On the Expressiveness of Reactive Turing Machines and ACP$_\tau$

Last update: 1 July 2010

Bas Luttik and Paul van Tilburg

Division of Computer Science, Eindhoven University of Technology,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands,
e-mail: {s.p.luttik,p.j.a.v.tilburg}@tue.nl

Abstract. We consider the expressiveness of reactive Turing machines that extend classical Turing machines with a process-theoretical notion of interaction. We prove that every effective transition system is simulated up to branching bisimilarity by an reactive Turing machine, and that every deterministic computable transition system is simulated up to branching bisimilarity with explicit divergence by an reactive Turing machine. We also establish a correspondence between reactive Turing machines and the process theory ACP$_\tau$, proving that the transition system associated with an reactive Turing machine is definable in ACP$_\tau$.

1 Introduction

The Turing machine [15] is widely accepted as a computational model suitable for exploring the theoretical boundaries of computing. Motivated by the existence of universal Turing machines, many textbooks on the theory of computation (e.g., [13,10]) present the Turing machine as an accurate theoretical model of the computer. In fact, Sipser even goes as far as writing that “[a] Turing machine can do everything a real computer can do.” [13] This statement is sometimes referred to as the strong Church-Turing thesis, as opposed to the normal Church-Turing thesis according to which every effectively calculable function is computable by a Turing machine.

There is a limitation to viewing the Turing machine as a model of a computer. A Turing machine operates from the assumptions that: (1) all the input it needs for the computation is available on the tape from the very beginning; (2) it performs a terminating computation; and (3) it leaves the output on the tape at the very end. That is, a Turing machine computes a function, and thus it abstracts from two key ingredients of contemporary computing: interaction and non-termination. Nowadays, most computing systems are so-called reactive systems [9], systems that are generally not supposed to terminate and typically consist of a number of computing devices that interact with each other and with their environment. A reactive system often unremittingly depends on input, and unremittingly produces output.
Towards the end of the 1970s, Milner observed that, for a thorough investigation of interaction and concurrency, it is better to study these notions in isolation rather than to try and add them to any of the existing models of computation. One of his desiderata for the design of CCS was “that there be only a single combinator for combining processes which interact or which coexist” [11]. In particular, also the interaction of a computing device with its memory can (and should) be modelled using a symmetric notion of interaction, modelling the memory as a separate process. It was probably this desideratum of Milner that caused the theory of algorithmic computation and the theory of interaction and concurrency to develop largely separately.

In 1997, Peter Wegner questioned in [19] the validity of the strong Church-Turing thesis, claiming that interaction makes computing systems more powerful than Turing machines. Since then there has been an ongoing philosophical debate in the literature as to whether the Turing machine model should still be considered an accurate model for exploring the boundaries of computing. To get a formal handle on the matter, several extensions of Turing machines have been proposed [5,8,18], each facilitating some form of interactive (sequential) computation. Our stance is that these attempts do not take full advantage of the results of concurrency theory, and do not give the notion of interaction the status it deserves. Interaction is only added to the extent that it may have a beneficial effect on computational power, e.g., by allowing an algorithm to query its environment, or by assuming that the environment periodically writes a write-only input tape and reads a read-only output tape of a Turing machine. Thus, the focus remains fully on the computational aspect, and interaction is treated as a second-class citizen.

Recently, we proposed in [3] an extension of the notion of Turing machine that facilitates interaction in a concurrency-theoretic style. The idea is very simple: we add an action to every transition of the Turing machine, which may either refer to an observable event, or to an unobservable event, in which case we write \( \tau \). The extended Turing machine is called a reactive Turing machine (RTM), and to fully tie in with concurrency theory, we give it operational semantics by associating a transition system with it.

In this paper we investigate the expressiveness of RTMs. Our first result, obtained in Section 4, states that every effective transition system (i.e., every transition system with an r.e. transition relation) is branching bisimilar with the transition system associated with an RTM. This is achieved by defining for every transition system an RTM that simulates it. Given a state \( s \) of the transition system, it nondeterministically guesses a pair of an action \( a \) and a next state \( t \) and verifies if the pair gives rise to a valid transition from \( s \). The verification may not terminate on invalid transitions, and therefore we need to include an abort-mechanism to escape non-terminating computations. The abort-mechanism allows the simulator to restart itself, and thus introduces an extra \( \tau \)-loop into the simulation. If the transition relation is deterministic and computable, then the set of outgoing transitions of a state can actually be recursively generated. Our
second result, obtained in Section 5 states that, in this case, introducing \( \tau \)-loops can be avoided.

Through its transition system semantics a reactive Turing machine ties in with concurrency theory. But it does not yet fully realise Milner’s desideratum that all interaction should be modelled using one symmetric notion of interaction: the interaction between the finite control and the tape memory is still implicit. Our third result, obtained in Section 6, realises Milner’s desideratum. We prove that every reactive Turing machine can be modelled as a regular (i.e., finite-state) process interacting with a process modelling the tape memory. Both will be specified by a finite specification in the process theory ACP\( \tau \) [4].

As corollaries to the combined results of this paper we can conclude that every effective transition system can be specified up to branching bisimilarity in ACP\( \tau \), and every deterministic computable transition system can be specified up to branching bisimilarity with explicit divergence in ACP\( \tau \). These results extend and improve the result of [2] to the effect that every finitely branching computable transition system can be specified with a finite ACP\( \tau \)-specification in several aspects: (1) we allow infinitely branching transition systems with a recursively enumerable transition relation; (2) we allow intermediate termination in our transition system; and (3) we establish a stronger behavioural equivalence in case the transition system is deterministic and computable. A further interesting aspect of our solution, compared to that in [2], is that we model the tape memory using a queue, rather than two stacks, which is essential in our setting with intermediate termination.

2 Effective and computable transition systems

We fix a finite set \( A \) of action symbols that we shall use to denote the observable events of a system. An unobservable event will be denoted with \( \tau \), assuming that \( \tau \not\in A \); we shall henceforth denote the set \( A \cup \{ \tau \} \) by \( A_\tau \). Throughout this this paper we fix a bijection \( \llbracket \cdot \rrbracket : A_\tau \rightarrow \{ 1, \ldots, |A_\tau| \} ; \llbracket a \rrbracket \) is the code of \( a \), and we denote the action with code \( i \) by \( a_i \).

An \( A \)-labelled labelled transition system \( L \) is a four-tuple \((S, \rightarrow, \uparrow, \downarrow)\) consisting of a set of states \( S \), an initial state \( \uparrow \in S \), a subset \( \downarrow \subseteq S \) of final states, and an \( A_\tau \)-labelled transition relation \( \rightarrow \subseteq S \times A_\tau \times S \). If \((s,a,t) \in \rightarrow \), we write \( s \xrightarrow{a} t \). If \( s \) is a final state, i.e., \( s \in \downarrow \), we write \( s \downarrow \).

A transition system is countable if its set of states is countable. We proceed to define when a transition system is effective [7]. Let \( L = (S, \rightarrow, \uparrow, \downarrow) \) be a (countable) transition system. By a coding of \( L \) we understand an injective mapping \( \llbracket \cdot \rrbracket : S \rightarrow \mathbb{N} ; \llbracket s \rrbracket \) is the code of \( s \in S \). For an arbitrary subset \( S' \) of \( S \) we shall write \( \llbracket S' \rrbracket \) to denote the set of all codes of states in \( S' \); so, in particular, if \( \downarrow \subseteq S \) is the subset of final states of some transition system, then by \( \llbracket \downarrow \rrbracket \) we denote the set of all codes associated with these final states. To extend the codings of \( A_\tau \) and \( L \) to a coding of transition relations, we use recursive functions.
\( \pi : \mathbb{N} \times \mathbb{N} \to \mathbb{N}, \pi_1 : \mathbb{N} \to \mathbb{N}, \) and \( \pi_2 : \mathbb{N} \to \mathbb{N} \) such that, for all \( x, y, z \in \mathbb{N} \),
\[
\begin{align*}
\pi_1(\pi(x, y)) &= x, \\
\pi_2(\pi(x, y)) &= y, \quad \text{and} \\
\pi(\pi_1(z), \pi_2(z)) &= z.
\end{align*}
\]

(For a proof of the existence of such recursive functions, see, e.g., [12, §5.3].)

Now, if \( \to \) is an \( \mathcal{A}_\tau \)-labelled transition relation on \( \mathcal{S} \), then we define \( \to^\gamma \) by
\[
\to^\gamma = \{ \pi(\pi^\gamma s^\gamma, \pi(\pi^\gamma a^\gamma, \pi^\gamma t^\gamma)) \mid (s, a, t) \in \to \}.
\]

**Definition 1.** A transition system \( L = (\mathcal{S}, \to, \uparrow, \downarrow) \) is effective if there exists a coding \( \cdot^\gamma : \mathcal{S} \to \mathbb{N} \) of \( \mathcal{S} \) such that \( \cdot^\gamma \) is recursively enumerable.

A transition system \( L = (\mathcal{S}, \to, \uparrow, \downarrow) \) is finitely branching if, for every state \( s \in \mathcal{S} \), the set
\[
\text{out}(s) = \{(a, t) \mid s \xrightarrow{a} t\}
\]
is finite. If \( L \) is countable and finitely branching, and \( \cdot^\gamma : \mathcal{S} \to \mathcal{S} \) is a coding of \( L \), then
\[
\text{out}(s)^\gamma = \{ \pi(\pi^\gamma a^\gamma, \pi^\gamma t^\gamma) \mid s \xrightarrow{a} t\}
\]
is a finite set of natural numbers. We denote by \( ci(\text{out}(s)^\gamma) \) the canonical index of \( \text{out}(s)^\gamma \), i.e., a unique code associated with a finite set of natural numbers such that there exists a recursive function \( ci^{-1} : \mathbb{N} \to \mathbb{N}^* \) such that if \( X = \{x_1, \ldots, x_n\} \) with \( x_1 < \cdots < x_n \), then \( ci^{-1}(ci(X)) = \{x_1, \ldots, x_n\} \). In Sect. 5 we shall further assume about \( ci \) that it satisfies \( ci(X) \leq \sum_{k=1}^{\max X} 2^k \); note that, e.g., the definition of canonical index in [12, p. 70] satisfies this property.

