \cup \mathsf{synch}(u_2, v)
\end{aligned}$$

In the definition of synch we omitted the cases for the higher path constructors, which are straightforward.

Finally, the auxiliary function

$$\mathsf{act}_\nu : \mathsf{P_{fin}}(\mathsf{Act}(n+1) \times \triangleright\mathsf{CCS}(n+1)) \to \mathsf{P_{fin}}(\mathsf{Act}(n) \times \triangleright\mathsf{CCS}(n))$$

is recursively defined as

$$\begin{aligned}
\mathsf{act}_\nu \varnothing &= \varnothing & \mathsf{act}_\nu(u_1 \cup u_2) &= \mathsf{act}_\nu u_1 \cup \mathsf{act}_\nu u_2 \\
\mathsf{act}_\nu \{(n, P)\} &= \varnothing & \mathsf{act}_\nu \{(\bar{n}, P)\} &= \varnothing \\
\mathsf{act}_\nu \{(b, P)\} &= \{(b, \lambda(\alpha : \mathbb{T}).\, \nu a.P\,[\alpha])\} & &(\text{if } b \neq n, \bar{n})
\end{aligned}$$

Again we omitted the cases for the higher path constructors.

Instantiating Theorem 5.3 with the GLTS $(\mathsf{CCS}(n), \mathsf{Act}(n), \mathsf{act})$, we obtain the following result.

Theorem 5.5. *Let $p, q : \mathsf{CCS}(n)$, then the types $p \sim_{\mathsf{act}} q$ and $[\![p]\!] \equiv [\![q]\!]$ are equivalent.*

5.3 Hennessy-Milner Logic

Basic properties of GLTSs can be expressed in Hennessy-Milner logic [Hennessy and Milner 1980]. The grammar for propositions

$$\phi ::= \mathsf{tt} \mid \mathsf{ff} \mid \phi \wedge \phi \mid \phi \vee \phi \mid [a]\,\phi \mid \langle a\rangle\,\phi$$

(where a ranges over the alphabet A) can be encoded as an inductive type

$$\mathsf{HML} = \mu X.1 + 1 + X \times X + X \times X + A \times X + A \times X$$

in the standard way. We shall use Hennessy-Milner logic as notation for this type, e.g., writing $[a]\,\phi$ for $\mathsf{in}_5\,(a, \phi)$ whenever $a : A$ and $\phi : \mathsf{HML}$. For any GLTS (X, A, f) we define the satisfiability relation $\models : X \to \mathsf{HML} \to \mathsf{Prop}$ by recursion on the second argument:

$$\begin{aligned}
x &\models \mathsf{tt} \leftrightarrow 1\\
x &\models \mathsf{ff} \leftrightarrow 0\\
x &\models \phi \wedge \psi \leftrightarrow (x \models \phi \times x \models \psi)\\
x &\models \phi \vee \psi \leftrightarrow (x \models \phi \vee x \models \psi)\\
x &\models [a]\,\psi \leftrightarrow \Pi x' : \triangleright X.(a, x') \in f(x) \to \triangleright(\alpha : \mathbb{T}).(x'\,[\alpha]) \models \phi\\
x &\models \langle a\rangle\,\psi \leftrightarrow \exists x' : \triangleright X.(a, x') \in f(x) \wedge \triangleright(\alpha : \mathbb{T}).(x'\,[\alpha]) \models \phi
\end{aligned}$$

Since bisimilarity for Proc coincides with equality, it is obvious that two bisimilar processes will satisfy the same propositions from Hennessy-Milner logic. It is a classical result [Hennessy and Milner 1980], that Hennessy-Milner logic is also complete in the sense that any two processes satisfying the same propositions are also bisimilar. The argument uses classical logic, and it is unlikely that it can be reproduced in type theory. Still, one can use propositions to distinguish between processes.

Example 5.6. Suppose A has elements a, b, c such that $\neg(b \equiv c)$ and consider the two processes expressed in CCS terms as

$$p = a.(b + c) \qquad q = a.b + a.c$$

and encoded as elements of Proc as

$$\begin{aligned}
p &= \{(a, \mathsf{next}(\{(b, \mathsf{next}\,\varnothing)\} \cup \{(c, \mathsf{next}\,\varnothing)\}))\}\\
q &= (\{(a, \mathsf{next}\{(b, \mathsf{next}\,\varnothing)\})\} \cup \{(a, \mathsf{next}\{(c, \mathsf{next}\,\varnothing)\})\})
\end{aligned}$$

In the definition above we omitted application of the function fold to improve readability. It is a classical result that these are not bisimilar, but in the guarded setting, the type $p \equiv q$ is not false, but logically equivalent to $\triangleright 0$, which can be proved as follows. Suppose $p \equiv q$, then since p satisfies $[a]\,\langle b\rangle\,\mathsf{tt}$, so does q. Since $(a, \mathsf{next}(\{(c, \mathsf{next}(\varnothing))\}) \in \mathsf{unfold}(q)$ this implies

$$\triangleright(\exists x' : \triangleright X.(b, x') \in \{(c, \mathsf{next}(\varnothing))\})$$

Reasoning under the $\triangleright$, this implies $\triangleright(\exists x' : \triangleright X.b \equiv c)$ which implies $\triangleright 0$.

In the opposite direction, we must prove that $\triangleright 0$ implies $p \equiv q$. By the extensionality principle (3) for $\triangleright$, the proposition $\triangleright 0$ implies $\mathsf{next}(x) \equiv \mathsf{next}(y)$ for all $x, y : X$, and thus $p \equiv \{(a, \mathsf{next}(\varnothing))\} \equiv q$.

6 GUARDED COALGEBRAS

We now move from finitely branching labelled transition systems to more general kind of systems. These are specified by a guarded recursive version of coalgebras, that we call guarded coalgebras. We define bisimulation for these systems and prove a coinduction proof principle: the greatest bisimulation is equivalent to path equality.

Let $F : \mathsf{U} \to \mathsf{U}$ be a functor. A *guarded coalgebra* for F is a small type X together with a function $f : X \to F(\triangleright X)$. Analogously, we could say that a guarded coalgebra is a coalgebra for the composed functor $F \circ \triangleright$. Proposition 3.2 states that the fixed point $\nu F^{\mathrm{g}} = \mathsf{fix}\, X.\, F(\triangleright(\alpha : \mathbb{T}).X\,[\alpha])$ is a final guarded coalgebra for F.

We now move to the representation of bisimulations. In the literature there exist several different notions of coalgebraic bisimulation [Staton 2009]. Here we consider the variant introduced by Hermida and Jacobs [1998], which relies on the notion of relation lifting [Kurz and Velebil 2016]. We adapt this notion to type theory in a way that is not directly a generalisation of isGLTSBisim as used in the previous section. Rather, we will show in Section 7 that the latter is a propositionally truncated version of the notion defined here. The truncated version is convenient for GLTSs because of the set-truncation used in the powerset functor, but truncating the general notion would falsify the equivalence of bisimilarity of paths as stated in Corollary 6.4 below.

6.1 Relation Lifting

Given a (proof-relevant) relation $R : X \to Y \to \mathsf{U}$, the *relation lifting* $\overline{F}R : FX \to FY \to \mathsf{U}$ of F on R is defined as

$$\overline{F}R\,x\,y \;=\; \Sigma t : F(\mathsf{tot}\,R).\, \mathsf{Path}_{FX}\,(F\pi_0\,t)\,x \times \mathsf{Path}_{FY}\,(F\pi_1\,t)\,y$$

for $x : FX$ and $y : FY$. Here $\mathsf{tot}\,R$ is the graph of R, i.e., $\mathsf{tot}\,R = \Sigma x : X.\,\Sigma y : Y.\,R\,x\,y$, and π_0 and π_1 refer to the projections out of the dependent product.

