Alcom-FT Technical Report Series
ALCOMFT-TR-01-162

On Algorithms and Interaction*

Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Department of Computer Science, Utrecht University,
Padualaan 14, 3584 CH Utrecht, the Netherlands.
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic.

Abstract. Many IT-systems behave very differently from classical machine models: they interact with an unpredictable environment, they never terminate, and their behavior changes over time. Wegner [25, 26] (see also [28]) recently argued that the power of interaction goes beyond the Church-Turing thesis. To explore interaction from a computational viewpoint, we describe a generic model of an 'interactive machine' which interacts with the environment using single streams of input and output signals over a simple alphabet. The model uses ingredients from the theory of ω-automata. Viewing the interactive machines as transducers of infinite streams of signals, we show that their interactive recognition and generation capabilities are identical. It is also shown that, in the given model, all interactively computable functions are limit-continuous.

1 Introduction

Many modern systems in information technology have features that are no longer adequately described by the models of computation as we know them: their input is unpredictable and is not specified in advance, they do not terminate (unless a fault occurs), and they 'learn' over time. The systems are usually not meant to compute some final result, but are designed to interact with the environment and to develop or maintain some well-defined action-reaction behavior. Systems of this kind have previously been called 'reactive systems' and received much attention in the theory of concurrent processes (see e.g. [10, 11] and [9, 12]). Reactive systems are infinite-state systems and have a program and a behavior that is evolving over time.

Wegner [25, 26] (see also Wegner and Goldin [28]) recently called for a more computational view of reactive systems, claiming that they have a richer behavior than 'algorithms' as we know them. He even challenged the validity of Church's Thesis by proclaiming that Turing machines cannot adequately model the interactive computing behavior of typical reactive systems in practice. Wegner [26] (p. 318) writes:

> *"The intuition that computing corresponds to formal computability by Turing machines ... breaks down when the notion of what is computable*

* This research was partially supported by GA ČR grant No. 201/98/0717 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

is broadened to include interaction. Though Church's thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid."

For a discussion of this claim from a theoretical viewpoint, see Prasse and Rittgen [14]). The formal aspects of Wegner's theory of interaction are studied in e.g. Wegner and Goldin [27, 29].

Irrespective of whether this claim is valid or not, it is interesting to look at the implications of reactiveness, or interactiveness, from a computational viewpoint. In [19] we have explored this for embedded systems, in [20] for 'dynamic' networks like the internet. The cornerstone in these studies is the notion of an interactive machine. In the present paper we describe a possible model of an interactive machine C, interacting with an environment E using single streams of input and output signals over a simple alphabet. Our notion of interactive machine (IM) is very similar to Broy's notion of component ([1]), but we restrict ourselves to deterministic components only. We will point to several connections to the well-known theory of ω-automata, and there clearly are game-theoretic aspects as well.

We will identify a special condition, referred to as the interactiveness condition, which we will impose throughout. Loosely speaking, the condition states that C is guaranteed to give some meaningful output within finite time any time after receiving a meaningful input from E and vice versa. We assume that C is a 'learning' program with unbounded memory, with a memory contents that is building up over time and that is never erased unless the component explicitly does so. This compares to the use of persistent Turing machines by Goldin [4] (see also Goldin and Wegner [5]) and Kosub [8]. No special assumptions are made about the 'speed' at which C and E can operate or generate responses. We assume that E can behave arbitrarily and unpredictably.

Viewing IM's as interactive transducers of the signals that they receive from the environment we show that, using suitable definitions, recognition and generation coincide just like they do in the case of Turing machines. The proof depends on some of the special operational assumptions in the model. We also define a general notion of interactively computable functions. We prove that interactively computable functions are limit-continuous, using a suitable extension of the notion of continuity known from the semantics of computable functions. It is shown that interactively computable 1-1 functions have interactively computable inverses.

The study of 'machines' working on infinite input streams (ω-words) is by no means new and has a sizable literature, with the first studies dating back to the nineteen sixties and seventies (cf. [15], [17]). In the model studied in the present paper a number of features are added that are meant to better capture some intuitive notions of interactiveness, inspired by Wegner's papers. The present paper derives from [19].

2 A Simple Model of Interactive Computation

Let C be a component (a software agent or a device) that interacts with an environment E. We assume that C and E interact by exchanging signals (symbols). Although general interactive systems do not need to have a limit on the nature and the size of the signals that they exchange, we assume that the signals are taken from a fixed and finite alphabet. More precisely:

> (*Alphabet*) C and E interact by exchanging symbols from the alphabet $\Sigma = \{0, 1, \tau, \sharp\}$.

Here 0 and 1 are the ordinary bits, τ is the 'silent' or empty symbol, and $\sharp$ is the fault or error symbol. Instead of the bits 0 and 1, one could use any larger (finite) set of symbols but this is easily coded back into binary form. We assume that the interaction is triggered by E.