**Definition 2.** A transition system \( L = (\mathcal{S}, \to, \uparrow, \downarrow) \) is computable if it is finitely branching and there exists a coding \( \cdot^\gamma : \mathcal{S} \to \mathbb{N} \) of \( \mathcal{S} \) such that the set \( \gamma^{-}\gamma \) is decidable and the (partial) function \( tr : \mathbb{N} \to \mathbb{N} \) defined by
\[
tr(\cdot^\gamma s) = ci(\text{out}(\cdot^\gamma s)^\gamma) \quad \text{for all states } s \in \mathcal{S}
\]
is recursive.

We proceed to define the behavioural equivalences that we shall employ in this paper to compare transition systems. Let \( \to \) be an \( \mathcal{A}_\tau \)-labelled transition relation on a set \( \mathcal{S} \), and let \( a \in \mathcal{A}_\tau \); we write \( s \xrightarrow{(a)} t \) if \( s \xrightarrow{a} t \) or \( a = \tau \) and \( s = t \). Furthermore, we write \( s \xrightarrow{} t \) if there exist \( s_0, \ldots, s_n \) such that \( s = s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n = t \).

**Definition 3.** Let \( L_1 = (\mathcal{S}_1, \to^1, \uparrow^1, \downarrow^1) \) and \( L_2 = (\mathcal{S}_2, \to^2, \uparrow^2, \downarrow^2) \) be transition systems. A branching bisimulation from \( L_1 \) to \( L_2 \) is a binary relation \( \mathcal{R} \subseteq \mathcal{S}_1 \times \mathcal{S}_2 \) such that \( \uparrow^1 \mathcal{R} \uparrow^2 \) and, for all states \( s_1 \) and \( s_2 \), \( s_1 \mathcal{R} s_2 \) implies
The transition systems $L_1$ and $L_2$ are branching bisimilar (notation: $L_1 \equiv_b L_2$) if there exists a branching bisimulation $R$ such that, for all states $s_1$ and $s_2$, $s_1 R s_2$ implies

1. if $s_1 \xrightarrow{a} s_1'$, then there exist $s_2', s_2'' \in S_2$ such that $s_2 \xrightarrow{a''} s_2'' \xrightarrow{(a)} s_2'$, $s_1 R s_2''$ and $s_1' R s_2'$;

2. if $s_2 \xrightarrow{a} s_2'$, then there exist $s_1', s_1'' \in S_1$ such that $s_1 \xrightarrow{a''} s_1'' \xrightarrow{(a)} s_1'$, $s_1' R s_2$ and $s_1' R s_2'$;

3. if $s_1 \downarrow_1$, then there exists $s_2'$ such that $s_2 \xrightarrow{2} s_2'$, $s_1 R s_2'$ and $s_2' \downarrow_2$; and

4. if $s_2 \downarrow_2$, then there exists $s_1'$ such that $s_1 \xrightarrow{1} s_1'$, $s_1' R s_2$ and $s_1' \downarrow_1$.

The transition systems $L_1$ and $L_2$ are branching bisimilar with explicit divergence (notation: $L_1 \equiv^\Delta_b L_2$) if there exists a branching bisimulation $R$ such that $L_1$ and $L_2$ are effective, and

5. 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 $s_1, i R s_2$ for all $i \in \mathbb{N}$, then there exists a state $s_2'$ such that $s_2 \xrightarrow{\tau} s_2'$ and $s_1, i R s_2'$ for some $i \in \mathbb{N}$; and

6. 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 $s_1 R s_{2,i}$ for all $i \in \mathbb{N}$, then there exists a state $s_1'$ such that $s_1 \xrightarrow{\tau} s_1'$ and $s_1' R s_{2,i}$ for some $i \in \mathbb{N}$.

The transition systems $L_1$ and $L_2$ are branching bisimilar with explicit divergence (notation: $L_1 \equiv^\Delta^\Delta_b L_2$) if there exists a branching bisimulation with explicit divergence from $L_1$ to $L_2$.

The notions of branching bisimilarity and branching bisimilarity with explicit divergence originate with [17]. The particular divergence conditions we use to define branching bisimulations with explicit divergence here are discussed in [16], where it is also proved that branching bisimilarity with explicit divergence is an equivalence.

In [6], Darondeau gives an example of an effective transition system that is (strongly) bisimilar to a transition system that is not effective. His example uses infinitely many actions; we present below an adaptation that needs only finitely many actions.

**Example 1.** Assume that $\mathcal{A}$ contains actions $\text{inc}$ and $\text{run}$ and consider the transition system $L_1 = (S_1, \rightarrow_1, \uparrow_1, \downarrow_1)$ with $S_1$, $\rightarrow_1$, $\uparrow_1$ and $\downarrow_1$ defined by

\[
S_1 = \{s_{1,x}, t_x \mid x \in \mathbb{N}\},
\]

\[
\rightarrow_1 = \{(s_{1,x}, \text{inc}, s_{x+1}) \mid x \in \mathbb{N}\} \cup \{(s_{1,x}, \text{run}, t_x) \mid x \in \mathbb{N}\},
\]

\[
\uparrow_1 = s_{1,0}, \text{ and}
\]

\[
\downarrow_1 = \{t_x \mid \varphi(x) \text{ converges}\}.
\]

(Here $\varphi$ denotes the partial recursive function with index $x$ in some exhaustive enumeration of partial recursive functions, see, e.g., [12].) Then both $\rightarrow_1$ and $\downarrow_1$ are easily seen to be recursively enumerable, so $L_1$ is effective.
Now consider the transition system \( L_2 = (S_2, \rightarrow_2, \uparrow_2, \downarrow_2) \) with \( S_2, \rightarrow_2, \uparrow_2, \downarrow_2 \) defined by

\[
S_2 = \{ s_{2,x} \mid x \in \mathbb{N} \} \cup \{ c, d \},
\]

\[
\rightarrow_2 = \{(s_{2,x}, inc, s_{2,x+1}) \mid x \in \mathbb{N} \}
\]

\[
\cup \{(s_{2,x}, run, c) \mid \varphi_x(x) \text{ converges} \}
\]

\[
\cup \{(s_{2,x}, run, d) \mid \varphi_x(x) \text{ diverges} \},
\]

\[
\uparrow_2 = s_0, \quad \text{and}
\]

\[
\downarrow_2 = \{ c \}.
\]

The transition system is not effective. To see this, suppose that \( \rightarrow_2 \) would be recursively enumerable. Then the partial decision procedure for \( \rightarrow_2 \) would yield a total decision procedure for deciding if \( \varphi_x(x) \) converges: execute the partial decision procedure for \( \rightarrow_2 \) simultaneously on \( s_x \xrightarrow{\text{run}}_2 c \) and \( s_x \xrightarrow{\text{run}}_2 d \); one of these executions will eventually terminate.

Despite the fact that \( L_1 \) is effective, while \( L_2 \) is not, these transition systems are branching bisimilar, as witnessed by the following binary relation

\[
R = \{(s_{1,x}, s_{2,x}) \mid x \in \mathbb{N} \}
\]

\[
\cup \{(t_x, c) \mid x \in \mathbb{N} \& \varphi_x(x) \text{ converges} \}
\]

\[
\cup \{(t_x, d) \mid x \in \mathbb{N} \& \varphi_x(x) \text{ diverges} \}.
\]

(To verify that \( R \) satisfies the conditions of Definition 3 is straightforward and left to the reader. In fact, the unobservable action \( \tau \) does not play a role in this example, so \( L_1 \) and \( L_2 \) are actually bisimilar in the strong sense.)

## 3 Reactive Turing Machines

In the previous section we have already fixed a finite set \( \mathcal{A}_\tau \) of action symbols, containing a special symbol \( \tau \) to denote unobservable activity. In addition, we now also fix a finite set \( \mathcal{D} \) of tape symbols. We add to \( \mathcal{D} \) a special symbol \( \square \) to denote a blank tape cell, assuming that \( \square \) is not already in \( \mathcal{D} \); we denote the set \( \mathcal{D} \cup \{\square\} \) by \( \mathcal{D}_\square \).

**Definition 4 (Reactive Turing machine).** An reactive Turing machine \((\text{RTM})\) \( M \) is defined as a quadruple \((S, \rightarrow, \uparrow, \downarrow)\) consisting of a finite set of states \( S \), a distinguished initial state \( \uparrow \in S \), a subset of final states \( \downarrow \subseteq S \), and a \((\mathcal{D}_\square \times \mathcal{A}_\tau \times \mathcal{D}_\square \times \{L,R\})\)-labelled transition relation

\[
\rightarrow \subseteq S \times \mathcal{D}_\square \times \mathcal{A}_\tau \times \mathcal{D}_\square \times \{L,R\} \times S.
\]

An RTM is deterministic if \((s,d,a,e_1,M_1,t_1) \in \rightarrow \) and \((s,d,a,e_2,M_2,t_2) \in \rightarrow \) implies \( e_1 = e_2, t_1 = t_2 \) and \( M_1 = M_2 \) for all \( s,t_1,t_2 \in S \), \( d,e_1,e_2 \in \mathcal{D}_\square \), \( a \in \mathcal{A}_\tau \), and \( M_1,M_2 \in \{L,R\} \).
If \((s, d, a, e, M, t) \in \rightarrow\), we write \(s \xrightarrow{d,a,c,M} t\). The intuitive meaning of such a transition is that whenever \(M\) is in state \(s\) and reads symbol \(d\) on the tape, 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 one position to the right on the tape (depending on whether \(M = L\) or \(M = R\)), and then continue in state \(t\).