We first show that the relation lifting of F applied to the identity relation Path_X is path equal to Path_{FX}. A proof of this fact in the classical set theoretic setting can be found in [Jacobs 2016]. Here we adapt the proof to type theory and prove it in the general setting of types that are not necessarily sets.

Proposition 6.1. *For all $x, y : FX$, the types $\overline{F}(\mathsf{Path}_X)\,x\,y$ and $\mathsf{Path}_{FX}\,x\,y$ are path equal.*

Proof. We have the following sequence of equalities:

$$\begin{aligned}
\overline{F}(\mathsf{Path}_X)\,x\,y &= \Sigma t : F(\mathsf{tot}\,(\mathsf{Path}_X)).\,\mathsf{Path}_{FX}(F\pi_0\,t)\,x \times \mathsf{Path}_{FX}(F\pi_1\,t)\,y \\
&\equiv \Sigma t : F(\mathsf{tot}\,(\mathsf{Path}_X)).\,\mathsf{Path}_{FX}(F\pi_0\,t)\,x \times \mathsf{Path}_{FX}(F\pi_0\,t)\,y & (7)\\
&\equiv \Sigma t : FX.\,\mathsf{Path}_{FX}\,t\,x \times \mathsf{Path}_{FX}\,t\,y & (8)\\
&\equiv \mathsf{Path}_{FX}\,x\,y & (9)
\end{aligned}$$

Equality (7) follows from the fact that the types $\mathsf{Path}_{FX}(F\pi_1\,t)\,y$ and $\mathsf{Path}_{FX}(F\pi_0\,t)\,y$ are path equal. This in turns follows from Lemma 2.1 and the existence of a path:

$$\lambda i.\, F(\lambda s.\,\pi_2\,s\,i)\,t : \mathsf{Path}_{FX}\,(F\pi_0\,t)\,(F\pi_1\,t).$$

Equality (8) follows from the univalence axiom and Lemma 2.2 instantiated with the function $f : F(\mathsf{tot}\,(\mathsf{Path}_X)) \to FX$, $f = F\pi_0$. The map f is an equivalence since $\pi_0 : \mathsf{tot}\,(\mathsf{Path}_X) \to X$ is an equivalence and functors preserve equivalences. The map π_0 is an equivalence with inverse $h = \lambda x.\,(x, x, \mathsf{refl}\,x) : X \to \mathsf{tot}\,(\mathsf{Path}_X)$.

Equality (9) follows from the univalence axiom and the existence of an equivalence

$$\lambda p.\,(x, \mathsf{refl}\,x, p) : \mathsf{Path}_{FX}\,x\,y \to \Sigma t : FX.\,\mathsf{Path}_{FX}\,t\,x \times \mathsf{Path}_{FX}\,t\,y$$

with inverse

$$\lambda(t,p,q).p^{-1}; q : \Sigma t : FX.\,\mathsf{Path}_{FX}\,t\,x \times \mathsf{Path}_{FX}\,t\,y \to \mathsf{Path}_{FX}\,x\,y \qquad \square$$

We next show that the mapping of R to $\overline{(F \circ \triangleright)}(R)$ factors through next, a property that allows for the notion of guarded bisimilarity to be defined below as a guarded fixed point. For this, given

$R : \triangleright(X \to Y \to \mathsf{U})$, write $R^{\triangleright} : \triangleright X \to \triangleright Y \to U$ for the relation $R^{\triangleright} x\, y = \triangleright(\alpha : \mathbb{T}).R\,[\alpha]\,(x\,[\alpha])\,(y\,[\alpha])$. Note that the extensionality principle (3) for the type former $\triangleright$ can be expressed by saying that the type $(\mathsf{next}\,\mathsf{Path}_X)^{\triangleright}$ is path equal to $\mathsf{Path}_{\triangleright X}$.

Lemma 6.2. *For any functor F, relation $R : X \to Y \to \mathsf{U}$, and elements $x : F(\triangleright X)$ and $y : F(\triangleright Y)$, the types $\overline{(F \circ \triangleright)}(R)\,x\,y$ and $\overline{F}((\mathsf{next}\,R)^{\triangleright})\,x\,y$ are equivalent.*

Proof. First note that

$$\triangleright(\mathsf{tot}\,R) \simeq \Sigma u : \triangleright X.\Sigma v : \triangleright Y.\triangleright(\alpha : \mathbb{T}).R(u\,[\alpha], v\,[\alpha])$$

by the maps mapping $t : \triangleright(\mathsf{tot}\,R)$ to $(\triangleright(\pi_0)(t), \triangleright(\pi_1)(t), \triangleright(\pi_2)(t))$ and

$$(u, v, p) : \Sigma u : \triangleright X.\Sigma v : \triangleright Y.\triangleright(\alpha : \mathbb{T}).R(u\,[\alpha], v\,[\alpha])$$

to $\lambda(\alpha : \mathbb{T}).(u\,[\alpha], v\,[\alpha], p\,[\alpha])$. Using this and Lemma 2.2, we get

$$\begin{aligned}\overline{(F \circ \triangleright)}(R)\,x\,y &= \Sigma t : F(\triangleright(\mathsf{tot}\,R)).\,\mathsf{Path}_{F(\triangleright X)}\,(F(\triangleright(\pi_0))\,t)\,x \times \mathsf{Path}_{F(\triangleright Y)}\,(F(\triangleright(\pi_1))\,t)\,y\\ &\simeq \Sigma t : F(\mathsf{tot}\,(\mathsf{next}\,R)^{\triangleright}).\,\mathsf{Path}_{F(\triangleright X)}\,(F(\pi_0)\,t)\,x \times \mathsf{Path}_{F(\triangleright Y)}\,(F(\pi_1)\,t)\,y\\ &= \overline{F}((\mathsf{next}\,R)^{\triangleright})\,x\,y\end{aligned}$$

□

6.2 Bisimulation for Guarded Coalgebras

We now implement the notion of coalgebraic bisimulation of Hermida and Jacobs [1998]. For a given guarded coalgebra $f : X \to F(\triangleright X)$, we call a relation $R : X \to X \to \mathsf{U}$ a *guarded bisimulation* iff the following type is inhabited:

$$\mathsf{is}\text{-}F\text{-}\mathsf{Bisim}_f\,R = \Pi\,x\,y : X.\,R\,x\,y \to \overline{(F \circ \triangleright)}(R)\,(f\,x)\,(f\,y)$$

Given a guarded coalgebra $f : X \to F(\triangleright X)$, *guarded bisimilarity* for X is the greatest bisimulation on X defined as follows:

$$\sim_f\; = \;\mathsf{fix}\,R.\,\lambda\,x\,y.\,\overline{F}R^{\triangleright}(f\,x)\,(f\,y)$$

which by Lemma 6.2 satisfies

$$x \sim_f y \simeq \overline{(F \circ \triangleright)}(\sim_f)(f\,x)(f\,y)$$

In particular, $\sim_f$ is a guarded bisimulation.

We now show that guarded bisimilarity for the final coalgebra νF^{g} is equivalent to path equality. This follows from the following more general proposition. Recall [Univalent Foundations Program 2013, Ch. 4.6] that a function $f : X \to Y$ is an *embedding*, if the map $\lambda p.\,\lambda i.\,f\,(p\,i) : \mathsf{Path}_X\,x\,y \to \mathsf{Path}_Y\,(f\,x)\,(f\,y)$ is an equivalence for all $x, y : X$. If both X and Y are sets, then the type of embeddings between X and Y is equivalent to the type of injections between those two sets.