In order to describe the interactions between C and E we assume a uniform time-scale of discrete moments $0, 1, 2 \ldots$. C and E are assumed to work synchronously. At any time t, E can send a symbol of Σ to C and C can send a symbol of Σ to E. It is possible that E or C remain 'silent' for a certain amount a time, i.e. that either of them does not send any real signal during some consecutive time moments. During these moments E or C is assumed to send the symbol τ just to 'record' this. For the symbol $\sharp$ a special convention is used:

> (*Fault rule*) If C receives a symbol $\sharp$ from E, then it will output a $\sharp$ in finite time after that as well (and vice versa).

If no $\sharp$'s are exchanged, the interaction between E and C is called fault-free (error-free). The communication between E and C can be described by the sequences $e = e_0 e_1 \ldots e_t \ldots$ and $c = c_0 c_1 \ldots c_t \ldots$, with e_t denoting the signal that E sends to C at time t and c_t denoting the signal that C sends to E at time t. It is assumed that E and C send at most one signal per time-moment and that a signal sent at time t is received by the other party at the same time-moment. When E or C is silent at time t, then the corresponding symbol sent is τ ('empty'). If two sequences correspond to the actual interactions that take place over time, we say that they form an 'interaction pair' and represent an *interactive computation* of C in response to the (unpredictable) environment E. We let $\overline{e}$ and $\overline{c}$ denote the sequences e and c *without* the τ's.

We assume the following property as being characteristic for interactive computations: E sends signals to C infinitely often and C is guaranteed to give an output within finite time, any time after receiving an input signal from the E. More precisely:

> (*Interactiveness*) For all times t, when E sends a non-empty signal to C at time t, then C sends a non-empty signal to E at some time t' with $t' > t$ (and vice versa).

The condition of interactiveness is assumed throughout this paper. Note that, in the definition of interactiveness, we do not make any special assumption about the causality between the signal sent (by E or C) at time t and the signal sent (by C or E respectively) at time t'. With the condition of interactiveness into effect, the behavior of a component C with respect to E is a relation on infinite strings over Σ, accounting for the intermittent empty symbols in the proper way (i.e. in the interaction they are present but in the reported sequences they are suppressed). We assume that E sends a non-empty signal at least once, thus always triggering an infinite interaction sequence.

We assume that C has only one channel of interaction with E, admits multi-threading (allowing it to run several processes simultaneously), and has a fixed but otherwise arbitrary speed of operation. As a consequence it is possible for C to have a foreground process doing e.g. the I/O-operations very quickly and have one or more background processes running at a slower pace. We also assume:

> *(Unbounded component memory)* C works without any a priori bounds on the amount of available memory, i.e. its memory space is potentially infinite.

In particular C's memory is never reset during an interactive computation, unless its program explicitly does so. We allow C to build up an 'infinite' database of knowledge that it can consult in the course of its interactions with the environment.

The environment E can behave like an adversary and can send signals in any arbitrary, unpredictable and 'non-algorithmic' way. We do assume it has the full choice of symbols from Σ at its disposal at all times. The fact that E can react in many different ways at any given moment accounts for the variety of output sequences C can produce (generate), when we consider all possible behaviors later on. We assume that, in principle, E can generate any possible signal at any moment but do not concern ourselves with the way E actually comes up with a specific signal in any given circumstance.

Despite the assumed generality, it is conceivable that E is constraint in some way and can generate at some or any given moment only a *selection* of the possible signals from Σ. The constraints could serve to define partial computations, defined only for subsets of possible inputs. If this is the case, we assume that the constraints are of an algorithmic nature (i.e. it can be checked algorithmically that E's response is within the allowed constraints) and never limit the infinity of the interaction. A component C that interacts with its environment according to the given assumptions, will be called an interactive machine (IM). We assume its operation is governed by an evolving process that can be effectively simulated.

It will be helpful to describe an interactive computation of C and E also as a mapping (transduction) of strings e (of environment input) to strings c (of component responses). In this way C acts as an ω-transducer on infinite strings, with the special incremental and interactive behavior as described here (and with game-theoretic connotations). Considerable care is needed in dealing with

sequences of τ's. A sequence of 'silent steps' can often be meaningful to C or E, e.g. for internal computation or timing purposes. It means that a given infinite string $\overline{e}$ may have a multitude of different transductions c, depending on the way E cares to intersperse $\overline{e}$ with 'empty' signals. One should note that E is fully unpredictable in this respect. However, the assumed interactiveness forces E to keep sequences of intermittent empty signals finite.

For the purposes of this paper we assume that the environment E sends a non-empty signal at *every* moment in time. We retain the possibility for C to emit τ's and thus to do internal computations for some time without giving output, even though the assumed interactiveness will force C to give some non-empty output after finite time also.

