Executable Behaviour and the π-Calculus

Bas Luttik
Department of Mathematics and Computer Science,
Eindhoven University of Technology
Eindhoven, The Netherlands
Department of Computer Science
VU University Amsterdam
Amsterdam, The Netherlands
s.p.luttik@tue.nl

Fei Yang
Department of Mathematics and Computer Science,
Eindhoven University of Technology
Eindhoven, The Netherlands
f.yang@tue.nl

Reactive Turing machines extend classical Turing machines with a facility to model observable interactive behaviour. We call a behaviour executable if, and only if, it is behaviourally equivalent to the behaviour of a 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 all 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 executable up to divergence-preserving branching bisimilarity. Motivated by an intuitive understanding of executability, we then consider a restriction on the operational semantics of the π-calculus that does associate with every π-term executable behaviour, at least up to the version of branching bisimilarity that does not require the preservation of divergence.

1 Introduction

The Turing machine [19] is generally accepted as the machine model that captures precisely which functions are algorithmically computable. As a theoretical model of the behaviour of actual computing systems, however, it has limitations, as was already observed, e.g., by Petri [16]. Most notably, Turing machines lack facilities to adequately deal with two important ingredients of modern computing: *interaction* and *non-termination*. Concurrency theory emerged from the work of Petri and developed into an active field of research. It resulted in a plethora of calculi for the formal specification of the behaviour of reactive systems, of which the π-calculus [15, 18] is probably the best-known to date.

Research in concurrency theory has focussed on defining expressive process specification formalisms, modal logics, studying suitable behavioural equivalences, etc. Expressiveness questions have also been addressed extensively in concurrency theory, especially in the context of the π-calculus (see, e.g., [12, 8]), but mostly pertaining to the so-called *relative expressiveness* of process calculi. The absolute expressiveness of process calculi, and in particular the question as to which interactive behaviour can actually be executed by a conventional computing system, has received less attention. In this paper, we consider the expressiveness of the π-calculus with respect to the model of reactive Turing machines, proposed in [3] as an orthogonal extension of classical Turing machines with a facility to model interaction in the style of concurrency theory.

Reactive Turing machines serve to define which behaviour can be executed by a computing system. Formally, we associate with every reactive Turing machine a transition system, which mathematically represents its behaviour. Then, we say that a transition system is executable if it is behaviourally equivalent to the transition system of a reactive Turing machine. Process calculi generally also have their

Submitted to:
ICE 2015

operational semantics defined by means of transition systems. Thus, we have a method to investigate 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. Note that the behavioural equivalence is a parameter of the method: if a behaviour specified in the process calculus is not executable up to some fine notion of behavioural equivalence (e.g., divergence-preserving branching bisimilarity), it may still be executable up to some coarser notion of behavioural equivalence (e.g., the divergence-insensitive variant of branching bisimilarity). The entire spectrum of behavioural equivalences (see [10]) is at our disposal to draw precise conclusions. We shall use the aforementioned method to characterize the expressiveness of the π-calculus.

We shall confirm that the π-calculus is expressive: every executable behaviour can be specified in the π-calculus up to divergence-preserving branching bisimilarity [9, 11], which is the finest behavioural equivalence discussed in van Glabbeek's seminal paper on behavioural equivalences [10]. Our proof explains how an arbitrary reactive Turing machine can be specified in the π-calculus. 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 analyze 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 reactive Turing machine is, however, purposely kept finite, since allowing reactive Turing machines to have an infinite alphabet of actions arguably leads to an unrealistic model of executability. As an alternative, we shall therefore investigate the executability of π-calculus behaviour subject to name restriction, considering only the observable behaviour of a π-calculus term that refers to a finite subset of the set of names. The underlying assumption is that any realistic system will be based on a finite alphabet of input symbols. The second cause is that, even under a finite name restriction, the transition system associated with a π-calculus term may still have unbounded branching. 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. We shall establish that, given a finite name restriction, the behaviour associated with a π-term is always executable up to (the divergence insensitive variant of) branching bisimilarity.

The paper is organized as follows. In Section 2, the basic definitions of reactive Turing machines and divergence-preserving branching bisimilarity are recapitulated, and we also recall the operational semantics of the π-calculus with replication. In Section 3, we prove the reactive Turing power of the π-calculus 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 executability of transition systems associated with π-calculus processes. First, we discuss reactive Turing machines based on an infinite alphabet of actions, and argue that then, trivially, every transition system associated with a π-calculus term can be simulated up to divergence-preserving branching bisimilarity, but that the ensued notion of executability is unrealistic. Then, we establish that every finite name restriction of a behaviour specifiable in the π-calculus is executable modulo the divergence-insensitive variant of branching bisimilarity. The paper ends with a discussion of related work and some conclusions in Section 5.

2 A Mathematical Theory of Behaviour

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 1 (Labelled Transition System)**.** *An $\mathcal{A}_\tau$-labelled transition system T is a triple $(\mathcal{S},\longrightarrow,\uparrow)$, where,*

1. *$\mathcal{S}$ is a set of* states*,*
2. *$\longrightarrow\subseteq\mathcal{S}\times\mathcal{A}_\tau\times\mathcal{S}$ is an $\mathcal{A}_\tau$-labelled* transition relation*. If $(s,a,t)\in\longrightarrow$, we write $s\xrightarrow{a}t$.*
3. *$\uparrow\in\mathcal{S}$ is the initial state.*

Let $(\mathcal{S},\longrightarrow,\uparrow)$ be an $\mathcal{A}_\tau$-labelled LTS; we define the set of reachable states from a state s as follows.

