Turing Machines, Transition Systems, and Interaction

Dina Q. Goldin [1]

Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269, USA

Scott A. Smolka [2]

Department of Computer Science, SUNY at Stony Brook Stony Brook, NY 11794-4400, USA

Peter Wegner [3]

Department of Computer Science, Brown University Providence, RI 02912, USA

Abstract

We present *Persistent Turing Machines* (PTMs), a new way of interpreting Turing-machine computation, one that is both interactive and persistent. We show that the class of PTMs is isomorphic to a very general class of effective transition systems. One may therefore conclude that the extensions to the Turing-machine model embodied in PTMs are sufficient to make Turing machines expressively equivalent to transition systems. We also define the *persistent stream language* (PSL) of a PTM and a corresponding notion of PSL-equivalence, and consider the infinite hierarchy of successively finer equivalences for PTMs over finite interaction-stream prefixes. We show that the limit of this hierarchy is strictly coarser than PSL-equivalence, a "gap" whose presence can be attributed to the fact that the transition systems corresponding to PTM computations naturally exhibit unbounded nondeterminism.

We also consider *amnesic* PTMs and a corresponding notion of equivalence based on *amnesic stream languages* (ASLs). It can be argued that amnesic stream languages are representative of the classical view of Turing-machine computation. We show that the class of ASLs is strictly contained in the class of PSLs. Furthermore, the hierarchy of PTM equivalence relations collapses for the subclass of amnesic PTMs. These results indicate that, in a stream-based setting, the extension of the Turing-machine model with persistence is a nontrivial one, and provide a formal foundation for reasoning about programming concepts such as objects with static attributes.

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

1 Introduction

Several researchers have recently observed that the Turing-machine model of computation, the focus of which is on a theory of computable functions, falls short when it comes to modeling modern computing systems whose hallmarks are interaction and reactivity. For example, van Leeuwen in [LW00b] states:

> ... the classical Turing paradigm may no longer be fully appropriate to capture all features of present-day computing.

Also, Wegner[Weg98] has conjectured that interactive models of computation are more expressive than "algorithmic" ones such as Turing machines. It would therefore be interesting to see what extensions are necessary to Turing machines to capture the salient aspects of interactive computing. Moreover, it would be desirable if the alterations made to the classical model could in some sense be kept minimal.

Motivated by these goals, we investigate a new way of interpreting Turing-machine computation, one that is both interactive and persistent. In particular, we present *persistent Turing machines* (PTMs). A PTM is a nondeterministic 3-tape Turing machine that upon receiving an input token from its environment, computes for a while and then outputs the result to the environment, and this process is repeated forever. A PTM is persistent in the sense that a notion of "state" (work-tape contents) is maintained from one computation to the next.

The main results we have obtained about PTMs are the following.

- We formalize the notions of interaction and persistence in PTMs in terms of the *persistent stream language* (PSL) of a nondeterministic 3-tape Turing machine (N3TM) (Section 2). Given an N3TM M and work-tape contents w, $PSL(M, w)$ is coinductively defined to be the set of infinite sequences (interaction streams) of pairs of the form (w_i, w_o) such that each pair represents a computation performed by the N3TM in response to receiving input token w_i from the environment, producing output token w_o. Moreover, the contents of the work tape (initially w) are left intact from the previous computation upon commencing a new computation. Persistent stream languages induce a natural, stream-based notion of equivalence for PTMs.
- We then define a very general kind of effective transition system called *interactive transition systems* (ITSs), and equip ITSs with three notions of behavioral equivalence: *ITS isomorphism*, *interactive bisimulation* and *in-*

[1] Research supported in part by NSF grant IRI-9733678. Email: dqg@cse.uconn.edu
[2] Research supported in part by NSF grant CCR-9988155 and ARO grants DAAD190110003 and DAAD190110019. Email: sas@cs.sunysb.edu
[3] Email: pw@cs.brown.edu

teractive stream equivalence (Section 3). We show that ITS isomorphism refines interactive bisimulation, and interactive bisimulation refines interactive stream equivalence.

Our main result concerning ITSs is that the class of ITSs is isomorphic to the class of PTMs, thereby allowing one to view PTMs as ITSs "in disguise" (Section 4). This addresses a question heretofore left unanswered concerning the relative expressive power of Turing machines and transition systems. Till now, the emphasis has been on showing that various kinds of process algebras, with transition-system semantics, are capable of simulating Turing machines in lock-step [Bou85,dS85,BBK87,BIM88,Dar90,Vaa93]. The other direction—namely: What extensions are required of Turing machines so that they can simulate transitions systems?—is answered by our result.

- We also define an infinite hierarchy of successively finer equivalences for PTMs over finite interaction-stream prefixes and show that the limit of this hierarchy does *not* coincide with PSL-equivalence (Section 5). The presence of this "gap" can be attributed to the fact that the transition systems corresponding to PTM computations naturally exhibit unbounded nondeterminism. In contrast, it is well known that classical Turing-machine computations are nondeterministically bounded.
- Finally, we define the *amnesic stream language* (ASL) of an N3TM and a corresponding notion of amnesic PTM (Section 6). In this case, the N3TM begins each new computation with a blank work tape. Our main result about ASLs is that the class of ASLs is strictly contained in the class of PSLs. Amnesic stream languages are representative of the classical view of Turing-machine computation. One may consequently conclude that, in a stream-based setting, the extension of the Turing-machine model with persistence is a nontrivial one, and provides a formal foundation for reasoning about programming concepts such as objects with static attributes. We additionally show that ASL-equivalence coincides with the equivalence induced by considering interaction-stream prefixes of length one, the bottom of our equivalence hierarchy; and that this hierarchy collapses in the case of amnesic PTMs.

2 Persistent Turing Machines

In this section, we show how classical Turing machines can be reinterpreted as interactive computing devices. We shall consider, in fact, *non-deterministic 3-tape Turing machines* (N3TMs), each equipped with an input, work, and output tape. It is well known that N3TMs are equivalent to single-tape TMs. That is, given an N3TM M accepting some language L, there exists a single-tape TM accepting L [HU79].

