Logical Methods in Computer Science
Volume 17, Issue 1, 2021, pp. 14:1–14:26
https://lmcs.episciences.org/

Submitted Jan. 23, 2020
Published Feb. 10, 2021

THE π-CALCULUS IS BEHAVIOURALLY COMPLETE AND ORBIT-FINITELY EXECUTABLE

BAS LUTTIK AND FEI YANG

Department of Mathematics and Computer Science, Eindhoven Unversity of Technology, The Netherlands
e-mail address: s.p.luttik@tue.nl

Zhejiang Lab, Hangzhou, China
e-mail address: yangf@zhejianglab.com

Abstract. Reactive Turing machines extend classical Turing machines with a facility to model observable interactive behaviour. We call a behaviour (finitely) executable if, and only if, it is equivalent to the behaviour of a (finite) reactive Turing machine. In this paper, we study the relationship between executable behaviour and behaviour that can be specified in the π-calculus. We establish that every finitely executable behaviour can be specified in the π-calculus up to divergence-preserving branching bisimilarity. The converse, however, is not true due to (intended) limitations of the model of reactive Turing machines. That is, the π-calculus allows the specification of behaviour that is not finitely executable up to divergence-preserving branching bisimilarity. We shall prove, however, that if the finiteness requirement on reactive Turing machines and the associated notion of executability is relaxed to orbit-finiteness, then the π-calculus is executable up to (divergence-insensitive) branching bisimilarity.

1. Introduction

In 2006, Jos Baeten initiated a research programme to explore and strengthen the connections between the classical theory of automata and formal languages and concurrency theory, with the aim to establish a unified theory. Such a unified theory should, in particular, upgrade the classical theory of automata and formal languages with a treatment of interaction, and reconsider standard notions and results modulo some form of bisimilarity instead of language equivalence.

Thus, non-deterministic finite automata and pushdown automata, and their correspondences with regular and context-free grammars were explored in the context of branching bisimilarity [BCLT09]. The expressiveness of regular expressions modulo bisimilarity was characterised [BCG07], and the expressiveness of extensions of regular expressions with various forms of parallel composition were studied [BLMT16]. The latter culminated in a concurrency-theoretic variant of Kleene's theorem establishing the correspondence, modulo bisimilarity, between finite automata and regular expressions extended with ACP-style

Key words and phrases: Reactive Turing machine; π-calculus; Behavioural completeness; Orbit-finite executability; Absolute expressiveness of process calculi; Divergence-preserving branching bisimilarity.

LOGICAL METHODS IN COMPUTER SCIENCE
DOI:10.23638/LMCS-17(1:14)2021

parallel composition and encapsulation. Also, the idea that a pushdown automaton is a finite automaton interacting with a stack was formalised [BCT08].

Finite automata and pushdown automata have a straightforward labelled transition system semantics, and are therefore directly amenable to investigation from a concurrency-theoretic perspective. Turing machines, on the other hand, are designed to compute mathematical functions and not to exhibit interactive behaviour: the input for the computation is assumed to be present on the Turing machine tape at the start of the computation, output is what is left on the tape after the computation, and the computation process itself is deemed internal. To add interactivity, an extension of the Turing machine was proposed that associates an action with every computation step [BLT13]. This so-called *reactive* Turing machine does have a labelled transition system semantics and can be studied from a concurrency-theoretic perspective.

In the same way as Turing machines have been used to define which functions are effectively computable, reactive Turing machines can be used to define which processes can be executed by a computing system. A process, mathematically modelled as a labelled transition system, is *executable* if it is behaviourally equivalent to the labelled transition system associated with a reactive Turing machine. Thus, reactive Turing machines provide a way to characterise the *absolute expressiveness* of a process calculus, by determining to what extent transition systems specified in the calculus are executable, and by determining to what extent executable transition systems can be specified in the calculus. If it is possible to specify every executable transition system in a process calculus, then we say that the process calculus is *behaviourally complete*. Note that the behavioural equivalence is a parameter of this method: if a process calculus is not behaviourally complete up to some fine notion of behavioural equivalence (e.g., divergence-preserving branching bisimilarity), it may still be behaviourally complete up to some coarser notion of behavioural equivalence (e.g., the divergence-insensitive variant of branching bisimilarity). The entire spectrum of behavioural equivalences (see van Glabbeek's seminal paper [Gla93]) is at our disposal to draw precise conclusions.

Expressiveness questions have received ample attention in concurrency theory, especially in the context of the π-calculus (see, e.g., [Gor10, FL10]), but mostly pertaining to so-called *relative* expressiveness: Is there a transformation of expressions of one calculus to expressions of another calculus preserving certain behavioural properties? In this article, instead, we consider the *absolute* expressiveness of the π-calculus using the tool of reactive Turing machine.

Our first contribution, which was already announced in [LY15], is a characterisation of the expressiveness of the π-calculus according to the method described above. We prove that the π-calculus is behaviourally complete up to divergence-preserving branching bisimilarity: every executable behaviour can be specified in the π-calculus up to divergence-preserving branching bisimilarity [GW96, GLT09a], which is the finest behavioural equivalence discussed in [Gla93]. Our proof explains how the behaviour of an arbitrary finite reactive Turing machine can be simulated by a π-calculus expression. The specification consists of a component that specifies the behaviour of the tape memory, and a component that specifies the behaviour of the finite control of the reactive Turing machine under consideration. The specification of the behaviour of the tape memory is generic and elegantly uses the link mobility feature of the π-calculus.

We also prove that the converse is not true: it is possible to specify, in the π-calculus, transition systems that are not executable up to divergence-preserving branching bisimilarity. We shall analyse the discrepancy and identify two causes.

The first cause is that the π-calculus presupposes an infinite supply of names, which is technically essential both for the way input is modelled and for the way fresh name generation is implemented. The infinite supply of names in the π-calculus gives rise to an infinite alphabet of actions. The presupposed alphabet of actions of a finite reactive Turing machine is, however, purposely kept finite. Allowing reactive Turing machines to have an infinite alphabet of actions only makes sense if the reactive Turing machine also has infinitely many states or infinitely many data symbols, and then, without any alternative restrictions we get an unrealistically expressive model of executability. In fact, every countable transition system is then executable up to divergence-preserving branching bisimilarity. We refer to [Yan18, Section 6.1] for an elaboration.

The second cause is that the transition system associated with a π-calculus term may have unbounded branching, even if it refers to only finitely many names. Transition systems with unbounded branching are not executable up to divergence-preserving branching bisimilarity, but unbounded branching behaviour can be simulated at the expense of sacrificing divergence preservation. In [LY15], we proved that π-calculus processes referring to only finitely many names are executable up to (the divergence insensitive variant of) branching bisimilarity.

Our second contribution is a refinement and generalisation of the aforementioned result presented in [LY15] regarding the executability of the π-calculus. Building on the foundations laid by Gabbay and Pitts on *nominal techniques* [GP02] and subsequent work by Bojańczyk et al. on Turing machines with atoms [BKLT13], we propose a notion of *orbit-finite* reactive Turing machine and *orbit-finite* executability. The components of an orbit-finite reactive Turing machine (i.e., its sets of states, transitions, data symbols, action alphabet) are allowed to be infinite, as long as they are finitely presentable. We argue that the π-calculus is orbit-finitely executable up to branching bisimilarity.

The paper is organised as follows. In Section 2, the basic definitions of labelled transition system, (divergence-preserving) branching bisimilarty, and reactive Turing machine are recapitulated, and we also recall the syntax and operational semantics of the π-calculus with replication. In Section 3, we prove that the π-calculus is behaviourally complete modulo divergence-preserving branching bisimilarity: a finite specification of reactive Turing machines in the π-calculus is proposed and verified. In Section 4, we discuss the orbit-finite executability of transition systems associated with π-calculus processes. We define the notion of orbit-finite reactive Turing machine and show that every π-calculus term can be simulated up to the divergence insensitive version of branching bisimilarity. The paper ends with some conclusions in Section 5.

2. Preliminaries

2.1. **Labelled Transition System and Behavioural Equivalence.** The transition system is the central notion in the mathematical theory of discrete-event behaviour. It is parameterised by a set $\mathcal{A}$ of *action symbols*, denoting the observable events of a system. We shall later impose extra restrictions on $\mathcal{A}$, e.g., requiring that it be finite or have a particular structure, but for now we let $\mathcal{A}$ be just an arbitrary abstract set. We extend $\mathcal{A}$ with a

special symbol τ, which intuitively denotes unobservable internal activity of the system. We shall abbreviate $\mathcal{A} \cup \{\tau\}$ by $\mathcal{A}_\tau$.

Definition 2.1. An *$\mathcal{A}_\tau$-labelled transition system* is a triple $(S, \longrightarrow, \uparrow)$, where

(1) S is a set of *states*;
(2) $\longrightarrow \subseteq S \times \mathcal{A}_\tau \times S$ is an $\mathcal{A}_\tau$-labelled *transition relation*; and
(3) $\uparrow \in S$ is the initial state.

Let $(S, \longrightarrow, \uparrow)$ be an $\mathcal{A}_\tau$-labelled LTS. We shall usually write $s \xrightarrow{a} t$ in lieu of $(s, a, t) \in \longrightarrow$. The set $\mathit{Reach}(s)$ of states reachable from a state s is defined by

$$\mathit{Reach}(s) = \{s' \in S \mid \exists n \geq 0,\, s_0, \ldots, s_n \in S,\, a_1, \ldots, a_n \in \mathcal{A}_\tau.\, s = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n = s'\} \ .$$

Transition systems can be used to give semantics to programming languages and process calculi. The standard method is to first associate with every program or process expression a transition system (its operational semantics), and then consider programs and process expressions modulo one of the many behavioural equivalences on transition systems that have been studied in the literature. In this paper, we shall use the notion of (divergence-preserving) branching bisimilarity [GW96, Lut20], which is the finest behavioural equivalence discussed in [Gla93], and also the coarsest behavioural equivalence that is a congruence for parallel composition and preserves CTL^*_{-X} formulas [GLT09b].

In the definition of (divergence-preserving) branching bisimilarity we need the following notation: let $\longrightarrow$ be an $\mathcal{A}_\tau$-labelled transition relation on a set S, and let $a \in \mathcal{A}_\tau$; we write $s \xrightarrow{(a)} t$ for "$s \xrightarrow{a} t$ or $a = \tau$ and $s = t$". Furthermore, we denote the transitive closure of $\xrightarrow{\tau}$ by $\longrightarrow^+$ and the reflexive-transitive closure of $\xrightarrow{\tau}$ by $\longrightarrow^*$.

Definition 2.2. Let $T_1 = (S_1, \longrightarrow_1, \uparrow_1)$ and $T_2 = (S_2, \longrightarrow_2, \uparrow_2)$ be transition systems. A *branching bisimulation* from T_1 to T_2 is a binary relation $\mathcal{R} \subseteq S_1 \times S_2$ such that for all states s_1 and s_2, $s_1 \mathcal{R} s_2$ implies

(1) if $s_1 \xrightarrow{a}_1 s_1'$, then there exist $s_2', s_2'' \in S_2$, such that $s_2 \longrightarrow_2^* s_2'' \xrightarrow{(a)} s_2'$, $s_1 \mathcal{R} s_2''$ and $s_1' \mathcal{R} s_2'$;
(2) if $s_2 \xrightarrow{a}_2 s_2'$, then there exist $s_1', s_1'' \in S_1$, such that $s_1 \longrightarrow_1^* s_1'' \xrightarrow{(a)} s_1'$, $s_1'' \mathcal{R} s_2$ and $s_1' \mathcal{R} s_2'$.

The transition systems T_1 and T_2 are *branching bisimilar* (notation: $T_1 \leftrightarrows_{\mathrm{b}} T_2$) if there exists a branching bisimulation $\mathcal{R}$ from T_1 to T_2 s.t. $\uparrow_1 \mathcal{R} \uparrow_2$.

A branching bisimulation $\mathcal{R}$ from T_1 to T_2 is *divergence-preserving* if, for all states s_1 and s_2, $s_1 \mathcal{R} s_2$ implies

(3) if there exists an infinite sequence $(s_{1,i})_{i \in \mathbb{N}}$ such that $s_1 = s_{1,0}$, $s_{1,i} \xrightarrow{\tau} s_{1,i+1}$ and $s_{1,i} \mathcal{R} s_2$ for all $i \in \mathbb{N}$, then there exists a state s_2' such that $s_2 \longrightarrow^+ s_2'$ and $s_{1,i} \mathcal{R} s_2'$ for some $i \in \mathbb{N}$; and
(4) if there exists an infinite sequence $(s_{2,i})_{i \in \mathbb{N}}$ such that $s_2 = s_{2,0}$, $s_{2,i} \xrightarrow{\tau} s_{2,i+1}$ and $s_1 \mathcal{R} s_{2,i}$ for all $i \in \mathbb{N}$, then there exists a state s_1' such that $s_1 \longrightarrow^+ s_1'$ and $s_1' \mathcal{R} s_{2,i}$ for some $i \in \mathbb{N}$.

The transition systems T_1 and T_2 are *divergence-preserving branching bisimilar* (notation: $T_1 \leftrightarrows^{\Delta}_{\mathrm{b}} T_2$) if there exists a divergence-preserving branching bisimulation $\mathcal{R}$ from T_1 to T_2 such that $\uparrow_1 \mathcal{R} \uparrow_2$.

