Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information and Computation 194 (2004) 101–128
Information and Computation
www.elsevier.com/locate/ic

Turing machines, transition systems, and interaction

Dina Q. Goldin,[a,*] Scott A. Smolka,[b] Paul C. Attie,[c] Elaine L. Sonderegger[a]

[a] *Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269, USA*
[b] *Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, USA*
[c] *College of Computer Science, Northeastern University, Boston, MA 02115, USA*

Received 24 September 2003; revised 24 March 2004
Available online 22 September 2004

Abstract

This paper presents *persistent Turing machines* (PTMs), a new way of interpreting Turing-machine computation, based on dynamic stream semantics. A PTM is a Turing machine that performs an infinite sequence of "normal" Turing machine computations, where each such computation starts when the PTM reads an input from its input tape and ends when the PTM produces an output on its output tape. The PTM has an additional worktape, which *retains* its content from one computation to the next; this is what we mean by *persistence*. A number of results are presented for this model, including a proof that the class of PTMs is isomorphic to a general class of effective transition systems called *interactive transition systems*; and a proof that PTMs without persistence (*amnesic PTMs*) are less expressive than PTMs. As an analogue of the Church-Turing hypothesis which relates Turing machines to algorithmic computation, it is hypothesized that PTMs capture the intuitive notion of *sequential interactive computation.*

Keywords: Models of interactive computation; Persistent Turing machine; Persistent stream language; Interactive transition system; Sequential interactive computation

* Corresponding author.
E-mail addresses: dqg@engr.uconn.edu, dqg@cse.uconn.edu (D.Q. Goldin), sas@cs.sunysb.edu (S.A. Smolka), attie@ccs.neu.edu (P.C. Attie), esonderegger@engr.uconn.edu (E.L. Sonderegger).
URLs: www.cse.uconn.edu/~dqg (D.Q. Goldin), www.cs.sunysb.edu/~sas (S.A. Smolka), www.ccs.neu.edu/groups/faculty/attie.html (P.C. Attie), www.cse.uconn.edu/~esonderegger (E.L. Sonderegger).

0890-5401/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2004.07.002

Preface

Interaction was a common theme in the research of Paris Kanellakis. His doctoral dissertation explored the computational complexity of concurrency control in distributed databases. Later, his research interests included object-oriented and constraint databases, complexity issues in process algebra and other formalisms for concurrent systems, and fault-tolerant parallel algorithms. Given that interaction is a hallmark of each of these areas and that the second author (Smolka) was Paris's first Ph.D. student and the first author (Goldin) was one of his last Ph.D. students, a paper on a formal framework for interactive computing seems appropriate for the special issue of *Information and Computation* commemorating the anniversary of Paris's 50th birthday. A preliminary version of this paper appeared in [1].

1. Introduction

A number of researchers have 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*. Milner, in his Turing Award lecture [2], asserts that:

> Through the seventies, I became convinced that a theory of concurrency and interaction requires a new conceptual framework, not just a refinement of what we find natural for sequential computing.

In [3], van Leeuwen states:

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

Wegner [4,5] 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 (N3TM) with a read-only input tape, a read/write work tape, and a write-only output tape. Upon receiving an input token from its environment on its input tape, a PTM computes for a while and then outputs the result to the environment on its output tape, and this process is repeated forever. A PTM performs *persistent computations* in the sense that a notion of "memory" (work-tape contents) is maintained from one computation step to the next, where each PTM computation step represents an N3TM computation. Fig. 1 illustrates the first two steps of a PTM computation.

Persistence extends the effect of inputs. An input token affects the computation of its corresponding macrostep, including the work tape. The work tape in turn affects subsequent computation steps. If the work tape were erased, then the input token could not affect subsequent macrosteps, but only

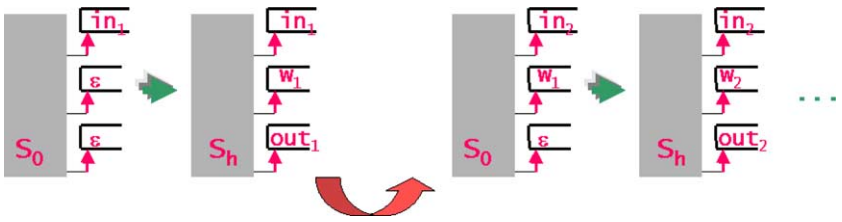

Fig. 1. Illustration of PTM macrosteps.

"its own" macrostep. With persistence, an input token can affect all subsequent macrosteps; this property is known as *history dependence*.

Our treatment of PTMs proceeds along the following lines. We first formalize the notions of interaction and persistence in PTMs in terms of the *persistent stream language* (PSL) of a PTM (Sections 2 and 3). Given a PTM, its persistent stream language is coinductively defined to be the set of infinite sequences (interaction streams) of pairs of the form (w_i, w_o) representing the input and output strings of a single PTM computation step. Persistent stream languages induce a natural, stream-based notion of equivalence for PTMs. *Decider PTMs* are an important subclass of PTMs; a PTM is a *decider* if it does not have divergent (non-halting) computations (Definitions 6 and 11).

We then define a very general kind of effective transition system called an *interactive transition system* (ITS), and equip ITSs with three notions of behavioral equivalence: *ITS isomorphism*, *interactive bisimulation*, and *interactive stream equivalence* (Section 4). We show that ITS isomorphism refines interactive bisimulation, and interactive bisimulation refines interactive stream equivalence.

Our first 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 5). A similar result is established for decider PTMs and decider ITSs. These results address a question heretofore left unanswered concerning the relative expressive power of Turing machines and transition systems. Until 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 [6–11]. The other direction, namely—What extensions are required of Turing machines so that they can simulate transitions systems?—is answered by our results.

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 6). The presence of this "gap" can be attributed to the fact that the transition systems corresponding to PTM computations naturally exhibit *unbounded nondeterminism*. This is an important phenomenon for *specification*; for example, modeling unbounded nondeterminism is crucial for supporting refinement between dialects of *timed* and *untimed CSP* [12]. In contrast, it is well known that classical Turing-machine computations have bounded nondeterminism, i.e., any nondeterministic TM can produce only a finite number of distinct outputs for a given input string. We note that this gap is not present for decider PTMs (Section 7).

We further introduce the class of *amnesic* PTMs and a corresponding notion of *amnesic stream languages* (ASL) (Section 8). In this case, the PTM begins each new computation with a blank work tape. We show that the class of ASLs is strictly contained in the class of PSLs. 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. ASLs 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.

The notion of a *universal PTM* is introduced next and a proof of its existence is presented (Section 9). Similarly to a *universal Turing machine*, a *universal PTM* can simulate the behavior of an arbitrary PTM. We also introduce the class of *sequential interactive computations* (Section 10), illustrating it with several examples. In an analogous fashion to the Church-Turing Thesis, we hypothesize that anything intuitively computable by a sequential interactive computation can be computed by a persistent Turing machine. This hypothesis, when combined with results earlier in the paper, implies that the class of sequential interactive computations is more expressive than the class of algorithmic computations, and thus is capable of solving a wider range of problems.

Finally, a discussion of related work (Section 11) and our concluding remarks are given (Section 12).

2. Nondeterministic 3-tape Turing machines

In this section, we define the notion of a nondeterministic 3-tape Turing machine. The definition is standard as far as Turing machines go (see e.g. [13]), modulo the fact that a 3-tape machine comes equipped with three tapes rather than one. We subsequently define how computation proceeds on a 3-tape machine.

Definition 1 (*3TM*)**.** A *nondeterministic 3-tape Turing machine* (N3TM) is a quadruple $\langle K, \Sigma, \delta, s_0 \rangle$ where:

- K is a finite set of *states*.
- Σ is a finite *alphabet* containing the blank symbol #, but not containing $\mathtt{L}$ (*left*) and $\mathtt{R}$ (*right*).
- $\delta \subseteq K \times \Sigma \times \Sigma \times \Sigma \times (K \cup \{\mathtt{h}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\})$ is the *transition relation*.
- $s_0 \in K$ is the *initial state*.
- $\mathtt{h} \notin K$ is the *halting state*.

An N3TM is *deterministic* (D3TM) if δ is a function $\delta : K \times \Sigma \times \Sigma \times \Sigma \to (K \cup \{\mathtt{h}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\}) \times (\Sigma \cup \{\mathtt{L}, \mathtt{R}\})$.

An N3TM has three semi-infinite tapes. Each of these tapes has an associated tape head and corresponding tape-head position. An N3TM makes a transition from its current state based on the (possibly blank) symbols found on the tapes at the current tape-head positions. Such a transition will take it to a new state (possibly the halt state $\mathtt{h}$) and for each of the three tapes, either a new symbol will be written at the current head position or the position of the head will be shifted by one location to the left ($\mathtt{L}$) or right ($\mathtt{R}$).

The following definition of an N3TM configuration is also standard. The contents of a tape refer to the string of symbols from the tape's beginning up through its last non-blank symbol. $\mathbb{N}$ is the natural numbers.

Definition 2 (*3TM configuration*)**.** Let $M = \langle K, \Sigma, \delta, s_0 \rangle$ be an N3TM. A *configuration* of M is a septuple $\langle s, w_1, w_2, w_3, n_1, n_2, n_3 \rangle$, where:

- $s \in K$ is the *state* of the configuration.
- $w_1, w_2, w_3 \in \Sigma^*$ are the *contents* of M's three tapes.
- $n_1, n_2, n_3 \in \mathbb{N}$ are the *tape head positions* for M's three tapes, respectively.

Let w be a word in Σ^*, $n \in \mathbb{N}$, and $c \in \Sigma$. Then $w[n]$ denotes the n th character of w, and $w[c/n]$ denotes w with its nth character replaced by c.