To formalise this intuition, we associate below with every RTM \(M\) a transition system \(T(M)\). The states of \(T(M)\) are the configurations of the RTM, consisting of a state of the RTM, its tape contents, and the position of the read/write head on the tape. We represent the tape contents by an element of \(D^*\), replacing precisely one occurrence of a tape symbol \(d\) by a marked symbol \(\overline{d}\), indicating that the read/write head is on this symbol. We denote by \(\overline{D} = \{d | d \in D\}\) the set of marked tape symbols; a tape instance is a sequence \(\delta \in (D \cup \overline{D})^*\) such that \(\delta\) contains exactly one element of \(D\).

A tape instance thus is a finite sequence of symbols that represents the contents of a two-way infinite tape. Henceforth, we shall not distinguish between tape instances that are equal modulo the addition or removal of extra occurrences of the symbol \(\square\) at the left or right extremes of the sequence. That is, we shall not distinguish tape instances \(\delta_1\) and \(\delta_2\) if \(\square \infty \delta_1 \infty \square = \square \infty \delta_2 \infty \square\).

**Definition 5 (Configuration).** A configuration of an RTM \(M = (S, \rightarrow, \uparrow, \downarrow)\) is a pair \((s, \delta)\) consisting of a state \(s \in S\), and a tape instance \(\delta\).

Our transition system semantics defines an \(A_\tau\)-labelled transition relation on configurations such that an RTM-transition \(s \xrightarrow{d,a,c,M} t\) corresponds with \(a\)-labelled transition from configurations consisting of the RTM-state \(s\) and a tape instance in which some occurrence of \(d\) is marked. The transition leads to a configuration consisting of \(t\) and a tape instance in which the marked symbol \(d\) is replaced by \(e\), and either the symbol to the left or to right of this occurrence of \(e\) is replaced by its marked version, according to whether \(M = L\) or \(M = R\). If \(e\) happens to be the first symbol and \(M = L\) or the last symbol and \(M = R\), then an additional blank symbol is added at the left or right end of the tape instance, respectively, to facilitate the movement.

It is convenient to introduce some notation to be able to concisely denote the new placement of the tape head marker. Let \(\delta\) be an element of \(D^*\). Then by \(\overrightarrow{\delta}\) we denote the element of \((D \cup \overline{D})^*\) obtained by placing the tape head marker on the right-most symbol of \(\delta\) if it exists, and \(\square\) otherwise, i.e.,

\[
\overrightarrow{\delta} = \begin{cases} 
\delta \overline{d} & \text{if } \delta = \delta' d \quad (d \in D, \delta' \in D^*) , \\
\square & \text{if } \delta' = \varepsilon .
\end{cases}
\]

(We use \(\varepsilon\) to denote the empty sequence.) Similarly, by \(\overleftarrow{\delta}\) we denote the element of \((D \cup \overline{D})^*\) obtained by placing the tape head marker on the left-most symbol of \(\delta\) if it exists, and \(\square\) otherwise i.e.,

\[
\overleftarrow{\delta} = \begin{cases} 
\overline{d}\delta' & \text{if } \delta = d\delta' \quad (d \in D, \delta' \in D^*) , \\
\square & \text{if } \delta' = \varepsilon .
\end{cases}
\]
Definition 6. Let $\mathcal{M} = (S, \rightarrow, \uparrow, \downarrow)$ be an RTM. The transition system $T(\mathcal{M})$ associated with $\mathcal{M}$ is defined as follows:

1. its set of states is the set of configurations;
2. its transition relation $\rightarrow$ is the least relation satisfying, for all $a \in A_\tau$, $d, e \in D_\square$ and $\delta_L, \delta_R \in D_\tau^*$:
   $$(s, \delta_L \downarrow d \delta_R) \xrightarrow{a} (t, \delta_L e \downarrow \delta_R) \text{ iff } s \xrightarrow{d,a,e,L} t,$$ and conversely
   $$(s, \delta_L \downarrow d \delta_R) \xrightarrow{a} (t, \delta_L e \rightarrow \delta_R) \text{ iff } s \xrightarrow{d,a,e,R} t.$$ 
3. its initial state is the configuration $(\uparrow, \square)$; and
4. its set of final states is the set of terminating configurations $\{(s, \delta) \mid s \downarrow\}$.

Proposition 1. The transition system associated with an RTM is computable.

4 Simulation of effective transition systems

To demonstrate the expressiveness of reactive Turing machines, we shall prove that, up to branching bisimilarity, every effective transition system can be simulated with a reactive Turing machine. For the remainder of this section let $L = (S_L, \rightarrow_L, \uparrow_L, \downarrow_L)$ be an effective transition system, and let $\gamma_L: S_L \rightarrow \mathbb{N}$ be a coding such that the sets $\gamma_L^{-1}$ and $\gamma_L^{-1}$ are recursively enumerable. We shall construct an RTM $\mathcal{M}_{\text{eff}} = (S_{\text{eff}}, \rightarrow_{\text{eff}}, \uparrow_{\text{eff}}, \downarrow_{\text{eff}})$ such that $T(\mathcal{M}_{\text{eff}}) \equiv_b L$; $\mathcal{M}_{\text{eff}}$ is called the effective transition system simulator for $L$.

Tape. Relying on the codings $\gamma_L: A_\tau \rightarrow \{1, \ldots, |A_\tau|\}$ and $\gamma_L: S_L \rightarrow \mathbb{N}$, $\mathcal{M}_{\text{eff}}$ stores actions, states and transitions on its tape as natural numbers. The way in which natural numbers are represented as sequences over some finite alphabet of tape symbols is largely irrelevant, but in our construction below it is sometimes convenient to have an explicit representation. In such cases, we assume that numbers are stored natural numbers in unary notation using the symbol 1. That is, a natural number $n$ is represented on the tape as the sequence $1^n + 1$ of $n + 1$ occurrences of the symbol 1. Mostly when specifying tape contents, however, we shall permit ourselves some sloppiness and write $n$ when we actually mean $1^n + 1$. In addition to the symbol 1, we use the symbol $\bot$ to mark an occurrence of 1, we use the symbols $[ ]$ and $\sqcup$ to enclose the representation on the tape of the state of $L$ that is currently being simulated, and # to separate the elements of a tuple of natural numbers. The RTM $\mathcal{M}_{\text{eff}}$ constructed below will incorporate the operation of some auxiliary Turing machines that may use some extra symbols; let $D'$ be the collection of all these extra symbols. Then the tape alphabet $D$ of $\mathcal{M}_{\text{eff}}$ is

$$D = \{1, \bot, [, ], #\} \cup D'.$$

For clarity of presentation, it is convenient to present $\mathcal{M}_{\text{eff}}$ as the union of six fragments, parts of $\mathcal{M}_{\text{eff}}$ representing some particular functionality of $\mathcal{M}_{\text{eff}}$. Each fragment has a set of states, an initial state, and a set of transitions. The union of the sets of states and transitions of the fragments will later constitute the sets of states and transitions of $\mathcal{M}_{\text{eff}}$. 


**Initialisation fragment.** Conventional Turing machines start with the tape containing all the input needed for the computation. In contrast, RTMs always start with an empty tape, represented by the tape instance $\llbracket\boxempty\rrbracket$. The *initialisation fragment* $\text{Init}$ prepares the tape for the simulation of $L$ by writing first the symbol $\llbracket\rrbracket$, then the code $\llbracket\uparrow\rrbracket$ of the initial state of $L$, and then the symbol $\rrbracket$.

The set of states of $\text{Init}$ is defined as

$$S_{\text{Init}} = \{\uparrow_{\text{Init}}\} \cup \{in_k \mid 0 \leq k \leq \uparrow_{\text{Init}} + 2\},$$

with $\uparrow_{\text{Init}}$ its start state, and its set of transitions is defined as

$$\rightarrow_{\text{Init}} = \{(\uparrow_{\text{Init}}, \square, \uparrow, R, in_0)\} \cup \{(in_k, \square, \tau, R, in_{k+1}) \mid 0 \leq k \leq \uparrow_{\text{Init}}\}$$

$$\cup \{(in_{\uparrow L + 1}, \square, \tau, L, in_{\uparrow L + 2})\}$$

$$\cup \{(in_{\uparrow L + 2}, d, \tau, d, \uparrow, \text{Term}) \mid d \in \mathcal{D}\}.$$  

Fig. 1. Diagram of the initialisation fragment.

**State fragment.** It is assumed that the *state fragment* $\text{State}$ starts with a tape instance of the form $\llbracket\llbracket s \rrbracket\rrbracket$ ($s \in S_L$). It starts with making a copy of $\llbracket s \rrbracket$, so that it reads $\llbracket s \rrbracket\llbracket s \rrbracket$. Then it non-deterministically chooses between making an attempt at determining if $s \downarrow_L$, or making an attempt at executing a transition $s \xrightarrow{a} t$ from $s$. The former is done by simply calling the termination fragment $\text{Term}$. The latter involves the nondeterministic guessing of $\pi(\llbracket a \rrbracket, \llbracket t \rrbracket)$ and then calling the transition fragment $\text{Trans}$. Both the termination fragment and the transition fragment will perform computations that are not guaranteed to terminate and will have a facility built-in to abort such computations. It is then important to be able to return to the situation at the beginning of the state fragment and to have a backup copy of $\llbracket s \rrbracket$ available on the tape. The termination fragment will not read from or write to the initial part of the tape containing $\llbracket s \rrbracket$ at all; the transition fragment will only do so after performing the actual $a$-transition, when it is clear that $\llbracket s \rrbracket$ is no longer needed.