For two LTSs $T_1 = (S_1, \longrightarrow_1, \uparrow_1)$ and $T_2 = (S_2, \longrightarrow_2, \uparrow_2)$, $s_1 \in S_1$ and $s_2 \in S_2$, we write $s_1 \leftrightarrows_{\mathrm{b}} s_2$ ($s_1 \leftrightarrows_{\mathrm{b}}^{\Delta} s_2$) if there is a (divergence-preserving) branching bisimilarity from T_1 to T_2 relating s_1 and s_2. Thus, $\leftrightarrows_{\mathrm{b}}$ is a relation from the states of T_1 to the states of T_2, and it can be shown that it satisfies the conditions of Definition 2.2. We can also write $s_1 \leftrightarrows_{\mathrm{b}} s_2$ ($s_1 \leftrightarrows_{\mathrm{b}}^{\Delta} s_2$) if s_1 and s_2 are states in a single LTS T and related by a (divergence-preserving) branching bisimulation from T to itself.

The relations $\leftrightarrows_{\mathrm{b}}$ and $\leftrightarrows_{\mathrm{b}}^{\Delta}$ are equivalence relations, both as relations on a single transition system, and as relations on a set of transition systems [Bas96, GLT09a].

Next we define the notion of bisimulation up to $\leftrightarrows_{\mathrm{b}}$. Note that we adapt a non-symmetric bisimulation up to relation, which is a useful tool to establish branching bisimilarity later.

Definition 2.3. Let $T_1 = (S_1, \longrightarrow_1, \uparrow_1)$ and $T_2 = (S_2, \longrightarrow_2, \uparrow_2)$ be two transition systems. A relation $\mathcal{R} \subseteq S_1 \times S_2$ is a bisimulation up to $\leftrightarrows_{\mathrm{b}}$ if, whenever $s_1 \mathcal{R} s_2$, then for all $a \in \mathcal{A}_\tau$:

(1) if $s_1 \longrightarrow^* s_1'' \xrightarrow{a} s_1'$, with $s_1 \leftrightarrows_{\mathrm{b}} s_1''$ and $a \neq \tau \vee s_1'' \not\leftrightarrows_{\mathrm{b}} s_1'$, then there exists s_2' such that $s_2 \xrightarrow{a} s_2'$, $s_1'' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_2$ and $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_2'$; and
(2) if $s_2 \xrightarrow{a} s_2'$, then there exist s_1', s_1'' such that $s_1 \longrightarrow^* s_1'' \xrightarrow{a} s_1'$, $s_1'' \leftrightarrows_{\mathrm{b}} s_1$ and $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_2'$.

Lemma 2.4. *If $\mathcal{R}$ is a bisimulation up to $\leftrightarrows_b$, then $\mathcal{R} \subseteq \leftrightarrows_b$.*

Proof. It is sufficient to prove that $\leftrightarrows_{\mathrm{b}} \circ \mathcal{R}$ is a branching bisimulation, since $\leftrightarrows_{\mathrm{b}}$ is reflexive. Let $s_1 \leftrightarrows_{\mathrm{b}} s_2 \mathcal{R} s_3$.

(1) Suppose $s_1 \xrightarrow{a} s_1'$. We distinguish two cases,
   (a) If $a = \tau$ and $s_1 \leftrightarrows_{\mathrm{b}} s_1'$, then $s_1' \leftrightarrows_{\mathrm{b}} s_1 \leftrightarrows_{\mathrm{b}} s_2$, so $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3$.
   (b) Otherwise, we have $a \neq \tau \vee s_1 \not\leftrightarrows_{\mathrm{b}} s_1'$. Then according to Definition 2.2, there exist s_2'' and s_2' such that $s_2 \longrightarrow^* s_2'' \xrightarrow{a} s_2'$, $s_1 \leftrightarrows_{\mathrm{b}} s_2''$ and $s_1' \leftrightarrows_{\mathrm{b}} s_2'$. Note that $s_2 \leftrightarrows_{\mathrm{b}} s_1 \leftrightarrows_{\mathrm{b}} s_2''$, so by Definition 2.3, there exist s_4'', s_4' and s_3' such that $s_3 \xrightarrow{a} s_3'$ and $s_2'' \leftrightarrows_{\mathrm{b}} s_4'' \mathcal{R} s_3$ and $s_2' \leftrightarrows_{\mathrm{b}} s_4' \mathcal{R} s_3'$. Since $s_1' \leftrightarrows_{\mathrm{b}} s_2' \leftrightarrows_{\mathrm{b}} s_4'$ and $s_4' \mathcal{R} s_3'$, it follows that $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3'$.
(2) If $s_3 \xrightarrow{a} s_3'$, then according to Definition 2.3, there exist s_2'' and s_2' such that $s_2 \longrightarrow^* s_2'' \xrightarrow{a} s_2'$, $s_2'' \leftrightarrows_{\mathrm{b}} s_2$ and $s_2' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3'$. Since $s_1 \leftrightarrows_{\mathrm{b}} s_2 \leftrightarrows_{\mathrm{b}} s_2''$ and $s_2'' \xrightarrow{a} s_2'$, by Definition 2.2, there exist s_1'' and s_1' such that $s_1 \longrightarrow^* s_1'' \xrightarrow{(a)} s_1'$ with $s_1'' \leftrightarrows_{\mathrm{b}} s_2''$ and $s_1' \leftrightarrows_{\mathrm{b}} s_2'$. Since $s_2'' \leftrightarrows_{\mathrm{b}} s_2 \mathcal{R} s_3$ and $s_2' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3'$, it follows that $s_1'' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3$ and $s_1' \leftrightarrows_{\mathrm{b}} \circ \mathcal{R} s_3'$.

Therefore, a branching bisimulation up to $\leftrightarrows_{\mathrm{b}}$ is included in $\leftrightarrows_{\mathrm{b}}$. □

2.2. **Reactive Turing Machines and Executability.** The notion of (finite) reactive Turing machine (RTM) was put forward in [BLT13] to mathematically characterise which behaviour is executable by a conventional computing system. In this section, we recall the definition of RTMs and the ensued notion of executable transition system. The definition of RTMs is parameterised by a set $\mathcal{A}_\tau$ of *action symbols* and a set $\mathcal{D}$ of *data symbols*. We extend $\mathcal{D}$ with a special symbol $\Box \notin \mathcal{D}$ to denote a blank tape cell; the elements of $\mathcal{D}_\Box = \mathcal{D} \cup \{\Box\}$ are called *tape symbols*.

Definition 2.5. A *reactive Turing machine* (RTM) $\mathcal{M}$ is a tuple $(S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow, \uparrow)$, where

(1) S is a set of *states*;
(2) $\mathcal{D}_\Box$ is a set of *tape symbols* including the special symbol $\Box$ denoting a blank tape cell;
(3) $\mathcal{A}_\tau$ is a set of *action symbols* including the special symbol τ denoting an unobservable event;

(4) $\longrightarrow \subseteq S \times \mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L, R\} \times S$ is a $(\mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L, R\})$-labelled *transition relation* (we write $s \xrightarrow{a[d/e]M} t$ for $(s, d, a, e, M, t) \in \longrightarrow$); and
(5) $\uparrow \in S$ is a distinguished *initial state*.

An RTM is *finite* if the sets S, $\mathcal{D}_\Box$ and $\mathcal{A}_\tau$ are all finite.

Remark 2.6. The reactive Turing machines proposed in [BLT13] are finite by definition. In Section 4 we wish to investigate a relaxation of the finiteness requirement, and therefore it is convenient to provide a more general definition of the notion here. Until Section 4 all RTMs are assumed to be finite, even if we do not explicitly say so.

Remark 2.7. The original definition of RTMs in [BLT13] includes an extra facility to declare a subset of the states of an RTM as final states, and so does the associated notion of executable transition system. In this paper, however, our goal is to explore the relationship between the transition systems associated with RTMs and those that can be specified in the π-calculus. Since the π-calculus does not include the facility to specify that a state has the option to terminate, we leave it out from the definition of RTMs too.

Intuitively, the meaning of a transition $s \xrightarrow{a[d/e]M} t$ is that whenever $\mathcal{M}$ is in state s, and d is the symbol currently read by the tape head, then it may execute the action a, write symbol e on the tape (replacing d), move the read/write head one position to the left or the right on the tape (depending on whether $M = L$ or $M = R$), and then end up in state t.

To formalise the intuitive understanding of the operational behaviour of RTMs, we associate with every RTM $\mathcal{M}$ an $\mathcal{A}_\tau$-labelled transition system $\mathcal{T}(\mathcal{M})$. The states of $\mathcal{T}(\mathcal{M})$ are the configurations of $\mathcal{M}$, which consist of a state from S, its tape contents, and the position of the read/write head. We denote by $\check{\mathcal{D}}_\Box = \{\check{d} \mid d \in \mathcal{D}_\Box\}$ the set of *marked* symbols; a *tape instance* is a sequence $\delta \in (\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ such that δ contains exactly one element of $\check{\mathcal{D}}_\Box$, indicating the position of the read/write head. We adopt a convention to concisely denote new placement of the tape head marker. Let δ be an element of $\mathcal{D}_\Box^*$. Then by $\delta^<$ we denote the element of $(\mathcal{D}_\Box \cup \check{\mathcal{D}}_\Box)^*$ obtained by placing the tape head marker on the right-most symbol of δ if δ is non-empty, and $\check{\Box}$ otherwise. Similarly ${}^>\delta$ is obtained by placing the tape head marker on the left-most symbol of δ if δ is non-empty, and $\check{\Box}$ otherwise.

Definition 2.8. Let $\mathcal{M} = (S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow, \uparrow)$ be an RTM. The *transition system* $\mathcal{T}(\mathcal{M})$ *associated with* $\mathcal{M}$ is defined as follows:

(1) its set of states is the set $Conf_{\mathcal{M}} = \{(s, \delta) \mid s \in S,\ \delta \text{ a tape instance}\}$ of all configurations of $\mathcal{M}$;
(2) its transition relation $\longrightarrow \subseteq Conf_{\mathcal{M}} \times \mathcal{A}_\tau \times Conf_{\mathcal{M}}$ is the least relation satisfying, for all $a \in \mathcal{A}_\tau$, $d, e \in \mathcal{D}_\Box$ and $\delta_L, \delta_R \in \mathcal{D}_\Box^*$:
   - $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L{}^< e \delta_R)$ iff $s \xrightarrow{a[d/e]L} t$, and
   - $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta_L e\, {}^>\delta_R)$ iff $s \xrightarrow{a[d/e]R} t$, and
(3) its initial state is the configuration $(\uparrow, \check{\Box})$.

Turing introduced his machines to define the notion of *effectively computable function* [Tur36]. By analogy, the notion of finite RTM can be used to define a notion of *effectively executable behaviour*.

A transition system is *(finitely) executable* if it is the transition system associated with some (finite) RTM. Usually, we shall be interested in executability up to some behavioural

equivalence (e.g., the divergence-preserving or divergence-insensitive variant of branching bisimilarity).

Definition 2.9. A transition system T is *finitely executable up to branching bisimilarity* if there exists a finite RTM $\mathcal{M}$ such that $T \leftrightarrows_{\mathrm{b}} \mathcal{T}(\mathcal{M})$. It is *finitely executable up to divergence-preserving branching bisimilarity* if there exists a finite RTM $\mathcal{M}$ such that $T \leftrightarrows_{\mathrm{b}}^{\Delta} \mathcal{T}(\mathcal{M})$.

2.3. **π-Calculus.** The π-calculus was proposed by Milner, Parrow and Walker in [Mil92] as a language to specify processes with link mobility. The expressiveness of many variants of the π-calculus has been extensively studied. In this paper, we shall consider the basic version presented in [SW01], excluding the match prefix. We recapitulate some definitions from [SW01] below and refer to the book for detailed explanations.

We presuppose a countably infinite set $\mathcal{N}$ of names; we use strings of lower case letters for elements of $\mathcal{N}$. The *prefixes*, *processes* and *summations* of the π-calculus are, respectively, defined by the following grammar:

$$\begin{aligned} \pi &\coloneqq \overline{x}\,y \mid x(z) \mid \tau \qquad (x, y, z \in \mathcal{N}) \\ P &\coloneqq M \mid P \,|\, P \mid (\nu z)P \mid\, !P \\ M &\coloneqq \mathbf{0} \mid \pi.P \,|\, M + M \ . \end{aligned}$$

In $x(z).P$ and $(\nu z)P$, the displayed occurrence of the name z is *binding* with scope P. An occurrence of a name in a process is *bound* if it is, or lies within the scope of, a binding occurrence in P; otherwise it is free. We use $\mathsf{fn}(P)$ to denote the set of names that occur free in P, and $\mathsf{bn}(P)$ to denote the set of names that occur bound in P. We write $P =_\alpha Q$ if P and Q are *α-convertible*, i.e., if Q can be obtained from P by a finite number of changes of bound names (see [SW01] for details).

We define the operational behaviour of π-processes by means of the structural operational semantics in Table 1, in which α ranges over the set of actions of the π-calculus

$$\mathcal{A}_\pi = \{x\,y, \overline{x}\,y, \overline{x}\,(z) \mid x, y, z \in \mathcal{N}\} \cup \{\tau\} \ . \tag{2.1}$$

The rules in Table 1 define on π-terms an $\mathcal{A}_\pi$-labelled transition relation $\longrightarrow$. Then, we can associate with every π-term P an $\mathcal{A}_\pi$-labelled transition system $\mathcal{T}(P) = (S_P, \longrightarrow_P, P)$. The set of states S_P of $\mathcal{T}(P)$ consists of all π-terms reachable from P, the transition relation $\longrightarrow_P$ of $\mathcal{T}(P)$ is obtained by restricting the transition relation $\longrightarrow$ defined by the structural operational rules to S_P (i.e., $\longrightarrow_P = \longrightarrow \cap\, (S_P \times \mathcal{A}_\pi \times S_P)$), and the initial state of $\mathcal{T}(P)$ is the π-term P.