The key concept we need to render Turing machines interactive is *interaction streams*: infinite sequences of token pairs of the form (w_i, w_o). Each such pair represents a computation performed by the N3TM, producing out-

put tape contents w_o, in response to w_i being placed on its input tape by the environment. Moreover, the N3TM is allowed to "remember" its previous "state" (work-tape contents) upon commencing a new computation. We shall therefore refer to such N3TMs as *persistent Turing machines* (PTMs) and to the sets of interaction streams they generate as *persistent stream languages*.

To begin our formal treatment of PTMs, we define a *macrostep* of an N3TM as a shorthand notation for a (possibly divergent) computation of a Turing machine. Our choice of terminology is inspired by the treatment of Statecharts semantics in [PS91].

Definition 2.1 *Let M be an N3TM having alphabet Σ, and let w_i, w, w' and w_o be words over Σ. We say that $\langle w_i, w\rangle \models\!\!\Rightarrow_M \langle w', w_o\rangle$* (yields in one macrostep) *if M, when started in its initial control state with w_i, w, ϵ on its input, work, and output tapes, respectively, has a halting computation that produces w_i, w', w_o as the respective contents of its input, work, and output tapes.*

Should M's computation diverge, we write $\langle w_i, w\rangle \models\!\!\Rightarrow_M \langle s_{div}, \tau\rangle$, where $s_{div}, \tau \notin \Sigma^*$ and s_{div} is a special "divergence state" such that $\langle w_i, s_{div}\rangle \models\!\!\Rightarrow_M \langle s_{div}, \tau\rangle$ for all inputs w_i.

We sometimes omit the subscript M from the macrostep notation when it is clear from the context.

We view macrosteps as transitions from one PTM "state" (encoded in the contents of the work tape) to another, an idea that is formalized in Section 4. Divergent computations of the underlying N3TM bring the PTM to the "divergence state," a special absorbing state not in Σ^* that outputs τ (in analogy with the internal τ action of CCS [Mil89]) in conjunction with the current and all subsequent inputs. Our treatment of divergence is consistent with the failures-divergence refinement model of CSP [BR85].

The contents of the input tape is not changed by a macrostep, reflecting the read-only nature of input tapes in our framework. Moreover, a macrostep begins with a blank output tape (ϵ is the empty word) reflecting a write-only semantics for output tapes. Note, however, that a macrostep may begin with a non-blank work tape, in contrast to the classical setting where the work tape is assumed to be blank at the start of computation. This convention plays an essential role in the definition of a PTM's "persistent stream language" given below (Definition 2.2).

To formally define interaction streams and persistent stream languages, fix the alphabet of an N3TM to be Σ, and let A be an enumerable set of *action tokens*. $\mathcal{S}_A$, the class of *streams over* A, is defined as follows: $\mathcal{S}_A = A \times \mathcal{S}_A$.[4] Then the class of interaction streams is given by $\mathcal{S}_{\Sigma^* \times \Sigma^* \uplus \{\tau\}}$, the members of which are pairs of the form $\langle (w_i, w_o), \sigma'\rangle$ with $(w_i, w_o) \in \Sigma^* \times (\Sigma^* \uplus \{\tau\})$ and $\sigma' \in \mathcal{S}_{\Sigma^* \times (\Sigma^* \uplus \{\tau\})}$.

[4] We have defined streams *coinductively*; see, e.g, [BM96]. This style of definition will allow us to apply coinduction as a proof technique later in the paper.

Definition 2.2 *Given an N3TM M and some $w \in \Sigma^* \uplus \{s_{div}\}$, $PSL(M,w)$ (the* persistent stream language of M with memory w*) is defined as follows:*

$$PSL(M,w) = \{\langle (w_i, w_o), \sigma' \rangle \in \mathcal{S}_{\Sigma^* \times \Sigma^* \uplus \{\tau\}} \mid \exists w' \in \Sigma^* \uplus \{s_{div}\} : \langle w_i, w \rangle \Longrightarrow\!\!\!\!\!| \; _M \langle w', w_o \rangle \text{ and } \sigma' \in PSL(M,w')\}$$

$PSL(M)$, the persistent stream language *of M, is defined as $PSL(M,\epsilon)$. N3TMs M_1 and M_2 are* PSL-equivalent*, notation $M_1 =_{PSL} M_2$, if $PSL(M_1) = PSL(M_2)$. We also have that $\mathcal{PSL} = \{PSL(M) \mid M \text{ is an N3TM}\}$.*

Example 2.3 *Consider the N3TM M_{Latch} that outputs the first bit of the input token it received in conjunction with its* previous *interaction with the environment (except for the first interaction where it outputs a 1). $PSL(M_{Latch})$ therefore contains interaction streams of the form*

$$\{(w_1, 1), (w_2, w_1[1]), (w_3, w_2[1]), \ldots\},$$

where, in general, $w[i]$ denotes the i th bit of the string w.

For example, if the input tokens M_{Latch} receives from the environment are single bits, and the first four of these form the bit sequence 1001*, then the corresponding interaction stream $\sigma_{io} \in PSL(M_{Latch})$ would be of the form:*

$$\sigma_{io} = \{(1,1), (0,1), (0,0), (1,0), \ldots\}$$

We also consider M'_{Latch}, an unreliable version of M_{Latch}. Apart from behaving like M_{Latch}, it can nondeterministically exhibit a divergent (nonterminating) computation in conjunction with any input but the first.

For an interaction stream $\sigma \in PSL(M_{Latch})$, $PSL(M'_{Latch})$ will contain σ as well as, for all $k > 1$, a k-divergent version of σ where the computation diverges at the k th macrostep. In this case, the first $k-1$ pairs in σ remain the same, and the output tokens for all subsequent pairs are replaced by τ. For example, a 3-divergent version of σ_{io} is $\{(1,1), (0,1), (0,\tau), (1,\tau), \ldots\}$.