Proposition 6.3. *If $f : X \to F(\triangleright X)$ is an embedding, then the type families $\sim_f$ and Path_X are path equal as elements of $X \to X \to \mathsf{U}$.*

Proof. The proof proceeds by guarded recursion. Suppose given $p : \triangleright(\mathsf{Path}_{X\to X\to\mathsf{U}}\,(\sim_f)\,(\mathsf{Path}_X))$. By function extensionality, it is sufficient to show that the types $x \sim_f y$ and $\mathsf{Path}_X\,x\,y$ are path equal for all $x, y : X$. We have the following sequence of equalities:

$$\begin{aligned}x \sim_f y &\equiv \overline{F}(\mathsf{next} \sim_f)^{\triangleright}(f\,x)\,(f\,y)\\ &\equiv \overline{F}(\mathsf{next}\,(\mathsf{Path}_X))^{\triangleright}(f\,x)\,(f\,y) &(10)\\ &\equiv \overline{F}(\mathsf{Path}_{\triangleright X})(f\,x)\,(f\,y)\\ &\equiv \mathsf{Path}_{F(\triangleright X)}(f\,x)\,(f\,y) &(11)\\ &\equiv \mathsf{Path}_X\,x\,y\end{aligned}$$

Equality (10) holds since the type families $(\mathsf{next} \sim_f)^{\triangleright}$ and $(\mathsf{next}\,(\mathsf{Path}_X))^{\triangleright}$ are path equal by the guarded recursive assumption p and (4). Equality (11) follows from Proposition 6.1. □

As a corollary, we obtain an extensionality principle for guarded recursive types. In particular, this implies the coinduction proof principle for the functor F.

Corollary 6.4. *For all $x, y : \nu F^{\mathrm{g}}$, the types $x \sim_{\mathsf{unfold}} y$ and $\mathsf{Path}_{\nu F^{\mathrm{g}}}\, x\, y$ are path equal.*

Proof. This follows from Proposition 6.3 and the fact that unfold is an equivalence of types, so in particular it is an embedding. □

7 EQUIVALENCE OF BISIMULATIONS FOR GLTS

In this section we show that the concrete notion of guarded bisimulation isGLTSBisim given in Section 5 and a truncated variant of the notion of coalgebraic guarded bisimulation is-F-Bisim given in Section 6.2, for the functor $F = \mathsf{P_{fin}}(A \times -)$, are equivalent. This is a reformulation to **TCTT** of a well-known set theoretic result [Rutten 2000].

First, notice that the types $\mathsf{isGLTSBisim}_f\, R$ and is-$\mathsf{P_{fin}}(A\times -)$-$\mathsf{Bisim}_f\, R$ are generally not equivalent. In fact, $\mathsf{isGLTSBisim}_f\, R$ is always a proposition, but that is not always the case for the type is-$\mathsf{P_{fin}}(A\times -)$-$\mathsf{Bisim}_f\, R$. To see this, let X be the inductive type with two distinct elements, $x, y : X$, A the unit type and R to be the always true relation. In this case the type $\mathsf{P_{fin}}(A \times \triangleright X)$ is equivalent to $\mathsf{P_{fin}}(\triangleright X)$ and we can consider the coalgebra f mapping both x and y to $\{\mathsf{next}(x), \mathsf{next}(y)\}$. The type is-$\mathsf{P_{fin}}(A\times -)$-$\mathsf{Bisim}_f\, R$ is then equivalent to

$$\Pi\, z\, w : X.\, \Sigma t : \mathsf{P_{fin}}(\triangleright(X \times X)).\, \mathsf{Path}_{\mathsf{P_{fin}}(\triangleright X)}(\mathsf{P_{fin}}(\triangleright\pi_0)(t))(f\, z) \times \mathsf{Path}_{\mathsf{P_{fin}}(\triangleright X)}(\mathsf{P_{fin}}(\triangleright\pi_1)(t))(f\, w)$$

and this type can be inhabited by the two distinct elements c_1, c_2 defined as

$$c_1\, z\, w = (t_1, \mathsf{refl}\,\{\mathsf{next}\, x, \mathsf{next}\, y\}, \mathsf{refl}\,\{\mathsf{next}\, x, \mathsf{next}\, y\})$$
$$c_2\, z\, w = (t_2, \mathsf{refl}\,\{\mathsf{next}\, x, \mathsf{next}\, y\}, \mathsf{refl}\,\{\mathsf{next}\, x, \mathsf{next}\, y\})$$

where $t_1 = \{\mathsf{next}(x,x), \mathsf{next}(y,y)\}$ and $t_2 = \{\mathsf{next}(x,y), \mathsf{next}(y,x)\}$

Notice though that despite the fact that $\mathsf{isGLTSBisim}_f$ and is-$\mathsf{P_{fin}}(A\times -)$-$\mathsf{Bisim}_f$ are generally different, the corresponding bisimilarity relations for $f = \mathsf{unfold} : \mathsf{Proc} \to \mathsf{P_{fin}}(A \times \triangleright\mathsf{Proc})$ are equal by Corollary 5.4 and Corollary 6.4.

In this section, we will prove that the type $\mathsf{isGLTSBisim}_f\, R$ is equivalent to a truncated variant of is-$\mathsf{P_{fin}}(A\times -)$-$\mathsf{Bisim}_f\, R$. For a given functor F and a guarded coalgebra $f : X \to F(\triangleright X)$, we call a relation $R : X \to X \to \mathsf{U}$ a *guarded truncated bisimulation* iff the following type is inhabited:

$$\text{is-}F\text{-}\mathsf{TrBisim}_f\, R = \Pi\, x\, y : X.\, R\, x\, y \to \exists t : F(\triangleright(\mathsf{tot}\, R)).\, \mathsf{Path}\,(F(\triangleright\pi_0)\, t)\,(f\, x) \times \mathsf{Path}\,(F(\triangleright\pi_1)\, t)\,(f\, y)$$

This differs from is-F-Bisim_f just by replacing the Σ by an existential quantifier.

We can also introduce a notion of *guarded truncated bisimilarity* as follows:

$$\sim_f^{\mathsf{Tr}} = \mathsf{fix}\, R.\, \lambda\, x\, y.\, \exists t : F(\mathsf{tot}\, R^{\triangleright}).\, \mathsf{Path}\,(F\pi_0\, t)\,(f\, x) \times \mathsf{Path}\,(F\pi_1\, t)\,(f\, y)$$

When $F X$ is a set for all types X, the two notions of guarded bisimilarity are equivalent on the final coalgebra for F.

Proposition 7.1. *If $F X$ is a set for all types X, then the types $x \sim_{\mathsf{unfold}} y$ and $x \sim^{\mathsf{Tr}}_{\mathsf{unfold}} y$ are path equal, for all $x, y : \nu F^{\mathrm{g}}$.*

Proof. By Corollary 6.4, the types $x \sim_{\mathsf{unfold}} y$ and $\mathsf{Path}_{\nu F^{\mathrm{g}}}\, x\, y$ are path equal. We prove by guarded recursion that $\sim^{\mathsf{Tr}}_{\mathsf{unfold}}$ and $\mathsf{Path}_{\nu F^{\mathrm{g}}}$ are path equal as terms of $\nu F^{\mathrm{g}} \to \nu F^{\mathrm{g}} \to \mathsf{U}$. Suppose given $p : \triangleright(\mathsf{Path}_{\nu F^{\mathrm{g}} \to \nu F^{\mathrm{g}} \to \mathsf{U}}\,(\sim^{\mathsf{Tr}}_{\mathsf{unfold}})\,(\mathsf{Path}_{\nu F^{\mathrm{g}}}))$. By function extensionality, it is sufficient to show that the types $x \sim^{\mathsf{Tr}}_{\mathsf{unfold}} y$ and $\mathsf{Path}_{\nu F^{\mathrm{g}}}\, x\, y$ are path equal for all $x, y : X$. We have the following sequence of equalities:

$$\begin{aligned}
x \sim^{\mathsf{Tr}}_{\mathsf{unfold}} y &\equiv \|\overline{F}(\mathsf{next} \sim^{\mathsf{Tr}}_{\mathsf{unfold}})^{\triangleright}(\mathsf{unfold}\, x)\,(\mathsf{unfold}\, y)\| \\
&\equiv \|\overline{F}(\mathsf{next}\,(\mathsf{Path}_{\nu F^{\mathrm{g}}}))^{\triangleright}(\mathsf{unfold}\, x)\,(\mathsf{unfold}\, y)\| && (12)\\
&\equiv \|\mathsf{Path}_{\nu F^{\mathrm{g}}}\, x\, y\| && (13)\\
&\equiv \mathsf{Path}_{\nu F^{\mathrm{g}}}\, x\, y && (14)
\end{aligned}$$

Here, (12) follows from the induction hypothesis and (4), and (13) follows from the fact that the two untruncated types are path equal, which we derived *en passant* in the proof of Proposition 6.3. Finally, (14) follows from the fact that νF^{g} is a set, since it is equivalent to the set $F\,(\triangleright \nu F^{\mathrm{g}})$. □

We now give a characterization of the truncated relation lifting of the functor $\mathsf{P}_{\mathsf{fin}}(A \times -)$, which is the key that allows us to prove that the concrete notion of bisimulation for labelled transition systems introduced in Section 5 is equivalent to the truncated variant of the general coalgebraic notion of bisimulation for the functor $\mathsf{P}_{\mathsf{fin}}(A \times -)$. First, we prove an auxiliary lemma.