Definition 3 (*Microstep*)**.** Let M be an N3TM and C, C' two configurations of M:

$$C = \langle s, w_1, w_2, w_3, n_1, n_2, n_3 \rangle, \quad C' = \langle s', w_1', w_2', w_3', n_1', n_2', n_3' \rangle.$$

We say that $C \mapsto C'$ (*yields in one microstep*) if $\langle s, w_1[n_1], w_2[n_2], w_3[n_3], s', c_1, c_2, c_3 \rangle \in \delta$ and, for $i = 1, 2, 3$:

$n_i' = n_i + 1$ if $c_i = \mathtt{R}$, $\qquad w_i' = w_i$ if $c_i = \mathtt{L}$ or $c_i = \mathtt{R}$,
$n_i' = n_i - 1$ if $c_i = \mathtt{L}$ and $n_i \neq 1$, $\qquad w_i' = w_i[c_i/n_i]$ otherwise.
$n_i' = n_i$ otherwise.

Definition 4 (*Microsequence*)**.** Let M be an N3TM and C, C' two configurations of M. We say that $C \overset{*}{\longmapsto} C'$ (*yields in zero or more microsteps*) if there exist configurations $C_0, \ldots, C_n$ for some $n \geqslant 0$ such that $C = C_0$, $C' = C_n$, and $C_i \mapsto C_{i+1}$ for $0 \leqslant i < n$. $\overset{*}{\longmapsto}$ is the reflexive transitive closure of $\mapsto$.

3. Persistent Turing machines

In this section, we show how classical Turing machines, N3TMs in particular, can be reinterpreted as interactive computing devices. The basic idea is to view the three tapes of an N3TM as a read-only input tape, a read/write work tape, and a write-only output tape, respectively. It is then possible to introduce the concept of *interaction streams*: infinite sequences of token pairs of the form (w_i, w_o). Each such pair represents a computation performed by an N3TM, producing output 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 refer to an N3TM of this nature as a *persistent Turing machine* (PTM).

Definition 5 (*Persistent Turing machine*)**.** A *persistent Turing machine* (PTM) is an N3TM having a read-only *input* tape, a read/write *work* tape, and a write-only *output* tape.

A *macrostep* is a shorthand notation for a (possibly divergent) computation step of a PTM. Our choice of terminology is inspired by the treatment of Statechart semantics in [14].

Definition 6 (*Macrostep*)**.** Let M be a PTM having alphabet Σ, and let w_i, w, w', and w_o be words over Σ. We say that $w \xrightarrow[M]{w_i/w_o} w'$ *(yields in one macrostep)* if M, when started in its initial control state with its heads at the beginning of its input, work, and output tapes, containing w_i, w, and ϵ, respectively, has a halting computation that produces w_i, w', and w_o as the respective contents of its input, work, and output tapes.

Should M's computation diverge, we write $w \xrightarrow[M]{w_i/\mu} s_{\text{div}}$, where $s_{\text{div}}, \mu \notin \Sigma^*$, and s_{div} is a special "divergence state" such that $s_{\text{div}} \xrightarrow[M]{w_i/\mu} s_{\text{div}}$ for all inputs w_i, and μ is a special output symbol signifying divergence.

The contents of the input tape are 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 the 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 contents of all tapes are assumed to be blank at the start of computation. Since the underlying N3TM transitions of a macrostep computation may be nondeterministic, a macrostep transition from one work-tape contents to another work-tape contents also may be nondeterministic.

Divergent computations of the underlying N3TM bring the PTM to the divergence state s_{div}, a special absorbing state not in Σ^* that outputs μ in conjunction with the current and all subsequent inputs. Our treatment of divergence is consistent with the failures-divergence refinement model of Theoretical CSP [12]. In CSP, once a process diverges (i.e., initiates an infinite sequence of internal actions), it is considered to be acting chaotically and able to do or refuse anything. This means that processes are considered to be identical after they have diverged. In our model, all diverging PTMs enter the divergence state s_{div} and remain there, outputting μ upon all subsequent inputs. Thus, like in CSP, PTMs are considered to be identical after they have diverged.

Fig. 1 graphically illustrates the concept of a macrostep, depicting the first two macrosteps of a PTM computation. The inputs for these macrosteps are in_1 and in_2, while the corresponding outputs are out_1 and out_2. The work-tape contents, which is initially ϵ, becomes w_1 at the end of the first macrostep and w_2 at the end of the second macrostep. The curved arrows between the macrosteps reflect the interactive nature of PTMs, when the input and the output tapes of the PTM are modified by its environment.

To formally define interaction streams and persistent stream languages, fix the alphabet of a PTM to be Σ, and let A be a recursively enumerable set of *action tokens*. $\mathbb{S}_A$, the class of streams over A, is defined coinductively [15] as follows: $\mathbb{S}_A = A \times \mathbb{S}_A$. Then the class of interaction streams is given by $\mathbb{S}_{\Sigma^* \times (\Sigma^* \cup \{\mu\})}$. We thus have that interaction streams are pairs of the form $\langle (w_i, w_o), \sigma' \rangle$ with $(w_i, w_o) \in \Sigma^* \times (\Sigma^* \cup \{\mu\})$ and $\sigma' \in \mathbb{S}_{\Sigma^* \times (\Sigma^* \cup \{\mu\})}$. The coinductive style of definition we have employed for interaction streams will allow us to apply coinduction as a proof technique later in the paper.

Definition 7 (*Persistent stream language*)**.** Given a PTM M and some $w \in \Sigma^* \cup \{s_{\text{div}}\}$, $PSL(M, w)$, the *persistent stream language of M with state w*, is defined as follows:

$$PSL(M, w) = \{\langle (w_i, w_o), \sigma' \rangle \in \mathbb{S}_{\Sigma^* \times (\Sigma^* \cup \{\mu\})} \mid \exists w' \in \Sigma^* \cup \{s_{\text{div}}\} : w \xrightarrow[M]{w_i/w_o} w' \wedge \sigma' \in PSL(M, w')\}.$$

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

Example 8. Consider the PTM 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). $PSL(M_{\text{Latch}})$ therefore contains interaction streams of the form

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

where $w[i]$ denotes the ith 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 $\sigma_{\text{io}} \in PSL(M_{\text{Latch}})$ would be of the form:

$$\sigma_{\text{io}} = [(1,1), (0,1), (0,0), (1,0), \ldots].$$

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

For an interaction stream $\sigma \in PSL(M_{\text{Latch}})$, $PSL(M'_{\text{Latch}})$ will contain σ as well as, for all $k > 1$, a k-divergent version of σ where the computation diverges at the kth 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 μ. For example, a 3-divergent version of σ_{io} is $[(1,1), (0,1), (0,\mu), (1,\mu), \ldots]$.

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 by the environment, possibly in response to outputs generated previously by the PTM. Therefore, σ cannot always be computed in advance.

The following is another, more practical, example of a PTM.

Example 9. An *answering machine AM* is a deterministic PTM whose work tape contains a sequence of recorded messages and whose operations are `record`, `play`, and `erase`. The Turing-computable function for AM is:

$$f_{\text{AM}}(\texttt{record}\ Y, X) = (\texttt{ok}, XY)$$
$$f_{\text{AM}}(\texttt{play}, X) = (X, X)$$
$$f_{\text{AM}}(\texttt{erase}, X) = (\texttt{done}, \epsilon)$$

Note that both the content of the work tape and the length of input for recorded messages are unbounded.

AM's behavior is such that for the input stream `[record A, erase, record BC, record D, play, ...]`, it generates the output stream `[ok, done, ok, ok, BCD, ...]`, resulting in the following interaction stream:

$$[(\texttt{record A}, \texttt{ok}), (\texttt{erase}, \texttt{done}), (\texttt{record BC}, \texttt{ok}), (\texttt{record D}, \texttt{ok}), (\texttt{play}, \texttt{BCD}), \ldots]$$

This example illustrates how sequential objects can be modeled by PTMs. It also underscores the need for a dynamic semantics of input/output streams in the PTM model.

Returning to our formal discussion of PTMs, we define the state space of a PTM to be those states that are reachable during a computation beginning with a blank work tape.

Definition 10 (*Reachable states of a PTM*)**.** Let M be a PTM with alphabet Σ. Then $reach(M)$, the *reachable states* of M, is defined as:

$$\begin{aligned} reach(M) = \{\epsilon\} \cup \{w \in \Sigma^* \cup \{s_{\text{div}}\} \mid & \exists k \geqslant 1,\ \exists w_i^1, \ldots, w_i^k \in \Sigma^*, \\ & \exists w_o^1, \ldots, w_o^k \in \Sigma^* \cup \{\mu\},\ \exists w^1, \ldots, w^k \in \Sigma^* \cup \{s_{\text{div}}\} : \\ & \epsilon \xrightarrow[M]{w_i^1/w_o^1} w^1,\ w^1 \xrightarrow[M]{w_i^2/w_o^2} w^2,\ \ldots,\ w^{k-1} \xrightarrow[M]{w_i^k/w_o^k} w^k \wedge w = w^k\}. \end{aligned}$$

An important subclass of Turing machines are "deciders": those that halt on all inputs, making a decision to accept or reject. In a similar fashion, we define the subclass of decider PTMs as those that have no divergent computations. Note that M_{Latch} and AM described above belong to this subclass.

Definition 11 (*DPTM*)**.** Let M be a PTM with alphabet Σ. Then M is a *decider PTM* if:

$$\forall w \in reach(M)\ \forall w_i \in \Sigma^*,\ w \xrightarrow[M]{w_i/w_o} w' \text{ is a halting computation, i.e., } w' \neq s_{\text{div}}.$$

If M is not a decider, i.e., if there exist some $w \in reach(M)$, $w_i \in \Sigma^*$ such that $w \xrightarrow[M]{w_i/\mu} s_{\text{div}}$, we say that *M diverges.*

It is easy to see that a PTM M is a decider if and only if $s_{\text{div}} \notin reach(M)$.