For convenience, we sometimes want to abbreviate interactions that involve the transmission of no name at all, or more than one name. Instead of giving a full treatment of the polyadic π-calculus (see [SW01]), we define the following abbreviations, assuming $w \notin \mathsf{fn}(P)$ in both:

$$\begin{aligned} &\overline{x}\,\langle y_1, \ldots, y_n\rangle.P \stackrel{\text{def}}{=} (\nu w)\overline{x}\,w.\overline{w}\,y_1.\cdots\overline{w}\,y_n.P, \text{ and} \\ &x(z_1, \ldots, z_n).P \stackrel{\text{def}}{=} x(w).w(z_1).\cdots w(z_n).P \ . \end{aligned}$$

Divergence-preserving branching bisimilarity is not a congruence with respect to π-calculus parallel composition and restriction, due to subtle issues regarding free and bound names stemming from scope extrusion. In the remainder of this section we introduce several

$$(\textsc{Tau})\ \frac{}{\tau.P \xrightarrow{\tau} P} \qquad (\textsc{Out})\ \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad (\textsc{Inp})\ \frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$$

$$(\textsc{Sum-l})\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad (\textsc{Alpha})\ \frac{P \xrightarrow{\alpha} P'}{Q \xrightarrow{\alpha} P'}\ P =_\alpha Q$$

$$(\textsc{Par-l})\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset \qquad (\textsc{Rep-act})\ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

$$(\textsc{Comm-l})\ \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad (\textsc{Close-l})\ \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} (\nu z)(P' \mid Q')}\ z \notin \mathsf{fn}(Q)$$

$$(\textsc{Res})\ \frac{P \xrightarrow{\alpha} P'}{(\nu z)P \xrightarrow{\alpha} (\nu z)P'}\ z \notin \alpha \qquad (\textsc{Open})\ \frac{P \xrightarrow{\overline{x}z} P'}{(\nu z)P \xrightarrow{\overline{x}(z)} P'}\ z \neq x$$

$$(\textsc{Rep-comm})\ \frac{P \xrightarrow{\overline{x}\,y} P',\ P \xrightarrow{x\,y} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P} \qquad (\textsc{Rep-close})\ \frac{P \xrightarrow{\overline{x}\,(z)} P',\ P \xrightarrow{x\,z} P''}{!P \xrightarrow{\tau} (\nu z)(P' \mid P'') \mid !P}$$

Table 1: The operational rules for the π-calculus; the symmetric variants (Par-r), (Comm-r) and (Close-r) of the rules (Par-l), (Comm-l) and (Close-l), respectively, have been omitted for conciseness.

$$\begin{array}{rcl@{\qquad}rcl}
P_1 + (P_2 + P_3) & \equiv & (P_1 + P_2) + P_3 & P_1 \mid (P_2 \mid P_3) & \equiv & (P_1 \mid P_2) \mid P_3 \\
P_1 + P_2 & \equiv & P_2 + P_1 & P_1 \mid P_2 & \equiv & P_2 \mid P_1 \\
P + \mathbf{0} & \equiv & P & P \mid \mathbf{0} & \equiv & P \\
(\nu z)(\nu w)P & \equiv & (\nu w)(\nu z)P & (\nu z)(P_1 \mid P_2) & \equiv & P_1 \mid (\nu z)P_2 \quad (\text{if } z \notin \mathsf{fn}(P_1)) \\
(\nu z)\mathbf{0} & \equiv & \mathbf{0} & !P & \equiv & P \mid !P
\end{array}$$

Table 2: The axioms of structural congruence.

technical tools that we shall use in our proof that the π-calculus is behaviourally complete up to divergence-preserving branching bisimilarity in Section 3.

Structural congruence, denoted by $\equiv$, is the least congruence on π-terms satisfying the axioms in Table 2. The following lemma establishes that structurally congruent π-terms are divergence-preserving branching bisimilar.

Lemma 2.10. *For all π-terms P and Q, if $P \equiv Q$, then $P \leftrightarrows_b^\Delta Q$.*

Proof. Using the Harmony Lemma [SW01, Lemma 1.4.15] it is straightforward to establish that $\equiv$ is a divergence-preserving branching bisimulation, from which the lemma immediately follows. □

Another useful tool in arguments establishing divergence-preserving branching bisimilarity will be the notion of deterministic internal computation.

Definition 2.11. Let P and P' be π-temrs. Then P has a *deterministic internal computation* to P' (notation: $P \rightsquigarrow P'$) if there exist π-terms $P_0, \ldots, P_n$ such that $P = P_0 \xrightarrow{\tau} \cdots \xrightarrow{\tau} P_n = P'$ and for every π-term P_i $(1 \leq i < n)$ it holds that $P_i \xrightarrow{\alpha} P_i'$ implies $\alpha = \tau$ and $P_i' = P_{i+1}$.

All π-terms on a deterministic internal computation are divergence-preserving branching bisimilar, and

Lemma 2.12. *Let P and P' be π-terms. If $P \rightsquigarrow P'$, then for all $z_1, \ldots, z_n \in \mathcal{N}$ and for all π-terms Q it holds that $(\nu z_1, \ldots, z_n)(P \mid Q) \leftrightarrows_b^\Delta (\nu z_1, \ldots, z_n)(P' \mid Q)$.*

We say that a π-term P has *reachable bound output* if there exist π-terms $P_0, \ldots, P_n, P'$, actions $\alpha_1, \ldots, \alpha_n \in \mathcal{A}_\pi$ and names $x, z \in \mathcal{N}$ such that

$$P = P_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P_n \xrightarrow{\overline{x}\,(z)} P' \ .$$

Lemma 2.13. *Let P_1, P_2, Q_1 and Q_2 be π-terms without reachable bound output. If $P_1 \leftrightarrows_b^\Delta Q_1$ and $P_2 \leftrightarrows_b^\Delta Q_2$, then $P_1 \mid P_2 \leftrightarrows_b^\Delta Q_1 \mid Q_2$.*

Let P be a π-term and let $z \in \mathcal{N}$. We say that P *eventually outputs* z if there exist π-terms $P_0, \ldots, P_n, P'$, actions $\alpha_1, \ldots, \alpha_n \in \mathcal{A}_\pi$ and a name $x \in \mathcal{N}$ such that

$$P = P_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} P_n \xrightarrow{\overline{x}\,z} P' \ .$$

Lemma 2.14. *Let P and Q, and let $z \in \mathcal{N}$ be such that neither P nor Q eventually outputs z. If $P \leftrightarrows_b^\Delta Q$, then $(\nu z)P \leftrightarrows_b^\Delta (\nu z)Q$.*

3. The π-Calculus is Behaviourally Complete

In the previous section, we have introduced the π-calculus as a language to specify behaviour of systems with link mobility, and we have proposed RTMs to define a notion of executable behaviour. In this section we prove that every executable behaviour can be specified in the π-calculus up to divergence-preserving branching bisimilarity. To this end, we associate with every RTM $\mathcal{M}$ a π-term P that simulates the behaviour of $\mathcal{M}$ up to divergence-preserving branching bisimilarity, that is, $\mathcal{T}(\mathcal{M}) \leftrightarrows_{\mathrm{b}}^\Delta \mathcal{T}(P)$.

The structure of our specification is illustrated in Figure 1. In this figure, each node represents a parallel component of the specification, each labelled arrow stands for a communication channel, with the arrow pointing from sender to receiver. The dashed lines represent links between cells that are achieved by instantiating parameters with the same name. The equalities on arrows and dashed lines indicate identifications that will thus be made; for instance, the equality $t_{i-1} = l_i$ indicates that the parameter t of the process C_{i-1} will be instantiated with the same name as the parameter l of the process C_i. The specification consists of a generic finite specification of the behaviour of a tape (parallel components H_k, $B_{l,k}$, C_k, $B_{r,k}$ in Figure 1), and a finite specification of a control process that is specific for the RTM $\mathcal{M}$ under consideration (parallel component S in Figure 1). We first discuss the generic specification of the tape in Section 3.1, then we discuss how to add

a suitable control process specific for $\mathcal{M}$ in Section 3.2 proving that $\mathcal{M}$ is simulated by the parallel composition of the two parts.

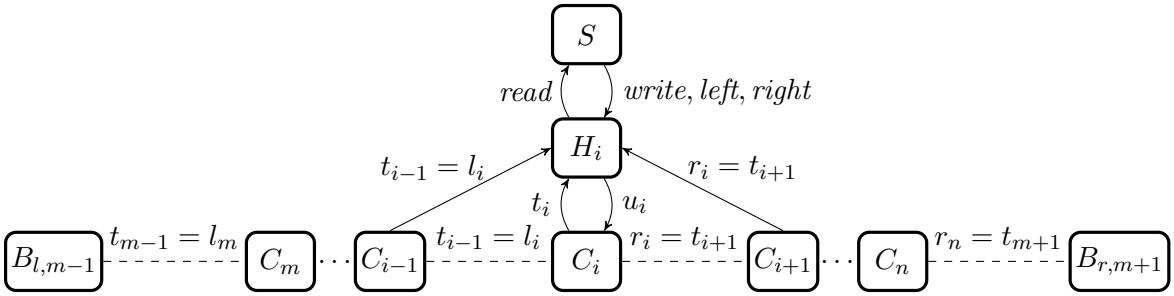

Figure 1: Specification of an RTM utilizing the linking structure of the π-calculus

3.1. **Tape.** In [BBK87], the behaviour of the tape of a Turing machine is finitely specified in ACP_τ making use of finite specifications of two stacks. That specification is not easily modified to take intermediate termination into account, and therefore, in [BLT13], an alternative solution is presented, specifying the behaviour of a tape in TCP_τ by using a finite specification of a queue (see also [BBR10]). In this paper, we will exploit the link passing feature of the π-calculus to give a more direct specification. In particular, we shall model the tape as a collection of cells endowed with a link structure that organises them in a linear fashion.

We first give an informal description of the behaviour of a tape. The state of a tape is characterised by a tape instance $\delta_L \check{d} \delta_R$, consisting of a finite (but unbounded) sequence of data with the current position of the tape head indicated by $\check{}$. The tape may then exhibit the following observable actions:

(1) $\overline{\mathit{read}}\, d$: the datum under the tape head is output along the channel *read*;
(2) $\mathit{write}(e)$: a datum e is written on the position of the tape head, resulting in a new tape instance $\delta_L \check{e} \delta_R$; and
(3) *left*, *right*: the tape head moves one position left or right, resulting in $\delta_L{}^{<} d \delta_R$ or $\delta_L d\, {}^{>}\delta_R$, respectively.

Henceforth, we assume that tape symbols are included in the set of names, i.e., that $\mathcal{D}_\Box \subseteq \mathcal{N}$.

In our π-calculus specification of the behaviour of a tape, each individual tape cell is specified as a separate component, and there is a separate component modelling the tape head. A tape cell stores a datum d, represented by a free name in the specification, and it has pointers l and r to its left and right neighbour cells. Furthermore, it has two links to the component modelling the tape head: the link u is used by the tape head for updating the datum, and the link t serves as a general communication channel for communicating all relevant information about the cell to the tape head.

The following π-term represents the behaviour of a tape cell:

$$\begin{aligned} C &\stackrel{\text{def}}{=} c(t,l,r,u,d).C(t,l,r,u,d) \\ C(t,l,r,u,d) &\stackrel{\text{def}}{=} u(e).\overline{c}\langle t,l,r,u,e\rangle.\mathbf{0} + \overline{t}\langle l,r,u,d\rangle.\overline{c}\langle t,l,r,u,d\rangle.\mathbf{0}\ . \end{aligned}$$

A cell is created by a synchronisation on name c, by which all relevant information about the cell is passed; we shall have a component $!C$ facilitate the generation of new incarnations of existing tape cells. Note that the behaviour of an individual tape cell $C(t, l, r, u, d)$ is as follows: either it receives along channel u an update e for its datum d, after which it recreates itself with datum e in place of d; or it outputs all relevant information about itself (i.e., the links to its left and right neighbours, its update channel u, and the stored datum d) to the tape head along channel t, after which it recreates itself. The following lemma, which is a straightforward consequence of the definition of C and the operational rules of the π-calculus, expresses that recreation proceeds via a deterministic internal computation.

Lemma 3.1. $(\nu c)(\overline{c}\langle t, l, r, u, d\rangle.\mathbf{0} \mid !C) \rightsquigarrow\equiv (\nu c)(C(t, l, r, u, d) \mid !C)$

At any moment, the number of tape cells will be finite. To model the unbounded nature of the tape, we define a process B that serves to generate new blank tape cells on either side of the tape whenever needed:

$$\begin{aligned} B &\stackrel{\text{def}}{=} b_l(t, r).\,(\nu u, l)B_l(t, l, r, u) + b_r(t, l).\,(\nu u, r)B_r(t, l, r, u) \\ B_l(t, l, r, u) &\stackrel{\text{def}}{=} \overline{t}\langle l, r, u, \Box\rangle.\overline{b_l}\langle l, t\rangle.\overline{c}\langle t, l, r, u, \Box\rangle.\mathbf{0} \\ B_r(t, l, r, u) &\stackrel{\text{def}}{=} \overline{t}\langle l, r, u, \Box\rangle.\overline{b_r}\langle r, t\rangle.\overline{c}\langle t, l, r, u, \Box\rangle.\mathbf{0}\ . \end{aligned}$$

Note that B offers the choice to either create a blank tape cell at the left-hand side of the tape through $B_l(t, l, r, u)$, or a blank tape cell at the right-hand side of the tape through $B_r(t, l, r, u)$. In the first case, suppose the original leftmost cell has the channels t_o and l_o, for itself and its left neighbour, respectively, then for the new cell, we have $t = l_o$ and $r = t_o$, in order to maintain the links to its neighbour. Moreover, at the creation of the new blank cell, two new links are created too: u is the update channel of the new blank cell, and l will later be used as the link to generate another cell. Thus, an extra blank cell is generated on the left through $\overline{b_l}\langle l, t\rangle.\mathbf{0}$ and the original blank cell on the left is promoted to a regular cell by $\overline{c}\langle t, l, r, u, \Box\rangle.\mathbf{0}$. In the second case, a symmetrical procedure is implemented by $B_r(t, l, r, u)$.