Lemma 7.2. *Given a relation $R : X \to Y \to \mathsf{U}$, two finite subsets $u : \mathsf{P}_{\mathsf{fin}}(A \times X)$, $v : \mathsf{P}_{\mathsf{fin}}(A \times Y)$ and a proof $p : \Pi x : X.\, \Pi a : A.\, (a, x) \in u \to \exists y : Y.\, (a, y) \in v \times R\, x\, y$, we have an inhabitant of the following type:*

$$\exists t : \mathsf{P}_{\mathsf{fin}}(A \times \mathsf{tot}\, R).\, \mathsf{Path}_{\mathsf{P}_{\mathsf{fin}}(A\times X)}\,(\mathsf{P}_{\mathsf{fin}}(A \times \pi_0)\, t)\, u \times \mathsf{P}_{\mathsf{fin}}(A \times \pi_1)\, t \subseteq v.$$

Proof. We construct a term $d\, u\, v\, p$ by induction on u. If $u = \varnothing$, define $d\, u\, v\, p = |(\varnothing, \mathsf{refl}\, \varnothing, q)|$, where q is the trivial proof of $\varnothing \subseteq v$. If $u = \{(a, x)\}$, define $p' : \exists y : Y.\, (a, y) \in v \times R\, x\, y$ as $p\, x\, a\, |\mathsf{refl}\,(a, x)|$. Since we are proving a proposition, we can use the induction principle for propositional truncation on p', to get $p' = |(y, q, r)|$. Now define $d\, u\, v\, p = |(\{(a, x, y, r)\}, \mathsf{refl}\, \{(a, x)\}, q')|$, where $q' : \{(a, y)\} \subseteq v$ follows from q.

If $u = u_1 \cup u_2$, by induction, we have proofs d_1 and d_2 of the statements of the lemma for u_1 and u_2 respectively. Define:

$$\begin{aligned}
d_1' &: \exists t : \mathsf{P}_{\mathsf{fin}}(A \times \mathsf{tot}\, R).\, \mathsf{Path}_{\mathsf{P}_{\mathsf{fin}}(A\times X)}\,(\mathsf{P}_{\mathsf{fin}}(A \times \pi_0)\, t)\, u_1 \times \mathsf{P}_{\mathsf{fin}}(A \times \pi_1)\, t \subseteq v\\
d_1' &= d_1\, u_1\, v\,(\lambda x\, a\, q.\, p\, x\, a\, |\mathsf{inl}\, q|)\\
d_2' &: \exists t : \mathsf{P}_{\mathsf{fin}}(A \times \mathsf{tot}\, R).\, \mathsf{Path}_{\mathsf{P}_{\mathsf{fin}}(A\times X)}\,(\mathsf{P}_{\mathsf{fin}}(A \times \pi_0)\, t)\, u_2 \times \mathsf{P}_{\mathsf{fin}}(A \times \pi_1)\, t \subseteq v\\
d_2' &= d_2\, u_2\, v\,(\lambda x\, a\, q.\, p\, x\, a\, |\mathsf{inr}\, q|)
\end{aligned}$$

Finally, by the induction principle of propositional truncation we can assume $d_1' = |(t_1, q_1, s_1)|$ and $d_2' = |(t_2, q_2, s_2)|$ and define $d\, u\, v\, p = |(t_1 \cup t_2\,,\ \lambda i.\, q_1\, i \cup q_2\, i\,,\ s)|$, where

$$s : \mathsf{P}_{\mathsf{fin}}(A \times \pi_1)\, t_1 \cup \mathsf{P}_{\mathsf{fin}}(A \times \pi_1)\, t_2 \subseteq v$$

follows from s_1 and s_2. □

Proposition 7.3. *For all $R : X \to Y \to \mathsf{U}$, $u : \mathsf{P}_{\mathsf{fin}}(A \times X)$ and $v : \mathsf{P}_{\mathsf{fin}}(A \times Y)$, the type $\|\overline{\mathsf{P}_{\mathsf{fin}}(A \times -)} R\, u\, v\|$ is path equal to the product type:*

$$\begin{gathered}
(\Pi\, x : X.\, \Pi\, a : A.\, (a, x) \in u \to \exists\, y : Y.\, (a, y) \in v \times R\, x\, y)\\
\times\\
(\Pi\, y : Y.\, \Pi\, a : A.\, (a, y) \in v \to \exists\, x : X.\, (a, x) \in u \times R\, x\, y)
\end{gathered}$$

Proof. By univalence, it suffices to show that the two propositions are logically equivalent. For the left-to-right implication, let $e : \|\overline{\mathsf{P}_{\mathsf{fin}}(A \times -)} R\, u\, v\|$. We construct a term

$$d_1 : \Pi\, x : X.\, \Pi\, a : A.\, (a, x) \in u \to \exists\, y : Y.\, (a, y) \in v \times R\, x\, y$$

A term d_2 of a similar type proving the second component in the product can be constructed similarly. Since we are proving a proposition, we can use the induction principle of propositional truncation on p. So let $e = |(t, p_1, p_2)|$, with

$$t : \mathsf{P_{fin}}(A \times \mathsf{tot}\, R) \qquad p_1 : \mathsf{Path}_{\mathsf{P_{fin}}(A \times X)}\, (\mathsf{P_{fin}}(A \times \pi_0)\, t)\, u \qquad p_2 : \mathsf{Path}_{\mathsf{P_{fin}}(A \times Y)}\, (\mathsf{P_{fin}}(A \times \pi_1)\, t)\, v.$$

Suppose $x : X$, $a : A$ and $q : (a, x) \in u$. Transporting over the path p_1, we obtain a proof $q' : (a, x) \in \mathsf{P_{fin}}(A \times \pi_0)\, t$. By Lemma 4.1 , we know that there merely exists $y : Y$ and $r : R\, x\, y$ such that $(a, x, y, r) \in t$. In particular, we have $(a, y) \in \mathsf{P_{fin}}(A \times \pi_1)\, t$. Transporting over the path p_2, we obtain $(a, y) \in v$.