The set of states of $\text{State}$ is defined as

$$S_{\text{State}} = \{st_0, \ldots, st_8\};$$

its initial state is defined as

$$\uparrow_{\text{State}} = st_0;$$ and
its set of transitions is defined as

\[ \rightarrow_{\text{State}} = \{(s_{t0}, [\tau, [\tau, [R, st_1]]) \}
\]

\[ \cup \{(st_1, 1, \tau, 1, R, st_2), (st_1, [\tau, [L, st_4])\}
\]

\[ \cup \{\{st_2, d, \tau, d, R, st_2 \mid d \in D\} \cup \{(st_2, \boxed{\square, \tau, 1, L, st_3}\}
\]

\[ \cup \{(st_3, d, \tau, d, L, st_3) \mid d \in D\{1]\} \cup \{(st_3, 1, \tau, 1, R, st_1)\}
\]

\[ \cup \{(st_4, d, \tau, d, R, \uparrow_{\text{Term}}), (st_4, d, \tau, d, R, st_5) \mid d \in D\{\}\}
\]

\[ \cup \{(st_5, 1, \tau, 1, R, st_5), (st_5, \boxed{\square, \tau, \#}, R, st_6)\}
\]

\[ \cup \{(st_6, \boxed{\square, \tau, 1, R, st_6}), (st_6, \boxed{\square, \tau, 1, L, st_7})\}
\]

\[ \cup \{(st_7, d, \tau, d, L, st_7) \mid d \in D\{\}\} \cup \{(st_7, [\tau, [L, st_8]\}
\]

\[ \cup \{(st_8, 1, \tau, 1, R, \uparrow_{\text{Trans}})\}. \]

Fig. 2. Diagram of the state fragment.

**Termination fragment.** The termination fragment Term makes an attempt at determining if the current state is final; if so, then the simulator will enter a final state. The fragment starts with a tape instance of the form \(\Gamma s\) \(\uparrow s\) \(s\) \((s \in S_L)\). That the set \(\Gamma_{\downarrow L}\) is recursively enumerable means that it is the domain of some partial recursive function \(\varphi_t\), and by the Church-Turing thesis, there exists a (conventional) deterministic Turing machine \(M_t\) that computes \(\varphi_t\). We assume, without loss of generality, that \(\varphi_t\) satisfies, for all \(n \in \mathbb{N}\),

\[
\varphi_t(n) = \begin{cases} 
0 & \text{if } n \in \Gamma_{\downarrow L} \setminus \Gamma_{\downarrow L}, \\
\text{undefined} & \text{if } n \not\in \Gamma_{\downarrow L}. 
\end{cases}
\]

Furthermore, we assume without loss of generality that \(M_t\) satisfies the following assumptions:

1. it has a unique halting state \(\downarrow_{M_t}\) without any outgoing transitions;
2. $M_t$ never moves its head to the left of the (left-most) occurrence of $\lceil$ on the tape;
3. $M_t$ never replaces the occurrence of a non-blank symbol $d \neq \square$ by $\square$, unless this occurrence is the right-most occurrence of a non-blank symbol on the tape;
4. to compute $\varphi_t(n)$, it should be started with tape instance $\delta \lceil n \rceil$ for some $\delta \in D_\square^*$ that is (as a consequence of the previous assumption) irrelevant for the proper working of $M_t$;
5. if $\varphi_t(n)$ is defined, then $M_t$ eventually reaches the configuration consisting of the unique halting state and the tape instance $\delta \rceil_T$; and
6. if $\varphi_t(n)$ is undefined, then the computation of $M_t$ never halts.

(The reader is referred, e.g., to the textbooks [14] and [10] for ample explanations as to why our assumptions can be met.)

All transitions of $M_t$ are included directly as transitions of $\mathsf{Term}$, extending each of them with the action $\tau$. Then, we also add extra transitions from all states but the final state $\downarrow_{M_t}$ of $M_t$ to the fragment $\mathsf{Abort}$, by which the computation of $\mathsf{Term}$ can be aborted as long as it has not yet reached the unique halting state. The latter is necessary to enable the interruption of a non-terminating computation of $M_t$ on $\rceil s \lceil$. From $\downarrow_{M_t}$, the state $\checkmark$ is reached; this state will later be declared as the unique final state of $M_{\mathsf{eff}}$; in this fragment, it moreover serves to realise the tape instance $\llbracket \rceil s \rceil$.

Let $S_{M_t}$ be the set of states of $M_t$, let $\uparrow_{M_t}$ be its initial state, and let $\downarrow_{M_t}$ be its unique halting state. We declare one extra state $\checkmark$ that will later be defined as the unique final state of $M_{\mathsf{eff}}$; it can be reached from $\downarrow_{M_t}$. The set of states of the fragment $\mathsf{Term}$ is defined as

$$S_{\mathsf{Term}} = S_{M_t} \cup \{ \checkmark \} \; ;$$

its initial state is defined by

$$\uparrow_{\mathsf{Term}} = \uparrow_{M_t} \; .$$

All transitions in $\rightarrow_t$ are included directly as transitions of $\mathsf{Term}$, extending each of them with the action $\tau$. Then, we also add extra transitions from every state in $S_{M_t} \setminus \{ \downarrow_{M_t} \}$ to the fragment $\mathsf{Abort}$, by which the computation of $\mathsf{Term}$ can be aborted as long as it has not yet reached the unique halting state. The latter is necessary to enable the interruption of a non-terminating computation of $M_t$ on $\rceil s \lceil$. From $\downarrow_{M_t}$, the state $\checkmark$ is reached; this state will later be declared as the unique final state of $M_{\mathsf{eff}}$; in this fragment, it moreover serves to realise the tape instance $\llbracket \rceil s \rceil$.

The set of transitions of $\mathsf{Term}$ is defined as

$$\rightarrow_{\mathsf{Term}} = \{(t_e, d, \tau, e, M, t_e') \mid (t_e, d, e, M, t_e') \in \rightarrow_t\}$$
$$\cup \{ (\downarrow_{M_t}, 1, \tau, \square, L, \checkmark) \}$$
$$\cup \{ (\checkmark, d, \tau, d, L, \checkmark) \mid d \in D \} \cup \{ (\checkmark, \square, \tau, \square, R, \uparrow_{\mathsf{State}}) \}$$
$$\cup \{ (t_e, d, \tau, d, R, \uparrow_{\mathsf{Abort}}) \mid t_e \in S_{M_t} \setminus \{ \downarrow_{M_t} \}, \; d \in D_\square \} \; .$$
**Transition fragment.** The transition fragment Trans makes an attempt at determining if the current state can make a transition to another state; if so, it calls the Step fragment. The fragment starts with a tape instance of the form \([s\] [s] #n (s \in S_L, n \in \mathbb{N}). That the set \(\rightarrow_L\) is recursively enumerable means that it is the domain of a partial recursive function \(\varphi_s : \mathbb{N} \rightarrow \mathbb{N}\), and, without loss of generality, we assume that \(\varphi_s\) satisfies, for all \(n \in \mathbb{N}\),

\[
\varphi_s(n) = \begin{cases} n & \text{if } n \in \rightarrow_L \\ \text{undefined} & \text{if } n \not\in \rightarrow_L \end{cases}
\]

We combine \(\varphi_s\) with the recursive pairing function \(\pi\) and the recursive projection functions \(\pi_1\) and \(\pi_2\) to obtain a partial recursive function \(\varphi'_s : \mathbb{N}^2 \rightarrow \mathbb{N}^3\) satisfying, for all \(m, n \in \mathbb{N}\),

\[
\varphi'_s(m, n) = \begin{cases} (\pi_1(n) , \pi_2(n)) & \text{if } \pi(m, n) \in \rightarrow_L \\ \text{undefined} & \text{if } \pi(m, n) \not\in \rightarrow_L \end{cases}
\]

By the Church-Turing thesis there exists a (conventional) deterministic Turing machine \(M_s\) that computes \(\varphi'_s\). We assume without loss of generality that \(M_s\) satisfies the following assumptions:

1. it has a unique halting state \(\downarrow M_s\), without outgoing transitions;
2. \(M_s\) never moves its head to the left of the (left-most) occurrence \([\square]\) on the tape;
3. \(M_s\) never replaces the occurrence of a non-blank symbol \(d \neq \square\) by \(\square\), unless this occurrence is the right-most occurrence of a non-blank symbol on the tape;
4. to compute \(\varphi'_s(m, n)\), it should be started with tape instance \(\delta [m] #n\) for some \(\delta \in D_{\square}\) that is, as a consequence of the previous assumption, irrelevant for the proper working of \(M_s\);
5. if \(\varphi'_s(m, n)\) is defined, then \(M_s\) eventually reaches the configuration consisting of its unique halting state and tape instance \(\delta [\pi_1(n)] #\pi_2(n)\);
6. if \(\varphi'_s(m, n)\) is undefined, then the computation of \(M_s\) never halts.

All transitions in \(\rightarrow_M\), except those with the halting state \(\downarrow M_s\) as target, are included directly as transitions of Trans, extending each of them with the action \(\tau\). Transitions in \(\rightarrow_M\), with \(\downarrow M_s\) as target state are also extended with \(\tau\).
and included as transitions of $\text{Trans}$, but the target state $\downarrow M_s$ is replaced with the target state $\uparrow_{\text{Step}}$ of the step fragment. Then, we also add extra transitions from every state of $\text{Trans}$ to the fragment $\text{Abort}$, by which the computation of $M_s$ can be aborted. The latter is necessary to enable the interruption of a non-terminating computation of $M_s$ on $(\langle s \rangle, n)$.