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

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 [9, 11], which is the finest behavioural equivalence from van Glabbeek's linear time - branching time spectrum [10].

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 $\mathcal{S}$, 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 (Branching Bisimilarity)**.** *Let $T_1=(\mathcal{S}_1,\longrightarrow_1,\uparrow_1)$ and $T_2=(\mathcal{S}_2,\longrightarrow_2,\uparrow_2)$ be transition systems. A* branching bisimulation *from T_1 to T_2 is a binary relation $\mathcal{R}\subseteq\mathcal{S}_1\times\mathcal{S}_2$ 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 $s_2',s_2''\in\mathcal{S}_2$, 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 $s_1',s_1''\in\mathcal{S}_1$, 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: $T_1\leftrightarrows_b T_2$) 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: $T_1\leftrightarrows_b^\Delta T_2$) 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 $T_1 = (\mathcal{S}_1, \longrightarrow_1, \uparrow_1)$ and $T_2 = (\mathcal{S}_2, \longrightarrow_2, \uparrow_2)$, $s_1 \in \mathcal{S}_1$ and $s_2 \in \mathcal{S}_2$, 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. 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 [4, 11].

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 the proofs of $\leftrightarrows_{\mathrm{b}}$ later.

Definition 3. *Let $T_1 = (\mathcal{S}_1, \longrightarrow_1, \uparrow_1)$ and $T_2 = (\mathcal{S}_2, \longrightarrow_2, \uparrow_2)$ be two transition systems. A relation $\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ is a bisimulation up to $\leftrightarrows_b$ 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 $s_1 \leftrightarrows_b s_1''$ and $a \neq \tau \vee s_1'' \not\leftrightarrows_b s_1'$, then there exists s_2' such that $s_2 \xrightarrow{a} s_2'$, $s_1'' \leftrightarrows_b \circ \mathcal{R}\, s_2$ and $s_1' \leftrightarrows_b \circ \mathcal{R}\, s_2'$; and*
2. *if $s_2 \xrightarrow{a} s_2'$, then there exist s_1', s_1'' such that $s_1 \longrightarrow^* s_1'' \xrightarrow{a} s_1'$, $s_1'' \leftrightarrows_b s_1$ and $s_1' \leftrightarrows_b \circ \mathcal{R}\, s_2'$.*

Lemma 1. *If $\mathcal{R}$ is a bisimulation up to $\leftrightarrows_b$, then $\mathcal{R} \subseteq \leftrightarrows_b$.*

2.1 Executable behaviour

The notion of reactive Turing machine (RTM) was put forward in [3] 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 with the set $\mathcal{A}_\tau$, which we now assume to be a finite set. Furthermore, the definition is parameterised with another finite set $\mathcal{D}$ of *data symbols*. We extend $\mathcal{D}$ with a special symbol $\Box \notin \mathcal{D}$ to denote a blank tape cell, and denote the set $\mathcal{D} \cup \{\Box\}$ of *tape symbols* by $\mathcal{D}_\Box$.

Definition 4 (Reactive Turing Machine)**.** *A* reactive Turing machine *(RTM) $\mathcal{M}$ is a triple $(\mathcal{S}, \longrightarrow, \uparrow)$, where*

1. *$\mathcal{S}$ is a finite set of* states*,*
2. *$\longrightarrow \subseteq \mathcal{S} \times \mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L, R\} \times \mathcal{S}$ is a finite collection of $(\mathcal{D}_\Box \times \mathcal{A}_\tau \times \mathcal{D}_\Box \times \{L, R\})$-labelled* transition rules *(we write $s \xrightarrow{a[d/e]M} t$ for $(s, d, a, e, M, t) \in \longrightarrow$),*
3. *$\uparrow \in \mathcal{S}$ is a distinguished* initial state*.*

Remark 1. *The original definition of RTMs in [3] includes an extra facility to declare a subset of the states of an RTM as being 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 $\mathcal{S}$, 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 it exists), and $\check{\Box}$ otherwise. Similarly ${}^>\delta$ is obtained by placing the tape head marker on the left-most symbol of δ (if it exists), and $\check{\Box}$ otherwise.

Definition 5. *Let $\mathcal{M} = (\mathcal{S}, \longrightarrow, \uparrow)$ 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 $C_\mathcal{M} = \{(s, \delta) \mid s \in \mathcal{S},\ \delta$ a tape instance$\}$ of all configurations of $\mathcal{M}$;*
2. *its transition relation $\longrightarrow \subseteq C_\mathcal{M} \times \mathcal{A}_\tau \times C_\mathcal{M}$ 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*. By analogy, the notion of RTM can be used to define a notion of *effectively executable behaviour*.