4. Interactive transition systems

In this section, we introduce a kind of "effective" transition system (see, for example [11]) that we shall refer to as an "interactive transition system." We show that interactive transition systems are isomorphic to PTMs (Theorem 24).

Definition 12 (*Interactive transition system*)**.** Given a finite alphabet Σ not containing μ, an *interactive transition system* (ITS) is a triple $\langle S, m, r\rangle$ where:

- $S \subseteq \Sigma^* \cup \{s_{\text{div}}\}$ is the set of *states*, where $s_{\text{div}} \notin \Sigma^*$ is a special "divergence" state.
- $m \subseteq S \times \Sigma^* \times S \times (\Sigma^* \cup \{\mu\})$ is the *transition relation*. We require that m, interpreted as the function $m : S \times \Sigma^* \to 2^{S \times (\Sigma^* \cup \{\mu\})}$, is a partial recursive function. Moreover, m is such that:
  - if $\langle s, w_i, s_{\text{div}}, w_o\rangle \in m$, then $w_o = \mu$, for all s,w_i; and
  - if $\langle s_{\text{div}}, w_i, s, w_o\rangle \in m$, then $s = s_{\text{div}} \wedge w_o = \mu$, for all w_i.
- $r \in S$ is the *initial state* (root).
- all the states of S are reachable from r.

An ITS is *deterministic* if m is a partial recursive function $m : S \times \Sigma^* \to S \times (\Sigma^* \cup \{\mu\})$.

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 computable. Divergent computation is modeled by a μ-transition to the absorbing state s_{div}.

A decider ITS is an ITS none of whose transitions are divergent, i.e., no transition places the ITS in s_{div}.

Definition 13 (*Decider interactive transition system*)**.** Given a finite alphabet Σ not containing μ, a *decider interactive transition system* (DITS) is a triple $\langle S, m, r\rangle$ where:

- $S \subseteq \Sigma^*$ is the set of *states*.
- $m \subseteq S \times \Sigma^* \times S \times \Sigma^*$ is the *transition relation*. We require that m, interpreted as the function $m : S \times \Sigma^* \to 2^{S \times \Sigma^*}$, is a total recursive function.
- $r \in S$ is the *initial state* (root).
- All the states of S are reachable from r.

A DITS is *deterministic* if m is a total recursive function $m : S \times \Sigma^* \to S \times \Sigma^*$.

Proposition 14. *Every DITS is an ITS.*

Proof. Follows from Definition 13. □

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

Definition 15 (*ITS isomorphism*)**.** 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 $T_1 =_{\text{iso}} T_2$, if there exists a bijection ψ: $S_1 \to S_2$ such that:

(1) $\psi(r_1) = r_2$,
(2) $\forall w_i \in \Sigma^*, w_o \in \Sigma^* \cup \{\mu\}, s, s' \in S : \langle s, w_i, s', w_o\rangle \in m_1$ iff $\langle \psi(s), w_i, \psi(s'), w_o\rangle \in m_2$.

Definition 16 (*ITS bisimulation*)**.** 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 $\mathbb{R} \subseteq S_1 \times S_2$ is a *(strong) interactive bisimulation* between T_1 and T_2 if it satisfies:

(1) $r_1 \,\mathbb{R}\, r_2$,
(2) if $s \,\mathbb{R}\, t \wedge \langle s, w_i, s', w_o\rangle \in m_1$, then $\exists t' \in S_2$ with $\langle t, w_i, t', w_o\rangle \in m_2 \wedge s' \,\mathbb{R}\, t'$,
(3) if $s \,\mathbb{R}\, t \wedge \langle t, w_i, t', w_o\rangle \in m_2$, then $\exists s' \in S_1$ with $\langle s, w_i, s', w_o\rangle \in m_1 \wedge s' \,\mathbb{R}\, t'$.

T_1 and T_2 are *interactively bisimilar*, notation $T_1 =_{\text{bis}} T_2$, if there exists an interactive bisimulation between them.

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

Definition 17 (*Interactive stream language*)**.** 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:

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

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

It is straightforward to show that $=_{\text{iso}}$, $=_{\text{bis}}$, and $=_{\text{ISL}}$ are equivalence relations.

Proposition 18. $=_{\text{iso}} \subset =_{\text{bis}}$ *and* $=_{\text{bis}} \subset =_{\text{ISL}}$.

Proof. The proof that ITS isomorphism (strictly) refines interactive bisimilarity is straightforward. To show that interactive bisimilarity refines interactive stream equivalence, suppose $T_1 =_{\text{bis}} T_2$. Then there exists an interactive bisimulation $\mathbb{R}$ between T_1 and T_2 such that $r_1 \,\mathbb{R}\, r_2$. Now let $\langle (w_i, w_o), \sigma_1\rangle$ be an arbitrary interaction stream in $ISL(T_1)$. In this case, $\exists s_1 \in S_1$ such that $\langle r_1, w_i, s_1, w_o\rangle \in m_1$ and $\sigma_1 \in ISL(T_1(s_1))$. Since $r_1 \,\mathbb{R}\, r_2$, $\exists s_2 \in S_2$ such that $\langle r_2, w_i, s_2, w_o\rangle \in m_2$ and $s_1 \,\mathbb{R}\, s_2$. This in turn implies that there exists an interaction stream $\langle (w_i, w_o), \sigma_2\rangle \in ISL(T_2)$ with $\sigma_2 \in ISL(T_2(s_2))$. By coinduction, we have that $ISL(T_1(s_1)) = ISL(T_2(s_2))$ and, since the interaction stream in $ISL(T_1)$ we considered above was arbitrary, $ISL(T_1) = ISL(T_2)$. This yields $T_1 =_{\text{ISL}} T_2$ as desired.

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:

$$m_1 = \{\langle r_1, 0, s_1, 1\rangle, \langle r_1, 0, t_1, 1\rangle, \langle s_1, 0, r_1, 1\rangle, \langle t_1, 1, r_1, 0\rangle\}$$
$$m_2 = \{\langle r_2, 0, s_2, 1\rangle, \langle s_2, 0, r_2, 1\rangle, \langle s_2, 1, r_2, 0\rangle\}$$

It is easy to see that $T_1 =_{\text{ISL}} T_2$ but $T_1 \neq_{\text{bis}} T_2$. □

5. 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 $\mathbb{M}$, and denote the class of ITSs with alphabet Σ by $\mathbb{T}$.

We show that $\mathbb{M}$ and $\mathbb{T}$ are isomorphic, preserving natural equivalence relations. For $\mathbb{T}$, the relation in question is ITS isomorphism (Definition 15), and for $\mathbb{M}$ it is *macrostep equivalence*, which we now define.

Definition 19 (*PTM macrostep equivalence*)**.** Two PTMs M_1, M_2 are *macrostep equivalent*, notation $M_1 =_{\text{ms}} M_2$, if there exists a bijection ϕ: $reach(M_1) \to reach(M_2)$ such that:

(1) $\phi(\epsilon) = \epsilon$,
(2) $\forall w_i \in \Sigma^*, w_o \in \Sigma^* \cup \{\mu\}, s, s' \in reach(M_1)$: $s \xrightarrow[M_1]{w_i/w_o} s'$ iff $\phi(s) \xrightarrow[M_2]{w_i/w_o} \phi(s')$.

The mapping $\xi : \mathbb{M} \to \mathbb{T}$ is given by $\xi(M) = \langle reach(M), m, \epsilon \rangle$, where $\langle s, w_i, s', w_o \rangle \in m$ iff $s \xrightarrow[M]{w_i/w_o} s'$. Note that $\xi(M)$ is indeed an ITS, as $reach(M)$ is enumerable, m is a partial recursive function, and the set of states of $\xi(M)$ is reachable from its root. By definition, ξ is a transition-preserving isomorphism from the reachable states of M to the states of T, where $T = \xi(M)$.

Example 20. The ITSs of Fig. 2 depict the image, under ξ, of the PTMs M_{Latch} and M'_{Latch} of Example 8. 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.

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

Proposition 21. *For all* $M, M' \in \mathbb{M}$,

(1) $PSL(M) = ISL(\xi(M))$ *and*
(2) $M =_{\text{PSL}} M'$ *iff* $\xi(M) =_{\text{ISL}} \xi(M')$.

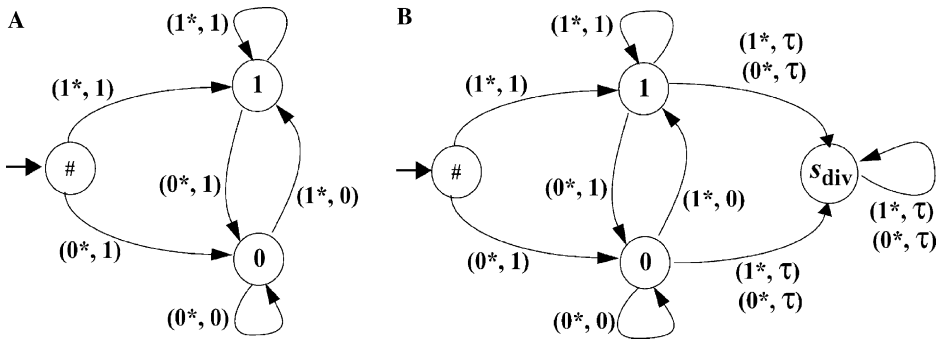

Fig. 2. (A) $\xi(M_{\text{Latch}})$ and (B) $\xi(M'_{\text{Latch}})$.