One might argue that the interaction between M_{Latch} and its environment is not essential; rather its behavior could be modeled by a machine that receives its entire (infinite) stream σ of input tokens prior to computation and then proceeds to output (the first bit of each element of) σ prepended with a 1. The problem with this approach is that, in general, the elements of σ are generated dynamically and therefore cannot be known in advance.

3 Interactive Transition Systems

In this section, we introduce a kind of "effective" transition system (see, for example, [Vaa93]) that we shall refer to as "interactive transition systems." An important result about interactive transition systems is that they are isomorphic to PTMs (Theorem 4.7).

Let Σ be a finite *alphabet* not containing τ, the "internal action."

Definition 3.1 *An* interactive transition system *(ITS) is a triple* $\langle S, m, r \rangle$ *where:*

- $S \subseteq \Sigma^* \uplus \{s_{div}\}$ *is the set of* states*, where* $s_{div} \notin \Sigma^*$ *is a special "divergence" state.*
- $m \subseteq S \times \Sigma^* \times S \times (\Sigma^* \uplus \{\tau\})$ *is the* transition relation*. We require that* m*, restricted to* $S \times \Sigma^* \times S \times \Sigma^*$*, is* recursive*, i.e., its interpretation as the function* $m : S \times \Sigma^* \to 2^{S \times \Sigma^*}$ *is recursively enumerable. Moreover,* m *is such that:*
  - *if* $\langle s, w_i, s_{div}, w_o \rangle \in m$*, then* $w_o = \tau$*, for all* s,w_i*; and*
  - *if* $\langle s_{div}, w_i, s, w_o \rangle \in m$*, then* $w_o = \tau$ *and* $s = s_{div}$*, for all* w_i*.*
- $r \in S$ *is the* initial state *(root).*

We use Σ to encode the states of an ITS. This is for convenience only; any effective encoding will do. Intuitively, a transition $\langle s, w_i, s', w_o \rangle$ of an ITS T means that T, while in state s and having received input string w_i from its environment, transits to state s' and outputs w_o. Moreover, such transitions are effective. Divergent computation is modeled by a τ-transition to the absorbing state s_{div}. We assume that all states in S, with the possible exception of s_{div}, are *reachable* from the root.

We now define three notions of equivalence for ITSs, each of which is successively coarser than the previous one.

Definition 3.2 *Two ITSs* $T_1 = \langle S_1, m_1, r_1 \rangle$ *and* $T_2 = \langle S_2, m_2, r_2 \rangle$ *are* isomorphic*, notation* $T_1 =_{iso} T_2$*, if there exists a bijection* ψ*:* $S_1 \to S_2$ *such that:*

(i) $\psi(r_1) = r_2$

(ii) $\forall w_i, w_o \in \Sigma^*, s, s' \in S : \langle s, w_i, s', w_o \rangle \in m_1$ *iff* $\langle \psi(s), w_i, \psi(s'), w_o \rangle \in m_2$

Definition 3.3 *Let* $T_1 = \langle S_1, m_1, r_1 \rangle$ *and* $T_2 = \langle S_2, m_2, r_2 \rangle$ *be ITSs. A relation* $\mathcal{R} \subseteq S_1 \times S_2$ *is a* (strong) interactive bisimulation *between* T_1 *and* T_2 *if it satisfies:*

(i) $r_1 \mathcal{R} r_2$

(ii) *if* $s \mathcal{R} t$ *and* $\langle s, w_i, s', w_o \rangle \in m_1$*, then there exists* $t' \in S_2$ *with* $\langle t, w_i, t', w_o \rangle \in m_2$ *and* $s' \mathcal{R} t'$*;*

(iii) *if* $s \mathcal{R} t$ *and* $\langle t, w_i, t', w_o \rangle \in m_2$*, then there exists* $s' \in S_1$ *with* $\langle s, w_i, s', w_o \rangle \in m_1$ *and* $s' \mathcal{R} t'$*.*

T_1 *and* T_2 *are* interactively bisimilar*, notation* $T_1 \sim T_2$*, if there exists an interactive bisimulation between them.*

Note that our definition of interactive bisimilarity is such that if $s \mathcal{R} t$, then s is divergent (has a τ-transition to s_{div}) if and only if t is divergent.

Definition 3.4 *Given an ITS* $T = \langle S, m, r \rangle$ *and a state* $s \in S$*,* $ISL(T(s))$ *(the* interactive stream language of T in state s*) is defined as follows:*

$$ISL(T(s)) = \{\langle (w_i, w_o), \sigma' \rangle \in \mathcal{S}_{\Sigma^* \times \Sigma^* \uplus \{\tau\}} \mid \exists s' \in S : \langle s, w_i, s', w_o \rangle \in m \ \wedge$$

$\sigma' \in ISL(T(s'))\}$

$ISL(T)$, the interactive stream language *of T, is defined as $ISL(T(r))$. Two ITSs T_1 and T_2 are* interactive stream equivalent*, notation $T_1 \approx T_2$, if $ISL(T_1) = ISL(T_2)$.*

It is straightforward to show that $=_{iso}$, $\sim$, and $\approx$ are equivalence relations.

Proposition 3.5 *$=_{iso} \subset \sim$ and $\sim \subset \approx$.*

Proof. The proof that ITS isomorphism (strictly) refines interactive bisimilarity is straightforward. The proof that interactive bisimilarity refines interactive stream equivalence is by coinduction on the structure of the interaction streams in $ISL(T_1)$.

To show that interactive bisimilarity strictly refines interactive stream equivalence, consider the following pair of ITSs over alphabet $\Sigma = \{0, 1\}$:

$T_1 = \langle \{r_1, s_1, t_1\}, , m_1, r_1 \rangle$ and $T_2 = \langle \{r_2, s_2\}, m_2, r_2 \rangle$, where
$m_1 = \{\langle r_1, 0, s_1, 1\rangle, \langle r_1, 0, t_1, 1\rangle, \langle s_1, 0, r_1, 1\rangle, \langle t_1, 1, r_1, 0\rangle\}$ and
$m_2 = \{\langle r_2, 0, s_2, 1\rangle, \langle s_2, 0, r_2, 1\rangle, \langle s_2, 1, r_2, 0\rangle\}$.

