Bachelor thesis
Computer Science

Radboud University

The Turing machine model extended with interaction

Author:
Rick Erkens
s4100573
rick.erkens@student.ru.nl

First supervisor/assessor:
prof. dr. Herman Geuvers
herman@cs.ru.nl

Second assessor:
dr. Freek Wiedijk
freek@cs.ru.nl

Abstract

Interaction is a concept that is different from the computable functions. The Turing machine model is widely accepted as a model of computation and two different extensions have been proposed to turn Turing machines into a model for interaction. This thesis describes the details of those models and provides several examples to illustrate the differences between them. The result is that the notion of a reactive Turing machine [2] is a suitable model for interaction.

Contents

1 Introduction

Since Alan Turing introduced his model of machine computation, Turing machines have been used as the basic model for computation. For every computable function there exists a Turing machine that can compute it and there exists a universal Turing machine that can simulate every Turing machine, with an appropriate input.

Modern computation is wider than merely computing functions. Not all input is available from the start and a system may run infinitely, still behaving correctly. Turing machines are no longer an appropriate model since they can't model interaction. Peter Wegner provides many examples in [11] like airline reservation systems and car drivers being able to take input from the world. Interaction goes beyond the current models of computation and therefore we need to explore the expressiveness of new models and their properties.

In this thesis we will focus on two models that have been proposed to extend the Turing machine paradigm: reactive Turing machines [2] and persistent Turing machines [6]. We will use various examples to explain them and we will make a comparison between those models. The research question is: When is an interactive process computable in terms of Turing Machines?

In Section 2 we will discuss standard Turing machines and the notion of transition systems as a preliminary. In Section 3 and Section 4 the models of reactive Turing machines and persistent Turing machines will be explained. Because they are rather new notions, most definitions from their corresponding works will be repeated in a similar fashion. Of course they are slightly adapted to fit our conventions and notation. We also discuss concrete examples that are similar for each model to make the differences and similarities easy to see. In the consecutive sections we will make the comparison in terms of behavioural equivalence, universality and divergence. Finally Section 8 includes a brief discussion on the λ-calculus, the π-calculus in relation to RTMs and other extensions to the Turing machine paradigm.

The result of this thesis is a comparison of two works that share the goal of modelling interaction by making a slight adaption to the Turing machine model. Reactive Turing machines focus on interactive processes and persistent Turing machines focus on dynamic stream semantics. The notion of computable transition system is exactly defined in [2][1]. The transition system associated with a persistent Turing machine is not necessarily computable while the transition system of an RTM is. Baeten et al. prove that for every deterministic computable transition system there exists an RTM that can simulate it up to divergence preserving branching bisimilarity, which is a nice property for the goal of this model.

[1] It is actually reformulated from [1].

2 Preliminaries

In this section we will discuss the knowledge that is needed to read this thesis, with the assumption that the reader is familiar with set theoretic notation.

As a preliminary and to fix the notation that we will use, we will first go through the definition of the standard Turing machine. Because we will compare models from different authors that all prefer a different notation, we will use one notation for comparable aspects of these models for the sake of consistency. We will use Q to denote the set of states for example.

2.1 Turing machines

In 1936 Alan Turing published his paper on computable numbers [9]. He introduced a notion of machines that could be built physically.

Definition 2.1. A standard Turing machine $M \in \mathbb{TM}$ is a 6-tuple $M = (Q, \Gamma, \Sigma, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ where

- Q is the set of *states*;
- Γ is the set of *tape symbols* and $\Gamma_\square = \Gamma \cup \{\square\}$, the set of tape symbols including the symbol for a blank cell $\square$;
- Σ is the *alphabet* with $\Sigma \subseteq \Gamma$;
- $\rightarrow \subseteq Q \times \Gamma_\square \times \Gamma_\square \times \{L, R\} \times Q$ is the *transition relation* where L and R describe the movement of the tape head;
- $q_{\mathcal{I}}$ is the *initial state* with $q_{\mathcal{I}} \in Q$;
- $Q_{\mathcal{F}}$ is the set of *final states* with $Q_{\mathcal{F}} \subseteq Q$.

2.1.1 Tape

A Turing machine executes on a tape, which is a sequence of tape symbols. A tape instance δ is a string that is made up from tokens $\in \Gamma_\square$. Furthermore there is one special symbol in the tape, namely the one that the tape head currently points to. Traditionally we use a tape that is one-way infinite. This means that the machine "crashes" if the head moves to the left when it's on the left end of the tape.

Definition 2.2. The *language of tape instances* is the language

$$L_\delta = \Gamma_\square^* \check{\Gamma}_\square \Gamma_\square^*$$

where $\check{\Gamma}_\square$ denotes the tape symbol that the tape head currently points to. The notation for the empty word and also the empty tape is ϵ.

The relation $\rightarrow$ requires more explanation. If a certain Turing machine is in state q_i and the tape instance is $w_L a\check{b}cw_R$ with $w_L, w_R \in \Gamma_{\square}^*$, then a transition $(q_i, b, d, L, q_j) \in \rightarrow$ results in writing the symbol d on the location of the tape head, moving the tape head to the left and going into state q_j. This particular transition gives us the tape instance $w_L\check{a}dcw_R$. Because this notation is difficult to read we prefer to visualise Turing machines as state machines. For single transitions we use the notation $q_i \xrightarrow{[b,d]L} q_j$.

The configuration of a Turing machine is not the same as its state. We also need to keep track of the tape and the location of the tape head. We will make use of a general definition of Turing machine configurations for n-tape Turing machines. This general definition will help us when we are going to use Turing machines that use multiple tapes.

Definition 2.3. A configuration of an n-tape Turing machine is a $(n+1)$-tuple $(q, \delta_0, \delta_1, \ldots, \delta_{n-1}) \in Q \times {L_\delta}^n$.

Initially the Turing machine is in configuration $(q_{\mathcal{I}}, \check{\square}i\square\square\ldots)$ where i denotes the input. The machine then performs state transitions from $\rightarrow$ until it reaches state $Q_{\mathcal{F}}$, leaving an output on the tape. This is also where the difference between Γ and Σ is. The input and the output can only be described as strings of Σ^*, but the creator of the machine is free to use any symbol in $\Gamma_{\square}$ to aid him in the computation.

There are many variations and extensions of Turing machines that use multiple tapes, multiple tracks, tapes that are finite or infinite in one or two ways. We can even declare only one final state, but these additions don't improve the expressiveness of Turing machines since they are all equivalent to each other. The proof that each of those variations is equivalent to this standard Turing machine is taught in many computer science courses on computability [8].

2.1.2 Languages

Turing machines can be used to accept languages that are defined over Σ. A TM accepts a language L if and only if it reaches a final state for every input $w \in L$. The highest level of formal languages in the Chomsky hierarchy is the set of recursively enumerable languages. A language is recursively enumerable in case its elements are accepted by a Turing machine, but the rejection of a word may result in non-termination. This means that the TM may never give us the answer $w \notin L$. The more specific layer, the language of recursive languages, is defined as the sets of words that can be accepted by a TM that always stops.

2.1.3 Computable functions

Although he explicitly wrote his article on the computable numbers, Turing also stated that the fundamental problems for computability on functions and numbers are the same. Functions like $\mathbb{N}^n \to \mathbb{N}$ can be described as machines that take n numbers as an input on the tape and produce an output that is left on the tape after termination. Natural numbers are represented as strings of 1's. A natural number n is denoted by 1^{n+1} where a single 1 on the tape denotes zero. A function is computable if and only if there exists a machine that can compute it.

2.1.4 Universal Turing machine

With the introduction of machines, Turing also proved the existence of a universal machine that could simulate the behaviour of any other machine. These days we refer to such a machine as a universal Turing machine denoted by $\mathcal{U}$. Let's assume that a Turing machine M takes i as an input and produces o as an output (notation: $M(i) = o$). Then the universal Turing machine produces the same output $\mathcal{U}(\lceil M \rceil, i) = o$. To give $\mathcal{U}$ a machine to simulate, we need to put M on the tape. We need to express this machine as a coding.