Definition 1. *The behavior of C with respect to E is the set $T_C = \{(e, \overline{c}) | (e, c)$ is an interaction pair of C and E $\}$. If (e, c) is an interaction pair of C and E, then we also write $T_C(e) = \overline{c}$ and say that $\overline{c}$ is the interactive transduction of e by C.*

Definition 2. *A relation T on infinite strings is called interactively computable if there is an interactive machine C such that $T = T_C$.*

Seemingly simple transductions may be impossible in interactive computation, due to the strict requirement of interactiveness and the unpredictability of the environment. Let $0^\star$ denote the set of finite sequences of 0's (including the empty sequence), $2^\star$ the set of all finite strings over $\{0, 1\}$, and 2^ω the set of infinite strings over $\{0, 1\}$. It can easily be argued that no interactive machine C can exist that transduces its inputs such that input streams that happen to be of the form $1\alpha 1\beta 1\gamma$ are transduced to $1\beta 1\alpha 1\gamma$, with $\alpha, \beta \in 0^\star$ and $\gamma \in 2^\omega$. The example crucially depends on the fact that C cannot predict that the input will be of the special form specified for the transduction.

3 Interactive relations and programs

Given a sequence $u \in 2^\omega$ and $t \geq 0$, let $pref_t(u)$ be the length-t prefix of u. For finite or infinite strings u we write $x \prec u$ if x is a finite (and strict) prefix of u. If x is a finite string, denote its lenght by $|x|$. Modify the common definition of monotonic functions (cf. [30]) for the case of partial functions as follows.

Definition 3. *A partial function $g : 2^\star \to 2^\star$ is called* monotonic *if for all $x, y \in 2^\star$, if $x \prec y$ and $g(y)$ is defined then $g(x)$ is defined as well and $g(x) \preceq g(y)$.*

The following observation captures some aspects of the computational model in terms of monotonic functions.

Theorem 1. *If a relation $T \subseteq 2^\omega \times 2^\omega$ is interactively computable, then there exists a classically computable, monotonic partial function $g : 2^\star \to 2^\star$ such that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if for all $t \geq 0$ $g(pref_t(u))$ is defined, $\lim_{t\to\infty} |g(pref_t(u))| = \infty$ and for all $t \geq 0$ $g(pref_t(u)) \prec v$.*

Proof. Let $T = T_C$. We define g by designing a (Turing) machine M_g for it. Given an arbitrary finite string $x = x_0 x_1 \dots x_{t-1} \in 2^\star$ on its input tape, M_g operates as follows. M_g simulates C using the 'program' of C, feeding it the consecutive symbols of x as input and checking every time it does so whether the next symbol is an input signal that E *could* have given on the basis of the interaction with C up until this moment. To check this, M_g employs the verifier $\mathcal{E}$ which checks the possible constraints on E (and which evolves along with the simulation). As long as no inconsistency is detected, M_g continues with the simulation of the interaction of E and C. Whenever the simulation leads C to output a signal 0 or 1, M_g writes the corresponding symbol to its output tape. When the simulation leads C to output a τ, M_g writes nothing. When the simulation leads C to output a $\sharp$ or when the verifier detects that the input is not consistent with E's possible behavior, then M_g is sent into an indefinite loop. If machine M_g has successfully completed the simulation up to and including the simulation of the (processing of the) final input symbol x_{t-1}, then M_g halts. It follows that M_g terminates if and only if x is a valid beginning of an interaction of E with C, with C's response appearing on the output tape if it halts. The result now follows by observing what properties of g are needed to capture that $(u, v) \in T$ in terms of M_g's action on the prefixes of u. The constraints capture the interactiveness of C and E and the fact that the interaction must be infinite. It is clear from the construction that g is monotonic.

For at least one type of interactively computable relation can the given observation be turned into a complete characterisation. Let a relation $T \subseteq 2^\omega \times 2^\omega$ be called *total* if for every $u \in 2^\omega$ there exists a $v \in 2^\omega$ such that $(u, v) \in T$. Note that the behaviors of interactive machines without special input constraints are always total relations. In the following result the monotonicity of g is *not* assumed beforehand.

Theorem 2. *Let $T \subseteq 2^\omega \times 2^\omega$ be a total relation. T is interactively computable if and only if there exists a classically computable total function $g : 2^\star \to 2^\star$ such that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if $\lim_{t\to\infty} |g(pref_t(u))| = \infty$ and for all $t \geq 0$ $g(pref_t(u)) \prec v$.*

Proof. The 'only if' part follows by inspecting the proof of theorem 1 again. If T is total, then the constructed function g is seen to be total as well and T and g satisfy the stated conditions.

For the 'if' part, assume that $T \subseteq 2^\omega \times 2^\omega$ is a total relation, that g is a computable total function and that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if $\lim_{t\to\infty} |g(pref_t(u))| = \infty$ and for all integers $t \geq 0$ $g(pref_t(u)) \prec v$. To prove that T is interactively computable, design an interactive machine C that operates as follows.