Let $S_{M_s}$ be the set of states of $M_s$, let $\uparrow_{M_s}$ be its initial state, and let $\downarrow_{M_s}$ be its unique halting state. The set of states of $\text{Trans}$ is defined as

$$ S_{\text{Trans}} = S_{M_s} - \{\downarrow_{M_s}\} ; $$

and its initial state is defined by

$$ \uparrow_{\text{Trans}} = \uparrow_{M_s} . $$

All transitions in $\rightarrow_{M_s}$ except those with the halting state $\downarrow_{M_s}$ as target, are included directly as transitions of $\text{Trans}$, extending each of them with the action $\tau$. Transitions in $\rightarrow_{M_s}$ with $\downarrow_{M_s}$ as target state are also extended with $\tau$ and included as transitions of $\text{Trans}$, but the target state $\downarrow_{M_s}$ is replaced with the target state $\uparrow_{\text{Step}}$ of the step fragment. Then, we also add extra transitions from every state of $\text{Trans}$ to the fragment $\text{Abort}$, by which the computation of $M_s$ can be aborted. The latter is necessary to enable the interruption of a non-terminating computation of $M_s$ on $(\langle s \rangle, n)$.

The set of transitions of $\text{Trans}$ is defined as

$$ \rightarrow_{\text{Trans}} = \{(tr, d, e, M, tr') | (tr, d, e, M, tr') \in \rightarrow_{M_s} , tr, tr' \in S_{\text{Trans}}\} \cup \{(tr, d, \tau, e, M, \downarrow_{\text{Step}}) | (tr, d, e, M, \downarrow_{M_s}) \in \rightarrow_{M_s}\} \cup \{(tr, d, \tau, d, R, \uparrow_{\text{Abort}}) | tr \in S_{\text{Trans}} , d \in D\} . $$

**Step fragment.** The step fragment $\text{Step}$ is responsible for executing a transition from the current state; the target state of the transition becomes the next current state. The fragment starts with a tape instance of the form $\langle \gamma s \rangle [a^\gamma \# t^\gamma]$. Its purpose is to execute the action $a$ and restart the state fragment with $\langle t^\gamma \rangle$. The step fragment first moves the tape head to the beginning of $\gamma a^\gamma$. It then starts
decoding $\gamma a \gamma$, and executes the action $a$ when it reaches the end of $\gamma a \gamma$. Note that for $T(M_{\text{eff}})$ to be branching bisimilar to $L$, it is important to allow for the option to abort the execution of the transition $s \xrightarrow{a} t$ until the actual execution of $a$. After the action $a$ has been executed, the simulation should continue with $t$ instead of $s$ as current state of $L$. That is, the step fragment should realise the tape instance \[ \llbracket t \rrbracket \].

The set of states of $\text{Step}$ is defined as
\[ S_{\text{Step}} = \{ sp_0, \ldots, sp_6 \} \cup \{ act_1, \ldots, act_{|A_\tau|} \} \];
its initial state is defined as
\[ \uparrow_{\text{Step}} = sp_0 \]; and
its set of transitions is defined as
\[ \rightarrow_{\text{Step}} = \{(sp_0, \llbracket \tau \rrbracket, R, sp_1), (sp_1, \llbracket \tau \rrbracket, R, act_1)\} \]
\[ \cup \{(act_{k}, \llbracket \tau \rrbracket, R, act_{k+1}) | 1 \leq k < |A_\tau|\} \]
\[ \cup \{(act_{k}, #, a_k, \llbracket R \rrbracket, sp_2) | 1 \leq k \leq |A_\tau|\} \]
\[ \cup \{(act_{k}, d, \llbracket R \rrbracket, \uparrow_{\text{Abort}}) | 1 \leq k \leq |A_\tau|, d \in D\} \]
\[ \cup \{(sp_2, \llbracket \tau \rrbracket, R, sp_3), (sp_2, \llbracket R \rrbracket, L, sp_3)\} \]
\[ \cup \{(sp_3, \llbracket \tau \rrbracket, L, sp_3)\} \]
\[ \cup \{(sp_4, d, \llbracket \tau \rrbracket, \llbracket R \rrbracket, sp_4) | d \in D\} \cup \{(sp_4, \llbracket \tau \rrbracket, \llbracket R \rrbracket, sp_5)\} \]
\[ \cup \{(sp_5, \llbracket \tau \rrbracket, \llbracket R \rrbracket, sp_5), (sp_5, \llbracket \tau \rrbracket, \llbracket L \rrbracket, sp_6)\} \]
\[ \cup \{(sp_6, \llbracket \tau \rrbracket, \llbracket R \rrbracket, \uparrow_{\text{State}})\} \].

---

**Abort fragment.** The termination and transition fragments involve computations that may not terminate. The simulator should always offer the possibility to
abort a computation. The *abort fragment* \(\text{Abort}\) then takes care of the garbage collection after a computation has been aborted. It is assumed that it starts with a tape instance of the form \(\llbracket \tau s \rrbracket \delta\) (\(s \in S_L\)), with \(\delta\) a tape instance satisfying the requirement that there is no occurrence of a non-blank symbol \(d \neq \Box\) at the right of a blank symbol \(\Box\). The purpose of the abort fragment is to restore the tape instance \(\llbracket s \rrbracket\).

The set of states of \(\text{Abort}\) is defined as
\[
S_{\text{Abort}} = \{ab_0, \ldots, ab_2\};
\]
its initial state is defined as
\[
\uparrow_{\text{Abort}} = ab_0;
\]
and its set of transitions is defined as
\[
\rightarrow_{\text{Abort}} = \{(ab_0, d, \tau, d, R, ab_0) \mid d \in D\}
\cup \{(ab_0, \Box, \tau, \Box, L, ab_1)\}
\cup \{(ab_1, d, \tau, L, ab_1) \mid d \in D \setminus \{\Box\}\}
\cup \{(ab_1, \Box, \tau, L, ab_2)\}
\cup \{(ab_2, d, \tau, d, L, ab_2) \mid d \in D\}
\cup \{(ab_2, \Box, \tau, R, \uparrow_{\text{State}})\}.
\]

**Effective transition system simulator.** The *effective transition system simulator* \(M_{\text{eff}} = (S_{\text{M_{eff}}}, \rightarrow_{\text{M_{eff}}}, \uparrow_{\text{M_{eff}}}, \downarrow_{\text{M_{eff}}})\) is the union of all fragments defined above.

The set of states of \(M_{\text{eff}}\) is defined as the union of the sets of states of all fragments:
\[
S_{M_{\text{eff}}} = S_{\text{Init}} \cup S_{\text{State}} \cup S_{\text{Term}} \cup S_{\text{Trans}} \cup S_{\text{Step}} \cup S_{\text{Abort}};
\]
the transition relation of \(M_{\text{eff}}\) is the union of the transition relations of all fragments:
\[
S_{M_{\text{eff}}} = \rightarrow_{\text{Init}} \cup \rightarrow_{\text{State}} \cup \rightarrow_{\text{Term}} \cup \rightarrow_{\text{Trans}} \cup \rightarrow_{\text{Step}} \cup \rightarrow_{\text{Abort}};
\]
the initial state of \(M_{\text{eff}}\) is the initial state of \(S_{\text{Init}}\):
\[
\uparrow_{M_{\text{eff}}} = \uparrow_{\text{Init}};
\]
and the set of final states of \(M_{\text{eff}}\) consists of the state \(\checkmark\) of the fragment \(\text{Term}\):
\[
\downarrow_{M_{\text{eff}}} = \{\checkmark\}.
\]
Fig. 7 illustrates how the fragments are combined to constitute the RTM $M_{\text{eff}}$. A fragment is shown as a box; an intermediate tape instance is shown as an ellipse. A dotted arrow from a tape instance to a box indicate that the tape instance matches the expected initial configuration for the corresponding reactive Turing machine; an outgoing dotted arrow from a box to a tape instance means that this tape instance is realised by the corresponding fragment. A double outgoing dotted arrow represents that it is possible to jump out of the respective fragment on multiple points with and have the resulting tape instance.

![Diagram of the effective transition system simulator.](image)

**Theorem 1.** Let $L$ be an effective transition system, and let $M_{\text{eff}}$ be the RTM associated to $L$ as above. Then $T(M_{\text{eff}}) \equiv_{b} L$.

**Corollary 1.** Every effective transition system is branching bisimilar with a transition system associated with an RTM.

5 Simulation of computable transition systems

The effective transition system simulator defined in the previous section introduces divergence. We shall now establish that deterministic computable transition systems can be simulated without introducing divergence. We shall prove that, up to branching bisimilarity with explicit divergence, every deterministic computable transition system can be simulated with an reactive Turing machine.

For the remainder of this section let $L = (S, \rightarrow, \uparrow, \downarrow)$ be a deterministic computable transition system, and let $\gamma: S \rightarrow \mathbb{N}$ be a coding such that the set $\gamma^{-1}(n)$ is decidable and the (partial) function $\text{tr}: \mathbb{N} \rightarrow \mathbb{N}$, which associates with the code of a state the canonical index of the set of all transitions from that state,

$$\text{tr}(\gamma(s)) = \text{ci}(\gamma(\text{out}(s)))$$

for all states $s \in S$

is recursive. We shall construct an RTM $M_{dc} = (S_{M_{dc}}, \rightarrow_{M_{dc}}, \uparrow_{M_{dc}}, \downarrow_{M_{dc}})$ such that $T(M_{dc}) \equiv_{b} L$; $M_{dc}$ is called the deterministic computable transition system simulator for $L$. 