Lemma 3.2. *We have*

$$\begin{aligned} &(\nu b_l, b_r, c)(\overline{b_l}\langle l, t\rangle.\overline{c}\langle t, l, r, u, \Box\rangle.\mathbf{0} \mid !B \mid !C) \\ &\qquad \rightsquigarrow\equiv (\nu b_l, b_r, c, u', l')(B_l(l, l', t, u') \mid C(t, l, r, u, \Box) \mid !B \mid !C) \end{aligned}$$

and

$$\begin{aligned} &(\nu b_l, b_r, c)(\overline{b_r}\langle r, t\rangle.\overline{c}\langle t, l, r, u, \Box\rangle.\mathbf{0} \mid !B \mid !C) \\ &\qquad \rightsquigarrow\equiv (\nu b_l, b_r, c, u', r')(B_r(r, t, r', u') \mid C(t, l, r, u, \Box) \mid !B \mid !C) \end{aligned}$$

Throughout the simulation of an RTM, the number of parallel components modelling individual tape cells will grow. We shall presuppose a numbering of these parallel components with consecutive integers from some interval $[m, n]$ (m and n are integers such that $m \leq n$), in agreement with the link structure. The numbering is reflected by a naming scheme that adds the subscript i to the links t, l, r, u and d of the ith cell ($m \leq i \leq n$). We abbreviate $C(t_i, l_i, r_i, u_i, d_i)$ by $C_i(d_i)$, and $B_l(t_i, l_i, r_i, u_i)$ and $B_l(t_i, l_i, r_i, u_i)$ by $B_{l,i}$ and $B_{r,i}$, respectively. Let $\vec{d}_{[m,n]} = d_m, d_{m+1}, \ldots, d_{n-1}, d_n$; we define:

$$\mathit{Cells}_{[m,n]}(\vec{d}_{[m,n]}) \stackrel{\text{def}}{=} (\nu b_l, b_r, c)(B_{l,m-1} \mid C_m(d_m) \mid \cdots \mid C_n(d_n) \mid B_{r,n+1} \mid !C \mid !B)\ .$$

The component modelling the tape head serves as the interface between the tape cells and the RTM-specific control process. It is defined as:

$$\begin{aligned} H &\stackrel{\text{def}}{=} h(t,l,r,u,d).H(t,l,r,u,d) \\ H(t,l,r,u,d) &\stackrel{\text{def}}{=} \overline{read}\,d.\overline{h}\langle t,l,r,u,d\rangle.\mathbf{0} + write(e).\overline{u}\,e.\overline{h}\langle t,l,r,u,e\rangle.\mathbf{0} \\ &\quad + left.l(l',r',u',d').\overline{h}\langle l,l',r',u',d'\rangle.\mathbf{0} \\ &\quad + right.r(l',r',u',d').\overline{h}\langle r,l',r',u',d'\rangle.\mathbf{0}\ . \end{aligned}$$

The tape head maintains two links to the current cell (a communication channel t and an update channel u), as well as links to its left and right neighbour cells (l and r, respectively). Furthermore, the tape head remembers the datum d in the current cell. The datum d may be output along the *read*-channel. Furthermore, a new datum e may be received through the *write*-channel, which is then forwarded through the update channel u to the current cell. Finally, the tape head may receive instructions to move left or right, which has the effect of receiving information about the left or right neighbours of the current cell through l or r, respectively. In all cases, a new incarnation of the tape head is started, with a call on the h-channel.

Lemma 3.3. $(\nu h)(\overline{h}\langle t,l,r,u,d\rangle.\mathbf{0} \mid !H) \rightsquigarrow\equiv (\nu h)(H(t,l,r,u,d) \mid !H)$

Let $\vec{t}_{[m,n]} = t_m, t_{m+1}, \ldots, t_{n-1}, t_n$, let $\vec{u}_{[m,n]} = u_m, u_{m+1}, \ldots, u_{n-1}, u_n$, and let $H_i(d_i) = H(t_i,l_i,r_i,u_i,d_i)$; we define

$$Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \stackrel{\text{def}}{=} (\nu \vec{t}_{[m-1,n+1]}, \vec{u}_{[m,n]})((\nu h)(H_i(d_i) \mid !H) \mid Cells_{[m,n]}(\vec{d}_{[m,n]}))\ .$$

We shall write $P \xrightarrow{a} \leftrightarrows^{\Delta}_{\mathrm{b}} P'$ for "there is a P'' such that $P \xrightarrow{a} P''$ and $P'' \leftrightarrows^{\Delta}_{\mathrm{b}} P'$".

Lemma 3.4. *$Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{\alpha} T'$ if, and only if, at least one of the following holds:*

(1) $\alpha = \overline{read}\,d_i$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^i_{[m,n]}(\vec{d}_{[m,n]})$, *or*
(2) $\alpha = write\,e$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^i_{[m,n]}(d_{[m,i-1]}, e, d_{[i+1,n]})$, *or*
(3) $\alpha = left$, $i > m$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^{i-1}_{[m,n]}(\vec{d}_{[m,n]})$, *or*
(4) $\alpha = left$, $i = m$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^{i-1}_{[m-1,n]}(\Box, \vec{d}_{[m,n]})$, *or*
(5) $\alpha = right$, $i < n$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^{i+1}_{[m,n]}(\vec{d}_{[m,n]})$, *or*
(6) $\alpha = right$, $i = n$ *and* $T' \leftrightarrows^{\Delta}_{b} Tape^{i+1}_{[m,n+1]}(\vec{d}_{[m,n]}, \Box)$.

Proof. The component $Cells_{[m,n]}(\vec{d}_{[m,n]})$ of $Tape^i_{[m,n]}(\vec{d}_{[m,n]})$ only admits interactions on the channels $\vec{t}_{[m-1,n+1]}$ and $\vec{u}_{[m,n]}$, which are restricted in $Tape^i_{[m,n]}(\vec{d}_{[m,n]})$. Hence, the only transitions afforded by $Tape^i_{[m,n]}(\vec{d}_{[m,n]})$ are those faciliated by the component $H_i(d_i)$, from which it is clear that $\alpha = \overline{read}\,d_i$, $\alpha = write\,e$, $\alpha = left$, or $\alpha = right$. After each of these transitions, the component $H_i(d_i)$ initiates deterministic internal computations of one or more parallel components:

If $\alpha = \overline{read}\,d_i$, then, note that, up to structural congruence, T' is obtained from $Tape^i_{[m,n]}(\vec{d}_{[m,n]})$ by replacing the component $H_i(d_i)$ by $\overline{h}\langle t_i,l_i,r_i,u_i,d_i\rangle.\mathbf{0}$. Since

$$(\nu h)(\overline{h}\langle t_i,l_i,r_i,u_i,d_i\rangle.\mathbf{0} \mid !H) \rightsquigarrow\equiv (\nu h)(H(t_i,l_i,r_i,u_i,d_i) \mid !H)\ ,$$

by Lemma 3.3, it follows by Lemma 2.12 that $T' \leftrightarrows^{\Delta}_{\mathrm{b}} Tape^i_{[m,n]}(\vec{d}_{[m,n]})$.

If $\alpha = \mathit{write}\, e$, then, note that, up to structural congruence, T' is obtained from $\mathit{Tape}^i_{[m,n]}(\vec{d}_{[m,n]})$ by replacing the component $H_i(d_i)$ by $\overline{u}\, e.\overline{h}\langle t_i, l_i, r_i, u_i, e\rangle.\mathbf{0}$. Then, $\overline{u}\, e$ interacts with $\mathit{Cells}_{[m,n]}(\vec{d}_{[m,n]})$ to update d_i to e, giving rise to a deterministic internal communication $T' \rightsquigarrow T''$. Moreover, T'' is obtained from $\mathit{Tape}^i_{[m,n]}(d_{[m,i-1]}, e, d_{[i+1,n]})$ by replacing the component $H_i(e)$ by $\overline{h}\langle t_i, l_i, r_i, u_i, e\rangle.\mathbf{0}$ and the component $C_i(d_i)$ by $\overline{c}\langle t_i, l_i, r_i, u_i, e\rangle.\mathbf{0}$. Since, by Lemma 3.1,

$$(\nu c)(\overline{c}\langle t_i, l_i, r_i, u_i, e\rangle.\mathbf{0} \mid !C) \rightsquigarrow\equiv (\nu c)(C(t_i, l_i, r_i, u_i, e) \mid !C) \ ,$$

and, by Lemma 3.3,

$$(\nu h)(\overline{h}\langle t_i, l_i, r_i, u_i, e\rangle.\mathbf{0} \mid !H) \rightsquigarrow\equiv (\nu h)(H(t_i, l_i, r_i, u_i, e) \mid !H) \ ,$$

it follows by Lemma 2.12 that $T'' \leftrightarrows^{\Delta}_{\mathrm{b}} \mathit{Tape}^i_{[m,n]}(d_{[m,i-1]}, e, d_{[i+1,n]})$, and hence

$$T'' \leftrightarrows^{\Delta}_{\mathrm{b}} \mathit{Tape}^i_{[m,n]}(d_{[m,i-1]}, e, d_{[i+1,n]}) \ .$$

If $\alpha = \mathit{left}$ or $\alpha = \mathit{right}$ then the arguments are similar as above, using also Lemma 3.2 if $i = m$ or $i = n$, respectively. □

3.2. **Finite control.** We associate with every RTM $\mathcal{M} = (S_{\mathcal{M}}, \mathcal{D}_{\square}, \mathcal{A}_{\tau}, \longrightarrow_{\mathcal{M}}, \uparrow_{\mathcal{M}})$ a finite specification of its control process. Here m can be either *left* or *right*.

$$S \stackrel{\text{def}}{=} \sum_{s \in S_{\mathcal{M}}} s. \sum_{d \in \mathcal{D}_{\square}} d.S_{s,d}$$

$$S_{s,d} \stackrel{\text{def}}{=} \sum_{(s,d,a,e,m,t) \in \longrightarrow_{\mathcal{M}}} a.\overline{\mathit{write}}\, e.\overline{m}.\mathit{read}(f).\overline{t}.\overline{f}.\mathbf{0}$$

We define

$$\mathit{Control}_{s,d} \stackrel{\text{def}}{=} S_{s,d} \mid !S \ .$$

The following lemma characterises the behaviour of the control process.

Lemma 3.5. *Let $\mathcal{M} = (S_{\mathcal{M}}, \mathcal{D}_{\square}, \mathcal{A}_{\tau}, \longrightarrow_{\mathcal{M}}, \uparrow_{\mathcal{M}})$ be an RTM. Then*

$$\mathit{Control}_{s,d} \xrightarrow{a} \xrightarrow{\overline{\mathit{write}}\, e} \xrightarrow{\overline{m}} \xrightarrow{\mathit{read}\, f} \rightsquigarrow\equiv \mathit{Control}_{t,f} \ .$$

if and only if $(s, d, a, e, m, t) \in \longrightarrow_{\mathcal{M}}$.

Given an RTM $\mathcal{M}$, we associate with every configuration $(s, \delta_L \check{d} \delta_R)$ a π-term $M_{s,\delta_L \check{d} \delta_R}$, consisting of a parallel composition of the specifications of its tape instance and control process. Let $\vec{s} = s_1, s_2, \ldots, s_m \in S_{\mathcal{M}}$, let $\vec{e} = e_1, e_2, \ldots, e_n \in \mathcal{D}_{\square}$, and let $\vec{r} = \mathit{read}, \mathit{write}, \mathit{left}, \mathit{right}$; we define

$$M_{s,\delta_L \check{d} \delta_R} = (\nu \vec{e}, \vec{s}, \vec{r})(\mathit{Control}_{s,d} \mid \mathit{Tape}^i_{[m,n]}(\vec{d}_{[m,n]})), \ \mathit{where}\ \vec{d}_{[m,n]} = \delta_L \check{d} \delta_R \ .$$

The following lemma establishes that $M_{s,\delta_L \check{d} \delta_R}$ simulates the execution steps of the RTM in the configuration $(s, \delta_L \check{d} \delta_R)$.

Lemma 3.6. *Given an RTM $\mathcal{M} = (S_{\mathcal{M}}, \mathcal{D}_{\Box}, \mathcal{A}_{\tau}, \longrightarrow_{\mathcal{M}}, \uparrow_{\mathcal{M}})$, for every configuration $(s, \delta_L \check{d} \delta_R)$, we have*

$$M_{s,\delta_L \check{d} \delta_R} \xrightarrow{a} \leftrightarrows_{b}^{\Delta} M_{t,\delta'_L \check{f} \delta'_R}$$

if and only if there is a transition $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta'_L \check{f} \delta'_R)$.