While E feeds input, a foreground process of C keeps buffering the input symbols in a queue $q = q_0 q_1 \dots q_t$ with $t \to \infty$. Let $r \in 2^\star$ be the (finite) output generated by C at any given moment. We will maintain the invariant that q is a prefix of u and r a prefix of v for some pair $(u, v) \in T$ (which exists by the totality of T). Letting q grow into 'u' by the input from E, we let r grow into

'v' by letting C carry out the following background process P every once in a while. C uses a counter c_q that is initially set to 1. C outputs 'empty' signals as long as a call to P is running.

When called, P copies the length-c_q prefix of the queue into x, it increments c_q by one, and computes $g(x)$ using the subroutine for g. (Note that the string now in x extends the string on which the previous call of P operated by precisely one symbol.) By totality of g the subroutine ends in finitely many steps and yields a string y as output. By totality of T and the second condition on g only two cases can occur: $r \prec y$ or $y \preceq r$. If $r \prec y$, then C outputs the symbols by which y extends r one after the other (implicitly updating r as well to account for the new output) and calls P again after it has done so. If $y \preceq r$, C does not generate any special output and simply moves on to another call of P provided at least one further input symbol has entered the queue in the meantime. Because $\lim_{t\to\infty} |g(pref_t(u))| = \infty$, there will be infinitely many calls to P in which the case '$r \prec y$' occurs. Thus r will grow to infinity, with $\lim_{t\to\infty} r = v$ precisely being the output generated by C.

Combining the two theorems we conclude: *the interactively computable total relations are precisely the relations implied 'in the limit' by classically computable, monotonic total functions on* $2^\star$. We will return to this characterisation of interactive computations in Section 6.

It is quite realistic to assume that the initial specification of C is a program, written in some acceptable programming system. For example, the internal operation of C could be modeled by a persistent Turing machine of some sort (as in Wegner and Goldin [27] and Goldin [4]). It is easily argued that interactiveness, as a property of arbitrary programs, is recursively undecidable. The following stronger but still elementary statement can be proved, thanking discussions with B. Rovan: *the set of interactive programs is not recursively enumerable* ([19]).

In the further sections we will study the given model of interactive computing from three different perspectives: recognition, generation and translation (i.e. the computation of mappings on infinite binary strings).

4 Interactive Recognition

Interactive machines typically perform a task in *monitoring*, i.e. the online perception or recognition of patterns in infinite streams of signals from the environment. The notion of recognition is well-known and well-studied in the theory of ω-automata (cf. [15], [16, 17]) and is usually based on the automaton passing an infinite number of times through one or more 'accepting' states during the processing of the infinite input string. In interactive systems this is not detectable (cf. proposition 1 (v) below) and thus not applicable. A component is normally placed in an environment that follows a certain specification, and the component has to observe that this is adhered to. This motivates the following definition.

Definition 4. *An infinite string $\alpha \in 2^\omega$ is said to be recognized by C if $(\alpha, 1^\omega) \in T_C$.*

The definition says that in our model an infinite string α is recognized if C outputs a 1 every once in a while and *no* other symbols except τ's in between during the interactive computation with E, where E generates the infinite string α as input during the computation.

Definition 5. *The set interactively recognized by C with respect to E is the set* $J_C = \{\alpha \in 2^\omega | \alpha$ is recognized by C $\}$.

Note that interactiveness is required on all infinite inputs, i.e. also on the strings that are not recognized. As a curious fact we observe that when it comes to recognition we may assume that C makes no silent steps: letting C output a 1 whenever it wants to output a τ does not affect the set of strings recognized by the component.

Definition 6. *A set $J \subseteq 2^\omega$ is called* interactively recognizable *if there exists a component C such that $J = J_C$.*

Proposition 1. *The following set is interactively recognizable:*
(i) $J = \{\alpha \in 2^\omega | \alpha$ contains at most k ones$\}$, for any fixed integer k,
but the following sets are not *interactively recognizable:*
(ii) $J = \{\alpha \in 2^\omega | \alpha$ contains precisely k ones$\}$, for any fixed integer $k \geq 1$,
(iii) $J = \{\alpha \in 2^\omega | \alpha$ contains finitely many 1's$\}$,
(iv) $J = \{\alpha \in 2^\omega | \alpha$ contains at least k ones$\}$, for any fixed integer $k \geq 1$,
(v) $J = \{\alpha \in 2^\omega | \alpha$ contains infinitely many 1's$\}$.

Proof. We illustrate the typical argument for *(iii)*. Suppose there was an interactive component C that recognized J. Let E input 1's. By interactiveness C must generate a non-empty signal σ at some moment in time. E can now fool C as follows. If $\sigma = 0$, then let E switch to inputting 0's from this moment onward: it means that the resulting input belongs to J but C does not respond with all 1's. If $\sigma = 1$, then let E continue to input 1's. Possibly C outputs a few more 1's but there must come a moment that it outputs a 0. If it didn't then C would recognize the string $1^\omega \notin J$. As soon as C outputs a 0, let E switch to inputting 0's from this moment onward: it means that the resulting input still belongs to J but C does not recognize it properly. Contradiction.