The RTM $M_{dc}$ again stores actions, states and transitions on its tape in unary notation using the symbol 1, # is again used to separate natural numbers. The RTM $M_{dc}$ constructed below will incorporate the operation of a Turing machine that may use some extra symbols. Let $D'$ be the collection of these extra symbols; then

$$D = \{1, \#\} \cup D'.$$

We shall define $M_{dc}$ as the union of three fragments.

**Initialisation fragment.** The initialisation fragment $\text{Init}$ prepares the tape for simulation of $L$ by first writing the symbol # on the tape, and then the code $\lceil \uparrow L \rceil$ of the initial state of $L$. Thereafter, it returns the head to the symbol #.

The set of states of $\text{Init}$ is defined as

$$S_{\text{Init}} = \{\uparrow \text{Init}\} \cup \{\text{in}_k \mid 0 \leq k \leq \lceil \uparrow L \rceil + 1\},$$

with $\uparrow \text{Init}$ its start state, and its set of transitions is defined as

$$\rightarrow_{\text{Init}} = \{(\uparrow \text{Init}, \Box, \tau, \#, \text{in}_0)\} \cup \{(\text{in}_k, \Box, \tau, 1, R, \text{in}_{k+1}) \mid 0 \leq k \leq \lceil \uparrow L \rceil\}$$

$$\cup \{(\text{in}_{\lceil \uparrow L \rceil}, \Box, 1, R, \text{in}_{\lceil \uparrow L \rceil +1})\} \cup \{(\text{in}_{\lceil \uparrow L \rceil +1}, d, \tau, d, L, \text{in}_{\lceil \uparrow L \rceil +1}) \mid d \in D\}.$$

**State fragment.** The state fragment $\text{State}$ replaces the code of the current state on the tape by a sequence of natural numbers that represents the behaviour of $L$ in the current state. It is assumed that it starts with a tape instance of the form $\# \lceil s \rceil$ ($s \in S_L$). First, we define a total recursive function $\varphi$ that, when applied to the code $\lceil s \rceil$ of a state $s$ of $L$, returns an $|A_{\tau}| + 1$-tuple of natural numbers that encodes all the information needed to simulate the behaviour of $s$ in $L$. To this end, let $\chi : \mathbb{N} \rightarrow \mathbb{N}$ be the mapping that, when applied to the code $\lceil s \rceil$ of $s$, return the canonical index of the subset of $\{0, \ldots, |A_{\tau}|\}$ that contains the element 0 if, and only if, $s \in \downarrow$, and the element $k$ ($1 \leq k \leq |A_{\tau}|$) if, and only if, there is a transition from $s$ labelled with $a_k$, i.e., $\chi$ is defined by

$$\chi(n) = ci(\{0 \mid n \in \lceil \downarrow L \rceil\} \cup$$

$$\{k \in \{1, \ldots, |A_{\tau}|\} \mid \exists m \in ci^{-1}(tr(n)) \text{ s.t. } k = \pi_1(m)\}).$$
Then, let $\psi_k : \mathbb{N} \rightarrow \mathbb{N}$ be a total recursive function such that

$$
\psi_k(n) = \begin{cases} 
\ell & \text{if } \pi(k, \ell) \in ci^{-1}(tr(n)); \\
0 & \text{otherwise.}
\end{cases}
$$

The total recursive function $\varphi : \mathbb{N} \rightarrow \mathbb{N}^{1 + |A_\tau|}$ can then be defined by

$$
\varphi(n) = (\chi(n), \psi_1(n), \ldots, \psi_{|A_\tau|}(n)).
$$

Note that, indeed, the function $\varphi$, when applied to the code $\lceil s \rceil$ of $s \in S_L$, conveniently returns all the information needed to simulate the behaviour of $L$ in state $s$: $s \in \downarrow$ if, and only if, $0 \in ci^{-1}(\chi(\lceil s \rceil))$; $a_k$ ($1 \leq k \leq |A_\tau|$) is enabled in $s$ if, and only if, $k \in ci^{-1}(\chi(\lceil s \rceil))$; and if $a_k$ ($1 \leq k \leq |A_\tau|$) is enabled in $s$, then the state of $L$ after performing the action $a_k$ from the $s$ can be found as the $k + 1$-th element of $\varphi(\lceil s \rceil)$. By the Church-Turing thesis there exists a (conventional) deterministic Turing machine $M_s$ that computes $\varphi$. We assume without loss of generality that, in order to compute $\varphi(n)$, we should start $M_s$ with a tape instance $\# n$, and it will halt in its unique halting state $\downarrow M_s$ with tape instance $\# \chi(n) \# \psi_1(n) \# \ldots \# \psi_{|A_\tau|}(n)$.

The set of states of $\textbf{State}$ is defined as

$$
\mathcal{S}_{\text{State}} = \mathcal{S}_{M_s} - \{\downarrow M_s\} ;
$$
its initial states is defined as

$$
\uparrow \text{State} = \uparrow M_s ; \text{ and}
$$
its set of transitions is defined as

$$
\rightarrow \text{State} = \{(st, d, \tau, e, st') | (st, d, e, st') \in \rightarrow M_s , st, st' \in \mathcal{S}_{\text{State}}\} \\
\cup \{(st, d, \tau, e, \uparrow \text{Step}) | (st, d, \tau, d, \downarrow M_s) \in \rightarrow M_s \} .
$$

\textbf{Fig. 9.} Diagram of the state fragment.

\textit{Step fragment.} The \textit{step fragment} $\text{Step}$ selects an action $a$ from the set of actions that are enabled in the current state, performs it, and removes all natural numbers from the tape, except the code of the state after the $a$-transition. It starts with a tape instance of the form $\# n_0 \# n_1 \# \ldots \# \# n_{|A_\tau|}$. The purpose of the
ment. A fragment is to first move to a special state $sp_{ci}$ that is declared final if, and only if, $0 \in ci^{-1}(n_0)$ and has an outgoing transition with the action $a_k$ if, and only if, $k \in ci^{-1}(n_0)$. Note that, by the definition of $\chi$, the maximum of the set encoded by $n_0$ is at most $|A|$, and hence is sufficient to include $N + 1$ such states, where $N = \sum_{k=1}^{|A|} 2^k$ (cf. our assumptions for $ci$ in Sect. 2). By taking an enabled action $a_k$ from the state $sp_{n_0}$, the state $nc_k$ is entered. The remaining goal from that state is to realise tape instance $\#n_k$, which means that $\#n_k$ should be found on the tape (at the same time removing all non-blank symbols preceding $\#n_k$), every non-blank symbol to the right of $\#n_k$ should be replaced by $\square$, and the tape head should move back to the initial occurrence of $\#$.

The set of states of $\text{Step}$ is defined as

$$S_{\text{Step}} = \{\uparrow_{\text{Step}}, sp_{\perp}\} \cup \{sp_k, sp'_k \mid 0 \leq k \leq N\} \cup \{ne_1, \ldots, ne_{|A|}\} \cup \{cr_0, cr_1, cr_2, cr_3\} ;$$

with $\uparrow_{\text{Step}}$ its initial state, and its set of transitions is defined as

$$\rightarrow_{\text{Step}} = \{\uparrow_{\text{Step}}, \#, \tau, \square, R, sp_{\perp}\}, \{sp_{\perp}, 1, \tau, \square, R, sp'_0\} \cup \{(sp_k, 1, \tau, \square, R, sp'_{k+1}) \mid 0 \leq k < N\} \cup \{(sp'_k, \#, \tau, \square, R, sp_k) \mid 0 \leq k \leq N\} \cup \{(sp_k, 1, a_1, 1, R, ne_1) \mid 0 \leq k \leq N, 1 \in ci^{-1}(k)\} \cup \{(sp_k, 1, a_\ell, \square, R, ne_\ell) \mid 0 \leq k \leq N, 1 < \ell \leq |A|, \ell \in ci^{-1}(k)\} \cup \{(ne_1, 1, \tau, \square, R, ne_\ell) \mid 1 < \ell \leq |A|\} \cup \{(ne_1, 1, \tau, 1, R, ne_1), (ne_1, \#, \tau, \square, R, cr_0), (ne_1, \square, \tau, \square, L, cr_1)\} \cup \{(cr_0, d, \tau, \square, R, cr_0) \mid d \in D\} \cup \{(cr_0, \square, \tau, \square, L, cr_1)\} \cup \{(cr_1, \tau, \square, L, cr_1) \mid cr_1 \in \text{Step}\} \cup \{(cr_2, 1, \tau, 1, L, cr_2), (cr_2, \square, \tau, \#, L, cr_3)\} \cup \{(cr_3, \square, \tau, \#, R, \uparrow_{\text{State}})\} .$$

**Deterministic computable transition system simulator**. To define the deterministically computable transition system simulator $M_{dc} = (S_{M_{dc}}, \rightarrow_{M_{dc}}, S_{init}, \downarrow_{M_{dc}})$ we simply combine all fragments defined above. The set $\downarrow_{M_{dc}}$ consists of those states $sp_k$ of the fragment $\text{Step}$, which are reached after decoding the first natural number on the tape, for which $0 \in ci^{-1}(k)$.

The set of states of $M_{dc}$ is defined as the union of the sets of states of all fragments:

$$S_{M_{dc}} = S_{\text{Init}} \cup S_{\text{State}} \cup S_{\text{Step}} ;$$

the transition relation of $M_{dc}$ is the union of the transition relations of all fragments:

$$S_{M_{dc}} = \rightarrow_{\text{Init}} \cup \rightarrow_{\text{State}} \rightarrow_{\text{Step}} ;$$
the initial state of $\mathcal{M}_{dc}$ is the initial state of $\text{Init}$:

$$\uparrow_{\mathcal{M}_{dc}} = \uparrow_{\text{Init}}; \text{ and}$$

the set of final states of $\mathcal{M}_{dc}$ consists of the states of Step $\text{sp}_k$ such that $0$ is in the finite set of natural numbers encoded by $k$:

$$\downarrow_{\mathcal{M}_{dc}} = \{ \text{sp}_k \mid 0 \in ci^{-1}(k) \}.$$  

Fig. 11 illustrates, in a similar way as Fig 7, how the fragments are combined to constitute the RTM $\mathcal{M}_{dc}$.