The proof uses coinduction to establish a stronger result, namely, if $\sigma \in PSL(M, w)$, for any $w \in reach(M)$, then $\sigma \in ISL(T(w))$.

Proof. 1. We prove only one direction, namely that $PSL(M) \subseteq ISL(\xi(M))$; the other direction is analogous. Let $w \in reach(M)$ and σ a stream in $PSL(M, w)$. According to Definition 7, there exists $w' \in \Sigma^* \cup \{s_{\text{div}}\}$ such that $\sigma = \langle (w_i, w_o), \sigma' \rangle$ where $w \xrightarrow[M]{w_i/w_o} w'$ and $\sigma' \in PSL(M, w')$.

Let $T = \xi(M)$; by definition, w is a state of T. Since w' is also in $reach(M)$, 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 17, $\sigma \in ISL(T(w))$. Since w was arbitrary, let $w = \epsilon$. It follows that for all $\sigma \in PSL(M)$, it is the case that $\sigma \in ISL(T)$.

2. This follows as a corollary of 1. □

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

Proposition 22. *For all* $M_1, M_2 \in \mathbb{M}, M_1 =_{\text{ms}} M_2$ *iff* $\xi(M_1) =_{\text{iso}} \xi(M_2)$.

Proof. Set ψ in the definition of $=_{\text{iso}}$ (Definition 15) to the ϕ in the definition of $=_{\text{ms}}$ (Definition 19) 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 up to isomorphic ITSs.

Proposition 23. *For all* $T \in \mathbb{T}$, *there exist* $M \in \mathbb{M}$ *and* $T_0 \in \mathbb{T}$ *such that* $T =_{\text{iso}} T_0$ *and* $T_0 = \xi(M)$.

Proof. Let $T = \langle S, m, r \rangle$. To prove the result, we exhibit an injective mapping $\varphi : S \to \Sigma^* \cup \{s_{\text{div}}\}$ and a PTM $M \in \mathbb{M}$ such that $\varphi(r) = \epsilon$, $\varphi(s_{\text{div}}) = s_{\text{div}}$, and $\langle s, w_i, s', w_o \rangle \in m$ iff $\varphi(s) \xrightarrow[M]{w_i/w_o} \varphi(s')$ where $w_o \in \Sigma^* \cup \{\mu\}$. M is a PTM since the initial state corresponds to an initial work-tape contents of ϵ, and each of the macrosteps is computable by an N3TM because m is recursively enumerable. Also, from the definition of an ITS, $reach(M) = \{\varphi(s) \mid s \in S\}$.

Let $T_0 = \langle S_0, m_0, \epsilon \rangle$, where $S_0 = \{\varphi(s) \mid s \in S\}$ and $m_0 = \{\langle \varphi(s), w_i, \varphi(s'), w_o \rangle \mid \langle s, w_i, s', w_o \rangle \in m\}$. Clearly, $\xi(M) = T_0$. Also, $T_0 =_{\text{iso}} T$, where φ 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 24. *The structures* $\langle \mathbb{M}, =_{\text{ms}} \rangle$ *and* $\langle \mathbb{T}, =_{\text{iso}} \rangle$ *are isomorphic*.

Proof. It follows from Propositions 22 and 23 that ξ is a structure-preserving bijection. □

The class of DPTMs and the class of DITSs also are isomorphic. Using similar reasoning, we denote the class of DPTMs with alphabet Σ by $\mathbb{DM}$ and the class of DITSs with alphabet Σ by $\mathbb{DT}$. The same mapping ξ, now restricted to ξ: $\mathbb{DM} \to \mathbb{DT}$, is used. Note that $\xi(M)$, with M restricted to a DPTM, is indeed a DITS, as m is a total recursive function and the set of states of $\xi(M)$ is reachable from its root.

It is easy to see that persistent stream languages of DPTMs are preserved by ξ (Proposition 21), and ξ maps equivalent DPTMs to equivalent DITSs (Proposition 22). In order to show that ξ is surjective, we need to show that the PTM M resulting from the ξ mapping in Proposition 23 is indeed a DPTM. But because m is a total recursive function, each of the macrosteps of M halts, and thus M is indeed a DPTM. Combining these results, we have the relationship between decider persistent Turing machines and decider interactive transition systems.

Theorem 25. *The structures* $\langle \mathbb{DM}, =_{\mathrm{ms}} \rangle$ *and* $\langle \mathbb{DT}, =_{\mathrm{iso}} \rangle$ *are isomorphic*.

6. 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 PTMs exhibit unbounded nondeterminism.

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

Definition 26 (*Stream prefix operators*)**.** Let $\mathbb{S}_A$ be the set of streams over some set A of tokens and let $\sigma \in \mathbb{S}_A$. Then $\sigma = \langle a, \sigma' \rangle$ for some $a \in A, \sigma' \in \mathbb{S}_A$. For all $k \geqslant 1$, $\mathrm{pref}_k(\sigma)$ is defined inductively as follows:

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

where $\frown$ denotes the concatenation operator on streams.

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

Definition 27 (*Stream prefix languages*)**.** For any $k \geqslant 1$ and any PTM M, the *k-prefix language* of M is given by

$$L_k(M) = \bigcup_{i \leqslant k} \{\mathrm{pref}_i(\sigma) \mid \sigma \in PSL(M)\}.$$

Moreover, the pair of PTMs M_1, M_2 are *k-equivalent*, notation $M_1 =_{\mathrm{k}} M_2$, if $L_k(M_1) = L_k(M_2)$.

Proposition 28. *For any* $k \geqslant 1$, $(k+1)$*-equivalence strictly refines k-equivalence, i.e.,* $=_{\mathrm{k+1}} \subset =_{\mathrm{k}}$.

Proof. That $(k+1)$-equivalence refines k-equivalence follows from Definition 27. To prove that the refinement is strict, consider the sequence of PTMs ${M_{\mathrm{Ct}}}^1, {M_{\mathrm{Ct}}}^2, \ldots$, where, for any k, ${M_{\mathrm{Ct}}}^k$ is the PTM with binary outputs that ignores the values it obtains from its input stream, outputting k 1's and thereafter outputting 0's only.

Essentially, these PTMs are counters, counting off k inputs. It is easily seen that for all $k \geqslant 1$ and for all $n \geqslant 0$, $L_k(M_{\mathrm{Ct}}{}^n) \subseteq L_k(M_{\mathrm{Ct}}{}^{n+1})$ and $(1,1)^{k+1} \in L_{k+1}(M_{\mathrm{Ct}}{}^{k+1}) - L_{k+1}(M_{\mathrm{Ct}}{}^k)$. This proves the proposition. □

Proposition 28 establishes an *infinite hierarchy* of stream-based equivalence relations for PTMs, the limit point of which is ∞-*equivalence*.

Definition 29 (∞-*equivalence*)**.** PTMs M_1, M_2 are called ∞-*equivalent*, notation $M_1 =_\infty M_2$, if $L_\infty(M_1) = L_\infty(M_2)$, where $L_\infty(M) = \bigcup_{k \geqslant 1} L_k(M)$.

∞-equivalence corresponds to properties that can be verified by checking the (finite) prefixes of computations; that is, it corresponds to *safety properties* [16,17]. Safety properties are those properties of a reactive system whose violation occurs in finite time. For example, mutually exclusive access to a shared resource is specified by a safety property. Thus, if a safety property is violated in an infinite computation, then there exists a finite prefix of that computation which demonstrates the violation.

PSL-equivalence, on the other hand, corresponds to arbitrary properties of reactive systems. In [17] it is shown that any property can be expressed as the intersection of a safety property and a *liveness* property. A liveness property is one that is violated only along an infinite computation; no finite prefix of a computation can demonstrate the violation of a liveness property. For example, the eventual granting of access to a shared resource is specified by a liveness property.

Clearly, $=_\infty$ strictly 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 in Section 3? We consider this question in Propositions 30 and 40.

Proposition 30. *PSL-equivalence strictly refines* ∞*-equivalence, i.e.,* $=_{\mathrm{PSL}} \subset =_\infty$.