Definition 2.4. The *coding* of a TM is a bijective function $\lceil - \rceil : \mathbb{TM} \to \Gamma_{\square}^{*}$.

2.1.5 Halting problem

The terms "decidable problems" and "computable functions" suggest that there also exist undecidable problems and noncomputable functions. One of the first problems that were proved undecidable is the halting problem. The machine that corresponds to the halting problem is the following:

$$M_{halting}(\lceil M \rceil, w) = \begin{cases} 1 & \text{if } M \text{ halts with input } w \\ 0 & \text{if } M \text{ diverges with input } w \end{cases}$$

This machine $M_{halting}$ does not exist. In Section 8 we will briefly discuss the Church-Turing thesis along with the λ-calculus, another model that captures the computable functions.

2.2 Transition systems

To reason about the behaviour of processes we use the general definition of an $\mathbb{A}$-labelled transition system. Transition systems are comparable to finite state automata in a sense that their state transition relation is labelled by a symbol from an alphabet. The main differences are that transition systems can contain an infinite number of states and transitions. Also the final state is optional. Throughout this thesis we will use a general definition of

transition systems. In later sections we will use two different models that have their own way of creating a transition system.

Definition 2.5. An $\mathbb{A}$-*labelled transition system* is a 5-tuple $(\mathcal{S}, \mathbb{A}, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ where:

$\mathcal{S}$ is the set of states;

$\mathbb{A}$ is the set of action symbols;

$\xrightarrow{\mathbb{A}} \subseteq \mathcal{S} \times \mathbb{A} \times \mathcal{S}$ is the transition relation;

$s_{\mathcal{I}}$ is the initial state with $s_{\mathcal{I}} \in \mathcal{S}$;

$s_{\mathcal{F}}$ is the set of final states with $s_{\mathcal{F}} \subseteq \mathcal{S}$.

The goal of this thesis is to study the computability of the transition systems associated with reactive or persistent Turing machines. We will use the notion of computable transition system that Baeten et al. describe in [2]. Let $out : \mathcal{S} \to 2^{\mathbb{A} \times \mathcal{S}}$ be the function that yields a set of all outgoing transitions for state every state s, defined as:

$$out(s) = \{(a, t) \in \mathbb{A} \times \mathcal{S} | s \xrightarrow{a} t\}.$$

Definition 2.6. Let $T = (\mathcal{S}, \mathbb{A}, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ be a finitely branching transition system (i.e. $|out(s)| < \infty$ for all s). Then T is *effective* if $\xrightarrow{\mathbb{A}}$ and $s_{\mathcal{F}}$ are recursively enumerable sets. T is *computable* if out is a recursive function and $s_{\mathcal{F}}$ is a recursive set.

Definition 2.7. The *branching degree* of a transition system is bounded by B if $\forall_{s \in \mathcal{S}}, |out(s)| \leq B$.

3 Reactive Turing machines

In [2] Baeten, Luttik and van Tilburg propose reactive Turing machines (RTMs for short) as an extension of standard Turing Machines. Every transition has an extra argument, which is an action symbol. This action symbol can be used to send or receive tape symbols to/from other processes along communication channels.

Definition 3.1. A reactive Turing machine $M \in \mathbb{RTM}$ is a 7-tuple $(Q, \Gamma, \mathcal{C}, \mathcal{A}_\tau, \rightarrow, q_\mathcal{I}, Q_\mathcal{F})$ where

- $\mathcal{C}$ is the set of *communication channels*;
- $\mathcal{A} = \{c!\gamma, c?\gamma | c \in \mathcal{C} \wedge \gamma \in \Gamma_\Box\}$ is the set of *action symbols* and $\mathcal{A}_\tau = \mathcal{A} \cup \{\tau\}$, the set of action symbols including the silent transition symbol τ;
- $\rightarrow \subseteq Q \times \mathcal{A}_\tau \times \Gamma_\Box \times \Gamma_\Box \times \{L, R\} \times Q$ is the *transition relation.*

An action $c!\gamma$ should be seen as the event that γ is sent along communication channel c. Similarly $c?\gamma$ is the event that γ is received along communication channel c, thus $c!\gamma$ and $c?\gamma$ are complementary actions. Transitions that use an action symbol must be executed simultaneously with transitions of their complementary action symbol. The complementary action must take place in another agent, for example another RTM. The transition $q_i \xrightarrow{c!\gamma[a/b]R} q_j$ can only be executed when a transition with $c?\gamma$ can be executed by another agent interacting with the RTM. To make internal computations an RTM can use the silent, unobservable transition τ. A standard Turing machine can be modelled by an RTM that only uses τ-transitions.

Example 3.2. In Figure 1 we see M_1, a simple RTM that sends the string $aaa\#aaa\#\ldots$ along channel i. This RTM in particular does not make use of silent $\tau-$transitions, which means that every step is externally observable.

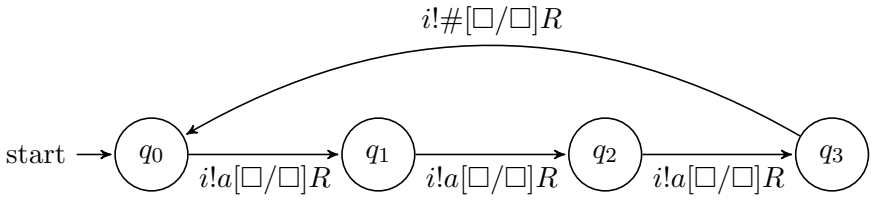

Figure 1: M_1, an example of an RTM.

3.1 Transition system of an RTM

With every RTM we can associate an $\mathbb{A}$-labelled transition system.

Definition 3.3. The transition system $\mathcal{T}_{RTM}(M) = (\mathcal{S}, \mathbb{A}, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ associated with a reactive Turing machine $M = (Q, \Gamma_{\square}, \mathcal{C}, \mathcal{A}_{\tau}, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ is an $\mathbb{A}$-labelled transition system such that:

$\mathcal{S}$ is the set of all configurations of M (see Definition 2.3);

$\mathbb{A} = \mathcal{A}_{\tau}$ is the set of labels;

$\xrightarrow{\mathbb{A}}$ is the least relation such that $\forall_{a \in \mathcal{A}_{\tau}}, \forall_{d,e,f \in \Gamma_{\square}}, \forall_{\delta_L, \delta_R \in \Gamma_{\square}^*}$,

$$(q, \delta_L d \check{e} \delta_R) \xrightarrow{a} (q', \delta_L \check{d} f \delta_R) \text{ if and only if } q \xrightarrow{a[e/f]L} q'$$

$$(q, \delta_L \check{e} d \delta_R) \xrightarrow{a} (q', \delta_L f \check{d} \delta_R) \text{ if and only if } q \xrightarrow{a[e/f]R} q';$$

$s_{\mathcal{I}} = (q_{\mathcal{I}}, \check{\square})$ is the *initial state*;

$s_{\mathcal{F}} = Q_{\mathcal{F}} \times L_{\delta}$.

Example 3.4. The transition system of M_1 in Example 3.2 is the following:

$$\mathcal{T}_{RTM}(M_1) = (q_0, \check{\square}) \xrightarrow{i!1} (q_1, \check{\square}) \xrightarrow{i!1} (q_2, \check{\square}) \xrightarrow{i!1} (q_3, \check{\square}) \xrightarrow{i!\#} (q_0, \check{\square}) \ldots$$

The RTM M_1 does not have any final states, thus the transition system is infinite.

3.2 Parallel composition

To describe interaction requires multiple RTMs running in parallel. This requires a new definition in which a set of communication channels is chosen for communication between those two machines. For this parallel composition we can also create a transition system. The notation for this composition is $[M_1||M_2]_{\mathcal{C}'}$ where $\mathcal{C}$ is a set of communication channels that are externally observable.