Proof. On the one hand, if $M_{s,\delta_L \check{d} \delta_R} \xrightarrow{a} \leftrightarrows_{\mathrm{b}}^{\Delta} M_{t,\delta'_L \check{f} \delta'_R}$, then

$$Control_{s,d} \xrightarrow{a} \xrightarrow{\overline{write}\, e} \xrightarrow{\overline{m}} \xrightarrow{read\, f} \rightsquigarrow\equiv Control_{t,f} \ ,$$

so by Lemma 3.5, $(s, d, a, e, m, t) \in \longrightarrow_{\mathcal{M}}$.

On the other hand, if $(s, \delta_L \check{d} \delta_R) \xrightarrow{a} (t, \delta'_L \check{f} \delta'_R)$, then $(s, d, a, e, m, t) \in \longrightarrow_{\mathcal{M}}$. Hence, by Lemma 3.5, we have

$$M_{s,\delta_L \check{d} \delta_R} \xrightarrow{a} (\nu \vec{e}, \vec{s}, \vec{r})(\overline{write}\, e.\overline{m}.read(f).\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid\, Tape^{i}_{[m,n]}(\vec{d}_{[m,n]})) = M' \ .$$

It then remains to prove that $M' \leftrightarrows_{\mathrm{b}}^{\Delta} M_{t,\delta'_L \check{f} \delta'_R}$.

To this end, first note that by Lemma 3.4, we get

$$M' \rightsquigarrow\equiv (\nu \vec{e}, \vec{s}, \vec{r})(\overline{m}.read(f).\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid T') \ ,$$

where $T' \leftrightarrows_{\mathrm{b}}^{\Delta} Tape^{i}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)$, so by Lemmas 2.10, 2.12, 2.13 and 2.14

$$M' \leftrightarrows_{\mathrm{b}}^{\Delta} (\nu \vec{e}, \vec{s}, \vec{r})(\overline{m}.read(f).\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid\, Tape^{i}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)) = M'' \ .$$

Then, again by Lemma 3.4, we get

$$M'' \rightsquigarrow\equiv (\nu \vec{e}, \vec{s}, \vec{r})(read(f).\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid T'') \ ,$$

where $T'' \leftrightarrows_{\mathrm{b}}^{\Delta} Tape^{j}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)$ and $j = i - 1$ if $m = \mathit{left}$ and $j = i + 1$ if $m = \mathit{right}$. So, by Lemmas 2.10, 2.12, 2.13 and 2.14,

$$M'' \leftrightarrows_{\mathrm{b}}^{\Delta} (\nu \vec{e}, \vec{s}, \vec{r})(read(f).\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid\, Tape^{j}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)) = M''' \ .$$

Then, with another application of Lemma 3.4, we get

$$M''' \rightsquigarrow\equiv (\nu \vec{e}, \vec{s}, \vec{r})(\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid T''') \ ,$$

where $T''' \leftrightarrows_{\mathrm{b}}^{\Delta} Tape^{j}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)$. So, by Lemmas 2.10, 2.12, 2.13 and 2.14,

$$M''' \leftrightarrows_{\mathrm{b}}^{\Delta} (\nu \vec{e}, \vec{s}, \vec{r})(\bar{t}.\overline{f}.\mathbf{0} \mid\, !S \mid\, Tape^{j}_{[m,n]}(d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n)) = M'''' \ .$$

And finally,

$$M'''' \rightsquigarrow\equiv M_{t,\delta'_L \check{f} \delta'_R} \ ,$$

where $\delta'_L f \delta'_R = d_m, \ldots, d_{i-1}, e, d_{i+1}, \ldots, d_n$. □

Theorem 3.7. *For every RTM $\mathcal{M}$, we have that $\mathcal{T}(M_{\uparrow,\check{\Box}}) \leftrightarrows_{b}^{\Delta} \mathcal{T}(\mathcal{M})$.*

Proof. Let $\mathcal{M} = (S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow', \uparrow)$, let $i \notin \mathcal{A}_\tau$, and let $\mathcal{M}' = (S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow', \uparrow)$ be the RTM obtained from $\mathcal{M}$ by replacing all τ-transitions by i-transitions. Clearly, $\mathcal{T}(\mathcal{M})$ is isomorphic to $\mathcal{T}(\mathcal{M}')$ up to a renaming of i to τ. Hence, in order to prove that $\mathcal{T}(M_{\uparrow,\check{\Box}}) \leftrightarrows_{\mathrm{b}}^{\Delta} \mathcal{T}(\mathcal{M})$, it suffices to prove that $\mathcal{T}(M'_{\uparrow,\check{\Box}}) \leftrightarrows_{\mathrm{b}}^{\Delta} \mathcal{T}(\mathcal{M}')$, where $M'_{\uparrow,\check{\Box}}$ is the π-term associated with $(\uparrow, \check{\Box})$ given $\mathcal{M}'$.

Using Lemma 3.6 it is straightforward to establish that the relation

$$\mathcal{R}' = \{(M'_{s,\delta_L \check{d} \delta_R}, (s, \delta_L \check{d} \delta_R)) \mid s \in S,\, \delta_L,\, \delta_R \in \mathcal{D}_\Box^*,\, \check{d} \in \check{\mathcal{D}}_\Box\}$$

is a branching bisimulation up to $\leftrightarrows_{\mathrm{b}}$. So, by Lemma 2.4, it follows that $\mathcal{T}(M'_{\uparrow,\check{\Box}}) \leftrightarrows_{\mathrm{b}} \mathcal{T}(\mathcal{M}')$.

It remains to show that divergence is preserved too.

Note that there is no τ-transition in $\mathcal{M}'$, which means $\mathcal{T}(\mathcal{M}')$ has no divergence. Then, by Lemma 3.6, the specification of a certain configuration $M'_{s,\delta_L \check{d} \delta_R}$ can only do a-labelled transitions, where $a \in \mathcal{A} \cup \{i\}$, i.e.

$$M'_{s,\delta_L \check{d} \delta_R} \xrightarrow{a} M' \leftrightarrows_{\mathrm{b}}^{\Delta} M'_{t,\delta'_L \check{f} \delta'_R} \ .$$

Since there is no τ transition from the term $M'_{t,\delta'_L \check{f} \delta'_R}$, it follows that M' has no divergence either. Hence, no π-term reachable from $M'_{s,\delta_L \check{d} \delta_R}$ admits a divergence, and therefore we have,

$$\mathcal{T}(M'_{\uparrow,\check{\Box}}) \leftrightarrows_{\mathrm{b}}^{\Delta} \mathcal{T}(\mathcal{M}') \ .$$

Finally, we switch back to $\mathcal{M}$, by changing all the i labelled transition to τ, and we let $M_{\uparrow,\check{\Box}}$ be the specification of the initial state of $\mathcal{M}$. We can also establish that the relation

$$\mathcal{R} = \{(M_{s,\delta_L \check{d} \delta_R}, (s, \delta_L \check{d} \delta_R)) \mid s \in S,\, \delta_L,\, \delta_R \in \mathcal{D}_\Box^*,\, \check{d} \in \check{\mathcal{D}}_\Box\} \ .$$

is a branching bisimulation up to $\leftrightarrows_{\mathrm{b}}$. Moreover, note that every infinite sequence of the form $\xrightarrow{i} \longrightarrow^* \xrightarrow{i} \longrightarrow^* \cdots$ from $M'_{\uparrow,\check{\Box}}$ corresponds with a infinite sequence of the form $\xrightarrow{i} \xrightarrow{i} \cdots$ from $\mathcal{M}'$, and vice versa. Additionally, there is no divergence from $M'_{\uparrow,\check{\Box}}$. Therefore, every infinite τ-labelled transition sequence from $M_{\uparrow,\check{\Box}}$ corresponds with an infinite τ-labelled transition sequence from $\mathcal{M}$. We conclude that $\mathcal{R} \subseteq \leftrightarrows_{\mathrm{b}}^{\Delta}$. □

Thus we have the following expressiveness result for the π-calculus.

Corollary 3.8. *The π-calculus is behaviourally complete up to divergence-preserving branching bisimilarity.*

4. On the Executability of the π-Calculus

We have proved that every executable behaviour can be specified in the π-calculus modulo divergence-preserving branching bisimilarity. We shall now investigate to what extent behaviour specified in the π-calculus is executable. Recall that we have defined executable behaviour as behaviour of an RTM. So, in order to prove that the behaviour specified by a π-term is executable, we need to show that the transition system associated with this π-term is behaviourally equivalent to the transition system associated with some RTM.

Note, however, that there is a mismatch between the formalisms of RTMs and π-calculus. On the one hand, the notion of RTM as we have defined it in Section 2.1 presupposes *finite*

sets $\mathcal{A}_\tau$ and $\mathcal{D}_\Box$ of actions and data symbols, and also the transition relation of an RTM is *finite*. As a consequence, the transition system associated with an RTM is finitely branching, and, in fact, its branching degree is bounded by a natural number. (Note that this does not mean that RTMs cannot deal with data of unbounded size; it only means that it has to be encoded using finitely many symbols.) The π-calculus, on the other hand, presupposes an infinite set of names by which an infinite set of actions $\mathcal{A}_\pi$ is generated. Furthermore, the transition system associated with a π-term by the structural operational semantics (see Table 1) may contain states with an infinite branching degree, even modulo branching bisimilarity, due to the rules for input prefix and bound output prefix.

From the assumption that the set of actions and the transition relation of an RTM must be finite it immediately follows that there are π-terms that cannot be simulated. The following example illustrates that it is not enough to relax just those two requirements.

Example 4.1. Consider the π-term $P = x(y).\overline{y}\,y.\mathbf{0}$. According to (Inp) (see Table 1), P affords, for every $z \in \mathcal{N}$, a transition $P \xrightarrow{xz} \overline{z}\,z.\mathbf{0}$, and since $\overline{z}\,z.\mathbf{0} \xrightarrow{\overline{z}\,z} \mathbf{0}$ is the only transition of $\overline{z}\,z.\mathbf{0}$, it follows that $\overline{z'}\,z'.\mathbf{0} \not\leftrightarrows_{\mathrm{b}} \overline{z''}\,z''.\mathbf{0}$ if $z' \neq z''$.

Now suppose that $\mathcal{M} = (S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow, \uparrow)$ is an RTM, except that we allow its set of actions to be the infinite set $\mathcal{A}_\pi$ and also we allow its transition relation to be infinite, and assume that $\mathcal{T}(\mathcal{M}) \leftrightarrows_{\mathrm{b}} \mathcal{T}(P)$.

Let $C = (\uparrow, \check{\Box})$ be the initial configuration of $\mathcal{M}$. We have $C \leftrightarrows_{\mathrm{b}} P$, so C affords, for every $z \in \mathcal{N}$, a transition sequence

$$C \longrightarrow^* \xrightarrow{x\,z} \longrightarrow^* C_z \xrightarrow{\overline{z}\,z} C'_z \ ,$$

with $C_z \leftrightarrows_{\mathrm{b}} \overline{z}\,z.\mathbf{0}$ and $C'_z \leftrightarrows_{\mathrm{b}} \mathbf{0}$.

The transition rules of RTMs are of the form $s \xrightarrow{a[d/e]M} t$, where s, $t \in S$, and d, $e \in \mathcal{D}_\Box$; we call the pair (s, d) the trigger of this rule. A configuration $(s', \delta_L \check{d'} \delta_R)$ satisfies the trigger (s, d) if $s = s'$ and $d = d'$. Now observe that a rule $s \xrightarrow{a[d/e]M} s'$ gives rise to an a-transition from every configuration satisfying its trigger (s, d). Since S and $\mathcal{D}_\Box$ are finite, there are finitely many triggers.

So, in the infinite collection of configurations C_z ($z \in \mathcal{N}$), there are at least two configurations, say $C_{z'}$ and $C_{z''}$ with $z' \neq z''$, satisfying the same trigger (s, d); these configurations must have the same outgoing transitions. It follows that $C_{z'} \xrightarrow{\overline{z''}\,z''} C''_{z'}$, which contradicts $C_{z'} \leftrightarrows_{\mathrm{b}} \overline{z'}\,z'.\mathbf{0}$. We conclude that $\mathcal{T}(\mathcal{M}) \not\leftrightarrows_{\mathrm{b}} \mathcal{T}(P)$.

The example illustrates that it is not enough, for the simulation modulo branching bisimilarity of π-terms, to relax the finiteness restriction on the set of actions and the transition relation. We should also allow the set of states S or the set of data sybols $\mathcal{D}_\Box$ to be infinite. Simply lifting the finiteness requirement on either yields a notion of RTM that is arguably too expressive: in both cases, every countable transition system can be simulated up to divergence-preserving branching bisimilarity [Yan18, Section 6.1]. The transition system associated with a π-term is clearly countable if the set of names $\mathcal{N}$ is countable.

Following the work of Bojańczyk, Klin, Lasota, and Toruńczyk [BKLT13], we consider in this section a more modest relaxation of the finiteness requirements on RTMs. We propose *orbit-finite* RTMs and an associated notion of *orbit-finite executability*, and then we prove that every π-term can be simulated up to branching bisimilarity by an orbit-finite RTM.

4.1. **Orbit-Finite Executability.** The notion of orbit-finite executability that we introduce below is based on the notion of *orbit-finite set* proposed in [BKL11]. Below, we briefly recap the definitions; we refer to [Boj19] for an extensive treatment and elaborate explanations.