It is easy to see that $T_1 \approx T_2$ but $T_1 \not\sim T_2$. □

4 Isomorphism of ITS and PTM

In this section, we show that the class of PTMs and the class of ITSs are isomorphic. For this purpose, we assume a fixed alphabet Σ, denote the class of PTMs with alphabet Σ by $\mathcal{M}$, and denote the class of ITSs with alphabet Σ by $\mathcal{T}$.

We begin by defining the "reachable memories" of a PTM. Recalling (Definition 3.1) that the states of an ITS are assumed to be reachable from the ITS's root, reachable memories provide us with an analogous concept for PTMs.

Definition 4.1 *Let $M \in \mathcal{M}$ be a PTM with alphabet Σ. Then $reach(M)$, the* reachable memories *of M, is defined as:*

$$\begin{aligned} reach(M) = \{ w \in \Sigma^* \uplus \{s_{div}\} \mid \\ \exists k \geq 0, \exists w_i^1, \ldots, w_i^k, w_o^1, \ldots, w_o^k, s^1, \ldots, s^k \in \Sigma^* \uplus \{s_{div}\} : \\ \langle w_i^1, \epsilon \rangle \models\!\!\Longrightarrow_M \langle s^1, w_o^1 \rangle, \langle w_i^2, s^1 \rangle \models\!\!\Longrightarrow_M \langle s^2, w_o^2 \rangle, \\ \ldots, \langle w_i^k, s^{k-1} \rangle \models\!\!\Longrightarrow_M \langle s^k, w_o^k \rangle \text{ and } w = s^k \} \end{aligned}$$

As noted above, we will show that $\mathcal{M}$ and $\mathcal{T}$ are isomorphic, preserving natural equivalence relations. For $\mathcal{T}$, the relation in question is ITS isomorphism (Definition 3.2), and for $\mathcal{M}$ it will be *macrostep equivalence*, which we now define.

Definition 4.2 *Two PTMs M_1, M_2 are* macrostep equivalent*, notation $M_1 =_{ms} M_2$, if there exists a bijection ϕ: $reach(M_1) \to reach(M_2)$ such that:*

(i) $\phi(\epsilon) = \epsilon$

(ii) $\forall w_i, w_o \in \Sigma^*, s, s' \in reach(M_1)$:
$\langle w_i, s, \epsilon\rangle \models\!\!\Longrightarrow_{M_1} \langle w_i, s', w_o\rangle$ *iff* $\langle w_i, \phi(s)\rangle \models\!\!\Longrightarrow_{M_2} \langle \phi(s'), w_o\rangle$

The mapping $\xi : \mathcal{M} \to \mathcal{T}$ is given by $\xi(M) = \langle reach(M), m, \epsilon\rangle$, where $\langle s, w_i, s', w_o\rangle \in m$ iff $\langle w_i, s\rangle \models\!\!\Longrightarrow_M \langle s', w_o\rangle$. Note that $\xi(M)$ is indeed an ITS, as $reach(M)$ is enumerable, m is effective, and the set of states of $\xi(M)$ is reachable from its root. By definition, ξ is a transition-preserving isomorphism from the reachable memories of M to the states of T.

Example 4.3 *The ITSs of Figure 1 depict the image, under ξ, of the PTMs M_{Latch} and M'_{Latch} of Example 2.3. Transitions such as $(1*, 0)$ represent the infinite family of transitions where, upon receiving a bit string starting with 1 as input, the ITS outputs a 0.*

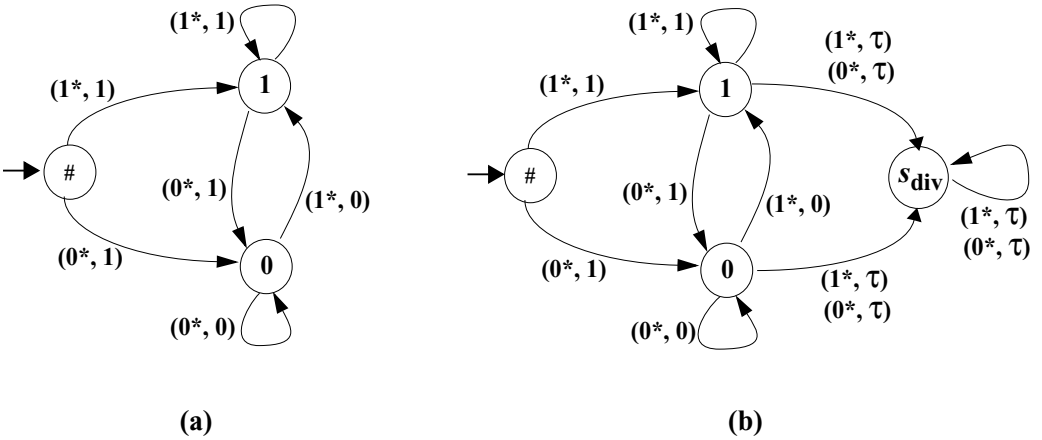

Fig. 1. (a) $\xi(M_{Latch})$ and (b) $\xi(M'_{Latch})$

It is easy to see that persistent stream languages are preserved by ξ.

Proposition 4.4 *For all $M, M' \in \mathcal{M}$, $PSL(M) = ISL(\xi(M))$ and $M =_{PSL} M'$ iff $\xi(M) \approx \xi(M')$.*

The proof uses coinduction to establish a stronger result, namely, if $\sigma \in PSL(M, w)$, for any reachable memory $w \in \Sigma^* \uplus \{s_{div}\}$ of M, then $\sigma \in ISL(T(w))$.