The different power of interactive recognition is perhaps best expressed by the following observation. We assume again that the internal operation of the components that we consider is specified by some program in an acceptable programming system. For sets $J \subseteq 2^\omega$, let $Init(J)$ be the set of all finite prefixes of strings from J. Similar to a result of Cohen and Gold ([2], theorem 7.22) one can show that the initial part of an environment input need not hold any clue about the recognizability of the whole string, as one would expect.

Corollary 1. *There are interactively recognizable sets J such that Init(J) is not recursively enumerable.*

Proof. Consider the set $J = \{0^n 1\{0,1\}^\omega | n \in A\} \cup 0^\omega$ for a non-recursively enumerable set A whose complement is recursively enumerable. It can be shown that J is interactively recognizable. If $Init(J)$ were recursively enumerable, then so would $Init(J) \cap 0^\star 1 = \{0^n 1 | n \in A\}$ and thus A. Contradiction.

Finally we note some rules for constructing new interactively recognizable sets from old ones. The proofs are slightly tedious because of the mechanism of feedback that may exist from the output of a component back to E.

Proposition 2. *The family of interactively recognizable sets of infinite strings is*
(i) closed under $\cup$ *and* $\cap$*, but*
(ii) not *closed under* ω*-complement (i.e. complement in* 2^ω*).*

Proof. For *(i)* we refer to [19]. For *(ii)*, consider the set $J = 0^\omega \cup 0^\star 10^\omega$. By proposition 1 (i) the set is interactively recognizable, but by proposition 1 (iv) its ω-complement is not.

5 Interactive Generation

Interactive machines typically also perform tasks in *controlling* other components. This involves the online translation of infinite streams into other, more specific streams of signals. In this section we will consider what infinite streams of signals an interactive machine can *generate*. The notion of generation is well-known in automata theory and related to matters of definability and expressibility, but it seems not to have been looked at very much in the theory of ω-automata (cf. [15]). Our aim will be to prove that generation and recognition are *duals* in our model. The following definition is intuitive.

Definition 7. *An infinite string* $\beta \in 2^\omega$ *is said to be generated by* C *if there exists an environment input* $\alpha \in \{0,1\}^\omega$ *such that* $(\alpha, \beta) \in T_C$.

Unlike the case for recognition (cf. section 3) one cannot simplify the output capabilities for components C now. In particular we have to allow that C outputs $\sharp$-symbols when it wants to signify that its generation process has gotten off on some wrong track. If C outputs a $\sharp$-symbol, it will automatically trigger E to produce a $\sharp$ as well some finite time later. Strings that contain a $\sharp$ are *not* considered to be validly generated.

Definition 8. *The set interactively generated by interactive machine* C *is the set* $L_C = \{\beta \in 2^\omega | \beta$ is generated by $C\}$.

Observe that, contrary to recognition, C may need to make silent steps while generating. It means that interactive generation is not necessarily an online process.

Definition 9. *A set* $L \subseteq 2^\omega$ *is called* interactively generable *if there exists a component* C *such that* $L = L_C$.

In the context of generating ω-strings, it is of interest to know what finite prefixes an interactive machine C can generate. To this end we consider the following intermediate problem:

> (*Reachability*) Given an interactive machine C and a finite string $\gamma \in \{0,1\}^\star$, is there an interactive computation of C such that the sequence of non-silent symbols generated and output by C at some finite moment equals γ.

It can be shown that the reachability problem for interactive machines C is effectively decidable. The proof involves an application of *König's Unendlichkeitslemma* ([6, 7]). We will now show that the fundamental law that 'what can be generated can be recognized and vice versa' holds for interactive machines. We prove it in two steps.

Lemma 1. *For all sets $J \subseteq 2^\omega$, if J is interactively generable then J is interactively recognizable.*

Proof. Let J be interactively generated by some interactive machine C, i.e. $J = L_C$. To show that J can be interactively recognized, design the following machine C'. Let the input from E be e. C' buffers the input that it receives from E symbol after symbol, and goes through the following cycle of activity: it takes the string r that is currently in the buffer, decides whether r is reachable for C (applying *Reachability*), and outputs a 1 if it is and a 0 if it is not. This cycle is repeated forever, each time taking the new (and longer) string r that is in the buffer whenever a new cycle starts again. Because the reachability problem is decidable in finite time, C' is an interactive machine. Clearly, if an ω-string belongs to J then all its prefixes are reachable for C and C' recognizes it. Conversely, if an ω-string e is recognized by C' then it must be interactively generated by C and hence belong to J. The latter can be argued using the *Unendlichkeitslemma* again.