Definition 6 (Executability)**.** *A transition system is* executable *if it is the transition system associated with some RTM.*

Usually, we shall be interested in executability up to some behavioural equivalence. In [3], a characterisation of executability up to (divergence-preserving) branching bisimilarity is given that is independent of the notion of RTM. In order to be able to recapitulate the results below, we need the following definitions, pertaining to the recursive complexity and branching degree of transition systems. Let $T = (\mathcal{S}, \longrightarrow, \uparrow)$ be a transition system. We say that T is *effective* if $\longrightarrow$ is a recursively enumerable set. The mapping $out : \mathcal{S} \to 2^{\mathcal{A}_\tau \times \mathcal{S}}$ associates with every state its set of outgoing transitions, i.e., for all $s \in \mathcal{S}$, $out(s) = \{(a, t) \mid s \xrightarrow{a} t\}$. We say that T is *computable* if *out* is a recursive function. We call a transition system *finitely branching* if $out(s)$ is finite for every state s, and *boundedly branching* if there exists $B \in \mathbb{N}$ such that $|out(s)| \leq B$ for all $s \in \mathcal{S}$.

The following results were established in [3].

Theorem 1.
1. *The transition system $\mathcal{T}(\mathcal{M})$ associated with an RTM $\mathcal{M}$ is computable and boundedly branching.*
2. *For every finite set $\mathcal{A}_\tau$ and every boundedly branching computable $\mathcal{A}_\tau$-labelled transition system T, there exists an RTM $\mathcal{M}$ such that $T \leftrightarrows_b^\Delta \mathcal{T}(\mathcal{M})$.*
3. *For every finite set $\mathcal{A}_\tau$ and every effective $\mathcal{A}_\tau$-labelled transition system T there exists an RTM $\mathcal{M}$ such that $T \leftrightarrows_b \mathcal{T}(\mathcal{M})$.*

Notice the role played by divergence preservation in the preceding theorem. Divergence can be used to simulate the behaviour in a state with a high branching degree using states with lower branching degrees; the idea stems from [1] and is generalised in [17] to prove that every effective $\mathcal{A}_\tau$-labelled transition system is weakly bisimilar to a computable transition system. We proceed to discuss a criterion to decide whether a transition system $T = (\mathcal{S}, \longrightarrow, \uparrow)$ is not executable up to divergence-preserving branching bisimilarity, which is based on the notion of branching degree up to $\leftrightarrows_{\mathrm{b}}^\Delta$. Let us denote the

equivalence class of $s \in \mathcal{S}$ modulo $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$ by $[s]_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}} = \{s' \in \mathcal{S} \mid s \underline{\leftrightarrow}^{\Delta}_{\mathrm{b}} s'\}$. The *branching degree* up to $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$ of s, denoted by $deg_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}}(s)$, is defined as the cardinality of the set

$$\left\{(a,[s']_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}}) \,\middle|\, \exists s''.\, s \longrightarrow^{*} s'' \xrightarrow{a} s' \ \&\ s \underline{\leftrightarrow}^{\Delta}_{\mathrm{b}} s'' \ \&\ (a = \tau \implies s'' \not\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}} s')\right\} \ .$$

The branching degree modulo $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$ of T is the least upper bound of the branching degrees of all reachable states, which is defined as $deg_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}}(T) = \sup\{deg_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}}(s) \mid s \in Reach(\uparrow)\}$. We say that T is *boundedly branching* up to $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$ if there exists $B \in \mathbb{N}$, such that $deg_{\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}}(T) \le B$, otherwise it is *unboundedly branching* up to $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$.

Lemma 2. *If $s \underline{\leftrightarrow}^{\Delta}_{b} t$, then $deg_{\underline{\leftrightarrow}^{\Delta}_{b}}(s) = deg_{\underline{\leftrightarrow}^{\Delta}_{b}}(t)$.*

A *divergence* (up to $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$) in a transition system is an infinite sequence of reachable states $s_1, s_2, \ldots$ such that $s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \cdots$ and $s_i \underline{\leftrightarrow}^{\Delta}_{\mathrm{b}} s_j$ for all $i, j \in \mathbb{N}$. The following lemma shows that, in the absence of a divergence, boundedly branching transition systems are boundedly branching up to $\underline{\leftrightarrow}^{\Delta}_{\mathrm{b}}$.

Lemma 3. *If a transition system is boundedly branching and does not have divergence up to $\underline{\leftrightarrow}^{\Delta}_{b}$, then it is boundedly branching up to $\underline{\leftrightarrow}^{\Delta}_{b}$.*

Thus we conclude the following theorem from Theorem 1(1) and Lemma 3.

Theorem 2. *If a transition system T has no divergence up to $\underline{\leftrightarrow}^{\Delta}_{b}$ and is unboundedly branching up to $\underline{\leftrightarrow}^{\Delta}_{b}$, then it is not executable modulo $\underline{\leftrightarrow}^{\Delta}_{b}$.*

2.2 π-Calculus

The π-calculus was proposed by Milner, Parrow and Walker in [15] 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 [18], excluding the match prefix. We recapitulate some definitions from [18] 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:

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

In $x(z).P$ and $(z)P$, 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.

An α-conversion between π-terms is defined in [18] as a finite number of changes of bound names. In this paper, we do not distinguish among π-terms that are α-convertible, and we write $P = Q$ if P and Q are α-convertible.