Proof. We prove only one direction, namely that $PSL(M) \subseteq ISL(\xi(M))$; the other direction is analogous. Let $w \in \Sigma^* \uplus \{s_{div}\}$ be a reachable memory of M and σ a stream in $PSL(M, w)$. According to Definition 2.2, there exists $w' \in \Sigma^* \uplus \{s_{div}\}$ such that $\sigma = \langle (w_i, w_o), \sigma'\rangle$ where $\langle w_i, w\rangle \models\!\!\Longrightarrow_M \langle w', w_o\rangle$ and $\sigma' \in PSL(M, w')$.

Let $T = \xi(M)$; by definition, w is a state of T. Since w' is also reachable memory of M, it is also a state of T. We prove coinductively that $\sigma \in ISL(T(w))$. By definition of ξ, $\langle w, w_i, w', w_o\rangle$ is a transition of T. By coinduction, we have that $\sigma' \in ISL(T(w'))$. Therefore, by Definition 3.4, $\sigma \in ISL(T(w))$. Since w was arbitrary, let $w = \epsilon$. It follows that for all $\sigma \in PSL(M)$, it is the case that $\sigma \in ISL(T)$. □

The following proposition shows that ξ maps equivalent PTMs to equivalent ITSs.

Proposition 4.5 *For all $M_1, M_2 \in \mathcal{M}$, $M_1 =_{ms} M_2$ iff $\xi(M_1) =_{iso} \xi(M_2)$.*

Proof. Set ψ in the definition of $=_{iso}$ (Definition 3.2) to the ϕ in the definition of $=_{ms}$ (Definition 4.2), for the $\Rightarrow$-direction of the proof, and vice versa for the $\Leftarrow$-direction of the proof. □

The following proposition shows that ξ is surjective.

Proposition 4.6 *For all $T \in \mathcal{T}$, there exists $M \in \mathcal{M}$ such that $T = \xi(M)$.*

Proof. Let $T = \langle S, m, \epsilon \rangle$. To prove the result, we exhibit a bijective mapping $\omega : S \rightarrow \Sigma^* \uplus \{s_{div}\}$ and a PTM $M \in \mathcal{M}$ such that that $\omega(r) = \epsilon$, $\omega(s_{div}) = s_{div}$ and

$$\langle s, w_i, s', w_o \rangle \in m \text{ iff } \langle w_i, \omega(s) \rangle \Longmapsto_M \langle \omega(s'), w_o \rangle$$

where $w_o \in \Sigma^* \uplus \{\tau\}$. Let $T_0 = \langle S_0, \Sigma, m_0, \epsilon \rangle$, where

$$S_0 = \{\omega(s) \mid s \in S\}; m_0 = \{\langle \omega(s), w_i, \omega(s'), w_o \rangle \mid \langle s, w_i, s', w_o \rangle \in m\}$$

Clearly, $\xi(M) = T$. Also, $T_0 =_{iso} T$, where ω is the desired mapping. □

The main result of this section, which essentially allows one to view persistent Turing machines and interactive transition systems as one and the same, now follows.

Theorem 4.7 *The structures $\langle \mathcal{M}, =_{ms} \rangle$ and $\langle \mathcal{T}, =_{iso} \rangle$ are isomorphic.*

Proof. It follows from Propositions 4.5 and 4.6 that ξ is a structure-preserving bijection. □

5 Equivalence Hierarchy

All stream-based notions of equivalence presented so far for PTMs are relative to infinite streams. In this section, we define equivalences over finite stream prefixes, to obtain an infinite hierarchy of equivalence relations for PTMs. We show that there is a gap between the limit of the hierarchy and PSL equivalence. When proving the existence of this gap, we also demonstrate that PTM computations exhibit unbounded nondeterminism.

We first define the family of stream prefix operators, pref_k.

Definition 5.1 *Let $\mathcal{S}_A$ be the set of streams over some set A of tokens and let $\sigma \in \mathcal{S}_A$. Then $\sigma = \langle a, \sigma' \rangle$ for some $a \in A, \sigma' \in \mathcal{S}_A$. For all $k \geq 1$, $\text{pref}_k(\sigma)$ is defined inductively as follows:*

$$\text{pref}_k(\sigma) = \begin{cases} \langle a, \epsilon \rangle & \text{if } k = 1 \\ \langle a, \text{pref}_{k-1}(\sigma') \rangle & \text{otherwise} \end{cases}$$

We next define the *k-prefix language* of a PTM M, the set of prefixes of length $\leq k$ of the interaction streams in $PSL(M)$. PTMs with the same k-prefix language are called *k-equivalent*.

Definition 5.2 *For any $k \geq 1$ and any PTM M, the* k-prefix language *of M is given by $L_k(M) = \cup_{i \leq k} \{\text{pref}_i(\sigma) \,|\, \sigma \in PSL(M)\}$. Moreover, the pair of PTMs M_1, M_2 are* k-equivalent*, notation $M_1 =_k M_2$, if $L_k(M_1) = L_k(M_2)$.*

Proposition 5.3 *For any $k \geq 1$, $(k+1)$-equivalence strictly refines k-equivalence, i.e., $=_{k+1} \subset =_k$.*

Proof. That $(k+1)$-equivalence refines k-equivalence follows from Definition 5.2. To prove that the refinement is strict, consider the sequence of PTMs ${M_{Ct}}^1, {M_{Ct}}^2, \ldots$, where, for any k, ${M_{Ct}}^k$ is the PTM with binary outputs that ignores its inputs, outputting k 1's and thereafter outputting 0's only.

Essentially, these PTMs are counters, counting off k inputs. It can be shown that for all $k \geq 1$, $L_k({M_{Ct}}^k) = L_k({M_{Ct}}^{k+1})$, but $L_{k+1}({M_{Ct}}^k) \neq L_{k+1}({M_{Ct}}^{k+1})$. This is accomplished by observing that the stream behavior of ${M_{Ct}}^k$ and ${M_{Ct}}^{k+1}$ is identical up to and including stream prefixes of length k, but $< (1,1),(1,1),\ldots,(1,1),(0,0) > \in L_{k+1}({M_{Ct}}^k) - L_{k+1}({M_{Ct}}^{k+1})$. □

Proposition 5.3 establishes an *infinite hierarchy* of stream-based equivalence relations for PTMs, the limit point of which is *∞-equivalence*.