Let $\mathbb{A}$ be a countably infinite set of *atoms*. We denote by $Sym(\mathbb{A})$ the group of all permutations on $\mathbb{A}$, and denote by id the neutral element of $Sym(\mathbb{A})$. An *action* of $Sym(\mathbb{A})$ on a set X is a binary operation $\cdot : Sym(\mathbb{A}) \times X \to X$ such that $id \cdot x = x$ for all $x \in X$ and $\pi\rho \cdot x = \pi \cdot (\rho \cdot x)$ for all $\pi, \rho \in Sym(\mathbb{A})$ and $x \in X$. The symbol $\cdot$ will often be omitted. For the sets with atoms appearing in the remainder of this paper, $\cdot$ will always be specified in the expected manner: the elements are denoted by expressions involving atoms and, for every element x it is assumed that $\pi \cdot x$ is obtained by replacing every atom a occurring in the expression denoting x by $\pi(a)$.

A set $A \subseteq \mathbb{A}$ *supports* an element $x \in X$ if for all $\pi \in Sym(\mathbb{A})$ such that $\pi(a) = a$ for all $a \in A$ it holds that $\pi x = x$. If for all $x \in X$ there exists a finite set $A_x \subseteq \mathbb{A}$ that supports x, then X is called a *nominal set*.

An action of $Sym(\mathbb{A})$ on X induces an equivalence relation $\sim$ on X, defined by $x \sim y$ if, and only if, there exists $\pi \in Sym(\mathbb{A})$ such that $\pi \cdot x = y$. The equivalence relation $\sim$ partitions X into equivalence classes called *orbits*, and X is *orbit-finite* if $\sim$ partitions X into finitely many orbits.

Definition 4.2. A reactive Turing machine $(S, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow, \uparrow)$ is *orbit-finite* if S, $\mathcal{D}_\Box$, $\mathcal{A}_\tau$ and $\longrightarrow$ are *orbit-finite* nominal sets. A transition system is *orbit-finitely executable* if it is the transition system associated with some orbit-finite reactive Turing machine.

4.2. **The π-Calculus is Orbit-Finitely Executable.** To prove that the π-calculus is orbit-finitely executable up to branching bisimilarity, we should establish that for every π-term P_0 there exists an orbit-finite RTM $\mathtt{Sim}(P_0)$ such that $\mathcal{T}(P_0) \leftrightarrows_\mathrm{b} \mathcal{T}(\mathtt{Sim}(P_0))$. The RTM $\mathtt{Sim}(P_0)$ will be orbit-finite with respect to a set of atoms $\mathbb{A}$ that consists of the set $\mathcal{N}$ of names of the π-calculus, i.e., we assume

$$\mathbb{A} = \mathcal{N} \ .$$

We shall define $\mathtt{Sim}(P_0)$ as the union of several smaller RTMs that take care of specific aspects of the simulation; we shall refer to these smaller RTMs as *fragments* of $\mathtt{Sim}(P_0)$. Before we proceed to describe these fragments in detail, we first give a broad overview of the fragments we need. In the overview we assume that π-terms, transitions and derivations of transitions can be stored on the tape of the RTM; this will also be discussed in more detail below.

Initialise: By design, an RTM starts with an empty tape, while for the simulation it is convenient to assume that (an encoding of) the π-term representing the current state is written on tape. The purpose of the *initialise* fragment $\mathtt{Init}(P_0)$ is to write an encoding of P_0, the π-term to be simulated, on tape.

Generate transition: It is assumed that the *generate* fragment `Gen` starts executing with the encoding of an arbitrary π-term P stored on tape. Its purpose is then to generate (the encoding of) a transition that is derivable according to the operational semantics of the π-calculus and has P as source. This is achieved by non-deterministically writing an arbitrary sequence of symbols on tape, and then verifying whether the sequence encodes a derivation of a transition and whether this transition has P as source. If so, then the *generate transition* fragment erases everything from the tape except for the

encoding of the derived transition and proceeds with the *execute transition* fragment described next; if not, then it erases the generated sequence of symbols and restarts the transition generation fragment with the same π-term on tape.

Execute transition: It is assumed that the *execute* fragment `Exec` starts executing with the encoding of a transition $P \xrightarrow{\alpha} P'$ written on the tape. Its purpose is to execute the associated action and thereafer leave only the target P' of the transition on the tape, returning to the *generate transition* fragment.

Note that only the $\mathtt{Init}(P_0)$ fragment will be specific for every π-term P_0. The fragments `Gen` and `Exec` are generic; their behaviour depends on their initial tape contents. We proceed to first define the sets $\mathcal{D}$ and $\mathcal{A}$ associated with $\mathtt{Sim}(P_0)$, and then discuss the definitions of the three fragments in more detail.

Orbit-finite sets of data and action symbols. We let the set of data symbols $\mathcal{D}$ of $\mathtt{Sim}(P_0)$ consist of the set of π-calculus names $\mathcal{N}$, extended with two *finite* sets of symbols $\mathcal{D}_\pi$ and $\mathcal{D}_{aux}$. To store a π-term on the tape of an RTM, we use the symbols in $\mathcal{N}$ and

$$\mathcal{D}_\pi = \{{}^{-}, (,), \tau, \mathbf{0}, ., +, |, \nu, !\} \ ,$$

writing $\overline{x}$ on the tape as ${}^{-}x$. The symbols in $\mathcal{D}_{aux}$ will serve as auxiliary symbols in computations; we assume that$\mathcal{D}_{aux}$ at least includes the symbols $\#$, $\langle$, $\rangle$, $-$, $>$, $\therefore$, [, and]. The sets $\mathcal{D}_\pi$ and $\mathcal{D}_{aux}$ are assumed to be mutually disjoint and also disjoint from $\mathcal{N}$, and they are assumed to have been constructed using standard set-theoretic methods, not involving atoms. We have

$$\mathcal{D} = \mathcal{D}_\pi \cup \mathcal{N} \cup \mathcal{D}_{aux} \ .$$

The orbits of $\mathcal{D}$ are $\mathcal{N}$ and all the singleton subsets of $\mathcal{D}_\pi$ and $\mathcal{D}_{aux}$, so $\mathcal{D}$ has $|\mathcal{D}_\pi|+1+|\mathcal{D}_{aux}|$ orbits. Hence $\mathcal{D}$ and clearly also $\mathcal{D}_\Box$ are orbit-finite. Trivially, every π-term P can be specified with a sequence of symbols from $\mathcal{D}$; for clarity of presentation, we shall distinguish a π-term P from its specification as a sequence of symbols from $\mathcal{D}$ writing $\lceil P \rceil$ for the latter.

The set of action symbols $\mathcal{A}_\tau$ of $\mathtt{Sim}(P_0)$ will be the set of all π-calculus actions given the set of names $\mathcal{N}$ (see Equation 2.1), i.e.,

$$\mathcal{A}_\tau = \{x\,y, \overline{x}\,y, \overline{x}\,(z) \mid x, y, z \in \mathcal{N}\} \cup \{\tau\} \ .$$

Note that $x_1\,y_1 \sim x_2\,y_2$ if, and only if, either $x_1 = y_1$ and $x_2 = y_2$, or $x_1 \neq y_1$ and $x_2 \neq y_2$. Hence, the set $\{x\,y \mid x, y \in \mathcal{N}\}$ has two orbits, and, by similar reasoning, so do $\{\overline{x}\,y \mid x, y \in \mathcal{N}\}$ and $\{\overline{x}\,(z) \mid x, z \in \mathcal{N}\}$. It follows that $\mathcal{A}_\tau$ has seven orbits in total, and hence is orbit-finite. Clearly, every π-calculus action α can be specified on tape using the symbols in $\mathcal{D}$; we denote this sequence by $\lceil \alpha \rceil$.

Initialise. The initialise fragment $\mathtt{Init}(P_0) = (S_{\mathtt{Init}(P_0)}, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow_{\mathtt{Init}(P_0)}, \uparrow_{\mathtt{Init}(P_0)})$ should simply write the encoding $\lceil P_0 \rceil$ of P_0 on the tape. This is straightforward: Let $n = |\lceil P_0 \rceil|$, and assume $\lceil P_0 \rceil = d_0 \cdots d_n$. We define $S_{\mathtt{Init}(P_0)} = \{\uparrow_{\mathtt{Init}(P_0)}, s_0, \ldots, s_n, \uparrow_{\mathtt{Gen}}\}$. The execution of $\mathtt{Init}(P_0)$ starts with a transition

$$\uparrow_{\mathtt{Init}(P_0)} \xrightarrow{\tau[\Box/d_0]R}_{\mathtt{Init}(P_0)} s_0 \ ,$$

has for every $0 \leq i < n$ a transition of the form

$$s_i \xrightarrow{\tau[\Box/d_i]R}_{\mathtt{Init}(P_0)} s_{i+1} \ ,$$

and ends with a transition

$$s_n \xrightarrow{\tau[\Box/\#]R}_{\mathtt{Init}(P_0)} \uparrow_{\mathtt{Gen}} .$$

The latter transition marks the end of the sequence $\lceil P_0 \rceil$ on the tape with the symbol #, and its target is the initial state $\uparrow_{\mathtt{Gen}}$ of the generate fragment `Gen`. Since both $S_{\mathtt{Init}(P_0)}$ and $\longrightarrow_{\mathtt{Init}(P_0)}$ are finite and $\mathcal{D}_\Box$ and $\mathcal{A}_\tau$ are orbit-finite, it follows that $\mathtt{Init}(P_0)$ is an orbit-finite RTM.

Generate transition. The generate transition fragment $\mathtt{Gen} = (S_{\mathtt{Gen}}, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow_{\mathtt{Gen}}, \uparrow_{\mathtt{Gen}})$ starts with the sequence of symbols $\lceil P \rceil \#$ written on tape and the tape head positioned on the first blank immediately to the right of the sequence. The purpose of the fragment is to compute a transition $P \xrightarrow{\alpha} P'$, specified on tape as a sequence $\langle \lceil P \rceil - \lceil \alpha \rceil > \lceil P' \rceil \rangle$. To this end, `Gen` will non-deterministically generate a sequence of symbols in $\mathcal{D} \backslash \{\#\}$ followed by #, and subsequently checks whether the sequence between the two # symbols on tape encodes a derivation of a transition of the form $P \xrightarrow{\alpha} P'$.

Generating a sequence of symbols is straightforward: it requires only two states $\uparrow_{\mathtt{Gen}}$ and $\uparrow_{\mathtt{Check}}$ and transitions

$$\uparrow_{\mathtt{Gen}} \xrightarrow{\tau[\Box/d]R}_{\mathtt{Gen}} \uparrow_{\mathtt{Gen}} \qquad (d \in \mathcal{D} \backslash \{\#\})$$

and

$$\uparrow_{\mathtt{Gen}} \xrightarrow{\tau[\Box/\#]R}_{\mathtt{Gen}} \uparrow_{\mathtt{Check}} .$$

Note that, since $\mathcal{D}$ is orbit-finite and hence also $\mathcal{D} \backslash \{\#\}$ is orbit-finite, there are orbit-finitely many transitions of the first type. So, this part of `Gen` is clearly orbit-finite.

The verification procedure, which starts in $\uparrow_{\mathtt{Check}}$ is tedious, but straightforward given the operational semantics. We shall refrain from specifying it in detail and only comment on the subprocedures that need to be carried out by this fragment, and argue that they are orbit-finite. First, however, we need to explain how a derivation can be (uniquely) encoded as a sequence of symbols from $\mathcal{D} - \{\#\}$. We adopt the convention that (sub)derivations are included in square brackets and the conclusion of a (sub)deriviation is preceded with the symbol $\therefore$. For instance, the derivation

$$\dfrac{\dfrac{\dfrac{}{\overline{x}\,z.\mathbf{0} \xrightarrow{\overline{x}\,z} \mathbf{0}}\,(\textsc{Out}) \qquad \dfrac{}{x(y).\mathbf{0} \xrightarrow{x\,z} \mathbf{0}}\,(\textsc{Inp})}{\overline{x}\,z.\mathbf{0} \mid x(y).\mathbf{0} \xrightarrow{\tau} \mathbf{0} \mid \mathbf{0}}\,(\textsc{Comm-l})}{(\nu z)(\overline{x}\,z.\mathbf{0} \mid x(y).\mathbf{0}) \xrightarrow{\tau} (\nu z)(\mathbf{0} \mid \mathbf{0})}\,(\textsc{Res})$$

is specified on tape as

$$[[[\therefore \langle {}^{-}xz.\mathbf{0} - {}^{-}xz > \mathbf{0}\rangle][\therefore \langle x(y).\mathbf{0} - xz > \mathbf{0}\rangle].\therefore \langle {}^{-}xz.\mathbf{0} | x(y).\mathbf{0} - \tau > \mathbf{0}|\mathbf{0}\rangle]$$
$$\therefore \langle (\nu z)({}^{-}xz.\mathbf{0}|x(y).\mathbf{0}) - \tau > (\nu z)(\mathbf{0}|\mathbf{0})\rangle] .$$

To verify whether a sequence of symbols from $\mathcal{D} \backslash \{\#\}$ indeed encodes a derivation, `Gen` implements a procedure consisting of the following steps:

(1) Check whether the occurrences of the symbols [, $\therefore$ and] between the two occurrences of the symbol # on the tape represent a tree structure. If so, then continue to the next step; otherwise, remove all symbols to the right of the left-most occurrence of the symbol # and return to $\uparrow_{\mathtt{Gen}}$.