For the right-to-left implication, by Lemma 7.2 we obtain a term:

$$s_1 : \exists t : \mathsf{P_{fin}}(A \times \mathsf{tot}\, R).\ \mathsf{Path}_{\mathsf{P_{fin}}(A \times X)}\, (\mathsf{P_{fin}}(A \times \pi_0)\, t)\, u \times \mathsf{P_{fin}}(A \times \pi_1)\, t \subseteq v.$$

Similarly, by applying a symmetric version of Lemma 7.2 we obtain another term:

$$s_2 : \exists t : \mathsf{P_{fin}}(A \times \mathsf{tot}\, R).\ \mathsf{Path}_{\mathsf{P_{fin}}(A \times Y)}\, (\mathsf{P_{fin}}(A \times \pi_1)\, t)\, v \times \mathsf{P_{fin}}(A \times \pi_0)\, t \subseteq u.$$

Since we are proving a proposition, we can use the induction principle of propositional truncation on s_1 and s_2. So let $s_1 = |(t_1, p_1, q_1)|$ and $s_2 = |(t_2, p_2, q_2)|$. We return $|(t_1 \cup t_2, p, q)| : \|\overline{\mathsf{P_{fin}}(A \times -)} R\, u\, v\|$, where $p : \mathsf{Path}_{\mathsf{P_{fin}}(A \times X)}\, (\mathsf{P_{fin}}(A \times \pi_0) t_1 \cup \mathsf{P_{fin}}(A \times \pi_0) t_2)\, u$ is taken to be the following sequential composition of equalities:

$$\mathsf{P_{fin}}(A \times \pi_0) t_1 \cup \mathsf{P_{fin}}(A \times \pi_0) t_2 \equiv u \cup \mathsf{P_{fin}}(A \times \pi_0) t_2 \equiv u$$

The first of these equalities follows from p_1 and the second from q_2. The proof q is constructed in a similar way. □

Notice that classically the proof of Proposition 7.3 is simpler than the constructive one presented here. In the classical proof, for the right-to-left direction one first constructs a subset $t : \mathsf{P_{fin}}(A \times \mathsf{tot}\, R)$ for which $(a, x, y, r) \in t$ if and only if $(a, x) \in u$, $(a, y) \in v$ and $r : R\, x\, y$. Then one proves, using the given hypothesis, that $\mathsf{P_{fin}}(A \times \pi_0)\, t$ is path equal to u and $\mathsf{P_{fin}}(A \times \pi_1)\, t$ is path equal v. Constructively, even assuming that the type A comes with decidable equality, we cannot proceed in two steps like this, since we do not have a principle of set comprehension as needed for constructing the term t. It is possible to construct a term $t' : \mathsf{P_{fin}}(A \times X \times Y)$ for which $(a, x, y) \in t'$ if and only if $(a, x) \in u$ and $(a, y) \in v$, but not to filter out of t' the triples (a, x, y) for which $R\, x\, y$ does not hold, since R in general is an undecidable relation.

As a direct consequence of Proposition 7.3, we obtain the equivalence of two notions of bisimulation for GLTSs.

Theorem 7.4. *Let (X, A, f) be guarded coalgebra for the functor $F = \mathsf{P_{fin}}(A \times -)$. The types* is-$F$-TrBisim$_f$ R *and* isGLTSBisim$_f$ R *are path equal.*

8 DENOTATIONAL SEMANTICS FOR TCTT

In this section, we show consistency of **TCTT** with higher inductive types by constructing a denotational model. The construction is an adaptation of the model of **GCTT** [Birkedal et al. 2016], which we extend with ticks using the constructions of Mannaa and Møgelberg [2018]. The extension with HITs is done exactly as in [Coquand et al. 2018], and includes ordinary inductive types, such as the type HML used in Section 5.3.

8.1 The Category of Cubical Trees

We first recall the definition of the category of cubical sets as used [Cohen et al. 2018] to model cubical type theory. Let $\{i, j, k, \dots\}$ be a countably infinite set of names. The *category of cubes* C has finite sets of names $I, J, K, \dots$ as objects, and as morphisms $f : J \to I$ functions $f : I \to \mathrm{DM}(J)$, where $\mathrm{DM}(J)$ is the free De Morgan algebra on J. These compose by the standard Kleisli composition.

We write $\widehat{C}$ for the category of contravariant presheaves on C, whose objects are called *cubical sets*. In elementary terms, a cubical set X comprises a family of sets $X(I)$ indexed over objects I in C and a family of maps indexed over morphisms $f : J \to I$ in C mapping an $x \in X(I)$ to $x \cdot f \in X(J)$, in a functorial way, i.e., $x \cdot \mathrm{id} = x$ and $(x \cdot f) \cdot g = x \cdot (f \circ g)$. We often leave $\cdot$ implicit, simply writing xf.

The category $\widehat{C}$ has an interval object $\mathbb{I}$ defined by $\mathbb{I}(I) = \mathrm{DM}(I)$, i.e., the Yoneda embedding applied to a singleton set. Recall that $\widehat{C}$, being a presheaf category is a topos [MacLane and Moerdijk 2012], and in particular has a subobject classifier Ω. The face lattice $\mathbb{F}$ is the cubical set defined as the subobject of Ω given as the image of the map $(-) = 1 : \mathbb{I} \to \Omega$ mapping an element $r : I$ to the truth value $r = 1$. In the internal language of $\widehat{C}$, this is

$$\mathbb{F} = \{p : \Omega \mid \exists r : \mathbb{I}. p \leftrightarrow (r = 1)\}$$

Faces in cubical type theory are modelled as elements of $\mathbb{F}$. This corresponds to the object of *cofibrant* propositions in [Orton and Pitts 2016].

As proved in [Birkedal et al. 2016] these constructions can be generalised to give a model of cubical type theory in presheaves over any category of the form $C \times \mathbb{D}$ as long as $\mathbb{D}$ has an initial object. We will extend this to a model of **TCTT** in the case of $\mathbb{D} = \omega$, the partially ordered category of natural numbers. Like $\widehat{C}$ the topos $\widehat{C \times \mathbb{D}}$ has an interval object defined as $\mathbb{I}_{\mathbb{D}}(I, d) = \mathrm{DM}(I)$ and a face lattice $\mathbb{F}_{\mathbb{D}}$ defined similarly to $\mathbb{F}$ in $\widehat{C}$.

Before we describe the model of **TCTT**, we first recall the notion of category with family, the standard notion of model of dependent type theory [Dybjer 1996].

Definition 8.1. A *category with family* (CwF) comprises

- A category $\mathbb{C}$. We refer to the objects of $\mathbb{C}$ as contexts, and the morphisms as substitutions
- For each object Γ in $\mathbb{C}$ a set of types in context Γ. We write $\Gamma \vdash A$ to mean that A is a type in context Γ.
- For each $\Gamma \vdash A$ a set of terms of type A. We write $\Gamma \vdash t : A$ to mean that t is a term of type A.
- For each substitution $\gamma : \Delta \to \Gamma$ a pair of reindexing maps associating to each $\Gamma \vdash A$ a type $\Delta \vdash A[\gamma]$ and to each term $\Gamma \vdash t : A$ a term $\Delta \vdash t[\gamma] : A[\gamma]$ such that
$$(A[\gamma])[\delta] = A[\gamma\delta] \qquad A[\mathrm{id}] = A \qquad (t[\gamma])[\delta] = t[\gamma\delta] \qquad t[\mathrm{id}] = t$$
- A comprehension map, associating to each $\Gamma \vdash A$ a context $\Gamma.A$, a substitution $\mathrm{p}_A : \Gamma.A \to \Gamma$, and a term $\Gamma.A \vdash \mathrm{q}_A : A[\mathrm{p}_A]$ such that for every $\gamma : \Delta \to \Gamma$ and $\Delta \vdash t : A[\gamma]$ there exists a unique $(\gamma, t) : \Delta \to \Gamma.A$ such that
$$\mathrm{p}_A \circ (\gamma, t) = \gamma \qquad \mathrm{q}_A[(\gamma, t)] = t$$

Categories with family model basic dependent types, and can be extended to models of Π and Σ types etc. [Dybjer 1996].

Any presheaf category $\widehat{\mathbb{C}}$ defines a category with family, in which the category of contexts is $\widehat{\mathbb{C}}$, a type $\Gamma \vdash A$ is a family of sets $A(C, \gamma)$ indexed over $C \in \mathbb{C}$ and $\gamma \in \Gamma(C)$, together with maps mapping $x \in A(C, \gamma)$ to $xf \in A(D, \gamma f)$ for $f : D \to C$, in a functorial way. A term $\Gamma \vdash t : A$ is a family of elements $t(C, \gamma) \in A(C, \gamma)$ such that $t(C, \gamma)f = t(D, \gamma f)$ for all f. We shall write $\Gamma \vdash_{C \times \mathbb{D}} A$ and $\Gamma \vdash_{C \times \mathbb{D}} t : A$ for the judgements of types and terms in the CwF associated with $\widehat{C \times \mathbb{D}}$.