Definition 3.5. Let $M_1 = (Q_1, \Gamma_1, \mathcal{C}_1, \mathcal{A}_{\tau 1}, \rightarrow_1, q_{\mathcal{I}1}, Q_{\mathcal{F}1})$ and $M_2 = (Q_2, \Gamma_2, \mathcal{C}_2, \mathcal{A}_{\tau 2}, \rightarrow_2, q_{\mathcal{I}2}, Q_{\mathcal{F}2})$ be RTMs. Then the parallel composition of M_1 and M_2 is defined as $[M_1||M_2]_{\mathcal{C}} = (Q, \mathcal{C}, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ where

$Q = Q_1 \times Q_2$ is the set of combined states of M_1 and M_2;

$\mathcal{C} \subseteq \mathcal{C}_1 \cup \mathcal{C}_2$ is a set of shared communication channels;

$\rightarrow \subseteq \rightarrow_1 \times \rightarrow_2$ is the transition relation such that $(q_1, q_2) \rightarrow (q_1', q_2')$ if and only if

(a) either $q_1 \xrightarrow{a[b/c]M}_1 q_1'$ and $q_2 = q_2'$ or $q_2 \xrightarrow{a[b/c]M}_2 q_2'$ and $q_1 = q_1'$ (either M_1 makes a transition or M_2 makes a transition) or

(b) $q_1 \xrightarrow{c!\gamma[b/c]M}_1 q_1'$ and $q_2 \xrightarrow{c?\gamma[b'/c']M'}_2 q_2'$ or $q_1 \xrightarrow{c?\gamma[b/c]M}_1 q_1'$ and $q_2 \xrightarrow{c!\gamma[b'/c']M'}_2 q_2'$ for some $c \in \mathcal{C}$ and $\gamma \in \Gamma_{\square 1} \cap \Gamma_{\square 2}$ (M_1 and M_2 make transitions of complementary action symbol);

$q_{\mathcal{I}} = (q_{\mathcal{I}1}, q_{\mathcal{I}2})$ is the initial state;

$Q_{\mathcal{F}} = Q_{\mathcal{F}1} \times Q_{\mathcal{F}2}$ is the set of final states.

Similarly to the transition systems of single RTMs, we can define a transition system over a parallel composition. Note that Baeten et al. do not mention the definition of the parallel composition itself, but only the definition of its transition system. The definition of a transition system over a parallel composition is defined likewise.

Definition 3.6. Let $M_1 = (Q_1, \Gamma_1, \mathcal{C}_1, \mathcal{A}_{\tau 1}, \rightarrow_1, q_{\mathcal{I}1}, Q_{\mathcal{F}1})$ and $M_2 = (Q_2, \Gamma_2, \mathcal{C}_2, \mathcal{A}_{\tau 2}, \rightarrow_2, q_{\mathcal{I}2}, Q_{\mathcal{F}2})$ be RTMs and let their corresponding transition systems be $\mathcal{T}_{RTM}(M_1) = (\mathcal{S}_1, \mathbb{A}_1, \xrightarrow{\mathbb{A}}_1, s_{\mathcal{I}1}, s_{\mathcal{F}1})$ and $\mathcal{T}_{RTM}(M_2) = (\mathcal{S}_2, \mathbb{A}_2, \xrightarrow{\mathbb{A}}_2, s_{\mathcal{I}2}, s_{\mathcal{F}2})$. Let $M = [M_1||M_2]_{\mathcal{C}}$ for some $\mathcal{C} \subseteq \mathcal{C}_1 \cap \mathcal{C}_2$. Then the transition system of the parallel composition of M_1 and M_2 is the transition system $\mathcal{T}_{RTM}(M) = \mathcal{T}_{RTM}([M_1||M_2]_{\mathcal{C}}) = (\mathcal{S}, \mathcal{A}_\tau, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ such that

$\mathcal{S} = \mathcal{S}_1 \times \mathcal{S}_2$;

$\mathbb{A} = \mathbb{A}_1 \cup \mathbb{A}_2$;

$\xrightarrow{\mathbb{A}}$ is the relation such that $(s_1, s_2) \xrightarrow{a} (s_1', s_2')$ if and only if $a \in \mathcal{A}_\tau \setminus \{c!\gamma, c?\gamma | c \in \mathcal{C} \wedge \gamma \in \Gamma_\square\}$ and either

- $s_1 \xrightarrow{a} s_1'$ and $s_2 = s_2'$ or $s_2 \xrightarrow{a} s_2'$ and $s_1 = s_1'$ or
- $a = \tau$ and either
  $s_1 \xrightarrow{c!\gamma} s_1'$ and $s_2 \xrightarrow{c?\gamma} s_2'$ or
  $s_1 \xrightarrow{c?\gamma} s_1'$ and $s_2 \xrightarrow{c!\gamma} s_2'$ for some $\gamma \in \Gamma_\square, c \in \mathcal{C}$;

$s_{\mathcal{I}} = (s_{\mathcal{I}1}, s_{\mathcal{I}2})$;

$s_{\mathcal{F}} = s_{\mathcal{F}1} \times s_{\mathcal{F}2}$.

The communication between two RTMs along some channel becomes just an invisible τ-transition when you look at the parallel composition. This is visualised in 2 and we will also see this in the following example.

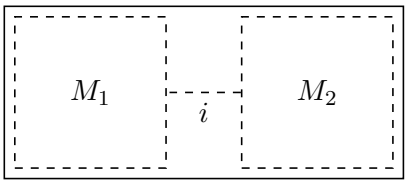

Figure 2: Internal communication in $[M_1||M_2]_{\{i\}}$ is unobservable

Example 3.7. To illustrate parallel composition we need another RTM to read the output of M_1 (Figure 1). In Figure 3 the RTM M_2 receives a string along channel i, copies is to its tape and decides whether the tape contains an even or an odd number of a's. After the input is separated by receiving #, the RTM submits the answer to this question to channel o (by sending either a 0 or a 1), followed by a #. The received input remains on the tape and M_1 also remembers the parity of the number of a's on the tape. At any time it can receive a new input from channel i to continue counting a's.

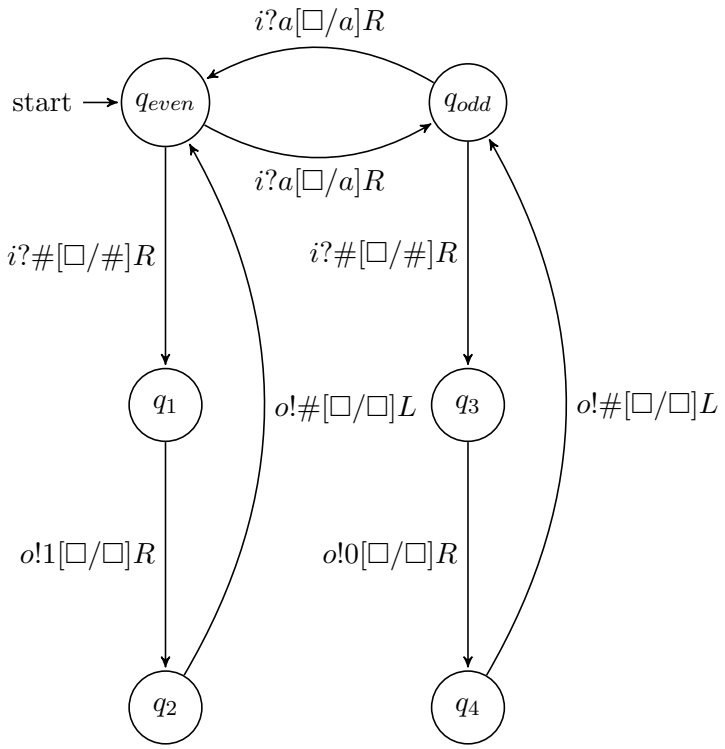

Figure 3: M_2, another example of an RTM.

We are going to create the transition system of the parallel composition of M_1 (recall Figure 1) and M_2. The communication channel i is used only for communication between the RTMs, thus $\mathcal{C}' = \{i\}$.