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 the values they obtain from their input streams, 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
```

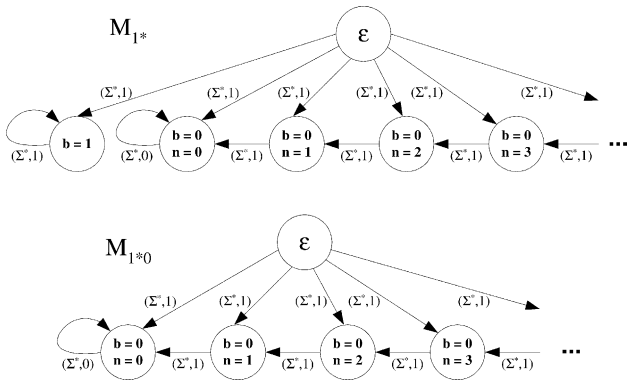

Fig. 3. The ITSs corresponding to the PTMs M_{1*} and M_{1*0}.

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 are 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 $j \leqslant k$) and 0 for the rest (if any). However, $PSL(M_{1*}) \neq PSL(M_{1*0})$; in particular, the stream $[(1,1),(1,1),\ldots] \in PSL(M_{1*}) - PSL(M_{1*0})$. □

The ITSs corresponding to M_{1*} and M_{1*0}, i.e., $\xi(M_{1*})$ and $\xi(M_{1*0})$, are depicted in Fig. 3. The ITS corresponding to M_{1*0} 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.

7. Unboundedness, divergence, and the gap

We now address the gap between $=_{\mathrm{PSL}}$ and $=_\infty$, which was illustrated by the example in Section 6. We show that for any two PTM's M and M', a gap may exist between $M =_{\mathrm{PSL}} M'$ and $M =_\infty M'$. Specifically, if at least one of M, M' exhibits unbounded nondeterminism (which we formalize below), then it is possible that $M =_\infty M'$ but not $M =_{\mathrm{PSL}} M'$. If neither of M, M' exhibits unbounded nondeterminism, then $M =_\infty M'$ and $M =_{\mathrm{PSL}} M'$ coincide (i.e., are either both true or both false). Thus, unbounded nondeterminism is a necessary condition for the existence of the gap.

Definition 31 (*Unbounded nondeterminism*)**.** A PTM M has *unbounded nondeterminism* if there exist some $w \in reach(M), w_i \in \Sigma^*$ such that there is an infinite number of $w_o \in \Sigma^* \cup \{\mu\}, w' \in \Sigma^* \cup \{s_{\mathrm{div}}\}$, such that $w \xrightarrow[M]{w_i/w_o} w'$.

Theorem 32. *If a PTM M has unbounded nondeterminism, then M diverges.*

Proof. Construct the transition diagram G of the underlying N3TM M. The nodes of G are configurations of M, and there is a directed edge from node C to node C' if and only if a single microstep,

i.e., regular TM step, of M can take configuration C to configuration C'. Attach to each node a step-counter giving the number of microsteps since the start of computation. This ensures that each node has only a finite number of incoming edges. If a node still has $m > 1$ incoming edges, then split the node into m nodes containing the same configuration and step-counter, and give each of the m nodes a sequence-number ranging from 1 to m. This style of construction ensures that the resulting G is a tree.

Unbounded nondeterminism means that the tree is infinite. Now the transition function of a Turing machine is itself bounded, i.e., for a given Turing-machine configuration there is only a finite number of possible successors. So, the tree has bounded out-degree, and by Koenig's lemma, the tree contains an infinite path, which means nontermination, i.e., divergence. □

Let $\frown$ denote the concatenation operator on streams, i.e., if σ_k is a sequence of k tokens from A ($k \geqslant 1$) and $\gamma \in \mathbb{S}_A$ then $\sigma_k \frown \gamma \in \mathbb{S}_A$.

Definition 33 (*Limit-closed stream language*)**.** Let L be a stream language over A, i.e., $L \subseteq \mathbb{S}_A$. Then, L is *limit-closed* if $\forall\, \sigma \in \mathbb{S}_A$, the following condition holds:

> if for all $k \geqslant 1$, there exist $\gamma_1 \in \mathbb{S}_A, \gamma_2 \in \mathbb{S}_A, \ldots, \gamma_k \in \mathbb{S}_A \ldots$ such that $pref_1(\sigma) \frown \gamma_1 \in L$, $pref_2(\sigma) \frown \gamma_2 \in L, \ldots,\ pref_k(\sigma) \frown \gamma_k \in L \ldots,$ then $\sigma \in L$.

In other words, whenever all of the finite prefixes of a stream σ are the prefix of some stream in a stream-language L, then σ itself must be in L.

Lemma 34. *Let T be an ITS in which every node has finite out-degree. Then $ISL(T)$ is limit-closed.*

Proof. Let σ be an infinite sequence all of whose finite prefixes are in $ISL(T)$. Consider the rooted directed acyclic graph G defined as follows. Nodes of G have the form $\langle \gamma, s \rangle$ where γ is a finite prefix of σ, and s is a state of T. The root node of G is $\langle \epsilon, r \rangle$, where r is the root of T. There is a directed edge from $\langle \gamma, s \rangle$ to $\langle \gamma', s' \rangle$ iff $\gamma' = \gamma \frown \langle w_i, w_o \rangle$ and there is a transition $\langle s, w_i, s', w_o \rangle$ in T. Clearly, there are an infinite number of nodes in G reachable from the root. By adding extra information to the nodes, and splitting nodes, we can convert this reachable portion of G into a tree G', just as in the proof of Theorem 32 above. By bounded nondeterminism, every node in G' has bounded out-degree. By Koenig's lemma, G' contains an infinite path. This path generates σ. Hence, $\sigma \in ISL(T)$. □

We remark that the above proof is adapted from [18].

Theorem 35. *Let M, M' be PTMs which do not have unbounded nondeterminism. Then $M =_{\mathrm{PSL}} M'$ iff $M =_{\infty} M'$.*

Proof. $M =_{\mathrm{PSL}} M'$ implies $M =_{\infty} M'$ follows from the definitions. We now show that $M =_{\infty} M'$ implies $M =_{\mathrm{PSL}} M'$.

Let $T = \xi(M)$, $T' = \xi(M')$, where ξ is as given in Section 5. Bounded nondeterminism of M, M' means that every node of T, T' has finite out-degree. By Lemma 34, $ISL(T)$ and $ISL(T')$ are both limit-closed.

Now assume $M =_\infty M'$. Hence, by Definition 29, $L_k(M) = L_k(M')$ for all $k \geqslant 1$. By Proposition 21, $PSL(M) = ISL(T)$ and $PSL(M') = ISL(T')$. Hence, $L_k(T) = L_k(T')$ for all $k \geqslant 1$, where $L_k(T) = \bigcup_{i \leqslant k} \{\text{pref}_i(\sigma) \mid \sigma \in ISL(T)\}$. Because $ISL(T)$ and $ISL(T')$ are both limit-closed, this implies that $ISL(T) = ISL(T')$. Hence $PSL(M) = PSL(M')$, and so $M =_{\text{PSL}} M'$. □

Hence, existence of the gap between $=_{\text{PSL}}$ and $=_\infty$ for two particular PTM's M and M' (i.e., $M =_\infty M'$ but not $M =_{\text{PSL}} M'$) requires unbounded nondeterminism of at least one of them, which also implies the divergence of at least one of them, as we now show.

Theorem 36. *Let M, M' be PTMs such that $M =_\infty M'$ and $M \neq_{\text{PSL}} M'$. Then at least one of M, M' diverges.*

Proof. Taking the contrapositive of Theorem 35, we see that $M =_\infty M'$ and $M \neq_{\text{PSL}} M'$ implies that at least one of M, M' has unbounded nondeterminism. Hence, by Theorem 32, at least one of M, M' diverges. □

Theorem 36 shows that unbounded nondeterminism is necessary for the gap to exist. We note that the connection between unbounded nondeterminism and divergence has been known for a long time, and is mentioned, for example, in [19, Chapter 9].

8. Amnesic stream computation

In this section, we present the notion of *amnesic* stream computation, where the contents of the persistent work tape are 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 6.

The amnesic stream language of a PTM is defined similarly to the PTM's persistent stream language (Definition 7). However, each computation of the PTM begins with a blank work tape; i.e., the PTM "forgets" the state it was in when the previous computation ended. As before, fix the alphabet of a PTM to be Σ.

Definition 37 (*Amnesic stream language*)**.** Given a PTM M, $ASL(M)$, the *amnesic stream language of* M, is defined as follows:

$$ASL(M) = \{\langle (w_i, w_o), \sigma' \rangle \in \mathbb{S}_{\Sigma^* \times (\Sigma^* \cup \{\mu\})} \mid \exists w' \in \Sigma^* : \epsilon \xrightarrow[M]{w_i/w_o} w' \wedge \sigma' \in ASL(M)\}.$$

PTMs M_1 and M_2 are *ASL-equivalent*, notation $M_1 =_{\text{ASL}} M_2$, if $ASL(M_1) = ASL(M_2)$. We also have that $\mathbb{ASL} = \{ASL(M) \mid M \text{ is a PTM}\}$.