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

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

$$\text{PREFIX}\quad \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P} \qquad \frac{}{x(y).P \xrightarrow{xz} P\{z/y\}}$$

$$\text{SUM}_{\text{L}}\quad \frac{P \xrightarrow{\alpha} P'}{(P+Q) \xrightarrow{\alpha} P'} \qquad \text{PAR}_{\text{L}}\quad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}\ \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset$$

$$\text{COM}_{\text{L}}\quad \frac{P \xrightarrow{\overline{x}y} P',\ Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{CLOSE}_{\text{L}}\quad \frac{P \xrightarrow{\overline{x}(z)} P',\ Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} (z)(P' \mid Q')}\ z \notin \mathsf{fn}(Q)$$

$$\text{RES}\quad \frac{P \xrightarrow{\alpha} P'}{(z)P \xrightarrow{\alpha} (z)P'}\ z \notin \alpha \qquad \text{OPEN}\quad \frac{P \xrightarrow{\overline{x}z} P'}{(z)P \xrightarrow{\overline{x}(z)} P'}\ z \neq x$$

$$\text{REP}\quad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \frac{P \xrightarrow{\overline{x}y} P',\ P \xrightarrow{xy} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P} \qquad \frac{P \xrightarrow{\overline{x}(z)} P',\ P \xrightarrow{xz} P''}{!P \xrightarrow{\tau} (z)(P' \mid P'') \mid !P}$$

Figure 1: Operational rules for the π-calculus

The rules in Fig. 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 $\mathcal{T}(P) = (\mathcal{S}_P, \longrightarrow_P, P)$. The set of states $\mathcal{S}_P$ 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 $\mathcal{S}_P$ (i.e., $\longrightarrow_P = \longrightarrow \cap (\mathcal{S}_P \times \mathcal{A}_\pi \times \mathcal{S}_P)$), 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 [18]), we define the following abbreviations:

$$\overline{x}\langle y_1,\ldots,y_n\rangle.P \stackrel{\text{def}}{=} (w)\overline{x}w.\overline{w}y_1.\cdots\overline{w}y_n.P \quad (w \notin \mathsf{fn}(P)), \text{ and}$$
$$x(z_1,\ldots,z_n).P \stackrel{\text{def}}{=} x(w).w(z_1).\cdots w(z_n).P \ .$$

The following lemma establishes that divergence-preserving branching bisimilarity is compatible with restriction and parallel composition. This will be a useful property when establishing the correctness of our simulation of RTMs in the π-calculus, in the next section.

Lemma 4. *For all π-terms P, P', Q, and Q':*

1. *if $P \leftrightarrows_b^\Delta P'$, then $(a)P \leftrightarrows_b^\Delta (a)P'$;*
2. *if $P \leftrightarrows_b^\Delta P'$ and $Q \leftrightarrows_b^\Delta Q'$, then $P \mid Q \leftrightarrows_b^\Delta P' \mid Q'$.*

3 Specifying Executable Behaviour in the π-Calculus

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, $\mathcal{T}(\mathcal{M}) \leftrightarrows_{\text{b}}^\Delta \mathcal{T}(P)$.

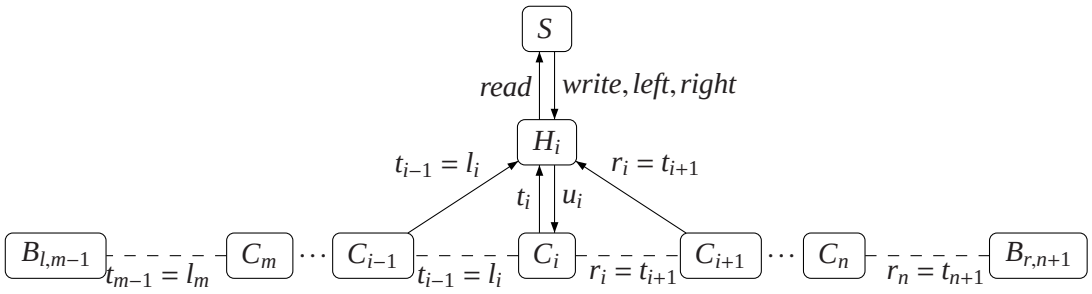

Figure 2: Specification of an RTM utilizing the linking structure of the π-calculus

The structure of our specification is illustrated in Figure 2. In this figure, each node represents a parallel component of the specification, each labelled arrow stands for a communication channel with certain labels, and the dashed lines represent the links between cells. Moreover, the equalities on arrows and dashed lines tell the correspondence between the names defined in the linked terms. 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 2), and a finite specification of a control process that is specific for the RTM $\mathcal{M}$ under consideration (parallel component S in Figure 2). 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.

3.1 Tape

In [1], the behaviour of the tape of a Turing machine is finitely specified in ACP_τ making use of finite specifications of two stacks. The specification is not easily modified to take intermediate termination into account, and therefore, in [3], an alternative solution is presented, specifying the behaviour of a tape in TCP_τ by using a finite specification of a queue (see also [2]). 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. $\overline{read}\, d$: the datum under the tape head is output along the channel *read*;
2. *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 $\delta_L \check{} d \delta_R$ or $\delta_L d \check{} \delta_R$, 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 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:

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

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. 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.

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:

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

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 utilized: u is the update channel of the new blank cell, and l will later be used as the link to generate another cell. Thus a new cell is generated from $\overline{c}\langle t,l,r,u,\Box\rangle.\mathbf{0}$, and the cell generator on the left is updated by $\overline{b_l}\langle l,t\rangle.\mathbf{0}$. In the second case, a symmetrical procedure is implemented by $B_r(t,l,r,u)$.

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. 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:

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

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

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

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.