$\mathcal{T}_{RTM}([M_1||M_2]_{\{i\}}) =$
$((q_{even}, \check{\square}), (q_0, \check{\square})) \xrightarrow{\tau}$
$((q_{odd}, a\check{\square}), (q_1, \check{\square})) \xrightarrow{\tau}$
$((q_{even}, aa\check{\square}), (q_2, \check{\square})) \xrightarrow{\tau}$
$((q_{odd}, aaa\check{\square}), (q_3, \check{\square})) \xrightarrow{\tau}$
$((q_3, aaa\#\check{\square}), (q_0, \check{\square})) \xrightarrow{o!0}$
$((q_4, aaa\#\square\check{\square}), (q_0, \check{\square})) \xrightarrow{o!\#}$
$((q_{odd}, aaa\#\check{\square}), (q_0, \check{\square})) \xrightarrow{\tau}$
$((q_{even}, aaa\#a\check{\square}), (q_1, \check{\square})) \xrightarrow{\tau}$
$((q_{odd}, aaa\#aa\check{\square}), (q_2, \check{\square})) \xrightarrow{\tau}$
$((q_{even}, aaa\#aaa\check{\square}), (q_3, \check{\square})) \xrightarrow{\tau}$
$((q_1, aaa\#aaa\#\square\check{\square}), (q_0, \check{\square})) \xrightarrow{o!1}$
$((q_2, aaa\#aaa\#\check{\square}), (q_0, \check{\square})) \xrightarrow{o!\#}$
$((q_{even}, aaa\#aaa\#\check{\square}), (q_0, \check{\square})) \xrightarrow{\tau} \ldots$

As we defined in Definition 3.6, the communication between M_1 and M_2 cannot be observed, as it is transformed into a τ-transition.

4 Persistent Turing machines

In [6] Goldin, Smolka, Attie and Sonderegger formally describe persistent Turing machines (PTMs for short) and their expressiveness. The article is from 2004, although Goldin introduced PTMs in earlier papers [4, 5]. A PTM is a minimal extension of a Turing machine that uses three tapes. We will go through the formal definition of PTMs, persistent stream languages and their corresponding transition systems.

Definition 4.1. A persistent Turing machine $M \in \mathbb{PTM}$ is a 5-tuple $(Q, \Gamma, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ where

- $\rightarrow \subseteq Q \times {\Gamma_{\square}}^2 \times {\Gamma_{\square}}^2 \times \{L, R, S\}^3 \times Q$ is the transition relation where L, R and S describe the movement of the tape head[2];

A PTM uses three tapes. The first tape is a read-only input tape, the second tape is a read/write work tape and the third tape is a write-only output tape. Therefore the transition relation looks different from what we have seen so far. We denote transitions in a PTM, also called *microsteps*, as $q_i \xrightarrow{[a/R,b/c/R,d/R]} q_j$. This particular microstep means that the PTM goes from state q_i into state q_j, reads a from the input tape and b from the work tape, writes c to the work tape and d to the write tape, and shifts all tape heads to the right. Note that we want to use the option to have a tape head stand still by using S. Otherwise the state machines would become very large. Another option would be to do what is mentioned in the footnote, but we'd rather be consistent.

Example 4.2. Consider the PTM M_3 in figure 4. On the input tape there should be an input and the work tape contains a word. The machine first checks the parity of the number of a's on the work tape. After that, the machine copies the input tape to the work tape, also keeping track of the parity. When M_3 is done copying, the output tape contains a 0 or a 1 depending whether the number of a's is odd or even respectively. The final state does not denote the end of this process' life. Afterwards the machine's configuration shifts back to $(q_1, \check{\square} w_i', \check{\square} w', \check{\square})$ to respond to the new input w_i'. Note that Goldin et al. did not discuss such a concrete example on a low level. We will discuss the abstraction to macrosteps in the next subsection.

4.1 Macrosteps

The articles about PTMs don't focus on the small computational steps that a mechanical PTM would execute. They focus on the 'big' steps rather than

[2]The definition in [6] allows either a movement or a writing action in every transition (using the notation $\Gamma_{\square} \cup \{L, R\}$). For the sake of consistency, we will stick with our convention of moving and writing at the same time. Yet for convenience we want to use the option to have a tape head stand still, which is indicated by the movement S.

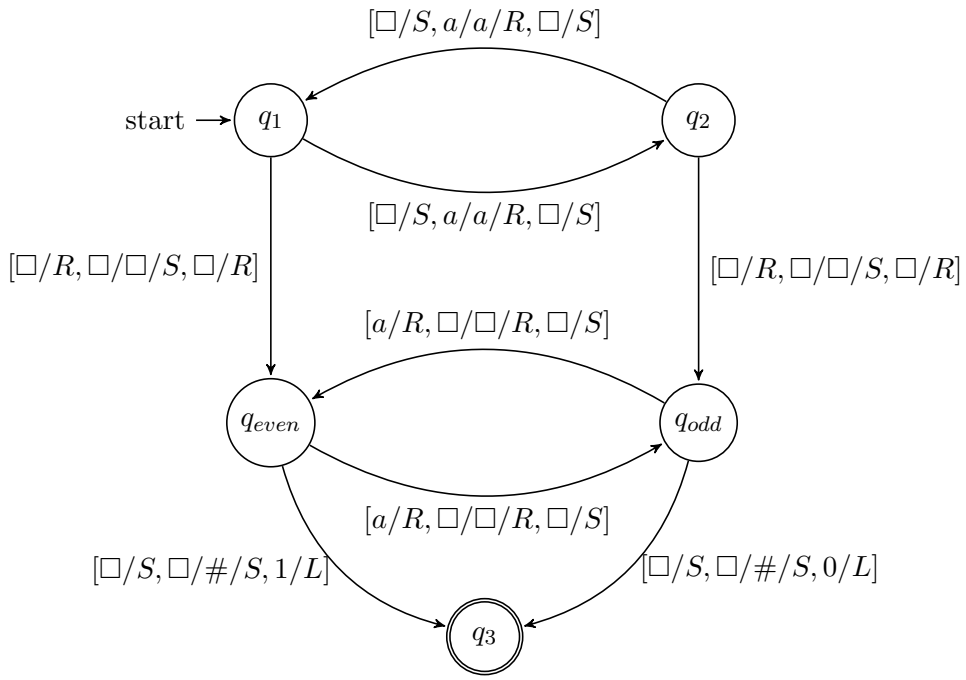

Figure 4: M_3, an example of a PTM.

the microsteps between interactions. Before a macrostep, a PTM has an input on the first tape. The work tape may contain a word and the output tape is empty. The machine then computes an output to write on the output tape. The content of the work tape may have changed during this process. After a macrostep the contents of the output tape are erased and the PTM receives a new input word. The content of the work tape persists between macrosteps, thus explaining the name 'persistent Turing machine '.

Definition 4.3. Let $w_i, w, w', w_o \in {\Gamma_\square}^*$. Let M be a PTM containing w_i on the input tape and w on the work tape. Then $w \xrightarrow{w_i/w_o} w'$ is a macrostep in M where as a result w_o is on the output tape and w' is on the work tape.

If the M diverges, we write $w \xrightarrow{w_i/\mu} s_\infty$ where s_∞ denotes a special divergence state and μ denotes a special divergence output. Since a PTM acts chaotically after it diverges we have $s_\infty \xrightarrow{w_i/\mu} s_\infty$ for all w_i.

For this thesis we assume that a macrostep (as a sequence of microsteps) ends when a PTM reaches its final state. We will refrain from describing detailed machines like the one in Example 4.2 for the remainder of this thesis and we will rather stick to describing PTMs as relations that deal with continuous interaction and persistent stream languages.

To describe the behaviour of a PTM we need to describe the output behaviour and the new work tape behaviour as a relation over the old work tape and the input tape, and the new work tape and the output tape. For this relation we use the notation

$$f_M(w_i, w) = (w_o, w')$$

where w_i, w, w_o and w' are the contents of the input tape, work tape, output tape and new work tape respectively. It is straightforward to show that f_M is a function if M is deterministic.

Example 4.4. The function corresponding to M_3 is:

$$f_{M_3}(w_i, w) = \begin{cases} (0, ww_i\#) & \text{if } ww_i \text{ contains an odd number of } a\text{'s} \\ (1, ww_i\#) & \text{if } ww_i \text{ contains an even number of } a\text{'s} \end{cases}$$

4.2 Interaction streams

PTMs take elements from streams as input and produce their output accordingly. A stream is defined coinductively over a set. Each stream contains an element and another stream. Goldin et al. use this definition in [6] to use coinduction as a proof technique.