Example 38. The interaction streams in $ASL(M_{\text{Latch}})$ (Example 8) are of the form $[(w_1, 1), (w_2, 1), \ldots]$.

It is also possible to define amnesic stream languages for ITSs. 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. In this case, a PTM's amnesic stream language would be preserved by the mapping ξ defined in Section 5. Amnesic behavior makes sense for Turing machines—in the classical, non-interactive setting, every Turing-machine computation commences with a blank work tape; the applicability of amnesic behavior to transition systems is questionable.

The following lemma is used in the proofs of Propositions 40 and 41.

Lemma 39. *Given a PTM M, let $L(M)$ be defined as follows*:

$$L(M) = \left\{(w_i, w_o) \in \Sigma^* \times (\Sigma^* \cup \{\mu\}) \mid \exists w' \in \Sigma^* \cup \{s_{\text{div}}\} : \epsilon \xrightarrow[M]{w_i/w_o} w'\right\}.$$

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

Proof. Follows from Definition 37. □

Proposition 40. $=_{\text{ASL}} \; = =_{\text{I}}$

Proof. Let M_1 and M_2 be arbitrary PTMs; $L(M_1)$ and $L(M_2)$ are as defined in Lemma 39. It follows from Lemma 39 that $L(M_1) = L(M_2)$ iff $ASL(M_1) = ASL(M_2)$. Also, it follows from Definition 27 that $L(M_1) = L(M_2)$ iff $L_1(M_1) = L_1(M_2)$. Therefore, $ASL(M_1) = ASL(M_2)$ iff $L_1(M_1) = L_1(M_2)$. □

Proposition 41. $\mathbb{ASL} \subset \mathbb{PSL}$

Proof. To show inclusion, it suffices to show that, given a PTM M, we can construct a PTM 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 7 and 37, it follows that $PSL(M') = ASL(M)$.

To prove that the inclusion of $\mathbb{ASL}$ in $\mathbb{PSL}$ is strict, we refer to M_{Latch} and $\sigma_{\text{io}} \in PSL(M_{\text{Latch}})$ defined in Example 8 to show that there does not exist a PTM M such that $ASL(M) = PSL(M_{\text{Latch}})$. Assume such a PTM M exists; then $\sigma_{\text{io}} \in ASL(M)$. Therefore, by Proposition 39, $(0,0)$, 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 $PSL(M_{\text{Latch}})$ can start with $(0,0)$, leading to a contradiction. Therefore, no such M exists. □

We say that a PTM M is *amnesic* if $PSL(M) \in \mathbb{ASL}$. Whereas PTMs extend Turing machines with stream-based semantics and persistence, amnesic PTMs only extend Turing machines with streams: like Turing-machine computations, their macrosteps all start in the same configuration, differing only in the input values.

Example 42. M_{Latch} is not amnesic. Neither are the M_{Ct} PTMs defined in the proof of Proposition 28. 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 [20, Fig. 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 43. *For any pair of amnesic PTMs* M_1 *and* M_2, $M_1 =_1 M_2$ *iff* $M_1 =_{\mathrm{PSL}} M_2$.

Proof. If a PTM M is amnesic, it can be shown that $PSL(M) = ASL(M)$. Combined with Proposition 40, it follows that for amnesic PTMs, $=_1$ is the same as $=_{\mathrm{PSL}}$. □

As this result shows, the infinite hierarchy of PTM equivalence relations (Proposition 30) collapses in the case of amnesic PTMs. In order to be PSL-equivalent, amnesic PTMs need only have the same 1-prefix language, i.e., the same set of input–output pairs for the first macrostep. This property differentiates amnesic PTMs from general PTMs, but is analogous to Turing-machine equivalence, which also can be defined in terms of sets of input–output pairs.

9. Universal PTMs

A *universal Turing machine* [21] is a Turing machine that, given the description of any other Turing machine, simulates the behavior of that machine. Turing used a universal Turing machine to prove the undecidability of the halting problem, i.e., whether an arbitrary Turing machine halts with a given input. Turing also hypothesized, in what is now called the Church-Turing Thesis, that any function-based computation can be performed by a Turing machine [21]. Computability theory has built upon these results, determining the classes of problems that are computable by Turing machines, and the models of computation that are equivalent to it.

Similarly, a *universal PTM* simulates the behavior of an arbitrary PTM. Anything that is computable by a PTM is computable by a universal PTM. Thus, a universal PTM can be used to show computability results for PTMs, including the classes of problems that are computable with a PTM and the models of computation that are equivalent to the PTM model of computation.

We begin our discussion of universal PTMs by reviewing the definition of a universal Turing machine. A Turing machine U is a *universal Turing machine* if it can simulate the behavior of every Turing machine.

Definition 44 (*Universal Turing machine*)**.** Let U be a Turing machine with alphabet Σ_U, let M be a Turing machine with alphabet Σ_M, let w be a string over Σ_M, and let $\eta : M, \Sigma_M^* \to \Sigma_U^*$ be a one-to-one encoding function for the transition relation and alphabet of M. Then U is a *universal Turing machine simulating* M if the following conditions hold:

- If M with input w has a halting computation producing w' on its tape, U with input $\langle \eta(M), \eta(w) \rangle$ has a halting computation producing $\eta(w')$ on its tape.
- If M with input w diverges, U with input $\langle \eta(M), \eta(w) \rangle$ diverges.

Theorem 45. *There is a Turing machine U that is a universal Turing machine.*

The literature contains many ways to construct such a universal Turing machine [21,13,22], and thereby prove its existence. All use encoding schemes both for the transition relation of M and for w, limiting the arbitrary alphabet of M to the alphabet of the universal machine. Some of these constructions halt in a rejecting state if either M or w is improperly encoded, while others diverge.

A PTM is a *universal persistent Turing machine* if it can simulate the behavior of every PTM. A universal PTM U, like a PTM, is an N3TM. It begins its computation with an *initializing macrostep*, taking on its input tape an encoding of the transition relation of the PTM M to be simulated and M's initial work-tape contents. U copies these to its work tape. Then for each subsequent macrostep, U takes an encoding of the next input to M on its input tape, simulates the operation of M on that input, and updates its work tape and output tape with encodings of the contents of the work tape and output tape of M. This is stated more formally in the following definition.

Definition 46 (*Universal PTM*)**.** Let U be a PTM with alphabet Σ_U, let M be a PTM with alphabet Σ_M, let w_i, w_o, w, and w' be strings over Σ_M, and let $\eta : M, \Sigma_M^* \to \Sigma_U^*$ be a one-to-one encoding function for the transition relation and alphabet of M. Then, U is a *universal PTM simulating M* if:

- U has an initializing macrostep $\epsilon \xrightarrow[U]{\langle \eta(M), \eta(w) \rangle / \epsilon} \langle \eta(M), \eta(w) \rangle$.
- If M has a halting computation $w \xrightarrow[M]{w_i/w_o} w'$, then U has a halting computation $\langle \eta(M), \eta(w) \rangle \xrightarrow[U]{\eta(w_i)/\eta(w_o)} \langle \eta(M), \eta(w') \rangle$.
- If M diverges, written $w \xrightarrow[M]{w_i/\mu} s_{\text{div}}$, then U diverges, written $\langle \eta(M), \eta(w) \rangle \xrightarrow[U]{\eta(w_i)/\mu} s_{\text{div}}$.
- If $s_{\text{div}} \xrightarrow[M]{w_i/\mu} s_{\text{div}}$, then $s_{\text{div}} \xrightarrow[U]{\eta(w_i)/\mu} s_{\text{div}}$.

Example 47. The PSL of PTM M_{Latch} (Example 8) contains the interaction stream:

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

The corresponding interaction stream for U is:

$$[(\langle \eta(M_{\text{Latch}}), \eta(\epsilon) \rangle, \epsilon), (\eta(w_1), \eta(1)), (\eta(w_2), \eta(w_1[1])), (\eta(w_3), \eta(w_2[1])), \ldots].$$

Similarly, the PSL of the answering machine AM (Example 9) contains the interaction stream:

```
[(record A, ok), (erase, done), (record BC, ok),
(record D, ok), (play, BCD), ...]
```

and the corresponding interaction stream for U is:

```
[(⟨η(AM), η(ϵ)⟩, ϵ), (η(record A), η(ok)), (η(erase), η(done)),
(η(record BC), η(ok)), (η(record D), η(ok)), (η(play), η(BCD)), ...].
```

The proof that a universal PTM exists relies on a construction similar to that used to show the existence of a universal Turing machine.

Theorem 48. *There is a PTM U that is a universal PTM.*

Proof. Let M be an arbitrary PTM, and let $w \xrightarrow[M]{w_i/w_o} w'$ be an arbitrary macrostep of M. By definition, M is a Turing machine. Thus, there exists a 1-tape Turing machine M' that is equivalent to M. In particular, M', when given $\langle w, w_i, \epsilon\rangle$ as input, produces $\langle w', w_i, w_o\rangle$ as output. Let U be a universal Turing machine, and let $\eta : M', \Sigma^*_{M'} \to \Sigma^*_U$ be U's encoding function for M' with the restriction that $\eta(\langle a, b\rangle) = \langle \eta(a), \eta(b)\rangle$ for all $a, b \in \Sigma^*_{M'}$. Then U, given input $\langle \eta(M'), \eta(\langle w, w_i, \epsilon\rangle)\rangle$, simulates the behavior of M' by producing $\eta(\langle w', w_i, w_o\rangle)$ as output.

Let U' be a Turing machine that behaves exactly like U except it maintains the description of the machine it is simulating on its tape, i.e., given input $\langle \eta(M), \eta(w)\rangle$, U' produces $\langle \eta(M), \eta(w')\rangle$ as output. U' exists since the class of Turing machines with semi-infinite tapes is equivalent to the class of Turing machines with doubly infinite tapes. Thus, when U' is given $\langle \eta(M'), \eta(\langle w, w_i, \epsilon\rangle)\rangle$ as input, U' simulates the behavior of M' by producing $\langle \eta(M'), \eta(\langle w', w_i, w_o\rangle)\rangle$. But this is equivalent to $\langle \eta(M'), \eta(w'), \eta(w_i), \eta(w_o)\rangle$, which in turn is equivalent to $\langle \langle \eta(M'), \eta(w')\rangle, \eta(w_i), \eta(w_o)\rangle$.

Let U'' be a 3-tape Turing machine equivalent to the 1-tape Turing machine U'. When U'' is given $\langle \eta(M'), \eta(w)\rangle$, $\eta(w_i)$, and ϵ on its three tapes, it simulates M' producing $\langle \eta(M'), \eta(w')\rangle$, $\eta(w_i)$, and $\eta(w_o)$ on its work, input, and output tapes, respectively. Therefore, by definition, U'' is a universal PTM. □

As with universal Turing machines, there are several ways a universal PTM U might handle improper encodings of M and w_i. Two possible designs are to have U diverge when it detects an improper encoding, or to have U halt after writing some special mark on its output tape. If a special mark is used, it must be a symbol or combination of symbols in the alphabet of U distinct from the encoding of any string in the alphabet of M.

A universal PTM U can also be used to simulate a PTM in the middle of a computation. In these cases the contents of the input tape used in the initializing macrostep of U are $\langle \eta(M), \eta(w)\rangle$, rather than $\langle \eta(M), \eta(\epsilon)\rangle$, where M, as above, is the PTM being simulated and w is an arbitrary string in $(\Sigma_M)^*$.

10. Sequential interactive computations

This section describes a class of interactive computations called *sequential interactive computations* which are characterized by dynamic streams of input/output pairs and saved "state" information. Several examples of sequential interactive computations are presented. We hypothesize,

in an analogous fashion to the Church-Turing Thesis, that any sequential interactive computation can be performed by a persistent Turing machine. Finally, we conclude that the class of sequential interactive computations can solve a wider range of problems than the class of algorithmic computations.

Definition 49 (*Sequential interactive computation*)**.** A *sequential interactive computation* continuously interacts with its environment by alternately accepting an input string and computing a corresponding output string. Each output-string computation may be both nondeterministic and history-dependent, with the resultant output string depending not only on the current input string, but also on all previous input strings.

Several examples of sequential interactive computations are briefly described, including descriptions of the information saved between interactions:

- In an object-oriented programming language such as Java or C++, a sequence of method invocations of an object instance constitutes a sequential interactive computation in that the value returned by one method invocation may depend on values stored in the object's instance variables from previous method invocations. The values of the instance variables comprise the state, and are maintained from one method invocation (interaction) to the next. The only restriction on these method-invocation sequences is that for any given object instance only one of its methods may be invoked at a time (no concurrency). Using Java terminology, each of the object's methods must be *synchronized*.
- In the C programming language, a series of calls to a function with static variables behaves like a sequential interactive computation, with the state of the computation saved in the values of the function's static variables.
- A sequence of queries and updates to a single-user database is a sequential interactive computation; the data in the database is the state of the computation that is maintained from one interaction to the next. This example is discussed further in [23].
- A simple line-drawing program is a sequential interactive computation. The user specifies "rubber-band" lines by dragging an input device from one endpoint of a line to the other. The first endpoint of a line is stored in the state of the computation while the user repeatedly repositions the second endpoint of the line until she is satisfied with its position. The computation also must maintain the positions of all previously drawn lines.
- Graphical applications with complex three-dimensional pictures may save intermediate rendering data between interactions, thereby taking advantage of temporal coherence to speed up the rendering calculations.
- Network communications using a protocol such as HTTP are sequential interactive computations. The communication strictly alternates between the client and the server. The protocol uses request and response headers to establish the parameters of the communication, and then uses this state information to properly transmit all subsequent communications.
- Driving home from work in an automatic car (one that drives itself) is a sequential interactive computation. The car constantly receives camera and sensor inputs from its environment about road conditions and the locations of other cars and pedestrians. These inputs, combined with

saved information such as the car's destination, are used to calculate a collision-free path home. This example is discussed further in [24].

In [5,25], the term *sequential interaction machine* (SIM) is used to refer to any device that performs a sequential interactive computation. In particular, a persistent Turing machine is a SIM. Other models of sequential interactive computations include transducers (Mealy and Moore machines), state-based agents [26], neural networks, non-networked embedded systems, dynamic algorithms [27], and on-line algorithms [28]. We conjecture that all of these models can be simulated by a PTM with a construction similar to that used for the universal PTM (Definition 46).

Interactive applications that require concurrency are not sequential interactive computations. As an example, any application that allows the user to type ahead or click ahead of the processor is allowing the user to perform actions concurrently with the processor. This category of application typically uses some type of queue to store user actions until the processor can handle them. Although the user actions are handled sequentially by the processor, these interactions cannot by modeled by a sequential interactive computation because such a computation cannot add new user actions to the queue while it is in the middle of processing earlier actions. More generally, sequential interactive computations do not allow any nonserializable behavior such as the coordination of inputs from multiple concurrent sources, referred to in [5,25] as *multistream interaction machines* (MIMs).

Sequential interactive computations also do not include any interactive behavior that is time-dependent. As an example of time-dependent interactive behavior, some character-recognition algorithms sample the position of the mouse or stylus at regular intervals of time and use the speed of position changes to distinguish between various characters.

The relationship between the intuitively defined sequential interactive computation and the formally defined persistent Turing machine can be summarized by the following thesis.

Thesis 50 (*Sequential interaction*)**.** Any sequential interactive computation can be performed by a persistent Turing machine.

Like the Church-Turing Thesis, this thesis cannot be proved. Informally, each step of a sequential interactive computation, corresponding to a single input/output-pair transition, is algorithmic. Therefore, by the Church-Turing Thesis, each step is computable by a Turing machine. A sequential interactive computation may be history-dependent, so state information must be maintained between steps. A persistent Turing machine is just a Turing machine that maintains state information on its work tape between steps. Thus, any sequential interaction machine can be simulated by a PTM.

This thesis, when combined with our earlier result establishing PTMs as more expressive than amnesic PTMs (Proposition 41), has the following implications. Since amnesic PTMs are an extension of Turing machines, and therefore at least as expressive, we can conclude that the class of sequential interactive computations, corresponding to PTM computations, is more expressive than the class of algorithmic computations, corresponding to classic Turing machine computations. In practical terms, this means that sequential interaction machines are capable of solving a wider range of problems than Turing machines; an example is the *driving home from work* problem, discussed in detail in [24].

11. Related work

A preliminary version of this paper appeared in [1]. Some of the notions formalized in this paper, including that of a persistent Turing machine, were put forth in earlier papers by one of the authors [29,30].

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 [31–36], process calculi [37,38], synchronous languages [39,40], finite/pushdown automata over infinite words [41,42], interaction games [43], concurrent constraint programming [44], transition systems [45], reactive modules [46], and I/O automata [47].

The main difference between these approaches and our own is that our focus is on the issues of *computability* and *expressiveness*—classical issues from the theory of computation reinterpreted in the new interactive context. In particular, our goal was to prove Wegner's conjecture [4] that "interaction is more powerful than algorithms." The models and the attendant definitions developed in the effort to formalize and justify this conjecture constitute the main contribution of this paper. By contrast, the other approaches tend to emphasize issues such as correctness and programming, and the computability of a transition step is often assumed. Moreover, these models of computation are typically purely functional in nature, without the notion of persistency or "memory" present in PTMs.

Persistency, however, can be captured in these models by one means or another. In dataflow models one can use "feedback loops" and in process calculi one can explicitly model the data store. For example, PTM M_{Latch} of Example 8 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 goes back only one interaction in time, while 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 [5]. A major emphasis of this 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 [48].