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

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

Lemma 5. *Suppose $C_i, B_{l,m}, B_{r,n}, H_i$ are as defined before, then the following statements are valid:*

1. $(c)(\overline{c}\langle t_i, l_i, r_i, u_i, d_i\rangle.\mathbf{0} \mid !C) \leftrightarrows^{\Delta}_{b} (c)(C_i(d_i) \mid !C)$
2. $(b_l, b_r)(\overline{b_l}\langle t_m, r_m\rangle.\mathbf{0} \mid !B) \leftrightarrows^{\Delta}_{b} (b_l, b_r, u_m, l_m)(B_{l,m} \mid !B)$
3. $(b_l, b_r)(\overline{b_r}\langle t_n, l_n\rangle.\mathbf{0} \mid !B) \leftrightarrows^{\Delta}_{b} (b_l, b_r, u_n, r_n)(B_{r,n} \mid !B)$
4. $(h)(\overline{h}\langle t_i, l_i, r_i, u_i, d_i\rangle.\mathbf{0} \mid !H) \leftrightarrows^{\Delta}_{b} (h)(H_i \mid !H)$

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 6. *There are four types of transitions from $Tape^i_{[m,n]}(\vec{d}_{[m,n]})$:*

1. $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{\overline{read}\, d_i} \leftrightarrows^{\Delta}_{b} Tape^i_{[m,n]}(\vec{d}_{[m,n]})$;
2. $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{write(e)} \leftrightarrows^{\Delta}_{b} Tape^i_{[m,n]}(d_{[m,i-1]}, e, d_{[i+1,n]})$;
3. $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{left} \leftrightarrows^{\Delta}_{b} Tape^{i-1}_{[m,n]}(\vec{d}_{[m,n]})$ *(if $i > m$);*
   $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{left} \leftrightarrows^{\Delta}_{b} Tape^{i-1}_{[m-1,n]}(\Box, \vec{d}_{[m,n]})$ *(if $i = m$);*
4. $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{right} \leftrightarrows^{\Delta}_{b} Tape^{i+1}_{[m,n]}(\vec{d}_{[m,n]})$ *(if $i < n$);*
   $Tape^i_{[m,n]}(\vec{d}_{[m,n]}) \xrightarrow{right} \leftrightarrows^{\Delta}_{b} Tape^{i+1}_{[m,n+1]}(\vec{d}_{[m,n]}, \Box)$ *(if $i = n$).*

3.2 Finite control

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

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

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

Let $\vec{s} = s_1, s_2, \ldots, s_m \in \mathcal{S}_{\mathcal{M}}$, and $\vec{e} = e_1, e_2, \ldots, e_n \in \mathcal{D}_{\Box}$; we define

$$Control_{s,d} \stackrel{\text{def}}{=} (\vec{s}, \vec{e})(S_{s,d} \mid !S) \ .$$

The following lemma illustrates the behaviour of the control process.

Lemma 7. *Given an RTM $\mathcal{M} = (\mathcal{S}_\mathcal{M}, \longrightarrow_\mathcal{M}, \uparrow_\mathcal{M})$, we have the following transition sequence:*

$$Control_{s,d} \xrightarrow{a} (\vec{s},\vec{e})(\overline{write}\, e.\overline{m}.read(f).\bar{t}.\bar{f}.\mathbf{0} \mid !S) \xrightarrow{\overline{write}\, e} \xrightarrow{\overline{m}} \xrightarrow{read\, f} \underline{\leftrightarrow}_b^\Delta Control_{t,f}.\mathbf{0} \ .$$

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

Finally, for a given 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 $\vec{r} = read, write, left, right$; we define

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

The following lemma shows that $M_{s,\delta_L\check{d}\delta_R}$ actually simulates every computation step of an RTM.

Lemma 8. *Given an RTM $\mathcal{M} = (\mathcal{S}_\mathcal{M}, \longrightarrow_\mathcal{M}, \uparrow_\mathcal{M})$, for every configuration $(s,\delta_L\check{d}\delta_R)$, its specification $M_{s,\delta_L\check{d}\delta_R}$ has the following transition*

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

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

Theorem 3. *Given an RTM $\mathcal{M}$, we have*

$$\mathcal{T}(M_{\uparrow,\check{\square}}) \underline{\leftrightarrow}_b^\Delta \mathcal{T}(\mathcal{M}) \ .$$

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

Corollary 1. *For every executable transition system T there exists a π-term P, such that $T \underline{\leftrightarrow}_b^\Delta \mathcal{T}(P)$.*

4 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 that there is an apparent mismatch between the formalisms of RTMs and the π-calculus. On the one hand, the notion of RTM as we have defined in Section 2 presupposes *finite* sets $\mathcal{A}_\tau$ and $\mathcal{D}_\square$ of actions and data symbols, and also the transition relation of an RTM is *finite*. As a consequence, we have observed, 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 Fig. 1) may contain states with an infinite branching degree, due to the rules for input prefix and bound output prefix. Regarding this gap, we shall explore two ways to establish simulation of π-calculus terms by RTMs. One is to extend the formalism of RTMs to presuppose an infinite set of actions, and the other is to restrict the π-calculus to use a finite set of names.

4.1 RTMs with Infinitely Many Actions

Let us first consider allowing RTMs to have infinitely many actions in order to accommodate for the infinitely many names in the π-calculus.