Definition 4.5. Let $\mathcal{A}$ be a recursively enumerable set of action tokens. The stream $\mathbb{S}_{\mathcal{A}}$ is the class of streams over $\mathcal{A}$ defined as

$$\mathbb{S}_{\mathcal{A}} = \mathcal{A} \times \mathbb{S}_{\mathcal{A}}$$

.

We are going to use streams over $\mathcal{A} = \Gamma \times (\Gamma \cup \{\mu\})$. This coninductive definition of streams, in particular $\mathbb{S}_{\Gamma\times(\Gamma\cup\{\mu\})}$, yields elements that look like $((w_i, w_o), \sigma)$ where $\sigma \in \mathbb{S}_{\Gamma\times(\Gamma\cup\{\mu\})}$.

Definition 4.6. The persistent stream language of a PTM M in state w is defined as:

$$PSL(M, w) = \{((w_i, w_o), \sigma) \in \mathbb{S}_{\Gamma\times(\Gamma\cup\{\mu\})} | \exists_{w'\in\Gamma_\square{}^*}, w \xrightarrow{w_i/w_o} w' \wedge \sigma \in PSL(M, w')\}.$$

Example 4.7. Consider example 4.2. Assume that M_3 receives the input stream $(aaa, aaa, \dots)$. The persistent stream language $PSL(M_3)$ contains the stream $((aaa, 0), (aaa, 1), (aaa, 0), (aaa, 1), \dots)$.

4.3 Transition system of a PTM

We reuse Definition 2.5 to define the transition system of a PTM (also called an interactive transition system in [6]).

Definition 4.8. The transition system $\mathcal{T}_{PTM}(M) = (\mathcal{S}, \mathbb{A}, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ associated with a persistent Turing machine $M = (Q, \Gamma_{\square}, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ is an $\mathbb{A}$-labelled transition system such that:

$\mathcal{S} \subseteq \Gamma^* \cup \{s_\infty\}$ is the set of all states of M;

$\mathbb{A} = \Gamma \times (\Gamma \cup \{\mu\})$ is the set of labels;

$\xrightarrow{\mathbb{A}} \subseteq \mathcal{S} \times \mathbb{A} \times \mathcal{S}$;

$s_{\mathcal{I}} \in \mathcal{S} \setminus \{s_\infty\}$ is the initial state;

$s_{\mathcal{F}} = \emptyset$.

Example 4.9. The transition system $\mathcal{T}_{PTM}(M_3) = (\mathcal{S}, \mathbb{A}, \xrightarrow{\mathbb{A}}, s_{\mathcal{I}}, s_{\mathcal{F}})$ is defined as follows:

$\mathcal{S} = \Gamma^* = \{\epsilon, a, aa, \dots\}$

$\xrightarrow{\mathbb{A}} = \{(w, w_i, w_o, w') \in (\Gamma^*)^4 | w = a^m \wedge w_i = a^n \Rightarrow w_o = (m+n) \bmod 2 \wedge w' = a^{m+n}\}$

$s_{\mathcal{I}} = \epsilon$

Figure 5 is a graphical representation of a small part of this transition system. Only the transitions from the initial state are depicted, but from any state a^i there is a transition to a^j with $i < j$.

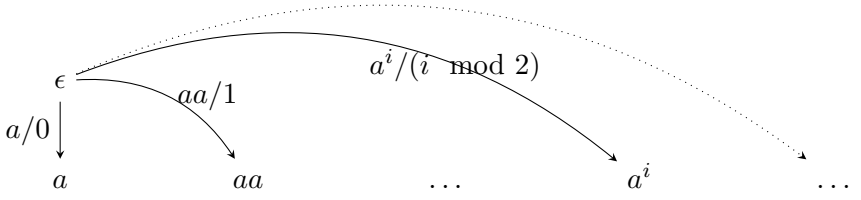

Figure 5: $\mathcal{T}_{PTM}(M_3)$

5 Behavioural equivalence

In the previous sections we discussed models that shared the goal of modelling interaction. Though RTMs clearly differ from PTMs in low level aspects, they become comparable when we talk about transition systems. Therefore we will study the process theoretic equivalence.

5.1 Equivalence relations

The goal of this thesis is to compare RTMs to PTMs and for that we need a notion of equivalence. In Definition 2.5 we defined a general transition system. We will now introduce the equivalence relations over those transition systems. For this we will assume two transition systems $T_1 = (\mathcal{S}_1, \mathbb{A}_1, \xrightarrow{\mathbb{A}}_1, s_{\mathcal{I}1}, s_{\mathcal{F}1})$ and $T_2 = (\mathcal{S}_2, \mathbb{A}_2, \xrightarrow{\mathbb{A}}_2, s_{\mathcal{I}2}, s_{\mathcal{F}2})$.

5.1.1 Isomorphism

The most straightforward relation holds when the structure of T_1 and T_2 are identical and their labelled transitions are equal.

Definition 5.1. Two transition systems T_1 and T_2 are *isomorphic* if there exists a bijective function f such that:

- $f(s_{\mathcal{I}1}) = f(s_{\mathcal{I}2})$ and
- $\forall_{a \in \mathbb{A}_1}, \forall_{s,s' \in \mathcal{S}_1}, s \xrightarrow{a}_1 s'$ if and only if $f(s) \xrightarrow{a}_2 f(s')$.

In [6] Goldin et al. prove that a PTM M is macrostep equivalent to M' if $\mathcal{T}_{PTM}(M)$ is isomorphic to $\mathcal{T}_{PTM}(M')$. We will not use this relation since it is too restrictive to be useful for our comparison.

5.1.2 Strong bisimulation

Definition 5.2. A *strong bisimulation* from T_1 to T_2 is a relation $\mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that:

- if $s_1 \mathcal{R} s_2$ and $s_1 \xrightarrow{a}_1 s_1'$ then $\exists_{s_2' \in \mathcal{S}_2}, s_2 \xrightarrow{a}_2 s_2'$ and $s_1' \mathcal{R} s_2'$;
- if $s_1 \mathcal{R} s_2$ and $s_2 \xrightarrow{a}_2 s_2'$ then $\exists_{s_1' \in \mathcal{S}_1}, s_1 \xrightarrow{a}_1 s_1'$ and $s_1' \mathcal{R} s_2'$;

T_1 and T_2 are *bisimilar* if there exists a strong bisimulation such that $s_{\mathcal{I}1} \mathcal{R} s_{\mathcal{I}2}$. We denote (strong) bisimilarity by $T_1 \underline{\leftrightarrow} T_2$.

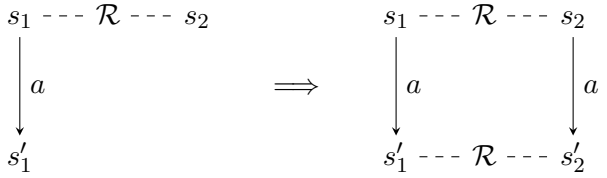

Figure 6: (Strong) bisimilarity

5.1.3 Weak bisimulation

Because of internal computations (τ-transitions) many transition systems are not bisimilar. The following equivalence relation ignores τ-transitions. The double arrow denotes the reflexive-transitive closure of a labelled transition (zero or more transitions).