Lemma 2. *For all sets $J \subseteq 2^\omega$, if J is interactively recognizable then J is interactively generable.*

Proof. Let J be interactively recognizable. Let C be an interactive machine such that $J = J_C$. To show that J can be interactively generated, design the following machine C'. C' buffers the input that it receives from E symbol after symbol, and copies it to output as well (at a slower pace perhaps). At the same time C' runs a simulation of C in the background, inputting the symbols from the buffer one after the other and reacting like E would, keeping track whether the buffered input is exactly the sequence that E would input when taking C's responses into account. Let C' continue to copy input to output as long as no inconsistency between the buffered string and E's simulated behavior arises and as long as C outputs τ's and 1's (in the simulation). If anything else occurs, let C' switch to outputting $\sharp$'s. C' is clearly interactive, and the generated strings are precisely those that C recognizes.

From the given lemmas we conclude the following basic result.

Theorem 3. *For all sets $J \subseteq 2^\omega$, J is interactively generable if and only if J is interactively recognizable.*

The theorem gives evidence that the concepts of interactive recognition and generation as intuitively defined, are well-chosen. Besides being of interest in its own right, theorem 3 also enables one to prove that certain sets are not interactively recognizable by techniques different from Section 4.

6 Interactive Translations

As noted in the previous section, interactive machines typically perform the online translation of infinite streams into other infinite streams of signals. We will consider this further, viewing interactive machines as transducers and viewing the translations (or: transductions) they realize as interactive mappings defined on infinite strings of 0's and 1's. The related notion of ω-transduction in the theory of ω-automata has received quite some attention before (cf. Staiger [15]), but mostly in the context of ω-regular languages only. In this section we will present some basic observations on interactive mappings. Let C be an interactive machine. Let T_C be the behavior of C (cf. definition 2).

Definition 10. *The interactive mapping computed by C is the mapping $f_C : 2^\omega \to 2^\omega$ defined by the following property: $f_C(\alpha) = \beta$ if and only if $(\alpha, \beta) \in T_C$.*

Generally speaking, an interactive mapping is a partial function defined over infinite binary strings. If $f_C(\alpha) = \beta$ (defined), then C actually outputs a sequence $s \in \{0, 1, \tau\}^\omega$ in response to input $\alpha \in \{0, 1\}^\omega$ such that $\overline{s} = \beta$.

Definition 11. *A partial function $f : 2^\omega \to 2^\omega$ is called interactively computable if there exists an interactive machine C such that $f = f_C$.*

Computable functions on incomplete inputs (like finite segments of infinite strings) should be continuous in the sense that, at any moment, any further extension of the input should lead to an extension of the output as it is at that moment and vice versa, without retraction of any earlier output signals. Interactively computable functions clearly all have this property on defined values, which can be more precisely formulated as follows. Modify the classical definition of 'continuous functions' (cf. [30]) for the case of functions on infinite strings as follows.

Definition 12. *A partial function $f : 2^\omega \to 2^\omega$ is called limit-continuous if there exists a classically computable partial function $g : 2^\star \to 2^\star$ such that the following conditions are satisfied:*
(1) g is monotonic, and
(2) for all strictly increasing chains $u_1 \prec u_2 \prec \ldots \prec u_t \prec \ldots$ with $u_t \in 2^\star$ for $t \geq 1$, one has $f(\lim_{t\to\infty} u_t) = \lim_{t\to\infty} g(u_t)$.

In condition (2) the identity is assumed to hold as soon as one of the sides is defined. Clearly monotonic functions map chains into chains, if they are defined on all elements of a chain, but they do not necessarily map strictly increasing chains into chains that are strictly increasing again. Definition 13 implies that if a total function f is limit-continuous, then the corresponding g must be total as well and also map strictly increasing chains into ultimately increasing chains. Using theorems 1 and 2 we now easily conclude the following facts.

Theorem 4. *If $f : 2^\omega \to 2^\omega$ is interactively computable, then f is limit-continuous.*

Theorem 5. *Let $f : 2^\omega \to 2^\omega$ be a total function. Then f is interactively computable if and only if f is limit-continuous.*

Several other properties of interactively computable functions are of interest. For example, without assuming that either f or g is total, it can be shown that if f and g are interactively computable, then so is $f \circ g$ (cf. [19]). The following result is more tedious. It shows that the inverses of 1-1 mappings defined by interactive machines are interactively computable again.

Theorem 6. *Let f be interactively computable and $1-1$. Then f^{-1} is interactively computable as well.*

Proof. Let $f = f_C$ and assume f is $1-1$. If $f(\alpha) = \beta$ (defined) then $f^{-1}(\beta) = \alpha$, and we seek an interactive machine to realize this. Design a machine C' that works as follows.

Let the input supplied so far be r, a finite prefix of 'β'. Assume the environment supplies further symbols in its own way, giving C' a longer and longer part r of the string β to which an original under f is sought. Let C' buffer r internally. We want the output $\sigma = \overline{u}$ of C' at any point to be a finite (and growing) prefix of 'α'. Let this be the case at some point. Now let C' do the following, with more and more symbols coming in and 'revealing' more and more of β.