Recall Definition 4, an RTM has a finite set of states $\mathcal{S}$ and a finite set of transition rules defining the associated transition relation. If we allow RTMs to have infinitely many actions, then, inevitably, we should also allow them to have infinitely many transition rules. The following lemma shows that we then also either need infinitely many states or infinitely many data symbols.

Lemma 9. *There does not exist an RTM with infinitely many actions but finitely many states and data symbols that simulates the π-term $P = x(y).\overline{y}.\mathbf{0}$ modulo branching bisimilarity.*

Now, assume we allow the alphabet of data symbols to be infinite. It is then straightforward to use it to encode an infinite set of control states. Allowing an infinite set of data symbols, in fact, greatly enhances the expressiveness of RTMs, as the following theorem shows.

Theorem 4. *Every infinitely branching effective transition system can be simulated up to divergence-preserving branching bisimilarity by an RTM with infinite sets of action symbols and data symbols.*

As a consequence, we can simulate every π-calculus term up to divergence-preserving branching bisimilarity with an RTM having infinite sets of action symbols and data symbols. So, if we would extend the formalism of RTMs allowing infinitely many action symbols and data symbols and define the notion of executability on the basis of it, then we would get that every π-calculus process is executable up to divergence-preserving branching bisimilarity. One may argue, however, that such extension is not in accordance with reality, referring to the finiteness of realistic computing systems. Actually, this result only shows the existence of such theoretical models, rather than giving a way of implementation. Moreover, the conclusion is valid for every model with an effective operational semantics, even if it has infinite branching.

4.2 Restricting the π-calculus

Now we proceed to consider the other option, which is to propose a restriction on the transition systems associated with π-terms such that they refer only to finitely many actions.

The infinity of the set of actions in the π-calculus arises in two ways, the free input names and the bound output names. The free input names allow a process to receive any potential input from the environment and the bound output names give a process the ability to generate unboundedly many distinct private channels to communicate with other processes. For both purposes, infinite branching of the transition system is essential. Observe, that the infinite branching caused by input prefix can be thought of as a technical device in the operational semantics to model the communication of an arbitrary name from one parallel component to another. The name that will be received, can either be a free name of the sending process (a value), or a restricted name (a private channel). Since the sending parallel component will only have a finite number of free names, only finitely many values can be communicated. Although, technically speaking, according to the operational semantics, infinitely many distinct private channels may be communicated when an input prefix synchronises with a bound output prefix, the communicated private channel is not observable, and the resulting π-terms are all equated by α-conversion, so the only observable effect of the interaction is that after the communication the sending and receiving parties share a private channel of which the name is irrelevant.

Our goal is to investigate to what extent the behaviour specified by an individual π-term is executable. Motivated by the above intuitive interpretation of interaction of a π-term with its environment, we assume

that the behaviour specified by that π-term is executed in an environment that may offer data values from some presupposed finite set on its input channels. This assumption seems reasonable as a machine should know in advance which symbols to expect as an input. Furthermore, we assume that there is a facility for establishing a private channel between the π-term and its environment. (Such a facility could, e.g., be implemented using encryption, but we will abstract from the actual implementation of the facility.) We define a restriction on the transition systems associated with π-terms that is based on these assumptions.

Definition 7. *Let $\mathcal{N}' \subseteq \mathcal{N}$ be a set of names, let $\mathcal{A}'_\pi = \mathcal{A}_\pi - (\{xy \mid x,y \in \mathcal{N},\ y \notin \mathcal{N}'\} \cup \{\overline{x}(z) \mid x,z \in \mathcal{N}\})$, and let P be a π-term. The transition system associated with P restricted to $\mathcal{N}'$, denoted by $\mathcal{T}(P) \upharpoonright \mathcal{N}'$, is a triple $(\mathcal{S}_P \upharpoonright \mathcal{N}', \longrightarrow_P \upharpoonright \mathcal{N}', P)$, obtained from $\mathcal{T}(P) = (\mathcal{S}_P, \longrightarrow_P, P)$ as follows:*

1. *$\mathcal{S}_P \upharpoonright \mathcal{N}'$ is the set of states reachable from P by means of transitions that are not labelled by xy ($y \notin \mathcal{N}'$); and*
2. *$\longrightarrow_P \upharpoonright \mathcal{N}'$ is the restriction of $\longrightarrow_P$ obtained by excluding all transitions labelled with xy ($y \notin \mathcal{N}'$), and relabelling all transitions labelled with $\overline{x}(z)$ ($x,z \in \mathcal{N}$) to $\nu\overline{x}$, i.e.,*

$$\longrightarrow_P \upharpoonright \mathcal{N}' = (\longrightarrow_P \cap (\mathcal{S}_P \upharpoonright \mathcal{N}' \times \mathcal{A}'_\pi \times \mathcal{S}_P \upharpoonright \mathcal{N}')) \cup \{(s, \nu\overline{x}, t) \mid s,t \in \mathcal{S}_P \upharpoonright \mathcal{N}', s \xrightarrow{\overline{x}(z)}_P t\} \ .$$

Using [18, Lemma 1.4.1], it is straightforward to show that for every π-term the set of actions of the π-calculus appearing as labels in $\mathcal{T}(P) \upharpoonright \mathcal{N}'$ is finite. Furthermore, the transition system associated with a π-term by the operational semantics, and also its restriction according to Definition 7 are clearly effective. Hence, as an immediate corollary of Theorem 1(3), we may conclude that the transition system associated with a π-term can be simulated by an RTM modulo (divergence-insensitive) branching bisimilarity.