Persistence in a Turing-machine setting was also investigated in [49], which introduced *information-conservative Turing machines* (later renamed to *persistent Turing machines* [50]) to explore the consequences of persistence on some classical concepts in recursion theory. These machines have a persistent input tape, appending a new input token for each subsequent computation at the end; the output of each computation is treated as a recursive function of the complete input sequence.

Earlier extensions of Turing machines that can be considered interactive include *on-line Turing machines* [51] and *interactive Turing machines* [52]. In on-line Turing machines, which model on-line computation, the ith output symbol must be written to the output tape before the $(i+1)$st input symbol is read. In interactive Turing machines, which capture interactive cryptographic protocols, macrostep-like computations alternate between two agents, preserving state between computations. Both devices were created to express formally the corresponding algorithms, so as to establish complexity-theoretic lower bounds; general computability issues were not considered.

By contrast, some recently proposed extensions of Turing-machine computation to the stream setting fail to capture persistence. In particular, the *squaring machine* of [20, Fig. 1] is amnesic; so

are the examples of "interactive computation" in [53]. Not surprisingly, both papers conclude that these extensions do not yield an interesting new model of computation.

An alternative approach to extending the Turing-machine model to interactive computation is captured by the *interactive Turing machines with advice* (ITMAs) of [3]. 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 (stream-based semantics and persistence of the work tape) needed to attain transition-system expressiveness.

PTMs, as well as other models of computation discussed above, are based on the notion of interaction as message passing. This *direct* form of interaction can be contrasted with *indirect interaction*, where computing agents share the same persistent environment, and changes by one agent can afterwards be observed by others [54,55]. *Concurrent Constraint Programming* [44], where concurrent *constraint logic programming* agents interact by placing, checking, and instantiating constraints on shared variables, serves as an example of indirect interaction. In general, however, the shared environment may be non-computable (i.e., involve the "real world"); the formalization of concurrent systems whose components interact indirectly via a non-computable environment is an open question.

Finally, one may wonder about PTM-like models for other forms of computation besides sequential interactive computations. Broy [56] proposes a stream-based theory of interaction for distributed interactive software systems composed of components. For future work, it would be interesting to develop a model of PTM computation where PTMs execute concurrently and communicate with each other through their input and output tapes, formalizing Wegner's *multistream interaction machines* [5,25]. We conjecture that concurrent PTMs are more expressive than sequential ones in terms of the stream languages they produce.

12. Conclusions

We have presented persistent Turing machines (PTMs), a stream-based extension of the Turing-machine model with appropriate notions of interaction and persistence. A number of expressiveness

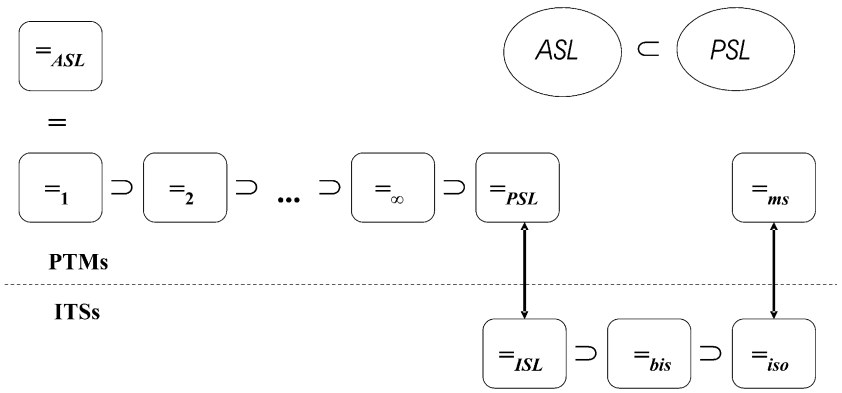

Fig. 4. Summary of results.

results concerning PTMs have been presented, including the expressive equivalence of PTMs and interactive transition systems; the strict inclusion of the set $\mathbb{ASL}$ of amnesic stream languages in the set $\mathbb{PSL}$ 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. These results are summarized in Fig. 4.

Perhaps the most interesting of our expressiveness results concerns the collapse of the equivalence hierarchy for amnesic PTMs. Amnesic computation extends Turing-machine computation with stream-based semantics but not persistence; 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.

As future 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 is abstracted away.

References

[1] D. Goldin, S. Smolka, P. Wegner, Turing machines, transition systems, and interaction, Electronic Notes in Theoretical Computer Science 52, Elsevier, 2002.
[2] R. Milner, Elements of interaction: Turing award lecture, Commun. ACM 36 (1) (1993) 78–89.
[3] J. van Leeuwen, J. Wiedermann, The Turing machine paradigm in contemporary computing, in: B. Enquist, W. Schmidt (Eds.), Mathematics Unlimited—2001 and Beyond, LNCS, Springer, Berlin, 2000.
[4] P. Wegner, Why interaction is more powerful than algorithms, Commun. ACM 40 (5), May 1997.
[5] P. Wegner, Interactive foundations of computing, Theoret. Comput. Sci. 192, February 1998.
[6] G. Boudol, Notes on algebraic calculi of processes, in: K. Apt (Ed.), Logics and Models of Concurrent Systems, LNCS, Springer, Berlin, 1985, pp. 261–303.
[7] R. de Simone, Higher-level synchronizing devices in MEIJE-SCCS, Theoret. Comput. Sci. 37 (1985) 245–267.
[8] J. Baeten, J. Bergstra, J. Klop, On the consistency of Koomen's fair abstraction rule, Theoret. Comput. Sci. 51 (1/2) (1987) 129–176.
[9] B.S. Bloom, S. Istrail, A.R. Meyer, Bisimulation can't be traced, in: Proc. 15th ACM Symposium on Principles of Programming Languages, 1988.
[10] P. Darondeau, Concurrency and computability, in: I. Guessarian (Ed.), Semantics of Systems of Concurrent Processes, Proc. LITP Spring School on Theoretical Computer Science, LNCS 469, Springer, Berlin, La Roche Posay, France, 1990.
[11] F.W. Vaandrager, Expressiveness results for process algebras, Tech. Rep. CS-R9301, Centrum voor Wiskunde en Informatica, Amsterdam, 1993.
[12] S.D. Brookes, A.W. Roscoe, An improved failures model for communicating sequential processes, in: Proc. NSF-SERC Seminar on Concurrency, Springer, Berlin, 1985.
[13] J. Hopcroft, R. Motwani, J. Ullman, Introduction to Automata Theory, Languages, and Computation, second ed., Addison-Wesley, Boston, 2001.
[14] A. Pnueli, M. Shalev, What is in a step: on the semantics of Statecharts, in: Theoretical Aspects of Computer Software, No. 526 in Lecture Notes in Computer Science, 1991, pp. 244–264.
[15] J. Barwise, L. Moss, Vicious circles, CSLI Lecture Notes #60, Cambridge University Press, Cambridge, 1996.
[16] L. Lamport, Proving the correctness of multiprocess programs, IEEE Transactions on Software Engineering SE-3 (2) (1977) 125–143.

[17] B. Alpern, F. Schneider, Recognizing safety and liveness, Distributed Comput. 2 (3) (1987) 117–126.
[18] N. Lynch, F. Vaandrager, Forward and backward simulations—Part I: Untimed systems, Inform. Comput. 121 (2) (1995) 214–233.
[19] E.W. Dijkstra, A Discipline of Programming, Prentice-Hall, Engle-wood Cliffs, NJ, 1976.
[20] M. Prasse, P. Rittgen, Why Church's thesis still holds: some notes on Peter Wegner's tracts on interaction and computability, Comput. J. 41 (6) (1998) 357–362.
[21] A.M. Turing, On computable numbers with an application to the Entscheidungsproblem, in: Proceedings of the London Math Society, vol. 2 (4), 1936, pp. 230–265, reprinted in Martin Davis, Ed., The Undecidable, Raven, New York, 1965, pp. 116–151.
[22] C. Papadimitriou, Computational Complexity, Addison-Wesley, Reading, MA, 1994.
[23] D. Goldin, S. Srinivasa, B. Thalheim, Information systems=databases + interaction: towards principles of information system design, in: Proc. ER 2000, Salt Lake City, Utah, 2000.
[24] E. Eberbach, D. Goldin, P. Wegner, Turing's ideas and models of computation, in: C. Teuscher (Ed.), Alan Turing: Life and Legacy of a Great Thinker, Springer, Berlin, 2004.
[25] P. Wegner, D. Goldin, Coinductive models of finite computing agents, Electronic Notes in Theoretical Computer Science 19, Elsevier, 2000.
[26] S.J. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Upper Saddle River, NJ, 1995.
[27] Y.-J. Chiang, R. Tamassia, Dynamic algorithms in computational geometry, in Proceedings of the IEEE, Special Issue on Computational Geometry, vol. 80 (9), 1992, pp. 362–381.
[28] S. Albers, Online algorithms, in: Goldin et al. [57], forthcoming.
[29] D. Goldin, P. Wegner, Persistence as a form of interaction, Brown University Tech. Rep. CS-98-07, July 1998.
[30] D. Goldin, Persistent turing machines as a model of interactive computation, in: K.-D. Schewe, B. Thalheim (Eds.), Foundations of Information and Knowledge Systems, First International Symposium, LNCS 1762, Springer, Berlin, 2000, pp. 116–135.
[31] G. Kahn, D.B. MacQueen, Coroutines and networks of parallel processes, in: Proceedings of the IFIP Congress 77, North-Holland, 1977.
[32] P. Panangaden, E.W. Stark, Computations, residuals, and the power of indeterminancy, in: Proceedings of the 15th ICALP, Lecture Notes in Computer Science, vol. 317, Springer-Barlin, 1988, pp. 439–454.
[33] A.M. Rabinovich, B.A. Trakhtenbrot, Communication among relations, in: Paterson [58], pp. 294–307.
[34] P. Panangaden, V. Shanbhogue, E.W. Stark, Stability and sequentiality in dataflow networks, in: Paterson [58], pp. 308–321.
[35] G. Kahn, G. Plotkin, Concrete domains, Theoret. Comput. Sci. 121 (1–2) (1993) 187–277.
[36] A. Bucciarelli, T. Ehrhard, Sequentiality in an extensional framework, Inform. Comput. 110 (2) (1994) 265–296.
[37] R. Milner, Communication and Concurrency, International Series in Computer Science, Prentice Hall, 1989.
[38] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes, Parts I and II, Inform. Comput. 100 (1992) 1–77.
[39] D. Harel, Statecharts: a visual formalism for complex systems, Sci. Comput. Program. 8 (1987) 231–274.
[40] G. Berry, G. Gonthier, The ESTEREL synchronous programming language: design, semantics, implementation, Sci. Comput. Program. 19 (1992) 87–152.
[41] J. Esparza, D. Hansel, P. Rossmanith, S. Schwoon, Efficient algorithms for model checking pushdown systems, in: Proc. CAV'2000, LNCS 1855, Springer, Berlin, 2000, pp. 232–247.
[42] O. Burkart, D. Caucal, F. Moller, B. Steffen, Verification on infinite structures, in: Handbook of Process Algebra, Elsevier, Amsterdam, 2001.
[43] S. Abramsky, Concurrent interaction games, in: J. Davies, A.W. Roscoe, J. Woodcock (Eds.), Millenial Perspectives in Computer Science, Pal-grave, 2000, pp. 1–12.
[44] V. Saraswat, M. Rinard, Concurrent constraint programming, in: Proceedings of the 17th Annual ACM Symposium on Principles of Programming Languages, San Francisco, CA, 1990, pp. 232–245.
[45] Z. Manna, A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems, Springer, Berlin, Heidelberg, New York, 1992.
[46] R. Alur, T.A. Henzinger, Reactive modules, Formal Methods System Design 15 (1999) 7–48.
[47] N.A. Lynch, Distributed Algorithms, Morgan Kaufmann, 1996.

[48] J. van Leeuwen, J. Wiedermann, A computational model of interaction in embedded systems, Tech. Rep. UU-CS-02-2001, Department of Computer Science, Utrecht University, 2001.
[49] S. Kosub, Some remarks towards dynamic analysis of computation, Manuscript, Institut fur Informatik, Julius Maximilians Universitat, Wurzburg, March 1998.
[50] S. Kosub, Persistent computations, Tech. Rep. 217, December 1998.
[51] M.J. Fischer, L.J. Stockmeyer, Fast on-line integer multiplication, J. Comput. System Sci. 9 (3) (1974) 317–331.
[52] O. Goldreich, Foundations of Cryptography, vol. I, Cambridge University Press, Cambridge, 2001.
[53] B. Ekdahl, Interactive computing does not supersede Church's thesis, in: Proceedings of the 17th Confernce on International Association of Management, 1999, pp. 261–265.
[54] D. Keil, D. Goldin, Modeling indirect interaction in open computational systems, in: WETICE 2003, IEEE Press, 2003, pp. 371–376.
[55] D. Goldin, D. Keil, Toward domain-independent formalization of indirect interaction, in: WETICE 2004, IEEE Press.
[56] M. Broy, Interactive computation: the new paradigm, in: Goldin et al. [57], forthcoming.
[57] D. Goldin, S. Smolka, P. Wegner (Eds.), Interaction: A New Computational Paradigm, Springer, Heidelberg, Germany, 2005, forthcoming.
[58] M. Paterson (Ed.), Automata, Languages and Programming (ICALP '90), vol. 443 of Lecture Notes in Computer Science, Springer, Berlin, Warwick, England, 1990.