(2) Check whether all sequences of symbols between subsequent occurrences of $\therefore$ and] encode transitions. If so, then continue to the next step; otherwise, remove all symbols to the right of the left-most occurrence of the symbol # and return to $\uparrow_{\texttt{Gen}}$.
(3) Check whether all individual inferences of the represented derivation are instances of a rule of the operational semantics (see Table 1). If so, then continue to the next step; otherwise, remove all symbols to the right of the left-most occurrence of the symbol # and return to $\uparrow_{\texttt{Gen}}$.
(4) Check whether the sequence of symbols that constitutes the source of the right-most transition, i.e., the conclusion of the derivation specified on tape, matches the sequence $\lceil P \rceil$ preceding the left-most occurrence of # on the tape. If so, then erase everything except the transition $\langle \lceil P \rceil - \lceil \alpha \rceil > \lceil P' \rceil \rangle$ from the tape and transition to $\uparrow_{\texttt{Exec}}$. Otherwise, remove all symbols to the right of the left-most occurrence of the symbol # and return to $\uparrow_{\texttt{Gen}}$.

The computations described in first two steps of the procedure can be done without any special consideration of individual names. The parts of `Gen` implementing these steps have finitely many states and orbit-finitely many transitions. Step 4 can be implemented as a simple comparison procedure: the sequence of symbols to the left of the left-most # must be compared to the sequence of symbols representing the source of the right-most transition. Such a comparison can be done by comparing one symbol at a time, but it will involve remembering that symbol in a state. Hence this part of `Gen` will require an infinite set of states and an infinite set of transitions, which can, however, both be partitioned into finitely many $\mathcal{D}_\pi \cup \mathcal{N}$-indexed orbits.

The implementation of step 3 is computationally more involved. We discuss, for each of the operational rules in Table 1 (see p. 8), what needs to be done:

(Tau): It needs to be checked that there are no premises (i.e., immediately left of the $\therefore$ symbol there is the [symbol). Furthermore, it needs to be checked that the source of the transition is a τ prefix (i.e., starts with the symbols τ and .), that the label of the transition is τ (i.e., the symbols $-$ and $>$ there is only the symbol τ), and that the operand of the prefix is equal to the target of the transition (i.e., the sequence of symbols between the symbol . and the symbol $-$ is identical to the sequence of symbols between $>$ and $\rangle$).
(Out): It needs to be checked that there are no premises. Furthermore, it needs to be checked that the source of the transition is of the form $\overline{x}\,y.P$, that the label of the transition is $\overline{x}\,y$ and that the target of the transition is P.
(Inp): It needs to be checked that there are no premises. Futhermore, it needs to be checked that the source of the transition is of the form $x(y).P$, that the label of the transition is $x\,z$ for some arbitrary name $z \in \mathcal{N}$, and that the target is obtained from P by substituting all free occurrences of y in P by z. The latter operation can, e.g., be carried out by remembering the pair (y, z) in states and carrying out a symbol-wise comparison in which the symbol y must be matched by z rather than y, except while in the scope of a y-binding construct $x(y)$ or (νy). The part of `Gen` that takes care of the comparison has finitely many (y, z)-indexed orbits.
(Sum-l): It needs to be checked that there is one premise, that the source of the conclusion is of the form $P + Q$, that P is the source of the premise, that the labels of the premise and the conclusion are identical, and so are the targets of the source and the premise.
(Sum-r): Analogous to (Sum-l).

(Alpha): It needs to be checked that there is one premise, that the target of the premise is identical to the target of the conclusion, and that the source of the premise and the source of the conclusion are α-convertible. The latter could, e.g., be implemented as a variation on the comparison procedure with a renaming operation built-in (cf. also [Pit16, Section 4]). Whenever, during the comparison of $\lceil P \rceil$ and $\lceil Q \rceil$, a binder, say $x(y_1)$ in $\lceil P \rceil$ and $x(y_2)$ in $\lceil Q \rceil$ or (νy_1) in $\lceil P \rceil$ and (νy_2) in $\lceil Q \rceil$, is encountered, then
(1) a name z is determined that is fresh for both the remainders of $\lceil P \rceil$ and $\lceil Q \rceil$,
(2) all occurrences of y_1 in the remainder of $\lceil P \rceil$ and all occurences of y_2 in the remainder of $\lceil Q \rceil$ are replaced by z, and
(3) the comparison continues.
Note that generating a name that is fresh with respect to some sequence of symbols on tape can be achieved by non-deterministically writing an arbitrary name on tape and subsequently checking whether the name already occurs in the sequence or not. If it does occur, then the name is not fresh for the sequence and the procedure must be repeated. If it does not occur, then the name is fresh for the sequence.

(Par-l): It needs to be checked that there is one premise. Furthermore, it needs to be checked that the source of the conclusion is of the form $P \mid Q$, that P is the source of the premise, that the labels of the premise and the conclusion are identical, and that the target of the conclusion is the parallel composition of the target of the premise and Q. Finally, it should be checked that the side condition $\mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset$ is satisfied. To this end, a sequence of all free names occurring in Q should be compiled and then it should be checked whether the bound name of α does not occur in that sequence.

(Par-r): Analogous to (Par-l).

(Comm-l): It needs to be checked that there are two premises. Futhermore, it needs to be checked that the source of the conclusion is of the form $P \mid Q$, that P is the source of the first premise, that Q is the source of the second premise, and that the target of the conclusion is the parallel composition of the targets of the premises. Finally, it needs to be checked that the label of the first premise is $\overline{x}\, y$ and the label of the second premise is $x\, y$ for some names $x, y \in \mathcal{N}$, and that the label of the conclusion is τ.

(Comm-r): Analogous to (Comm-l).

(Close-l): It needs to be checked that there are two premises. Furthermore, it needs to be checked that the source of the conclusion is of the form $P \mid Q$, that P is the source of the first premise, that Q is the source of the second premise, and that the target of the conclusion is of the form $(\nu z)(P' \mid Q')$, where P' and Q' are the targets of the premises. It also needs to be checked that the label of the first premise is $\overline{x}\,(z)$, for some $x \in \mathcal{N}$, that the label of the second premise is $x\, z$, and that the label of the conclusion is τ. Finaly, it should be checked that $z \notin \mathsf{fn}(Q)$, by first compiling the sequence of names with a free occurrence in Q and then checking whether z appears in the sequence.

(Close-r): Analogous to (Close-l).

(Res): It needs to be checked that there is one premise. Furthermore, it needs to be checked that the source of the conclusion is of the form $(\nu z)P$, that P is the source of the premise, that the target is $(\nu z)P'$ where P' is the target of the conclusion, that the labels of the premise and the conclusion are identical, and that the name z does not appear in α.

(Open): It needs to be checked that there is one premise. Furthermore, it needs to be checked that the source of the conclusion is of the form $(\nu z)P$, that P is the source of the premise, and that the targets of the premise and the conclusion are identical.

Finally, it needs to be checked that the label of the premise is $\overline{x}\,z$ for some names $x \neq z$, and that the label of the conclusion then is $\overline{x}\,(z)$.

(Rep-comm): It needs to be checked that there are two premises. Furthermore, it needs to be checked that the source of the conclusion is of the form $!P$, that the source of both premises is P, and that the target of the conclusion is the parallel composition of, on the one hand, the parallel composition of the targets of the premises and, on the other hand, $!P$. Finally, it needs to be checked that the label of the first premise is $\overline{x}\,y$ for some names $x, y \in \mathcal{N}$, that the label of the second premise is then $x\,y$, and that the label of the conclusion is τ.

(Rep-close): It needs to be checked that there are two premises. Furthermore, it needs to be checked that the source of the conclusion is of the form $!P$, that the source of both premises is P, and that the target of the conclusion is a restriction (νz) applied to a parallel composition of, on the one hand, the parallel composition of the targets of the premises and, on the other hand, $!P$. Finally, it needs to be checked that the label of the first premise is $\overline{x}\,(z)$ for some name $x \in \mathcal{N}$, that the label of the second premise is $x\,z$, and that the label of the conclusion is τ.

In step 3, `Gen` applies, for every individual inference, the procedures associated above with the operational rules of the π-calculus one after the other until either one of them succeeds or it has been determined that none of them succeeds. In the first case, all symbols to the right of the left-most occurrence of the symbol # are removed and `Gen` returns to $\uparrow_{\texttt{Gen}}$. In the second case, `Gen` proceeds with the next individual inference.

All transitions associated with `Gen` are deemed internal, i.e., are labelled with τ. Moreover, from every state of `Gen` at least one of $\uparrow_{\texttt{Gen}}$ and $\uparrow_{\texttt{Exec}}$ is reachable.

Execute transition. The execute transition fragment $\texttt{Exec} = (S_{\texttt{Exec}}, \mathcal{D}_{\Box}, \mathcal{A}_{\tau}, \longrightarrow_{\texttt{Exec}}, \uparrow_{\texttt{Exec}})$ starts with the encoding of a transition $\langle \lceil P \rceil - \lceil \alpha \rceil > \lceil P' \rceil \rangle$ written on the tape; we assume that the tape head is positioned on the first symbol of $\lceil \alpha \rceil$. The initial state $\uparrow_{\texttt{Exec}}$ admits the following transition sequences:

$$\uparrow_{\texttt{Exec}} \xrightarrow{\tau[\tau/\tau]R}_{\texttt{Exec}} s_{\tau} \xrightarrow{\tau[>/>]R}_{\texttt{Exec}} \mathit{cnt} \ ,$$

$$\uparrow_{\texttt{Exec}} \xrightarrow{\tau[x/x]R}_{\texttt{Exec}} s^{x}_{in} \xrightarrow{\tau[y/y]R}_{\texttt{Exec}} s^{x,y}_{in} \xrightarrow{x\,y[>/>]R}_{\texttt{Exec}} \mathit{cnt} \qquad (x, y \in \mathcal{N}) \ , \text{ and}$$

$$\uparrow_{\texttt{Exec}} \xrightarrow{\tau[\lceil/\rceil]R}_{\texttt{Exec}} s_{\text{out}} \xrightarrow{\tau[x/x]R}_{\texttt{Exec}} s^{x}_{\text{out}} \qquad (x \in \mathcal{N}) \ .$$

From the states s^{x}_{out} we need to distinguish two possible continuations, depending on whether α represents a regular or a bound output:

$$s^{x}_{out} \xrightarrow{\tau[y/y]R}_{\texttt{Exec}} s^{x,y}_{out} \xrightarrow{\overline{x}\,y[>/>]R}_{\texttt{Exec}} \mathit{cnt} \qquad (x, y \in \mathcal{N}) \ ,$$

and

$$s^{x}_{out} \xrightarrow{\tau[(/(]R}_{\texttt{Exec}} t^{x} \xrightarrow{\tau[z/z]R}_{\texttt{Exec}} t^{x,z} \xrightarrow{\tau[)/)R]}_{\texttt{Exec}} s^{x,z}_{bout} \xrightarrow{\overline{x}\,(z)[>/>]R}_{\texttt{Exec}} \mathit{cnt} \qquad (x, z \in \mathcal{N}) \ .$$

In the state *cnt* an internal (i.e., τ-labelled) deterministic *continue* procedure is started which ensures that the contents of the tape is $\lceil P' \rceil \#$. The last transition of this procedure leads to $\uparrow_{\texttt{Gen}}$ with the tape head positioned on the first blank immediately to the right of the sequence.

Furthermore, for every $s \in \{s_\tau\} \cup \{s_{in}^{x,y}, s_{out}^{x,y} \mid x, y \in \mathcal{N}\} \cup \{s_{bout}^{x,z} \mid x, z \in \mathcal{N}\}$, $\mathtt{Exec}$ includes a transition

$$s \xrightarrow{\tau[>/>]R}_{\mathtt{Exec}} abrt \ .$$

In the state *abrt* an internal (i.e., τ-labelled) deterministic *abort* procedure is started which serves to restore the contents of the tape to $\lceil P \rceil \#$ leading to the state $\uparrow_{\mathtt{Gen}}$ with the tape head positioned on the first blank immediately to the right of the sequence. It is necessary to have this abort procedure to ensure that the (non-deterministic) choice for this particular transition is made upon its actual execution and not before. At the same time, note that the abort procedure introduces divergence into the simulation: $\uparrow_{\mathtt{Gen}}$ affords a non-empty sequence of τ-transitions to *abrt*, which, in turn, affords a non-empty sequence of τ-transitions back to $\uparrow_{\mathtt{Gen}}$.

From the descriptions above, which are parameterised in at most two names, it is clear that the part of $\mathtt{Exec}$ transitions leading up to the action execution is orbit-finite. Furthermore, the continue and abort procedures do not give any special treatment to individual elements of $\mathcal{N}$ (names are either skipped or erased), so these procedures require finitely many states and orbit-finitely many transitions.

We now define $\mathtt{Sim}(P_0)$ as the union of the three fragments $\mathtt{Init}(P_0)$, $\mathtt{Gen}$ and $\mathtt{Exec}$, i.e.,

$$\mathtt{Sim}(P_0) = (S_{\mathtt{Init}(P_0)} \cup S_{\mathtt{Gen}} \cup S_{\mathtt{Exec}}, \mathcal{D}_\Box, \mathcal{A}_\tau, \longrightarrow_{\mathtt{Init}(P_0)} \cup \longrightarrow_{\mathtt{Gen}} \cup \longrightarrow_{\mathtt{Exec}}, \uparrow_{\mathtt{Init}(P_0)}) \ .$$

(We assume that $\uparrow_{\mathtt{Gen}}$ is shared between all three fragments and $\uparrow_{Exec}$ is shared between the fragmens $\mathtt{Gen}$ and $\mathtt{Exec}$, but otherwise the sets of states of the three fragments are disjoint.) We now have the following theorem.