**Theorem 2.** Let $L$ be a deterministic computable transition system, and let $\mathcal{M}_{dc}$ be the RTM associated to $L$ as above. Then $T(\mathcal{M}_{dc}) \leftrightarrow^{\Delta} L$.

**Corollary 2.** Every deterministic computable transition system is branching bisimilar with explicit divergence with a transition system associated with an RTM.
6 ACP_τ specifications

In this section we establish a correspondence between reactive Turing machines and the process theory ACP_τ [4]. First, we introduce an instance of ACP_τ with

\[ \tau \]

handshaking communication presented earlier in [3].

We presuppose a finite set \( C \) of channels, reuse the set of data \( D \) and define the set of special actions \( I = \{ c?d, c!d, c\!\!d \mid d \in D, c \in C \} \). The actions \( c?d, c!d, c\!\!d \) respectively denote the event that a datum \( d \) is received, sent, or communicated along channel \( c \). Let \( \mathcal{N} \) be a set of names. The set of process expressions \( \mathcal{P} \) is generated by the following grammar (\( a \in A \cup I \), \( N \in \mathcal{N} \), \( c \in C \)):

\[
\begin{align*}
p &::= 0 \mid 1 \mid a.p \mid p \cdot p \mid p + p \mid p \parallel p \mid \partial_c(p) \mid \tau_c(p) \mid N.
\end{align*}
\]

Let us briefly comment on the operators in this syntax. The constant \( 0 \) denotes deadlock, the unsuccessfully terminated process. The constant \( 1 \) denotes the successfully terminated process. For each action \( a \in A \cup I \) there is a unary action prefix; the process denoted by \( a.p \) can do an \( a \)-transition to the process denoted by \( p \). The binary operator \( \cdot \) denotes sequential composition. The binary operator \( + \) denotes alternative composition or choice. The binary operator \( \parallel \) denotes parallel composition; actions of both arguments are interleaved, and in addition a communication \( c?d \) of a datum \( d \) on channel \( c \) can take place if one argument can do an input action \( c?d \) that matches an output action \( c!d \) of the other component. The unary operator \( \partial_c(p) \) encapsulates the process \( p \) in such a way that all input actions \( c?d \) and output actions \( c!d \) are blocked (for all data) so that communication is enforced. Finally, the unary operator \( \tau_c(p) \) denotes abstraction from communication over channel \( c \) in \( p \) by renaming all communications \( c\!\!d \) to \( \tau \)-transitions. We shall abbreviate \( \tau_c(\partial_c(p)) \) with \( [p]_c \).

A recursive specification \( E \) is a set of equations of the form: \( N \stackrel{\text{def}}{=} p \), with as left-hand side a name \( N \) and as right-hand side a process expression \( p \). It is required that a recursive specification \( E \) contains, for every \( N \in \mathcal{N} \), precisely one equation with \( N \) as left-hand side; this equation will be referred to as the defining equation for \( N \) in \( \mathcal{N} \). Each recursive specification has a distinguished initial name.

Definition 7. Let \( E \) be a recursive specification. The transition system \( T(E) \) associated with \( E \) is defined as follows:

1. its states are the process expressions reachable from the initial name;
2. its transition relation \( \rightarrow \) is given by the operational rules in Table 1;
3. its initial state is the distinguished initial name of \( E \); and
4. its final states are the process expressions \( p \) reachable from the initial name such that \( p \downarrow \) according to the operational rules in Table 1.

We prove that for every reactive Turing machine \( M \) there exists a finite recursive ACP_τ specification \( E_M \) such that \( T(M) \cong T(E_M) \). Our finite recursive specification \( E_M \) will have two parts: one part specifying the tape of an
Queue. The following infinite recursive specification specifies the behaviour of the process $Q_\delta$ modelling a queue with contents $\delta$ that receive inputs over channel $i$ and sends output over channel $o$ (for every $d \in D_\square, \delta \in D_\square$):

$$Q_\varepsilon \overset{\text{def}}{=} 1 + \sum_{d \in D} i?d.Q_d,$$

$$Q_{\delta d} \overset{\text{def}}{=} 1 + o!d.Q_\delta + \sum_{e \in D} i?e.Q_{\epsilon \delta d}.$$

We can also give a finite recursive specification $Q^{\infty}_l$ defined in [4,1] which is branching bisimilar with explicit divergence with $Q_\varepsilon$. For clarity reasons, we shall continue to use the infinite specification.

Tape. The following infinite recursive specification specifies the behaviour of a tape processes $T_{\delta L, \delta R}$ for every possible tape instance. This tape process sends
the datum \(d\) under the head over channel \(r\) (read), receives a datum \(e\) to replace the datum under the head over channel \(w\) (write), and receives commands to move one position to the left (on \(\delta_L\)) or right (on \(\delta_R\)) over channel \(m\) (move) for \(d \in \mathcal{D}, \delta_L, \delta_R \in \mathcal{D}^*\):

\[
T_{\delta_L \delta_R}^{\delta_L \delta_R} \overset{\text{def}}{=} 1 + r!d.T_{\delta_L \delta_R}^{\delta_L \delta_R} + \sum_{e \in \mathcal{D}^*} w?e.T_{\delta_L \delta_R}^{\delta_L \delta_R} + m?L.T_{\delta_L \delta_R}^{\delta_L \delta_R} + m?R.T_{\delta_L \delta_R}^{\delta_L \delta_R}.
\]

Again, this recursive specification is infinite, but intuitive. Below we will give a finite specification of a tape process and show that it is branching bisimilar with explicit divergence with the tape process specified above. In the following recursive specification \(E_T\) we have the main process \(H_d\) which represents the head being located on datum \(d\). The process \(H_d\) is put in parallel with the previously defined queue process \(Q_{\delta_R \perp \delta_L}\). Here, \(\delta_R\) is the sequence to the right of the head followed by a special marker \(\perp\) and then the sequence \(\delta_L\) that is to the left of the head. The process \(H_d\) can remove a datum from the queue over channel \(o\) and then insert it over channel \(i\); we call this shifting. Doing this once is called a shift operation and effectively moves the contents of the queue one position to the right, see Fig. 12 for a schematic overview.

We define the finite specification \(E_T\) (with initial name \(T\)) as follows:

\[
T \overset{\text{def}}{=} [H_{\square} \parallel Q_{\perp}]_o,
\]

using the following auxiliary defining equations for all \(d \in \mathcal{D}_{\square}\):

\[
H_d \overset{\text{def}}{=} 1 + r!d.H_d + \sum_{e \in \mathcal{D}_{\square}} w?e.H_e + m?L.H_d + m?R.H_d,
\]
\[
H_d^{L_d} \overset{\text{def}}{=} \delta_d \left( \sum_{e \in \mathcal{D}_{\square}} o?e.H_e + o?\perp\perp\dollar\dollar\dollar.H_d \right),
\]
\[
\text{Back} \overset{\text{def}}{=} \sum_{d \in \mathcal{D}_{\square}} o?d.H_d + o?\perp\perp\dollar\dollar\dollar.H_d.
\]
$$H^R_d \overset{\text{def}}{=} i$$. For $\epsilon \in \mathcal{D}$, we define
$$F_{\text{wd}d} \overset{\text{def}}{=} \sum_{\epsilon \in \mathcal{D}} \epsilon \cdot \text{Fwd}_d + o?\cdot \perp \cdot \text{Fwd}_{\perp}$$,
$$F_{\text{wd}L} \overset{\text{def}}{=} \sum_{\epsilon \in \mathcal{D}} \epsilon \cdot \perp \cdot \text{Fwd}_d + o?\cdot \perp \cdot \text{Fwd}_{\perp} + o? \cdot H_d$$.

When we move to the left ($H^L_d$), we can simply remove a datum from the queue and insert datum $d$ into the queue. But, we have to make sure that it does not accidentally remove the special marker $\perp$ after inserting datum $d$ in the case that the sequence to the left of the head ($\delta_L$) is empty. If this happens, we insert the search marker $\$ followed by $\perp$ and

Because the queue only shifts in one direction, we need a different approach for moving to the right ($H^R_d$). We first insert the search marker $\$ followed by $d$.

Then, $\text{Fwd}_d$ (for some $\epsilon \in \mathcal{D}$) shifts through the queue contents, remembering the previously removed datum $e$, thus acting as a lookahead. When it encounters the search marker $\$, the previously removed datum becomes the current datum under the head and the queue is in the correct state again.

We argue that the tape process given by the infinite and finite recursive specifications are branching bisimilar with explicit divergence and in the process explain the functionality of the auxiliary equations and show how the different types of commands (reading, writing and moving) are handled.

**Lemma 1.** For each tape instance $\delta_L \perp \delta_R$ ($\delta_L, \delta_R \in \mathcal{D}$, $d \in \mathcal{D}$) we have that $T_{\delta_L \perp \delta_R} \overset{\sim}{\overset{\Delta}{\rightarrow}} \{ [H_d \parallel Q_{\delta_R \perp \delta_L}], o \}$.