Definition 5.4 *PTMs M_1 and M_2 are called* ∞-equivalent*, notation $M_1 =_\infty M_2$, if $L_\infty(M_1) = L_\infty(M_2)$, where $L_\infty(M) = \cup_{k \geq 1} L_k(M)$.*

Clearly, $=_\infty$ refines $=_k$, for all k. But how do $=_\infty$ and $=_1$ (the end points of the hierarchy) relate to the stream-based equivalences we defined earlier in Section 2? We consider this question in Propositions 5.5 and 6.4.

Proposition 5.5 *PSL-equivalence strictly refines ∞-equivalence, i.e., $=_{PSL} \subset =_\infty$.*

Proof. That PSL-equivalence refines ∞-equivalence follows from the definitions. To prove that the refinement is strict, we define PTMs M_{1*} and M_{1*0}, which ignore their inputs, and output a zero or a one with each macrostep. PTM M_{1*} has a persistent bit b and a persistent string n representing some natural number in unary notation, both of which are initialized at the beginning of the first macrostep. In particular b is nondeterministically set to 0 or 1, and n is initialized to some number of 1's using the following loop:

```
while true do
  write a 1 on the work tape and move head to the right;
  nondeterministically choose to exit the loop or continue
od
```

M_{1*}'s output at every macrostep is determined as follows:

```
if b = 1
```

```
then output 1;
else if n > 0
       then decrement n by 1 and output 1;
       else output 0
```

PTM M_{1^*0} behaves the same as M_{1^*} except that b is always initialized to 0. Now note that the L_∞-languages of these PTMs is the same, consisting of all finite sequences of pairs of the form:

$\{(in_1, out_1), \ldots, (in_k, out_k)\}$,

where the $in_j \in \Sigma^*$ are input tokens and out_j is 1 for the first j pairs in the sequence (for some $j \leq k$) and 0 for the rest (if any). However, $PSL(M_{1^*}) \neq PSL(M_{1^*0})$; in particular, the stream $\{(1,1),(1,1),...\} \in PSL(M_{1^*}) - PSL(M_{1^*0})$. □

The ITS corresponding to M_{1^*0}, i.e., $\xi(M_{1^*0})$, is depicted in Figure 2 and demonstrates that PTMs are capable of exhibiting *unbounded nondeterminism* in a natural way. Though the number of 1's at the beginning of each interaction stream is always finite, it is unbounded. The ITS for M_{1^*} is similar.

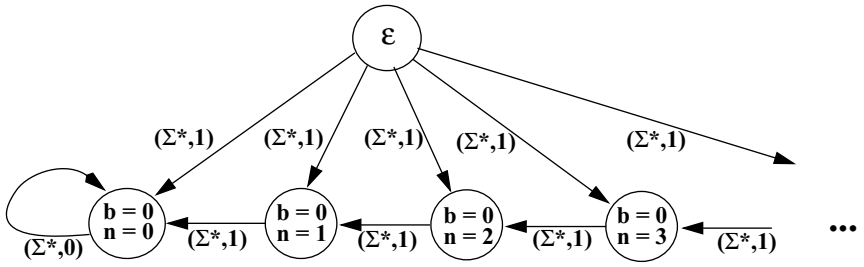

Fig. 2. The ITS corresponding to the PTM M_{1^*0}.

6 Amnesic Stream Computation

In this section, we present the notion of *amnesic* stream computation, where the contents of the persistent work tape is erased (or simply ignored) at each macrostep. We show that *amnesic stream languages* (ASLs) constitute a proper subset of PSLs, and that ASL equivalence coincides with the bottom of the infinite equivalence hierarchy presented in Section 5.

The amnesic stream language for an N3TM is defined similarly to the NT3M's persistent stream language (Definition 2.2). However, each computation of the N3TM begins with a blank work tape; i.e., the N3TM "forgets" the state it was in when the previous computation ended. As before, fix the alphabet of an N3TM to be Σ.

Definition 6.1 *Given an N3TM M, $ASL(M)$ (the* amnesic stream language of *M) is defined as follows:*

$$ASL(M) = \{\langle (w_i, w_o), \sigma' \rangle \in \mathcal{S}_{\Sigma^* \times \Sigma^* \uplus \{\tau\}} \mid \exists w' \in \Sigma^* : \langle w_i, \epsilon \rangle \models\!\!\Longrightarrow_M \langle w', w_o \rangle \wedge \sigma' \in ASL(M)\}.$$

N3TMs M_1 and M_2 are ASL-equivalent*, notation $M_1 =_{ASL} M_2$, if $ASL(M_1) = ASL(M_2)$. We also have that $\mathcal{ASL} = \{ASL(M) \mid M$ is an N3TM$\}$.*

Example 6.2 *The interaction streams in $ASL(M_{Latch})$ (Example 2.3) are of the form $\{(w_1, 1), (w_2, 1), \ldots\}$.*

It is also possible to define amnesic stream languages for ITSs, and a PTM's amnesic stream language would be preserved by the mapping ξ defined in Section 4. The interaction streams contained in the amnesic stream language of an ITS T would be constructed by always returning to T's initial state before moving on to the next input-output token pair in the stream. Although amnesia makes sense for Turing machines—in the classical, non-interactive setting, every Turing-machine computation commences with a blank work tape—its applicability to transition systems is questionable.

The following proposition is used in the proofs of Propositions 6.4 and 6.5.

Proposition 6.3 *Given an N3TM M, let $L(M)$ be defined as follows:*

$$\{(w_i, w_o) \in \Sigma^* \times \Sigma^* \uplus \{\tau\} \mid \exists w' \in \Sigma^* : \langle w_i, \epsilon \rangle \models\!\!\Longrightarrow_M \langle w', w_o \rangle\}$$

Then, $ASL(M) = \mathcal{S}_{L(M)}$, the set of all streams over $L(M)$.