Theorem 4.3. *For every π-term P there exists an orbit-finite reactive Turing machine $\mathtt{Sim}(P_0)$ such that $\mathcal{T}(P_0) \leftrightarrows_b \mathcal{T}(\mathtt{Sim}(P_0))$.*

Proof. We have already argued that for every P_0 there exists an orbit-finite RTM $\mathtt{Sim}(P_0)$. It remains to establish a branching bisimulation $\mathcal{R}$ from $\mathcal{T}(P_0)$ to $\mathcal{T}(\mathtt{Sim}(P_0))$ such that $P_0 \ \mathcal{R} \ (\uparrow_{\mathtt{Init}(P_0)}, \check{\Box})$.

We first introduce the following notations

(1) $\mathtt{Init}(P_0, \check{\Box})$ denotes the set of all configurations reachable from $(\uparrow_{\mathtt{Init}(P_0)}, \check{\Box})$;
(2) for every P reachable from P_0 in $\mathcal{T}(P_0)$, $\mathtt{Gen}(P)$ denotes the set of all configurations reachable from $(\uparrow_{\mathtt{Gen}}, \lceil P \rceil \# \check{\Box})$ in $\mathcal{T}(\mathtt{Sim}(P_0))$.
(3) for every P reachable from P_0 in $\mathcal{T}(P_0)$, $\mathtt{Exec}(P)$ denotes the set of all configurations that are either on a path from $(\uparrow_{\mathtt{Exec}}, \langle \lceil P \rceil - \lceil \check{\alpha} \rceil > \lceil P' \rceil)$ to a configuration $(s, \langle \lceil P \rceil - \lceil \alpha \rceil \check{>} \rangle)$ (with $s \in \{s_\tau\} \cup \{s_{in}^{x,y}, s_{out}^{x,y} \mid x, y \in \mathcal{N}\} \cup \{s_{bout}^{x,z} \mid x, z \in \mathcal{N}\}$) or on a path from $(abrt, \langle \lceil P \rceil - \lceil \check{\alpha} \rceil > \lceil \check{P'} \rceil \rangle)$ to $(\uparrow_{\mathtt{Gen}}, \lceil P \rceil \# \check{\Box})$; and
(4) for every P' reachable from P_0 in $\mathcal{T}(P_0)$, $\mathtt{Exec}'(P')$ denotes the set of all configurations that are on a path from $(cnt, \langle \lceil P \rceil - \lceil \alpha \rceil > \lceil \check{P'} \rceil \rangle$ to $(\uparrow_{\mathtt{Gen}}, \lceil P' \rceil \# \check{\Box})$.

We can now define the relation $\mathcal{R}$ by

$$\begin{aligned}\mathcal{R} = \{(P_0, c) \mid c \in \mathtt{Init}(P_0, \check{\Box})\} \cup \\ \{(P, c) \mid P \text{ reachable from } P_0 \text{ in } \mathcal{T}(P_0),\ c \in \mathtt{Gen}(P) \cup \mathtt{Exec}(P)\} \cup \\ \{(P', c) \mid P' \text{ reachable from } P_0 \text{ in } \mathcal{T}(P_0),\ c \in \mathtt{Exec}'(P')\} \ .\end{aligned}$$

From $(\uparrow_{\mathtt{Init}(P_0)}, \check{\Box})$ there is a deterministic internal computation leading to $(\uparrow_{\mathtt{Gen}}, \lceil P \rceil \# \check{\Box})$ and P_0 is related according to $\mathcal{R}$ to all intermediate states of this internal computation. The fragments `Gen` and `Exec` have been designed such that all configurations in $\mathtt{Gen}(P)$ and $\mathtt{Exec}(P)$ are reachable from the configuration $(\uparrow_{\mathtt{Gen}}, \lceil P \rceil \# \check{\Box})$ by τ-transitions and, moreover, from all those configurations the configuration $(\uparrow_{\mathtt{Gen}}, \lceil P \rceil \# \check{\Box})$ is reachable by τ-transitions. It follows that $\mathtt{Gen}(P) \cup \mathtt{Exec}(P)$ is a so-called τ-cluster: every configuration is reachable from every other configuration and the only exits from the cluster are the transitions of the form

$$(s, \langle \lceil P \rceil - \lceil \alpha \rceil \check{>} \lceil P' \rceil \rangle) \xrightarrow{\alpha} (c, \langle \lceil P \rceil - \lceil \alpha \rceil > \lceil \check{P}' \rceil \rangle) \quad (s \in \{s_\tau\} \cup \{s_{in}^{x,y}, s_{out}^{x,y} \mid x, y \in \mathcal{N}\} \cup \{s_{bout}^{x,z} \mid x, z \in \mathcal{N}\}) \ .$$

Note that these transitions simulate and are simulated by transitions $P \xrightarrow{\alpha} P'$. Finally, we have that from $(c, \langle \lceil P \rceil - \lceil \alpha \rceil > \lceil \check{P}' \rceil \rangle)$ there is a deterministic internal computation leading to $(\uparrow_{\mathtt{Gen}}, \lceil P' \rceil \# \check{\Box})$, which concludes the argument that $\mathcal{R}$ is a branching bisimulation. □

Corollary 4.4. *The π-calculus is orbit-finitely executable modulo branching bisimilarity.*

5. Conclusions

Milner already established in [Mil92] that the π-calculus is *computationally complete*, by exhibiting an encoding of the λ-calculus in the π-calculus by which every reduction in the λ-calculus is simulated by a sequence of reductions in the π-calculus. We have established that the π-calculus is *behaviourally complete* up to divergence-preserving branching bisimilarity, which is the finest reasonable notion of behavioural equivalence [Gla93]. This implies that the π-calculus is also behaviourally complete up to the weaker notions of behavioural equivalence usually used in the context of the π-calculus [SW01]. Interestingly, the proof does not rely on recursion and the finite specification of a queue, as does the proof in [BLT13] that the process calculus TCP_τ is behaviourally complete. Instead, the specification of finite RTMs in the π-calculus uses replication and link mobility to directly specify a tape process. An alternative specification of finite RTMs in a process calculus without recursion is presented in [BLY17], which, instead of replication, uses iteration and nesting operators [BBP94, BP01], and also relies on a sequencing operator [BLB19].

Our specification of the behaviour of an RTM seems to make essential use of all the constructions of the π-calculus. Interesting future work would be to consider the various subcalculi of the π-calculus and determine to what extent these are behaviourally complete. It could then also be worthwhile to consider deterministic reactive Turing machines. Another interesting approach to simulate reactive Turing machines in the π-calculus could proceed via a universal process of the latter [Fu17].

We have also established that the π-calculus is orbit-finitely executable up to branching bisimilarity, by associating with every π-calculus process an orbit-finite RTM that simulates it. The simulation is non-deterministic and introduces divergence. We leave it as an open problem whether there exists a simulation that does not introduce divergence and proves that the π-calculus is executable up to divergence-preserving branching bisimilarity. It is established in [BLT13] that every boundedly branching computable transition system is finitely executable up to divergence-preserving branching bisimilarity and that every effective transition system is finitely executable up to the divergence-insensitive variant of branching

bisimilarity. Similar general characterisations of the notion of orbit-finite executability may be needed for solving the aforementioned open problem.

The generic specification of the behaviour of a Turing machine tape, presented in Section 3.1, does not rely on a finiteness assumption regarding the set of tape symbols; in fact, any π-calculus name can be stored on tape. The specification of the control process of an RTM in Section 3.2, however, does essentially rely on the RTM having finitely many states, finitely many tape symbols and finitely many action symbols, for it has summations indexed by the sets of states, tape symbols and transitions of the RTM to be simulated. Thus, orbit-finite RTMs can be simulated by a variant of the π-calculus that allows summations indexed by orbit-finite sets. We conjecture that, on the one hand, infinite summations are essential for a simulation up to divergence-preserving branching bisimilarity, whereas, on the other hand, up to the divergence-insensitive variant of branching bisimilarity they are not.

Acknowledgments

The first author thanks Jos Baeten for being an inspiration, for instigating the research on the integration of automata theory and concurrency theory, and for the pleasant collaboration on this and other topics over the years. Furthermore, the authors thank the anonymous reviewers for their suggestions, which led to improvements of the presentation.

References

[Bas96] T. Basten. Branching bisimilarity is an equivalence indeed! *Information Processing Letters*, 58(3):141–147, 1996.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the Consistency of Koomen's Fair Abstraction Rule. *Theoretical Computer Science*, 51:129–176, 1987.

[BBP94] J. A. Bergstra, I. Bethke, and A. Ponse. Process algebra with iteration and nesting. *The Computer Journal*, 37(4):243–258, 1994.

[BBR10] J. C. M. Baeten, T. Basten, and M. A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*, volume 50. Cambridge university press, 2010.

[BCG07] J.C.M Baeten, F. Corradini, and C. Grabmayer. A Characterization of Regular Expressions under Bisimulation. *J. ACM*, 54(2):6, 2007.

[BCLT09] J. C. M. Baeten, P. J. L. Cuijpers, B. Luttik, and P. J. A. van Tilburg. A Process-theoretic Look at Automata. In F. Arbab and M. Sirjani, editors, *Fundamentals of Software Engineering, Third IPM International Conference, FSEN 2009, Kish Island, Iran, April 15-17, 2009, Revised Selected Papers*, volume 5961 of *Lecture Notes in Computer Science*, pages 1–33. Springer, 2009.

[BCT08] J. C. M. Baeten, P. J. L. Cuijpers, and P. J. A. van Tilburg. A Context-Free Process as a Pushdown Automaton. In F. van Breugel and M. Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 98–113. Springer, 2008.

[BKL11] M. Bojańczyk, B. Klin, and S. Lasota. Automata with group actions. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 355–364. IEEE Computer Society, 2011.

[BKLT13] M. Bojańczyk, B. Klin, S. Lasota, and S. Toruńczyk. Turing Machines with Atoms. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 183–192. IEEE Computer Society, 2013.

[BLB19] A. Belder, B. Luttik, and J. C. M. Baeten. Sequencing and intermediate acceptance: Axiomatisation and decidability of bisimilarity. In M. Roggenbach and A. Sokolova, editors, *Proceedings of CALCO 2019*, volume 139 of *LIPIcs*, pages 11:1–11:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BLMT16] J. C. M. Baeten, B. Luttik, T. Muller, and P. J. A. van Tilburg. Expressiveness modulo Bisimilarity of Regular Expressions with Parallel Composition. *Mathematical Structures in Computer Science*, 26:933–968, 2016.

[BLT13] J.C.M. Baeten, B. Luttik, and P.J.A. van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013.

[BLY17] J. C. M. Baeten, B. Luttik, and F. Yang. Sequential composition in the presence of intermediate termination (extended abstract). In K. Peters and S. Tini, editors, *Proceedings Combined 24th International Workshop on Expressiveness in Concurrency and 14th Workshop on Structural Operational Semantics, EXPRESS/SOS 2017, Berlin, Germany, 4th September 2017*, volume 255 of *EPTCS*, pages 1–17, 2017.

[Boj19] M. Bojańczyk. Slightly infinite sets, September 2019. The latest version can be downloaded from: mimuw.edu.pl/~bojan/paper/atom-book.

[BP01] J. A. Bergstra and A. Ponse. Non-regular iterators in process algebra. *Theoretical Computer Science*, 269(1):203–229, 2001.

[FL10] Y. Fu and H. Lu. On the expressiveness of interaction. *Theoretical Computer Science*, 411(11–13):1387 – 1451, 2010.

[Fu17] Y. Fu. The universal process. *Log. Methods Comput. Sci.*, 13(4), 2017.

[Gla93] R. J. van Glabbeek. The Linear Time - Branching Time Spectrum II. In E. Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[GLT09a] R. J. van Glabbeek, B. Luttik, and N. Trčka. Branching Bisimilarity with Explicit Divergence. *Fundamenta Informaticae*, 93(4):371–392, 2009.

[GLT09b] R. J. van Glabbeek, B. Luttik, and N. Trčka. Computation tree logic with deadlock detection. *Logical Methods in Computer Science*, 5(4), 2009.

[Gor10] D. Gorla. Towards a Unified Approach to Encodability and Separation Results for Process Calculi. *Information and Computation*, 208(9):1031–1053, 2010.

[GP02] M. J. Gabbay and A. M. Pitts. A New Approach to Abstract Syntax with Variable Binding. *Formal Aspects of Computing*, 13(3):341–363, 2002.

[GW96] R. J. van Glabbeek and W. P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *J. ACM*, 43(3):555–600, 1996.

[Lut20] B. Luttik. Divergence-preserving branching bisimilarity. In O. Dardha and J. Rot, editors, *Proceedings EXPRESS/SOS 2020*, volume 322 of *EPTCS*, pages 3–11, 2020.

[LY15] B. Luttik and F. Yang. Executable behaviour and the π-calculus (extended abstract). In S. Knight, I. Lanese, A. Lluch-Lafuente, and H. Torres Vieira, editors, *Proceedings 8th Interaction and Concurrency Experience, ICE 2015, Grenoble, France, 4-5th June 2015*, volume 189 of *EPTCS*, pages 37–52, 2015.

[Mil92] R. Milner. Functions as Processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Pit16] A. M. Pitts. Nominal techniques. *SIGLOG News*, 3(1):57–72, 2016.

[SW01] D. Sangiorgi and D. Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[Tur36] A.M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *J. of Math*, 58:345–363, 1936.

[Yan18] F. Yang. *A Theory of Executability (with a focus on the expressivity of process calculi)*. PhD thesis, Eindhoven University of Technology, June 2018.