Definition 5.3. A *weak bisimulation* from T_1 to T_2 is a relation $\mathcal{R}_w \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that:

- $s_{\mathcal{I}1}\mathcal{R}s_{\mathcal{I}2}$
- if $s_1\mathcal{R}_w s_2$ and $s_1 \xrightarrow{\tau} s_1'$ then $\exists_{s_2' \in \mathcal{S}_2}, s_2 \xrightarrow{\tau}{}^* s_2'$ and $s_1'\mathcal{R}_w s_2'$
- if $s_1\mathcal{R}_w s_2$ and $s_2 \xrightarrow{\tau} s_2'$ then $\exists_{s_1' \in \mathcal{S}_1}, s_1 \xrightarrow{\tau}{}^* s_1'$ and $s_1'\mathcal{R}_w s_2'$
- if $s_1\mathcal{R}_w s_2$ and $s_1 \xrightarrow{a} s_1'$ then $\exists_{s_2' \in \mathcal{S}_2}, s_2 \xrightarrow{\tau}{}^* \xrightarrow{a} \xrightarrow{\tau}{}^* s_2'$ and $s_1'\mathcal{R}_w s_2'$
- if $s_1\mathcal{R}_w s_2$ and $s_2 \xrightarrow{a} s_2'$ then $\exists_{s_1' \in \mathcal{S}_1}, s_1 \xrightarrow{\tau}{}^* \xrightarrow{a} \xrightarrow{\tau}{}^* s_1'$ and $s_1'\mathcal{R}_w s_2'$

T_1 and T_2 are *weakly bisimilar* if there exists a weak bisimulation such that $s_{\mathcal{I}1}\mathcal{R}_w s_{\mathcal{I}2}$. We denote weak bisimilarity by $T_1 \underline{\leftrightarrow}_w T_2$.

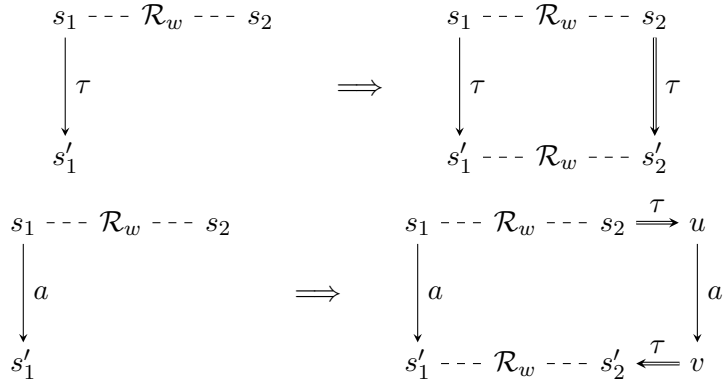

Figure 7: Weak bisimilarity

In Figure 7 we can see a visual representation of weak bisimulation. If $T_1 \underline{\leftrightarrow} T_2$, then $T_1 \underline{\leftrightarrow}_w T_2$. In the definition of weak bisimulation a special case holds where it takes exactly zero τ-transitions from s_2 to u (thus $s_2 = u$) and from v to s_2' (thus $s_2' = v$).

5.1.4 Branching bisimulation

Definition 5.4. A *branching bisimulation* from T_1 to T_2 is a relation $\mathcal{R}_b \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that:

- $s_{\mathcal{I}1} \mathcal{R} s_{\mathcal{I}2}$
- if $s_1 \mathcal{R}_w s_2$ and $s_1 \xrightarrow{a} s_1'$ then $(\exists_{s_2', s_2'' \in \mathcal{S}_2}, s_2 \xrightarrow{\tau}{}^* s_2'' \xrightarrow{a})$ or $(\exists_{s_2'}, s_2 \xrightarrow{\tau} s_2'$ and $s_2' \xrightarrow{\tau} s_2')$ and $s_1' \mathcal{R}_w s_2'$
- if $s_1 \mathcal{R}_w s_2$ and $s_1 \xrightarrow{a} s_1'$ then $(\exists_{s_2', s_2'' \in \mathcal{S}_2}, s_2 \xrightarrow{\tau}{}^* s_2'' \xrightarrow{a})$ or $(\exists_{s_2'}, s_2 \xrightarrow{\tau} s_2'$ and $s_2' \xrightarrow{\tau} s_2')$ and $s_1' \mathcal{R}_w s_2'$
- if $s_1 \mathcal{R}_w s_2$ and $s_1 \xrightarrow{a} s_1'$ then $\exists_{s_2' \in \mathcal{S}_2}, s_2 \xrightarrow{\tau}{}^* \xrightarrow{a} \xrightarrow{\tau}{}^* s_2'$ and $s_1' \mathcal{R}_w s_2'$
- if $s_1 \mathcal{R}_w s_2$ and $s_2 \xrightarrow{a} s_2'$ then $\exists_{s_1' \in \mathcal{S}_1}, s_1 \xrightarrow{\tau}{}^* \xrightarrow{a} \xrightarrow{\tau}{}^* s_1'$ and $s_1' \mathcal{R}_w s_2'$

T_1 and T_2 are *branching bisimilar* if there exists a branching bisimulation such that $s_{\mathcal{I}1} \mathcal{R}_b s_{\mathcal{I}2}$. We denote branching bisimilarity by $T_1 \underline{\leftrightarrow}_b T_2$.

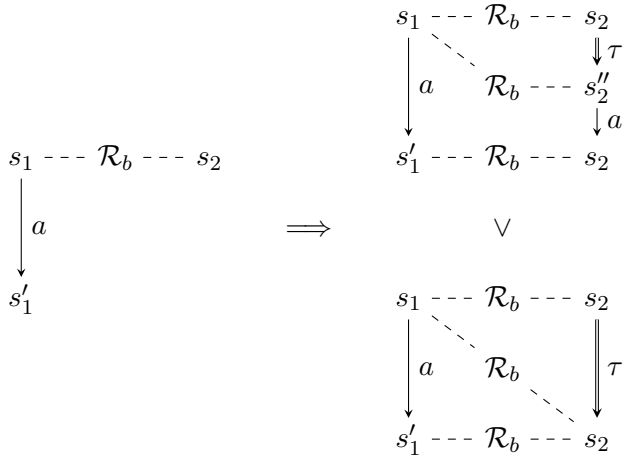

Figure 8: Branching bisimilarity

5.1.5 Divergence preserving branching bisimulation

Definition 5.5. A *divergence preserving branching bisimulation* from T_1 to T_2 is a relation $\mathcal{R}_b^\Delta \subseteq \mathcal{S}_1 \times \mathcal{S}_2$ such that:

- $\mathcal{R}_b^\Delta$ is a branching bisimulation $\mathcal{R}_b$;
- if there exists an infinite sequence $\{s_{1,i} \in \mathcal{S}_1 | i \in \mathbb{N}\}$ such that $\forall_i s_{1,i} \xrightarrow{\tau}_1 s_{1,i+1}$, then $\exists_{s_2'}, s_2 \xrightarrow{\tau}_2^+ s_2'$ and $s_{1,i} \mathcal{R}_b^\Delta s_2'$for some i;
- if there exists an infinite sequence $\{s_{1,i} \in \mathcal{S}_1 | i \in \mathbb{N}\}$ such that $\forall_i s_{1,i} \xrightarrow{\tau}_1 s_{1,i+1}$, then $\exists_{s_2'}, s_2 \xrightarrow{\tau}_2^+ s_2'$ and $s_{1,i} \mathcal{R}_b^\Delta s_2'$ for some i.

T_1 and T_2 are *divergence preserving branching bisimilar* if there exists a divergence preserving branching bisimulation such that $s_{\mathcal{I}1} \mathcal{R}_b^\Delta s_{\mathcal{I}2}$. We denote this bisimilarity by $T_1 \underline{\leftrightarrow}_b^\Delta T_2$.

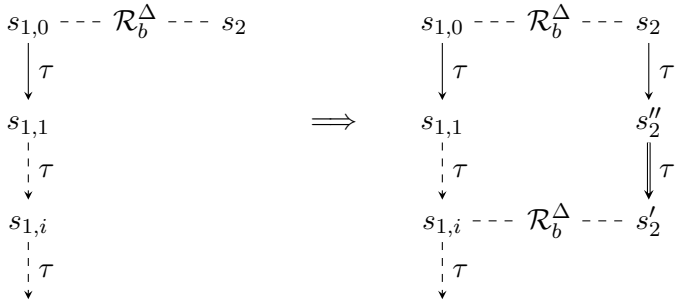

Figure 9: Divergence preserving branching bisimilarity

5.2 Expressiveness comparison

This subsection includes the expressiveness results from [2] and [6]. In the preliminaries we defined effective and computable transition systems.