Proposition 6.4 $=_{ASL} \; = \; =_1$

Proposition 6.5 $\mathcal{ASL} \subset \mathcal{PSL}$

Proof. To prove $\mathcal{ASL} \subseteq \mathcal{PSL}$, it suffices to show that, given an N3TM M, we can construct an N3TM M' such that $PSL(M') = ASL(M)$. The construction is as follows:

> M' always starts its computation by erasing the contents of its work tape and moving the work-tape head back to beginning of tape; it then proceeds just like M.

From Definitions 2.2 and 6.1, it follows that $PSL(M') = ASL(M)$.

To prove that the inclusion of $\mathcal{ASL}$ in $\mathcal{PSL}$ is strict, we refer to M_{Latch} and $\sigma_{io} \in PSL(M_{Latch})$ defined in Example 2.3 to show that there does not exist an N3TM M such that $ASL(M) = PSL(M_{Latch})$. Assume such an N3TM M exists; then, $\sigma_{io} \in ASL(M)$. Therefore, by Proposition 6.3, $(0, 0)$, the third element of σ_{io}, is in $L(M)$. This in turn implies that there are interaction streams in $ASL(M)$ starting with $(0, 0)$. But no stream in $PSL(M_{Latch})$ can start with $(0, 0)$, leading to a contradiction. Therefore, no such M exists. □

We say that a PTM M is *amnesic* if $PSL(M) \in \mathcal{ASL}$.

Example 6.6 *M_{Latch} is not amnesic. Neither are the M_{Ct} PTMs defined in the proof of Proposition 5.3. Even though they ignore their input values, these*

PTMs remember the number *of inputs they have consumed, and are therefore not amnesic.*

On the other hand, some recently proposed extensions of Turing-machine computation to the stream setting do not capture persistence. For example, the squaring machine *of [PR98, Figure 1], which repeatedly accepts an integer n from its environment and outputs n^2, is clearly amnesic.*

Most of the results obtained in this paper rely on the persistence of PTMs; that is, they do not hold if we restrict our attention to amnesic PTMs. For example, the whole equivalence hierarchy collapses in this case.

Proposition 6.7 *For any pair of amnesic PTMs M_1 and M_2, $M_1 =_{ASL} M_2$ iff $M_1 =_{PSL} M_2$.*

7 Related Work

The notions of persistency and interaction embodied in PTMs and ITSs can be found in one form or another in various models of reactive computation including dataflow and related areas [KM77,PS88,RT90,PSS90,KP93,BE94], process calculi [Mil89,MPW92], synchronous languages [Har87,BG92], finite/pushdown automata over infinite words [EHRS00,BCMS01], interaction games [Abr00], reactive modules [AH99], and I/O automata[Lyn96]. The main difference between these approaches and our own is that our focus is on the relationship between Turing machines and transition systems, and on the effects of unbounded nondeterminism on the equivalence hierarchy for PTMs. The other approaches tend to emphasize issues such as correctness and programming, and the computability of a transition step is often left implicit. Moreover, these models of computation are typically purely functional in nature, and, therefore, the notion of persistency or "state" present in PTMs is absent.

Persistency, however, can be captured in dataflow models by "feedback loops" and in process calculi by explicitly modeling the data store. For example, PTM M_{Latch} of Example 2.3 can be modeled in a dataflow setting by the stream transformer $f(s) = (1, s)$, which can be evaluated lazily/on-the-fly. M_{Latch} is a simple example of a PTM: its history dependence only goes back one interaction in time and PTMs are in general capable of expressing history dependence of an unbounded nature. It would therefore be interesting to determine whether stream transformers can encode the behavior of *all* PTMs.

Persistent Turing machines formalize the notion of *Sequential Interaction Machines* introduced in earlier papers by the first and third authors, including [Weg98,GST00]. A major emphasis of this body of work is to show how such a computational framework can be used as a basis for modeling various forms of interactive computing, such as object-oriented, agent-based, and dynamical systems. PTMs also formalize the notion of *embedded components*, according to the criteria presented in [LW00a].

An alternative approach to extending the Turing-machine model to interactive computation is captured by the *Interactive Turing Machines with Advice* (ITMAs) of [LW00b]. Like PTMs, ITMAs are persistent, interactive, and stream-based. Additionally, they incorporate several features aimed at capturing "practical" computing devices, including multiple input/output ports and advice, a kind of oracle modeling hardware and software upgrades. In contrast, PTMs, which have single input and output tapes and do not appeal to oracles, represent a minimal extension to the classical Turing-machine model (persistence of the work tape) needed to attain transition-system expressiveness.

8 Conclusions

We have presented Persistent Turing Machines (PTMs), a stream-based extension of the Turing-machine model with appropriate notions of interaction and persistency. A number of expressiveness results concerning PTMs have been presented, including the expressive equivalence of PTMs and interactive transition systems; the strict inclusion of the set $\mathcal{ASL}$ of amnesic stream languages in the set $\mathcal{PSL}$ of persistent stream languages (showing that "persistence pays"); the "gap" between the limit of the equivalence hierarchy based on finite interaction-stream prefixes and PSL-equivalence; and the collapse of the equivalence hierarchy in the case of amnesic PTMs.

Our results are summarized in Figure 3.

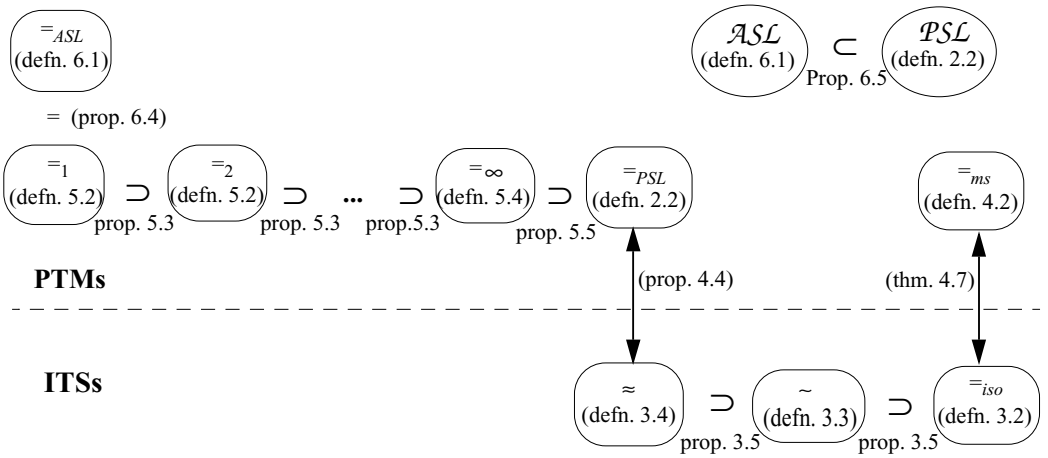

Fig. 3. Summary of results.