**Proof (Sketch).** Let $\mathcal{R}_S$ be defined by
$$\mathcal{R}_S = \{ (\delta_L, \delta_R \in \mathcal{D}, d \in \mathcal{D}) \mid \exists (T_{\delta_L \perp \delta_R}, [H_d \parallel Q_{\delta_R \perp \delta_L}]). \}

First, we observe that $T_{\delta_L \perp \delta_R}$ can terminate regardless of its contents. The same holds for $[H_d \parallel Q_{\delta_R \perp \delta_L}]$ since both $H_d$ and $Q_{\delta_R \perp \delta_L}$ have a $1$-summand. Now, let us consider the other summands present both in $T_{\delta_L \perp \delta_R}$ and $H_d$ separately.

**Reading:** If $T_{\delta_L \perp \delta_R} \overset{rld}{\rightarrow} T_{\delta_L \perp \delta_R}$, then also $[H_d \parallel Q_{\delta_R \perp \delta_L}] \overset{rld}{\rightarrow} [H_d \parallel Q_{\delta_R \perp \delta_L}]$ and vice versa. The resulting states are identical to the starting states and therefore related by $\mathcal{R}_S$.

**Writing:** For some $\epsilon \in \mathcal{D}$, if $T_{\delta_L \perp \delta_R} \overset{wrd}{\rightarrow} T_{\delta_L \perp \delta_R}$, then also $[H_d \parallel Q_{\delta_R \perp \delta_L}] \overset{wrd}{\rightarrow} [H_d \parallel Q_{\delta_R \perp \delta_L}]$ and vice versa. The pair of resulting states are related by $\mathcal{R}_S$.

**Moving left:** If we move to the left on the tape, we have to take into account that the sequence to the left of the head ($\delta_L$) might be empty. If $\delta_L$ is not empty, then $\delta_L = \delta_L d_l$ for some $l \in \mathcal{D}$, and if $T_{\delta_L \perp \delta_R} \overset{mld}{\rightarrow} T_{\delta_L \perp \delta_R}$, also
[H_d \parallel Q_{\delta_L} \parallel \delta_L]_{\io} \xrightarrow{m^L} [H^L_d \parallel Q_{\delta_R} \parallel \delta_R, a]_{\io} \rightarrow [H_d \parallel Q_{\delta_R} \parallel \delta_R]_{\io}$. So, $H^L_d$ moves the tape head to the left by performing one shift operation. This results in datum $d$ getting prefixed to the part to the sequence to right of the head ($\delta_R$) and taking the right-most datum ($d_R$) of the sequence to the left of the head ($\delta_L$) and putting it under the head.

In the case that $\delta_L$ is empty, we get $[H_d \parallel Q_{\delta_R} \parallel \delta_R]_{\io} \xrightarrow{m^L} [H^L_d \parallel Q_{\delta_R} \parallel \delta_R]_{\io}$.

When moving left or right either one shift operation happens or we shift until the case.

What happens now depends on whether the sequence to the right of the head ($\delta_R$) is empty or not. If $\delta_R$ is not empty, then $\delta_R = d_e \delta_R'$ for some $d_e \in D$, and

$[Fwd_L \parallel Q_{\delta_L \parallel \delta_R} \parallel \delta_R']_{\io} \rightarrow [Fwd_L \parallel Q_{\delta_L \parallel \delta_R} \perp \delta_R']_{\io}$. If $\delta_R$ is empty, we get $[Fwd_L \parallel Q_{\delta_L} \parallel \delta_R]_{\io} \rightarrow [H \parallel Q_{\delta_L} \parallel \delta_R]_{\io}$. So, the special marker $\perp$ is reinserted and we get $H_m$.

Again, in both cases the pair of resulting states are in $\mathcal{R}_S$. We define $\mathcal{R}_L$ as the relation that contains pairs $(T_{\delta_L} \parallel \delta_L, p)$ for all reachable states $p$ from $[H^L_d \parallel Q_{\delta_R} \parallel \delta_L]_{\io}$ to either $[H_d \parallel Q_{\delta_R} \parallel \delta_L]_{\io}$ or $[H \parallel Q_{\delta_R} \parallel \delta_L]_{\io}$ depending on the case.

Moving right: Because shifting through the queue contents only goes in one direction, we have to use a different approach for moving the head to the right. This time we need to have the left-most datum of the sequence to the right of the head ($\delta_R$) and we will have to shift through the entire queue contents to reach it. We do this by inserting a search marker $\$ into the queue and shifting through it using a lookahead that remembers the datum that was previously removed from the queue. Once we encounter the search marker, we put this previously encountered datum under the head.

So, if $T_{\delta_L} \parallel \delta_L \xrightarrow{m^R} T_{\delta_L} \parallel \delta_L$, then also $[H_d \parallel Q_{\delta_R} \parallel \delta_R]_{\io} \xrightarrow{m^R} [H^R_d \parallel Q_{\delta_R} \parallel \delta_R]_{\io}$.

What happens now depends on whether the sequence to the right of the head ($\delta_R$) is empty or not. If $\delta_R$ is not empty, then $\delta_R = d_e \delta_R'$ for some $d_e \in D$, and

$[Fwd_L \parallel Q_{\delta_L \parallel \delta_R} \parallel \delta_R']_{\io} \rightarrow [Fwd_L \parallel Q_{\delta_L \parallel \delta_R} \parallel \delta_R']_{\io}$. If $\delta_R$ is empty, we get $[Fwd_L \parallel Q_{\delta_L} \parallel \delta_R]_{\io} \rightarrow [H \parallel Q_{\delta_L} \parallel \delta_R]_{\io}$. So, the special marker $\perp$ is reinserted and we get $H_m$.

We can observe that there are no $\tau$-loops introduced by the specification. When moving left or right either one shift operation happens or we shift until the search marker is found, both yield a finite number of $\tau$-steps. Hence, there is no divergence.

The relation $\mathcal{R} = \mathcal{R}_S \cup \mathcal{R}_L \cup \mathcal{R}_R$ is a branching bisimulation relation with explicit divergence. The proof that $\mathcal{R}$ is a branching bisimulation with explicit divergence is left to the reader. $\square$

The RTM. To be able to simulate an RTM $\mathcal{M}$ we construct a finite recursive specification $E_{\mathcal{M}}$. We start by adding the finite recursive specification of the tape
Now, we consider the finite recursive specification for the finite control of the RTM, the RTM specific part. For every state $s \in S$ and datum $d \in D_{\Box}$ then we add the following equation for the name $C_{s,d}$:

$$C_{s,d} \overset{\text{def}}{=} \sum_{(s,d,a,e,M,t) \in \rightarrow} \left( a.w!e.m!M, \sum_{f \in D_{\Box}} r?f.C_{t,f} \right) \left[ +1 \right]_{s_{\downarrow}}.$$

Finally, we add the following equation for the initial name $M$ of $E_M$:

$$M \overset{\text{def}}{=} \left[ C_{\uparrow,\Box} \parallel T_{\Box} \right]_{\text{rwm}}.$$

**Theorem 3.** For every reactive Turing machine $M$ there exists a finite recursive specification $E_M$ such that $T(M) \equivb T(E_M)$.

**Proof.** Let $M = (S, \rightarrow, \uparrow, \downarrow)$ be an RTM. For every state $s \in S$ and for every datum $d \in D_{\Box}$ the recursive specification $E_M$ contains the name $C_{s,d}$ as defined above. So, each defining equation of $C_{s,d}$ has a summand for every transition $s \stackrel{d,a,e,M \rightarrow}{\longrightarrow} t$ ($a \in A_{\tau}$, $e \in D_{\Box}$, $M \in \{L, R\}$) and it optionally has a $1$-summand if $s$ is a final state. The relation between a transition in an RTM and the transitions in the ACP$_{\tau}$ specification are shown in Fig. 13.

The specification $E_M$ also contains the following defining equation for the distinguished initial name $M$:

$$M \overset{\text{def}}{=} \left[ C_{\uparrow,\Box} \parallel T_{\Box} \right]_{\text{rwm}},$$

and includes the finite specifications of the tape process $E_T$. We use Lemma 1 to justify the use of the infinite tape process specifications below.

We define the relation $R_L$ as and $R_R$ as

$$\{ (s, \delta_L \overrightarrow{d}_R; [C_{s,d} \parallel T_{\delta_L \overrightarrow{d}_R}]_{\text{rwm}}) \mid s \in S, d \in D_{\Box}, \delta_L, \delta_R \in D_{\Box}^* \}.$$
Note that $\mathcal{R}$ relates the initial configuration $(\uparrow, \Boxempty)$ with the process expression $[C_{\uparrow, \Boxempty} \parallel T_{\Boxempty\text{rwm}}]$, which is the defining equation of the initial name $M$.

The relation $\mathcal{R} = \mathcal{R}_L \cup \mathcal{R}_R$ is a branching bisimulation relation with explicit divergence.

**Corollary 3.** For every effective transition system $L$ there exists a recursive specification $E_L$ over $\text{ACP}_\tau$ such that $L \leftrightarrow_b T(E_L)$, and for every deterministic computable transition system $L$ there exists a recursive specification $E_L$ over $\text{ACP}_\tau$ such that $L \leftrightarrow_b T(E_L)$.

### 7 Conclusion

We have discussed reactive Turing machines (RTMs), an extension of conventional Turing machines with a notion of interaction in the style of concurrency theory. We have established that effective transition systems can be simulated up to branching bisimilarity by RTMs, and that deterministic computable transition systems can be simulated up to branching bisimilarity with explicit divergence. Our results can be adapted to define a notion of universal RTM, which first inputs the code of an arbitrary RTM, and then simulates the (computable) transition system associated with it. The simulation works up to branching bisimilarity, and preserves divergence in the deterministic case.

**Acknowledgement.** We have greatly benefited from extensive discussions with Jos Baeten on the topics of this paper. We thank Clemens Grabmayer for suggesting to us the term reactive Turing machine.

### References