We introduced the machines in Figure 1, 3 and 4 and to behave in such a way that $\mathcal{T}_{RTM}([M_1||M_2]_{\{i\}})$ is comparable to $\mathcal{T}_{PTM}(M_3)$. Intuitively M_2 does the same as M_3, though the corresponding transition systems in Example 3.7 and Example 4.9 are not ((divergence preserving) branching) bisimilar. There is a difference between intuitive equivalence and formal equivalence. In the following example we create an RTM for an existing PTM.

Example 5.6. Let $M_{AM} \in \mathbb{PTM}$ be the answering machine that Goldin et al. used as an example in [6]. The machine can receive a command '*record*' with a parameter Y that M_{AM} records on the work tape. The command

'*play*' puts the entire work tape on the output tape and finally the command '*erase*' removes all content from the work tape.

$$\begin{aligned} f_{AM}(record\ Y, X) &= (ok, XY) \\ f_{AM}(play, X) &= (X, X) \\ f_{AM}(erase, X) &= (done, \epsilon) \end{aligned}$$

It is easy to define an RTM M'_{AM}that acts like this. Let $\{record, play, erase, ok, done\} \subset \Gamma$ and let i and o be the input and output channel respectively. Then M'_{AM} has three possible outgoing transitions from $q_{\mathcal{I}}$, one for each of the symbols {record, play, erase}. The internal computations that M'_{AM} has to do for each command resemble those of M_{AM}. The way that input and output is treated, is once again different as RTMs can output one symbol for each state transition and PTMs can only output an entire output tape in each macrostep.

While the intuition for transforming RTMs into PTMs and reversed is simple, we need to know whether this is possible for all cases. This is not easy since Baeten et al. stick to process theory and Goldin et al. were more interested in dynamic stream semantics and interaction streams associated with PTMs. We will continue to discuss the process theoretic notions of interaction.

Theorem 5.7. Let M be an RTM. Then the transition system $\mathcal{T}_{RTM}(M)$ is computable.

Baeten et al. gave the essence of the proof in [2]. We will give the formal details to illustrate the meaning of computable transition systems.

Proof. Assume that $M = (Q, \Gamma_\Box, \mathcal{C}, \mathcal{A}_\tau, \rightarrow, q_{\mathcal{I}}, Q_{\mathcal{F}})$ is an RTM. Let M be in configuration (q, δ_L). Then we can compute the set of outgoing transitions $out(q, \delta)$ by using Definition 3.3. We know that $\rightarrow$ is finite, thus $out(s)$ is finite for all configurations (q, δ). Then the function out is also recursive.

We can compute the set of final configurations $s_{\mathcal{F}} = \{(q, \delta) | q \in Q_{\mathcal{F}} \wedge \delta \in L_\delta\}$. We know that $Q_{\mathcal{F}}$ is a finite set. Then there is an algorithm that decides whether a configuration is final, thus $s_{\mathcal{F}}$ is a recursive set. □

One of the main results in [2] is that these exists a simulator that can take the coding of a computable transition system as an input to simulate it up to divergence preserving bisimilarity.

Theorem 5.8. Let T be a computable transition system. Then there exists an RTM Sim such that $T \underline{\leftrightarrow}_b^\Delta \mathcal{T}_{RTM}(Sim)$

The question at hand is whether RTMs and PTMs can simulate each other's behaviour up to some equivalence relation. Baeten et al. proved that RTMs can simulate PTMs up to branching bisimilarity. The proof

transforms the transition system of a PTM into an effective transition system, which can only be simulated up to branching bisimilarity instead of divergence preserving bisimilarity if the transition system of a PTM would be computable.

The transition system of a PTM is not always computable since PTMs allow infinite branching. Example 4.9 is infinitely branching for every state including the initial state. The transition system of an RTM is computable because it relies on the definition of an RTM containing finite sets. A PTM is also composed of finite sets, but the abstraction into macrosteps causes many internal computations completely disappear in the transition system, resulting in infinite branching. τ-transitions that we don't see in its corresponding transition system.

6 Divergence

In this section we discuss the behaviours of RTMs and PTMs once they diverge. In process theory divergence is defined as an infinite path of τ-transitions. We also saw in Section 4 that divergence in a PTM is denoted by $w \xrightarrow{w_i/\mu} s_\infty$ and for all $w_i \in \Gamma_\square{}^*, s_\infty \xrightarrow{w_i/\mu} s_\infty$. Intuitively these notions of divergence capture the same infinite path of externally unobservable computations. The following examples illustrate two instances of both models that behave in a comparable way.

Example 6.1. Let RTM_{div} be the RTM with 3 states as depicted in Figure 10. Once the machine reaches state q_2 it won't be able to communicate with the environment anymore since there is no transition that allows it. The corresponding transition system is depicted in Figure 11.

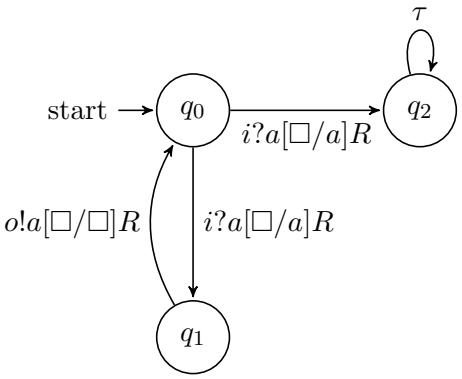

Figure 10: RTM_{div}, a nondeterministic RTM that has the possibility to diverge.

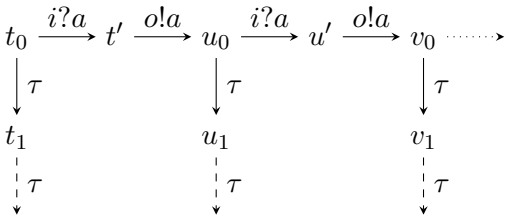

Figure 11: $\mathcal{T}_{RTM}(RTM_{div})$

Similarly we have the PTM PTM_{div} that does the same as RTM_{div}. The transition system of this machine is depicted in Figure 12.

PTMs have a special divergence state that cannot be escaped once reached. We assumed in Section 4 that the final state of a PTM marks

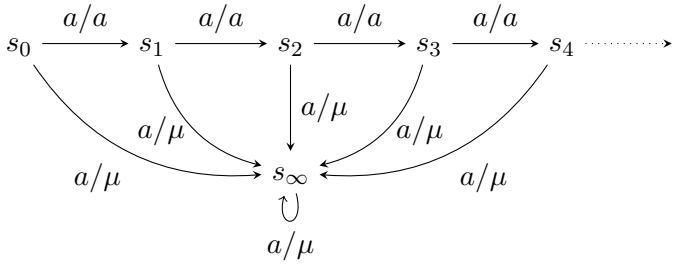

Figure 12: $\mathcal{T}_{PTM}(PTM_{div})$

the end of a macrostep. This assumption makes the transition from s_∞ to s_∞ for all inputs unnecessary. Also if one creates a divergence state, the transition system can never be computable since we can never decide whether a PTM diverges. The divergence state s_∞ doesn't exist. Since we cannot even observe internal computations of a PTM in transition systems, we cannot reason about divergence preserving branching bisimilarity between both models.

7 Universality

In the preliminaries we discussed the universal TM that Turing used to prove the undecidability of the halting problem. In [2] and [6] the universality of both RTMs and PTMs is discussed. In this section we compare $\mathcal{U}_P$ and $\mathcal{U}_R$, the universal machines that simulate PTMs and RTMs respectively.

7.1 Universal reactive Turing machine

Baeten et al. specified that an RTM initialises in configuration $(q_{\mathcal{I}}, \check{\Box})$, thus an RTM cannot start with the code of some other machine on the tape. Thus the universal RTM needs another machine $\overline{M}$ with the purpose of sending the code $\lceil M \rceil$ along communication channel u. The RTM $\mathcal{U}_R$ is defined as follows.

Definition 7.1. A universal RTM $\mathcal{U}_R$ is an RTM such that:

$$\forall_{M \in \mathbb{RTM}}, [\mathcal{U}_R || \overline{M}]_{\{u\}} \leftrightarrows_b \mathcal{T}_{RTM}(M)$$

where the machine $\overline{M}$ is the RTM that only sends $\lceil M \rceil$ over communication channel u.