Corollary 2. *For every closed π-term P, and for every finite set of input names $\mathcal{N}' \subseteq \mathcal{N}$, there exists an RTM M such that $\mathcal{T}(P) \upharpoonright \mathcal{N}' \leftrightarrows_b \mathcal{T}(M)$.*

The following example shows that there exist π-terms with which the structural operational semantics associates a transition system without divergence that is unboundedly branching up to $\leftrightarrows_{\mathrm{b}}^{\Delta}$. Note that by Theorem 2 such π-terms are not executable modulo divergence-preserving branching bisimilarity.

Example 1. *Consider the π-process $P \stackrel{def}{=} (c,i,d,s,flip)(\overline{i}\,s.\mathbf{0} \mid flip.\mathbf{0} \mid !C \mid !I \mid !D)$, with C, I and D defined as follows:*

$$\begin{aligned} C &\stackrel{def}{=} c(h,t,b).(\overline{h}\langle t,b\rangle.\mathbf{0} + flip.\overline{c}\langle h,t,1\rangle.\mathbf{0}) \\ I &\stackrel{def}{=} i(h).(inc.(h')\overline{c}\langle h',h,0\rangle.\overline{i}\,h'.\mathbf{0} + flush.\overline{flip}.\overline{d}\,h.\mathbf{0}) \\ D &\stackrel{def}{=} d(h).(h(t,b).\overline{b}.\overline{d}\,t.\mathbf{0}) \end{aligned}$$

Intuitively, the process $!C$ facilitates the generation of a linked list of one-bit cells with a pointer h to the head of the list, a pointer t to the tail of the list, and a bit b. Each cell may either output, along h, the link t to the tail of the list and its bit b, or it may receive the instruction flip after which it recreates itself with the value 1. The process I serves as the interface process. It maintains a link to the head of the list. Upon receiving an inc-instruction, it adds another one-bit cell to the list, and upon receiving the flush-instruction, it flips at most one of the bits, and then calls D. The process D then simply outputs the bits in reverse sequence.

Consider the state reached after performing n inc-actions, followed by a flush-action. In this state, the list contains a string of n 0s. The τ-transitions that correspond to the interaction of $\overline{flip}$ between

I and one of the flips of one of the one-bit cells or flip in the definition of P have the effect of non-deterministically changing (at most) one of the 0*s to a* 1*. Note that there are $n+1$ such τ-transitions, and since D will subsequently output the sequence in order, the states reached by these τ-transitions are (pairwise) not divergence-preserving branching bisimilar. Hence, it follows that for every n, the transition system associated with P has a reachable state with a branching degree modulo $\underline{\leftrightarrow}_b^{\Delta}$ of at least $n+1$. It follows that the transition system associated with P is unboundedly branching up to $\underline{\leftrightarrow}_b^{\Delta}$.*

Note that the only names occurring as part of the labels on the transitions in the transition system associated with the π-term P in the preceding example are $\overline{0}$, $\overline{1}$, *inc* and *flush*, so if $\mathcal{N}'$ contains at least these four names, then P satisfies $\mathcal{T}(P) \upharpoonright \mathcal{N}' = \mathcal{T}(P)$. Let us say, in general, that a π-term P has *finitely many observable names* if there exists a finite set $\mathcal{N}' \subseteq \mathcal{N}$ such that $\mathcal{T}(P) \upharpoonright \mathcal{N}' = \mathcal{T}(P)$. Note that, in this case, P cannot have parameterised free inputs, nor bound outputs. For π-terms with finitely many observable names, we have the following corollary as a consequence of a combination of Corollary 2 and Example 1.

Corollary 3. *Every closed π-term P with finitely many observable names is executable up to (divergence-insensitive) branching bisimilarity, but there exist closed π-terms with finitely many observable names that are not executable up to divergence-preserving branching bisimilarity.*

Our notion of restriction is introduced to restrict labelled transition systems associated with π-terms to finitely many names. Alternatively, we could define a finite version π_{fin} of the π-calculus, presupposing a *finite* set of names $\mathcal{N}$ right from the beginning. If $\mathcal{T}(P)$ is the labelled transition system associated with P according to the operational semantics of π_{fin}, then $\mathcal{T}(P) \upharpoonright \mathcal{N}$ is obtained from $\mathcal{T}(P)$ by replacing all transitions with the label $\overline{x}(z)$ by transitions with the label $\nu\overline{x}$. Apart from this modification, restriction keeps all observable behaviour.

5 Conclusions and Related Work

We have investigated the expressiveness of the π-calculus in relation to the theory of executability provided by reactive Turing machines. The issue of the expressiveness of the π-calculus has been extensively studied (see [12] for a comprehensive overview of research in this area). A distinction is usually made between absolute and relative expressiveness results. The absolute expressiveness results focus on proving the (im)possibility of expressing a computational phenomenon in a calculus; the relative expressiveness results are mostly about encoding one calculus in another. Our results pertain to the absolute expressiveness of the π-calculus.