It should be noted that, by virtue of our isomorphism result, every equivalence defined for PTMs can be carried over to ITSs, and vice versa. For example, a relation can be defined for PTMs that is analogous to ITS bisimulation; by contrast, bisimulation makes no sense in the traditional Turing-machine context. On the other hand, the transition-system analog of ASL equivalence makes little sense, even though it is natural in the traditional (i.e., non-stream-based) Turing-machine world.

As ongoing work, we are developing a model of PTM computation where PTMs execute concurrently and communicate with each other through their input and output tapes. We conjecture that concurrent PTMs are more expressive than sequential ones in terms of the stream languages they produce. We are also interested in developing a "weak" theory of persistent stream languages and interactive bisimulation in which divergent computation (τ-transitions) is abstracted away.

Acknowledgement

We would like to thank Peter Fejer and the anonymous referees for their valuable comments, and Paul Attie for bringing to our attention the phenomenon of unbounded nondeterminism in PTMs.

References

[Abr00] S. Abramsky. Concurrent interaction games. In J. Davies, A. W. Roscoe, and J. Woodcock, editors, *Millenial Perspectives in Computer Science*, pages 1–12. Palgrave, 2000.

[AH99] R. Alur and T. A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the consistency of Koomen's fair abstraction rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[BCMS01] O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*. Elsevier, 2001.

[BE94] A. Bucciarelli and T. Ehrhard. Sequentiality in an extensional framework. *Information and Computation*, 110(2):265–296, 1994.

[BG92] G. Berry and G. Gonthier. The ESTEREL synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19:87–152, 1992.

[BIM88] B. S. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. In *Proceedings of the 15th ACM Symposium on Principles of Programming Languages*, 1988.

[BM96] J. Barwise and L. Moss. *Vicious Circles*. CSLI Lecture Notes #60. Cambridge University Press, 1996.

[Bou85] G. Boudol. Notes on algebraic calculi of processes. In K. Apt, editor, *Logics and Models of Concurrent Systems*, pages 261–303. LNCS, Springer-Verlag, 1985.

[BR85] S. D. Brookes and A. W. Roscoe. An improved failures model for communicating sequential processes. In *Proceedings NSF-SERC Seminar on Concurrency*. Springer-Verlag, 1985.

[Dar90] P. Darondeau. Concurrency and computability. In I. Guessarian, editor, *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science*, La Roche Posay, France, 1990. LNCS **469**, Springer-Verlag.

[dS85] R. de Simone. Higher-level synchronizing devices in meije-sccs. *Theoretical Computer Science*, 37:245–267, 1985.

[EHRS00] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon. Efficient algorithms for model checking pushdown systems. In *Proc. of CAV'2000*, pages 232–247. number 1855 in Lecture Notes in Computer Science, Springer-Verlag, 2000.

[GST00] D. Goldin, S. Srinivasa, and B. Thalheim. Information systems = databases + interaction: Towards principles of information system design. In *Proceedings of ER 2000*, Salt Lake City, Utah, 2000.

[Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[KM77] G. Kahn and D. B. MacQueen. Coroutines and networks of parallel processes. In *Proc. of the IFIP Congress 77*. North-Holland, 1977.

[KP93] G. Kahn and Gordon D. Plotkin. Concrete domains. *Theoretical Computer Science*, 121(1&2):187–277, 1993.

[LW00a] J. van Leeuwen and J. Wiedermann. On algorithms and interaction. In *Proc. of MFCS'2000*, Bratislava, Slovak Republic, August 2000. Springer-Verlag.

[LW00b] J. van Leeuwen and J. Wiedermann. The Turing machine paradigm in contemporary computing. In B. Enquist and W. Schmidt, editors, *Mathematics Unlimited - 2001 and Beyond*. LNCS, Springer-Verlag, 2000.

[Lyn96] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100, 1992.

[Pat90] M.S. Paterson, editor. *Automata, Languages and Programming (ICALP '90)*, volume 443 of *Lecture Notes in Computer Science*, Warwick, England, July 1990. Springer-Verlag.

[PR98] M. Prasse and P. Rittgen. Why Church's thesis still holds: Some notes on Peter Wegner's tracts on interaction and computability. *Computer Journal*, 41(6), 1998.

[PS88] P. Panangaden and E. W. Stark. Computations, residuals, and the power of indeterminancy. In *Proceedings of 15th ICALP*, pages 439–454. Springer-Verlag, Lecture Notes in Computer Science, Vol. 317, 1988.

[PS91] A. Pnueli and M. Shalev. What is in a step: On the semantics of Statecharts. In *Theoretical Aspects of Computer Software*, number 526 in Lecture Notes in Computer Science, pages 244–264, 1991.

[PSS90] P. Panangaden, V. Shanbhogue, and E. W. Stark. Stability and sequentiality in dataflow networks. In Paterson [Pat90], pages 308–321.

[RT90] A. M. Rabinovich and B. A. Trakhtenbrot. Communication among relations. In Paterson [Pat90], pages 294–307.

[Vaa93] F. W. Vaandrager. Expressiveness results for process algebras. Technical Report CS-R9301, Centrum voor Wiskunde en Informatica, Amsterdam, 1993.

[Weg98] P. Wegner. Interactive foundations of computing. *Theoretical Computer Science*, 192, February 1998.