Unfortunately $\mathcal{U}_R$ does not exist when we demand divergence preserving branching bisimilarity because an RTM has a fixed maximum number of transitions (see Definition 2.7). If $\mathcal{U}_R$ has branching degree B, then one can always find a machine M with branching degree $B + 1$ that cannot be simulated by $\mathcal{U}_R$. Baeten et al. proved this in [2]. They also proved the following result.

Theorem 7.2. For every $B \in \mathbb{N}$ there exists an RTM $\mathcal{U}_R^B$ such that:

$\forall_{M \in \mathbb{RTM}}$ if M has a branching degree $B' \leq B$ then $[\mathcal{U}_R^B || \overline{M}]_{\{u\}} \leftrightarrows_b^{\Delta} \mathcal{T}_{RTM}(M)$

A universal RTM $\mathcal{U}_R$ can only simulate up to divergence preserving branching bisimilarity up to a fixed branching degree.

7.2 Universal persistent Turing machine

The universal PTM is also limited by the fact that all behaviour is influenced by interaction. Therefore it is not allowed to have content on the work tape initially. This is not a problem though because the code of a PTM M can be put on the input tape in its initial macrostep. $\mathcal{U}_P$ is defined as follows.

Definition 7.3. A universal PTM $\mathcal{U}_P$ is a PTM such that for all $M \in \mathbb{PTM}$:

- there is an initializing macrostep $\epsilon \xrightarrow{\lceil M \rceil, \lceil w \rceil / \epsilon} \lceil M \rceil, \lceil w \rceil$;

- if M has a halting computation $w \xrightarrow{w_i/w_o} w'$ then $\mathcal{U}_P$ has a halting computation $\lceil M \rceil, \lceil w \rceil \xrightarrow{\lceil w_i \rceil / \lceil w_o \rceil} \lceil M \rceil, \lceil w' \rceil$ and
- if M in state w diverges ($w \xrightarrow{w_i/\mu} s_\infty$) then $\mathcal{U}_P$ also diverges ($\lceil M \rceil \lceil w \rceil \xrightarrow{w_i/\mu} s_\infty$).

7.3 Comparison

Clearly Baeten et al. and Goldin et al. discuss different notions of universality. The universal PTM simulates an arbitrary PTM up to macrostep equivalence (modulo the first macrostep) while the universal RTM simulates an arbitrary RTM up to branching bisimilarity and up to divergence preserving branching bisimilarity for a fixed branching degree. Goldin et al. showed in [6] that two isomorphic transition systems are also bisimilar and that two PTMs are macrostep equivalent if and only if their transition systems are isomorphic. Thus a universal PTM can also simulate an arbitrary PTM up to strong bisimulation. This difference is due to the possibility of infinite branching of PTMs that we discussed in Section 5.

8 Related Work

In this section we will look at other notions of modelling computability and attempts at introducing interaction to them.

The λ-calculus [3], is a model of computation based on substitution. When Turing published his article [9] in 1936, Alonzo Church also published an article about the λ-calculus. Both Turing's machines and the lambda calculus are able to capture the computable functions.

The π-calculus is a process calculus that is based on the λ-calculus. Similar to the models that we discussed in this thesis, the π-calculus is about communication and interaction instead of only the computable functions. For the π-calculus it also holds that any computable term in the λ-calculus can be expressed in the π-calculus. The models that we compared in this paper are similar in a sense that any Turing machine can be simulated by an RTM (only allowing τ-transitions) and a PTM (erasing the contents of the work tape after a macrostep, removing persistence). Bas Luttik, one of the writers of [2], compared the π-calculus to the notion of reactive Turing machines in [7] together with Fei Yang. The result of this comparison is that any executable behaviour (that according to [2] can be simulated by an RTM) can also be specified by the π-calculus, while the converse is not true.

Interaction is not the only extension to the Turing machine model. There are works of Turing machines that have access to an advice function and also TMs that have access to an all-knowing oracle machine. Van Leeuwen and Wiedermann propose interactive Turing machines in [10]. They also discuss non-uniform evaluation as it takes place in distributed systems. The underlaying hardware of a system may partially change during the execution and it is not desirable to stop the execution of important systems such as those of banks. TMs are also limited in the sense that the entire machine must be defined before execution and so are RTMs and PTMs.

9 Conclusions

Reactive Turing machines and persistent Turing machines are two models proposed in [2] and [6] respectively. We discussed the formal definitions of RTMs and PTMs and provided several examples. This showed the intiution of the differences between both models.

In Section 5 we discussed the formal equivalence relations that are used in both articles to compare instances of the models to each other. These relations cannot be used to compare RTMs to PTMs directly, but Baeten et al. showed that the transition systems of PTMs can be transformed into an effective transition system. Since RTMs can simulate any effective transition system up to branching bisimilarity, any PTM's transition system can be simulated by an RTM up to branching bisimilarity. PTMs do not have the property of their transition system being computable because of the microsteps that don't appear in the transition system.

In Section 6 we briefly discussed divergence in both models. Divergence preserving branching bisimulation is an equivalence relation over transition systems that can be used to reason about divergence. A transition system diverges if it performs an infinite number of internal computations. RTMs have τ-transitions to indicate internal steps while PTMs have a special divergence state. We saw that PTMs don't provide a way of reasoning about divergence since transitions to s_∞ can never be computable. There are results about divergence preserving branching bisimilarity for RTMs, which are nice to have.

Section 7 is about the universal machines that are proposed along with each model. The universal PTM can simulate any PTM up to macrostep equivalence, thus also strong bisimilarity. This is also due to the ability of PTMs to yield an infinitely branching transition system. The universal RTM is limited to a finitely branching transition system.

For these reasons the notion of reactive Turing machine is more suitable. PTMs abstract from all internal computation with the introduction of macrosteps. The examples that we discussed provided an insight for this main difference.

Discussion

RTMs and PTMs clearly describe a notion of interaction. The set of computable functions can be captured by Turing machines, the λ-calculus, primitive recursion and so on. The Church-Turing thesis states that they are equally powerful, but the concept of interaction is still vague. The expressiveness of RTMs, in particular with respect to computable transition systems, uses the notion of recursive functions. Since this notion provides a strict limit to what is possible to compute, the notion of RTM is a suitable model for interaction.

Acknowledgements

I would like to thank Herman Geuvers for introducing me to this subject and for providing help, and very useful and detailed criticism. To my fellow board members of Olympus I want to express my regrets for not being able to fulfil all of my duties because of the amount of work that I required for this bachelor thesis. They were very understanding in this, for which I want to thank them.

References

[1] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. On the consistency of koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1):129–176, 1987.

[2] Jos CM Baeten, Bas Luttik, and Paul Van Tilburg. Reactive Turing machines. *Information and Computation*, 231:143–166, 2013.

[3] Hendrik Pieter Barendregt. *The lambda calculus, its syntax and semantics*, volume 3. North-Holland Amsterdam, 1984.

[4] Dina Goldin and Peter Wegner. Persistence as a form of interaction. 1998.

[5] Dina Q Goldin. Persistent Turing machines as a model of interactive computation. In *Foundations of information and knowledge systems*, pages 116–135. Springer, 2000.

[6] Dina Q Goldin, Scott A Smolka, Paul C Attie, and Elaine L Sonderegger. Turing machines, transition systems, and interaction. *Information and Computation*, 194(2):101–128, 2004.

[7] Bas Luttik and Fei Yang. Executable behaviour and the π-calculus. *CoRR*, abs/1410.4512, 2014.

[8] Thomas A. Sudkamp. *Languages and Machines: An Introduction to the Theory of Computer Science (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[9] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936.

[10] Jan van Leeuwen and Jiří Wiedermann. Beyond the Turing limit: Evolving interactive systems. In *SOFSEM 2001: Theory and Practice of Informatics*, pages 90–109. Springer, 2001.

[11] Peter Wegner. Why interaction is more powerful than algorithms. *Commun. ACM*, 40(5):80–91, May 1997.