8.2 A Model of TCTT

Following [Birkedal et al. 2016], we construct the model using the internal language of the CwF $\widehat{C \times \mathbb{D}}$. First we define the notion of a composition structure.

Definition 8.2. Let $\Gamma \vdash_{C\times\mathbb{D}} A$ be given. A composition structure on A is an operation taking as input a path $\mathbb{I}_{\mathbb{D}} \vdash_{C\times\mathbb{D}} \gamma : \Gamma$, a face $1 \vdash_{C\times\mathbb{D}} \phi : \mathbb{F}_{\mathbb{D}}$, and terms $1 \vdash_{C\times\mathbb{D}} u : [\phi] \to \Pi(i : \mathbb{I}_{\mathbb{D}})A[\gamma(i)]$ and $1 \vdash_{C\times\mathbb{D}} u_0 : A[\gamma(0)]$ such that $\phi \to u \star 0 = u_0$ and producing a term $1 \vdash_{C\times\mathbb{D}} c_A\, \gamma\, \phi\, u\, u_0 : A[\gamma(1)]$ such that $\phi \to u \star 1 = c_A\, \gamma\, \phi\, u\, u_0$.

Note that this makes sense, because the CwF is democratic: contexts Γ correspond to types in context 1 (the terminal object). The definition uses the subsingleton $[\phi]$ associated to an element $\phi : \mathbb{F}_{\mathbb{D}}$ defined in the internal logic of $\widehat{C \times \mathbb{D}}$ as $\{\star \mid \phi\}$.

Theorem 8.3 ([Birkedal et al. 2016]). *Let $\mathbb{D}$ be a small category with an initial object. The CwF defined as follows is a model of cubical type theory.*

- *The category of contexts is $\widehat{C \times \mathbb{D}}$*
- *A type in context Γ is a pair of a type $\Gamma \vdash_{C\times\mathbb{D}} A$ and a composition structure c_A on A.*
- *Terms of type $(\Gamma \vdash_{C\times\mathbb{D}} A, c_A)$ are terms $\Gamma \vdash_{C\times\mathbb{D}} t : A$.*

We now specialise to the case of $\mathbb{D} = \omega$ and show how to extend this model with ticks to a model of **TCTT**, using the techniques developed in [Mannaa and Møgelberg 2018] to model the ticks of Clocked Type Theory [Bahr et al. 2017]. First note that there is an adjunction $◀ \dashv ▶$ of endofunctors on $\widehat{C \times \omega}$ defined as follows

$$(▶X)(I, 0) = \{\star\}$$
$$(▶X)(I, n+1) = X(I, n)$$
$$(◀X)(I, n) = X(I, n+1)$$

The right adjoint $▶$ is the obvious adaptation of the interpretation of $\triangleright$ in the topos of trees model of guarded recursion [Birkedal et al. 2012], and the left adjoint $◀$ is discussed in the same paper.

Extension of contexts with ticks is modelled using the left adjoint

$$[\![\Gamma, \alpha : \mathbb{T} \vdash]\!] = ◀[\![\Gamma \vdash]\!]$$

and tick weakening is modelled by the natural transformation $◀ \to \mathrm{id}$ induced by the maps $(I, n) \to (I, n+1)$ in $C \times \omega$. Using this, one can define a context projection $p_{\Gamma;\Gamma'} : [\![\Gamma, \Gamma' \vdash]\!] \to [\![\Gamma \vdash]\!]$ by induction on Γ'.

Lemma 8.4. *The tick exchange rules are sound, i.e.,*

$$[\![\Gamma, \alpha : \mathbb{T}, i : \mathbb{I}]\!] = [\![\Gamma, i : \mathbb{I}, \alpha : \mathbb{T}]\!] \qquad\qquad [\![\Gamma, [\phi], \alpha : \mathbb{T}]\!] = [\![\Gamma, \alpha : \mathbb{T}, [\phi]]\!]$$

Proof. The first of these equalities follows directly from the fact that $◀\mathbb{I}_{\omega} = \mathbb{I}_{\omega}$:

$$[\![\Gamma, \alpha : \mathbb{T}, i : \mathbb{I}]\!] = ◀[\![\Gamma]\!] \times \mathbb{I}_{\omega} = ◀[\![\Gamma]\!] \times ◀\mathbb{I}_{\omega} = ◀([\![\Gamma]\!] \times \mathbb{I}_{\omega}) = [\![\Gamma, i : \mathbb{I}, \alpha : \mathbb{T}]\!]$$

For the second of these, note first that $\Gamma, \alpha : \mathbb{T} \vdash \phi : \mathbb{F}$ if and only if $\Gamma \vdash \phi : \mathbb{F}$. The interpretation associates to each (I, n) and $\gamma \in [\![\Gamma]\!](I, n+1)$, an element $[\![\Gamma, \alpha : \mathbb{T} \vdash \phi : \mathbb{F}]\!](I, n, \gamma)$ in $\mathbb{F}_{\omega}(I, n)$, which is a subset of the subobject classifier at (I, n). An easy induction shows that $[\![\Gamma, \alpha : \mathbb{T} \vdash \phi : \mathbb{F}]\!](I, n, \gamma)$ is true iff $[\![\Gamma \vdash \phi : \mathbb{F}]\!](I, n+1, \gamma)$ is true, which implies the second equality. □

Given a type $◀\Gamma \vdash_{C\times\omega} A$, we define the semantic later modality $\Gamma \vdash_{C\times\omega} ▶_{\Gamma} A$ as:

$$(▶_{\Gamma}A)(I, 0, \gamma) = \{\star\}$$
$$(▶_{\Gamma}A)(I, n+1, \gamma) = A(I, n, \gamma)$$

Lemma 8.5. *If* $\blacktriangleleft\Gamma \vdash_{C\times\omega} A$ *has a composition structure, so does* $\Gamma \vdash_{C\times\omega} \blacktriangleright_\Gamma A$.

Proof. Given input data γ, ϕ, u, u_0, consider the term $\mathbb{I}_\omega \vdash_{C\times\omega} \tilde{\gamma} : \blacktriangleleft\Gamma$ defined as $\tilde{\gamma}(I,n,r) = \gamma(I,n+1,r)$. Note that

$$\begin{aligned}\mathbb{I}_\omega \vdash_{C\times\omega} A[\tilde{\gamma}](I,n,r) &= A(I,n,\tilde{\gamma}(r)) \\ &= (\blacktriangleright_\Gamma A)(I,n+1,\gamma(r)) \\ &= (\blacktriangleright_{\mathbb{I}_\omega} A)[\gamma](I,n+1,r)\end{aligned}$$

so we can define $1 \vdash \tilde{u}_0 : A[\tilde{\gamma}(0)]$ as $\tilde{u}_0(I,n) = u_0(I,n+1)$ and likewise

$$1 \vdash_{C\times\omega} \tilde{u} : [\phi] \to \Pi(i : \mathbb{I}_\omega)A[\tilde{\gamma}(i)]$$

as $\tilde{u}(I,n) = u(I,n+1)$. Finally, to define $1 \vdash_{C\times\omega} c_{\blacktriangleright_\Gamma A}\, \gamma\, \phi\, u\, u_0 : (\blacktriangleright_\Gamma A)[\gamma(1)]$ we define

$$c_{\blacktriangleright_\Gamma A}\, \gamma\, \phi\, u\, u_0(I,0) = \star \qquad\qquad c_{\blacktriangleright_\Gamma A}\, \gamma\, \phi\, u\, u_0(I,n+1) = c_A\, \tilde{\gamma}\, \phi\, \tilde{u}\, \tilde{u}_0(I,n).$$ □