We have established that, up to divergence-preserving branching bisimilarity, every executable transition system can be specified in the π-calculus, showing that the π-calculus is reactively Turing powerful. Milner already established in [14] that the π-calculus is Turing powerful, 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. Our result that all executable behaviour can be specified in the π-calculus up to divergence-preserving branching bisimilarity also implies that the π-calculus is Turing powerful, and thus it subsumes Milner's result. Similarly, in [5] several expressiveness results for variants of CCS are obtained via an encoding of Random Access Machines, and also those results only make claims about the computational expressiveness of the calculi. Notice that the results in [14] and [5] confirm the computational power of the respective calculi, but do not make a qualitative statement about its interactive expressiveness. By showing that reactive Turing machines can be faithfully simulated, we at the same time confirm the interactive expressiveness of the π-calculus.

In his recent work [7], Fu also proposes to study computation and interaction in an integrated theory. His theory is built on four fundamental principles, rather than on a machine model. One of the contributions of his theory is a calculus including a bare minimum of primitives to be computationally and interactively complete, and he uses it to confirm the completeness of the π-calculus. We leave it for future work to explore the relationship between Fu's theory of interaction and the theory of executability based on reactive Turing machines.

We have observed that it is possible to specify behaviour in the π-calculus that is not executable up to any reasonable notion of behavioural equivalence, simply because it uses infinitely many observable names. For the presentation of the π-calculus it is technically important to presuppose an infinite set of names especially to model the feature of dynamic creation of private channels between components. In a real system, however, private channels between components may be implemented differently, e.g., using some form of encryption. We have shown that a behaviour specified in the π-calculus is executable up to the divergence-insensitive variant of branching bisimilarity if one restricts to finitely many observable names and does not associate a unique identifier with every dynamically created private channel.

It has been claimed (e.g., in [6]) that the π-calculus provides a model of computation that is behaviourally more expressive than Turing machines. Our results provide further justification for this claim, and characterise the difference. It should be noted that the difference in expressive power is at the level of interaction (allowing interaction between an unbounded number of components), rather than at the level of computation.

Acknowledgement We thank the reviewers for the elaborate discussions on the ICE 2015 forum, which led to several improvements of the article.

References

[1] J. C. M. Baeten, J. A. Bergstra & J. W. Klop (1987): *On the consistency of Koomen's Fair Abstraction Rule*. *Theoretical Computer Science* 51(12), pp. 129 – 176.

[2] Jos C. M. Baeten, Twan Basten & Michel A. Reniers (2010): *Process algebra: equational theories of communicating processes*. 50, Cambridge University Press.

[3] Jos C. M. Baeten, Bas Luttik & Paul van Tilburg (2013): *Reactive Turing Machines. Inform. Comput.* 231, pp. 143–166.

[4] Twan Basten (1996): *Branching Bisimilarity is an Equivalence Indeed! Inf. Process. Lett.* 58(3), pp. 141–147, doi:10.1016/0020-0190(96)00034-8. Available at http://dx.doi.org/10.1016/0020-0190(96)00034-8.

[5] Nadia Busi, Maurizio Gabbrielli & Gianluigi Zavattaro (2009): *On the expressive power of recursion, replication and iteration in process calculi*. *Mathematical Structures in Computer Science* 19(06), pp. 1191–1222.

[6] Eugene Eberbach (2007): *The $-calculus process algebra for problem solving: A paradigmatic shift in handling hard computational problems*. *Theoretical Computer Science* 383(2), pp. 200–243.

[7] Yuxi Fu (2014): *Theory of Interaction*. Available at http://basics.sjtu.edu.cn/~yuxi/.

[8] Yuxi Fu & Hao Lu (2010): *On the expressiveness of interaction*. *Theoretical Computer Science* 411(1113), pp. 1387 – 1451.

[9] R. J. van Glabbeek & W. Peter Weijland (1996): *Branching time and abstraction in bisimulation semantics*. *Journal of the ACM (JACM)* 43(3), pp. 555–600.

[10] Rob J. van Glabbeek (1993): *The linear time branching time spectrum II*. In: *CONCUR'93*, Springer, pp. 66–81.

[11] Rob J. van Glabbeek, Bas Luttik & Nikola Trčka (2009): *Branching bisimilarity with explicit divergence*. *Fundamenta Informaticae* 93(4), pp. 371–392.

[12] Daniele Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Inf. Comput.* 208(9), pp. 1031–1053.

[13] Bas Luttik & Fei Yang (2014): *Executable Behaviour and the π-Calculus*. *CoRR* abs/1410.4512. Available at `http://arxiv.org/abs/1410.4512`.

[14] Robin Milner (1990): *Functions as processes*. In Michael S. Paterson, editor: *Automata, Languages and Programming, Lecture Notes in Computer Science* 443, Springer Berlin Heidelberg, pp. 167–180, doi:`10.1007/BFb0032030`.

[15] Robin Milner, Joachim Parrow & David Walker (1992): *A calculus of mobile processes, I & II*. *Information and computation* 100(1), pp. 1–77.

[16] C. A. Petri (1962): *Kommunikation mit Automaten*. Ph.D. thesis, Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2.

[17] I. C. C. Phillips (1993): *A Note on Expressiveness of Process Algebra*. In G. L. Burn, S. Gay & M. D. Ryan, editors: *Proceedings of the First Imperial College Department of Computing Workshop on Theory and Formal Methods*, Workshops in Computing, Springer-Verlag, pp. 260–264.

[18] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.

[19] Alan Mathison Turing (1936): *On computable numbers, with an application to the Entscheidungsproblem*. *J. of Math* 58, pp. 345–363.