The dilemma C' faces is whether to output a 0 or a 1 (or, of course, a $\sharp$). In other words, C' must somehow decide whether $\sigma 0$ or $\sigma 1$ is the next longer prefix of the original α under f in the unfolding of β. We argue that this is indeed decidable in finite time. The idea is to look 'into the future' and see which of the two possibilities survives. To this end, we create two processes P_b $(b = 0, 1)$ that explore the future for σb. A P_b works as follows. Remember that symbols continue to come into r and that C' outputs τ's until it knows better.

P_b works on the infinite binary tree $\mathcal{T}$ with its left and right branches labeled 0 and 1, respectively. Every node q of $\mathcal{T}$ can be represented by a finite string p_q, consisting of the 0's and 1's on the path from the root down to q. P_b labels the root by 'Y'. Then it works through the unlabeled nodes q level by level down the tree, testing for every node q whether the string $\sigma b p_q$ is output (i.e. is reachable) by C as it operates on (a prefix of) r, i.e. on a prefix of β. (P_b does this in the usual way, by running the simulation of the interactive computation of C and E.) If $\sigma b p_q$ is reachable, then label q by 'Y'. If the output of C does not reach the end of $\sigma b p_q$ but is consistent with it as far as it gets, then P_b waits at this

node and continues the simulation as more and more symbols come into r. (By interactivity, r will eventually be long enough for C to give an output at least as long as $\sigma b p_q$.) If the output of C begins to differ from $\sigma b p_q$ before the end of $\sigma b p_q$ is reached, then label q by 'N' and prune the tree below it. If the simulation runs into an inconsistency between E's behavior and the string r that is input, then label q by 'N' and prune the tree below it as well. If P_b reaches a tree level where all nodes have been pruned away, it stops. Denote the tree as it gets labeled by P_b by $\mathcal{T}_b$.

Let C' run P_0 and P_1 'in parallel'. We claim that one of the two processes must stop in finite time. Suppose that neither of the two stopped in finite time. Then $\mathcal{T}_0$ and $\mathcal{T}_1$ would both extend to infinite trees as r extends to 'infinity' (i.e. turning into the infinite string β). By the *Unendlichkeitslemma*, $\mathcal{T}_0$ will contain an infinite path δ_0 and likewise $\mathcal{T}_1$ will contain an infinite path δ_1. This clearly implies that both $\sigma 0 \delta_0$ and $\sigma 1 \delta_1$ would be mapped by C to β, which contradicts that f is $1-1$. It follows that at least one of the processes P_0 and P_1 must stop in finite time. (Stated in another way, the process that explores the wrong prefix of α will die out in finite time.) Note that both processes could stop, which happens at some point in case the limit string β has *no* original under f.

Letting C' run P_0 and P_1 'in parallel', do the following as soon as one of the processes stops. If both processes stop, C' outputs $\sharp$. If P_0 stopped but P_1 did not, then output 1. If P_1 stopped but P_0 did not, then output 0. If C' output a b (0 or 1), then it repeats the whole procedure with σ replaced by σb. If it output a $\sharp$ it will continue to output $\sharp$'s from this moment onwards. It is easily seen that C' is interactive and that it computes precisely the inverse of f.

7 Conclusion

In this paper we considered a simple model of interactive computing, consisting of a machine interacting with an 'environment' on infinite streams of input and output symbols that are exchanged in an online manner. The motivation stems from the renewed interest in the computation-theoretic capabilities of interactive computing, in particular by Wegner's claim (cf. [25]) that 'interactive computing is more powerful than algorithms'. In the model we have tried to identify a number of properties which one would intuitively ascribe to components of interactive system, to obtain a generic model of an interactive machine. In [19] and [20] we have used it to model some fundamental interactive features of embedded systems and of the internet, respectively. In the latter case, the model includes the possibility of letting external information enter into the interaction process. Such an intervention may take the form of e.g. a software or hardware upgrade of the underlying device in the course of its interaction with its environment. The resulting model presents a kind of a non-uniformly evolving, interactive device that possesses a computational power that outperforms that of the interaction machines considered in this paper. The steadily increasing role of interactive computing and the understanding of its computational power has led us to propose an extension of the paradigm of 'classical', non-interactive Turing machine

computing in favor of an interactive, possibly non-uniform machine model [21] that captures the nature of contemporary computing. Many interesting problems remain in the further understanding of interactive computing.

Acknowledgements

We are grateful to D.Q. Goldin and T. Muntean for several useful comments and suggestions on a preliminary version of [19].