There is a bijective correspondence between terms $\Gamma \vdash_{C\times\omega} t : \blacktriangleright_\Gamma A$ and terms $\blacktriangleleft\Gamma \vdash_{C\times\omega} u : A$ given by

$$t(I,0,\gamma) = \star \qquad t(I,n+1,\gamma) = u(I,n,\gamma) \qquad u(I,n,\gamma) = t(I,n+1,\gamma)$$

Writing $\overline{(-)}$ for both directions of this bijection, we define

$$\begin{aligned}[\![\Gamma \vdash \triangleright(\alpha : \mathbb{T}).A]\!] &= \blacktriangleright_{[\![\Gamma]\!]} [\![\Gamma, \alpha : \mathbb{T} \vdash A]\!] \\ [\![\Gamma \vdash \lambda(\alpha : \mathbb{T}).t]\!] &= \overline{[\![\Gamma, \alpha : \mathbb{T} \vdash t]\!]} \\ [\![\Gamma, \alpha : \mathbb{T}, \Gamma' \vdash t\,[\alpha]]\!] &= \overline{[\![\Gamma \vdash t]\!]} \circ p_{\Gamma,\alpha;\Gamma'}\end{aligned}$$

This can be verified to satisfy the β and η equalities [Mannaa and Møgelberg 2018]. Likewise fixed points can be modelled in the standard way [Birkedal et al. 2016]. Indeed the fixed point equality holds definitionally in the model.

Higher inductive types can be modelled in $\widehat{C \times \omega}$ essentially in the same way as in [Coquand et al. 2018], and we refer the reader to loc.cit. for details. Note that in [Coquand et al. 2018, Section 2.5] higher inductive types defining operations on types, such as the finite powerset functor, preserve the universe level, so indeed the universe is closed under such constructions.

It remains to show that the universe is closed under the $\triangleright(\alpha : \mathbb{T}).(-)$ operation. As is standard, the universe is modelled in [Birkedal et al. 2016] under the assumption of the existence of a Grothendieck universe in the ambient set theory, by modelling $[\![\mathrm{U}]\!](I,n)$ as the set of small types in context $y(I,n)$. Since the operation $\blacktriangleright_\Gamma(-)$ preserves smallness, the universe is closed under $\triangleright(\alpha : \mathbb{T}).(-)$.

9 CONCLUSION AND FUTURE WORK

We have shown that in the type theory **TCTT** combining cubical type theory with guarded recursion, the notions of bisimilarity and path equality coincide for guarded coalgebras, in the sense of equivalence of types. As a consequence of this, representing processes as guarded labelled transition systems allows for proofs of bisimilarity to be done in a simple way using guarded recursion.

As stated in the introduction, the use of the finite powerset functor is motivated by the desire to use this work as a stepping stone towards similar results for coinductive types. For this, the use of finiteness, as opposed to some other cardinality restriction, appears to be non-essential. Veltri [2017] has given a description of the countable powerset functor as a HIT, and we believe that the results presented here can be extended to this functor as well. This would allow, e.g., to extend our presentation of CCS with replication $!P$.

In future work we plan to extend **TCTT** with clocks and universal quantification over these. This should allow for coinductive types to be encoded using guarded recursive types [Atkey and McBride 2013], and for guarded recursion to be used for coinductive reasoning [Bizjak et al. 2016]. In particular, the reasoning of Example 5.6, which shows that the two processes p and q satisfying $p \equiv q \leftrightarrow \triangleright 0$ can be used to show that their corresponding elements in the coinductive type of labelled transition systems are genuinely not equal.

We expect the general proof of coincidence of bisimilarity and path equality to lift easily to the coinductive case, but encoding the coinductive type of processes as a final coalgebra for the functor $\mathsf{P}_{\mathsf{fin}}(A \times -)$ is a challenge. This requires $\mathsf{P}_{\mathsf{fin}}$ to commute with universal quantification over clocks [Atkey and McBride 2013; Møgelberg 2014], a result which holds in the model, but which we are currently unable to express in syntax. Doing this will most likely require new syntactical constructions. Universal quantification over clocks does commute with ordinary inductive types and W-types, also in the syntax of **GDTT**, which allows nested inductive and coinductive types to be defined using guarded recursion. Unfortunately, the techniques used in these results do not appear to be applicable to HITs.

Future work also includes investigating more advanced proofs of bisimulation e.g., using up-to-techniques [Danielsson 2018; Milner 1983; Pous and Sangiorgi 2012] and weak bisimulation, which is generally challenging for guarded recursion [Møgelberg and Paviotti 2016]. It would also be desirable to have an implementation of guarded recursion in a proof assistant, which would allow proofs such as those presented in this paper to be formally checked by a computer. There does exist a prototype implementation of Guarded Cubical Type Theory [Birkedal et al. 2016], but we found this inadequate for larger proofs.

ACKNOWLEDGMENTS

This work was supported by a research grant (13156) from VILLUM FONDEN, and by DFF-Research Project 1 Grant no. 4002-00442, from The Danish Council for Independent Research for the Natural Sciences (FNU). We thank the anonymous referees for valuable comments.

REFERENCES

Andreas Abel, Stephan Adelsberger, and Anton Setzer. 2017. Interactive programming in Agda - Objects and graphical user interfaces. *J. Funct. Program.* 27 (2017), e8. https://doi.org/10.1017/S0956796816000319

Jirí Adámek, Paul Blain Levy, Stefan Milius, Lawrence S. Moss, and Lurdes Sousa. 2015. On Final Coalgebras of Power-Set Functors and Saturated Trees - To George Janelidze on the Occasion of His Sixtieth Birthday. *Applied Categorical Structures* 23, 4 (2015), 609–641. https://doi.org/10.1007/s10485-014-9372-9

Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. 2015. Non-Wellfounded Trees in Homotopy Type Theory. In *13th International Conference on Typed Lambda Calculi and Applications, TLCA 2015, July 1-3, 2015, Warsaw, Poland (LIPIcs)*, Thorsten Altenkirch (Ed.), Vol. 38. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 17–30. https://doi.org/10.4230/LIPIcs.TLCA.2015.17

Andrew W. Appel and David A. McAllester. 2001. An indexed model of recursive types for foundational proof-carrying code. *ACM Trans. Program. Lang. Syst.* 23, 5 (2001), 657–683. https://doi.org/10.1145/504709.504712

Robert Atkey and Conor McBride. 2013. Productive coprogramming with guarded recursion. In *ACM SIGPLAN International Conference on Functional Programming, ICFP'13, Boston, MA, USA - September 25 - 27, 2013*, Greg Morrisett and Tarmo Uustalu (Eds.). ACM, 197–208. https://doi.org/10.1145/2500365.2500597

Patrick Bahr, Hans Bugge Grathwohl, and Rasmus Ejlers Møgelberg. 2017. The clocks are ticking: No more delays!. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017.* IEEE Computer Society, 1–12. https://doi.org/10.1109/LICS.2017.8005097

Marc Bezem, Thierry Coquand, and Simon Huber. 2013. A Model of Type Theory in Cubical Sets. In *19th International Conference on Types for Proofs and Programs, TYPES 2013, April 22-26, 2013, Toulouse, France (LIPIcs)*, Ralph Matthes and Aleksy Schubert (Eds.), Vol. 26. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 107–128. https://doi.org/10.4230/

LIPIcs.TYPES.2013.107

Lars Birkedal, Ales Bizjak, Ranald Clouston, Hans Bugge Grathwohl, Bas Spitters, and Andrea Vezzosi. 2016. Guarded Cubical Type Theory: Path Equality for Guarded Recursion. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs)*, Jean-Marc Talbot and Laurent Regnier (Eds.), Vol. 62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 23:1–23:17. https://doi.org/10.4230/LIPIcs.CSL.2016.23

Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. 2012. First steps in synthetic guarded domain theory: step-indexing in the topos of trees. *Logical Methods in Computer Science* 8, 4 (2012). https://doi.org/10.2168/LMCS-8(4:1)2012

Aleš Bizjak, Lars Birkedal, and Marino Miculan. 2014. A Model of Countable Nondeterminism in Guarded Type Theory. In *Rewriting and Typed Lambda Calculi - Joint International Conference, RTA-TLCA 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings (Lecture Notes in Computer Science)*, Gilles Dowek (Ed.), Vol. 8560. Springer, 108–123. https://doi.org/10.1007/978-3-319-08918-8_8

Ales Bizjak, Hans Bugge Grathwohl, Ranald Clouston, Rasmus Ejlers Møgelberg, and Lars Birkedal. 2016. Guarded Dependent Type Theory with Coinductive Types. In *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, Bart Jacobs and Christof Löding (Eds.). Springer, 20–35. https://doi.org/10.1007/978-3-662-49630-5_2

Edwin Brady. 2016. *Type-driven Development with Idris.* Manning Publications Company.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In *21st International Conference on Types for Proofs and Programs (TYPES 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Tarmo Uustalu (Ed.), Vol. 69. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 5:1–5:34. https://doi.org/10.4230/LIPIcs.TYPES.2015.5

Thierry Coquand. 1993. Infinite Objects in Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'93, Nijmegen, The Netherlands, May 24-28, 1993, Selected Papers (Lecture Notes in Computer Science)*, Henk Barendregt and Tobias Nipkow (Eds.), Vol. 806. Springer, 62–78. https://doi.org/10.1007/3-540-58085-9_72

Thierry Coquand, Simon Huber, and Anders Mörtberg. 2018. On Higher Inductive Types in Cubical Type Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, Anuj Dawar and Erich Grädel (Eds.). ACM, 255–264. https://doi.org/10.1145/3209108.3209197

Nils Anders Danielsson. 2018. Up-to techniques using sized types. *PACMPL* 2, POPL (2018), 43:1–43:28. https://doi.org/10.1145/3158131

Peter Dybjer. 1996. Internal Type Theory. In *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers (Lecture Notes in Computer Science)*, Stefano Berardi and Mario Coppo (Eds.), Vol. 1158. Springer, 120–134. https://doi.org/10.1007/3-540-61780-9_66

Dan Frumin, Herman Geuvers, Léon Gondelman, and Niels van der Weide. 2018. Finite sets in homotopy type theory. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018*, June Andronick and Amy P. Felty (Eds.). ACM, 201–214. https://doi.org/10.1145/3167085

Matthew Hennessy and Robin Milner. 1980. On Observing Nondeterminism and Concurrency. In *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings (Lecture Notes in Computer Science)*, J. W. de Bakker and Jan van Leeuwen (Eds.), Vol. 85. Springer, 299–309. https://doi.org/10.1007/3-540-10003-2_79

Claudio Hermida and Bart Jacobs. 1998. Structural Induction and Coinduction in a Fibrational Setting. *Inf. Comput.* 145, 2 (1998), 107–152. https://doi.org/10.1006/inco.1998.2725

John Hughes, Lars Pareto, and Amr Sabry. 1996. Proving the Correctness of Reactive Systems Using Sized Types. In *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, Hans-Juergen Boehm and Guy L. Steele Jr. (Eds.). ACM Press, 410–423. https://doi.org/10.1145/237721.240882

Bart Jacobs. 2016. *Introduction to Coalgebra: Towards Mathematics of States and Observation.* Cambridge Tracts in Theoretical Computer Science, Vol. 59. Cambridge University Press. https://doi.org/10.1017/CBO9781316823187

Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Aleš Bizjak, Lars Birkedal, and Derek Dreyer. 2018. Iris from the ground up. (2018). https://people.mpi-sws.org/~dreyer/papers/iris-ground-up/paper.pdf

Alexander Kurz and Jiri Velebil. 2016. Relation lifting, a survey. *J. Log. Algebr. Meth. Program.* 85, 4 (2016), 475–499. https://doi.org/10.1016/j.jlamp.2015.08.002

Xavier Leroy. 2006. Formal certification of a compiler back-end or: programming a compiler with a proof assistant. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11-13, 2006*, J. Gregory Morrisett and Simon L. Peyton Jones (Eds.). ACM, 42–54. https://doi.org/10.1145/1111037.1111042

Saunders MacLane and Ieke Moerdijk. 2012. *Sheaves in geometry and logic: A first introduction to topos theory.* Springer Science & Business Media.

Bassel Mannaa and Rasmus Ejlers Møgelberg. 2018. The Clocks They Are Adjunctions Denotational Semantics for Clocked Type Theory. In *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK (LIPIcs)*, Hélène Kirchner (Ed.), Vol. 108. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 23:1–23:17. https://doi.org/10.4230/LIPIcs.FSCD.2018.23

Robin Milner. 1980. *A Calculus of Communicating Systems*. Lecture Notes in Computer Science, Vol. 92. Springer. https://doi.org/10.1007/3-540-10235-3

Robin Milner. 1983. Calculi for Synchrony and Asynchrony. *Theor. Comput. Sci.* 25 (1983), 267–310. https://doi.org/10.1016/0304-3975(83)90114-7

Robin Milner. 1999. *Communicating and mobile systems: the pi calculus*. Cambridge University Press.

R. E. Møgelberg. 2014. A type theory for productive coprogramming via guarded recursion. In *CSL-LICS*. 71:1–71:10.

Rasmus Ejlers Møgelberg and Marco Paviotti. 2016. Denotational semantics of recursive types in synthetic guarded domain theory. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, Martin Grohe, Eric Koskinen, and Natarajan Shankar (Eds.). ACM, 317–326. https://doi.org/10.1145/2933575.2934516

Hiroshi Nakano. 2000. A Modality for Recursion. In *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 255–266. https://doi.org/10.1109/LICS.2000.855774

Ian Orton and Andrew M. Pitts. 2016. Axioms for Modelling Cubical Type Theory in a Topos. In *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France (LIPIcs)*, Jean-Marc Talbot and Laurent Regnier (Eds.), Vol. 62. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 24:1–24:19. https://doi.org/10.4230/LIPIcs.CSL.2016.24

Damien Pous and Davide Sangiorgi. 2012. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*, Davide Sangiorgi and Jan Rutten (Eds.). Cambridge University Press.

Jan J. M. M. Rutten. 2000. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.* 249, 1 (2000), 3–80. https://doi.org/10.1016/S0304-3975(00)00056-6

Daniel Schwencke. 2010. Coequational logic for accessible functors. *Inf. Comput.* 208, 12 (2010), 1469–1489. https://doi.org/10.1016/j.ic.2009.10.010

Arnaud Spiwack and Thierry Coquand. 2010. Constructively Finite? In *Contribuciones científicas en honor de Mirian Andrés Gómez*, Laureano Lambán Pardo, Ana Romero Ibáñez, and Julio Rubio García (Eds.). Universidad de La Rioja, 217–230.

Sam Staton. 2009. Relating Coalgebraic Notions of Bisimulation. In *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7-10, 2009. Proceedings (Lecture Notes in Computer Science)*, Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki (Eds.), Vol. 5728. Springer, 191–205. https://doi.org/10.1007/978-3-642-03741-2_14

The Agda Team. 2018. The Agda wiki. (2018). http://wiki.portal.chalmers.se/agda/.

The Project Everest Team. 2018. The Everest Project. (2018). https://project-everest.github.io/.

The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study.

Niccolò Veltri. 2017. *A Type-Theoretical Study of Nontermination*. Ph.D. Dissertation. Tallinn University of Technology. https://digi.lib.ttu.ee/i/?7631

Andrea Vezzosi. 2017. Streams for Cubical Type Theory. (2017). http://www.cse.chalmers.se/~vezzosi/streams-ctt.pdf