References

1. M. Broy. A logical basis for modular software and systems engineering, in: B. Rovan (Ed.), *SOFSEM'98: Theory and Practice of Informatics*, Proc. 25th Conference on Current Trends, Lecture Notes in Computer Science, Vol. 1521, Springer-Verlag, Berlin, 1998, pp. 19-35.
2. R.S. Cohen, A.Y. Gold. ω-Computations on Turing machines, *Theor. Comput. Sci.* 6 (1978) 1-23.
3. J. Engelfriet, H.J. Hoogeboom. X-automata on ω-words, *Theor. Comput. Sci.* 110 (1993) 1-51.
4. D.Q. Goldin. Persistent Turing machines as a model of interactive computation, in: K-D. Schewe and B. Thalheim (Eds.), *Foundations of Information and Knowledge Systems*, Proc. First Int. Symposium (FoIKS 2000), Lecture Notes in Computer Science, vol. 1762, Springer-Verlag, Berlin, 2000, pp. 116-135.
5. D. Goldin, P. Wegner. Persistence as a form of interaction, Techn. Rep. CS-98-07, Dept. of Computer Science, Brown University, Providence, RI, 1998.
6. D. König. Sur les correspondances multivoques des ensembles, *Fundam. Math.* 8 (1926) 114-134.
7. D. König. Über eine Schlussweise aus dem Endlichen ins Unendliche (Punktmengen. – Kartenfärben. — Verwantschaftsbeziehungen. – Schachspiel), Acta Litt. Sci. (Sectio Sci. Math.) 3 (1927) 121-130.
8. S. Kosub. Persistent computations, Technical Report No. 217, Institut für Informatik, Julius-Maximilians-Universität Würzburg, 1998.
9. Z. Manna, A. Pnueli. Models for reactivity, *Acta Informatica* 30 (1993) 609-678.
10. R. Milner. *A calculus of communicating systems*, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin, 1980.
11. R. Milner. Elements of interaction, *C.ACM* 36:1 (1993) 78-89.
12. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, in: J.W. de Bakker, W.-P. de Roever and G. Rozenberg, *Current Trends in Concurrency*, Lecture Notes in Computer Science, Vol. 224, Springer-Verlag, Berlin, 1986, pp. 510-585.
13. A. Pnueli. Specification and development of reactive systems, in: H.-J. Kugler (Ed.), *Information Processing 86*, Proceedings IFIP 10th World Computer Congress, Elsevier Science Publishers (North-Holland), Amsterdam, 1986, pp. 845-858.
14. M. Prasse, P. Rittgen. Why Church's Thesis still holds. Some notes on Peter Wegner's tracts on interaction and computability, *The Computer Journal* 41 (1998) 357-362.

15. L. Staiger. ω-Languages, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 6, Springer-Verlag, Berlin, 1997, pp. 339-387.
16. W. Thomas. Automata on infinite objects, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B: *Models and Semantics*, Elsevier Science Publishers, Amsterdam, 1990, pp. 135-191.
17. W. Thomas. Languages, automata, and logic, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 7, Springer-Verlag, Berlin, 1997, pp. 389-455.
18. B.A. Trakhtenbrot. Automata and their interaction: definitional suggestions, in: G. Ciobanu and G. Păun (Eds.), *Fundamentals of Computation Theory*, Proc. 12th International Symposium (FCT'99), Lecture Notes in Computer Science, Vol. 1684, Springer-Verlag, Berlin, 1999, pp. 54-89.
19. J. van Leeuwen, J. Wiedermann. A computational model of interaction, Technical Report, Dept of Computer Science, Utrecht University, Utrecht, 2000.
20. J. van Leeuwen, J. Wiedermann. Breaking the Turing barrier: the case of the Internet, Technical Report, Inst. of Computer Science, Academy of Sciences, Prague, 2000.
21. J. van Leeuwen, J. Wiedermann. Extending the Turing Machine Paradigm, in: B. Engquist and W. Schmidt (Eds.), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2000 (to appear).
22. K. Wagner, L. Staiger. Recursive ω-languages, in: M. Karpinsky (Ed.), *Fundamentals of Computation Theory*, Proc. 1977 Int. FCT-Conference, Lecture Notes in Computer Science, Vol. 56, Springer-Verlag, Berlin, 1977, pp. 532-537.
23. P. Wegner. Interactive foundations of object-based programming, *IEEE Computer* 28:10 (1995) 70-72.
24. P. Wegner. Interaction as a basis for empirical computer science, *Comput. Surv.* 27 (1995) 45-48.
25. P. Wegner. Why interaction is more powerful than algorithms, *C.ACM* 40 (1997) 80-91.
26. P. Wegner. Interactive foundations of computing, *Theor. Comp. Sci.* 192 (1998) 315-351.
27. P. Wegner, D. Goldin. Co-inductive models of finite computing agents, in: B. Jacobs and J. Rutten (Eds.), *CMCS'99-Coalgebraic Methods in Computer Science*, TCS: Electronic Notes in Theoretical Computer Science, Vol. 19, Elsevier, 1999.
28. P. Wegner, D. Goldin. Interaction as a framework for modeling, in: P. Chen *et al.* (Eds.), *Conceptual Modeling - Current Issues and Future Directions*, Lecture Notes in Computer Science, Vol. 1565, Springer-Verlag, Berlin, 1999, pp 243-257.
29. P. Wegner, D.Q. Goldin. Interaction, computability, and Church's thesis, *The Computer Journal* 1999 (to appear).
30. G. Winskel. *The formal semantics of programming languages: an introduction*, The MIT Press, Cambridge (Mass.), 1993.