# A coinductive animation of Turing Machines

Alberto Ciaffaglione

Dipartimento di Matematica e Informatica
Università di Udine, Italia
`alberto.ciaffaglione@uniud.it`

**Abstract.** We adopt corecursion and coinduction to formalize Turing Machines and their operational semantics in the proof assistant `Coq`. By combining the formal analysis of converging and diverging evaluations, our approach allows us to certify the implementation of the functions computed by concrete Turing Machines. Our effort may be seen as a first step towards the formal development of basic computability theory.

## 1 Introduction

In this paper we present and discuss an encoding of *Turing Machines* (TMs) [12] and their semantics in the `Coq` implementation of the *Calculus of (Co)Inductive Constructions* ($\mathrm{CC}^{(\mathrm{Co})\mathrm{Ind}}$). Actually, we do not find in the literature much formalization work dealing with *computability theory*, a foundational, major area of computer science, whereas several other domains have benefited, in recent years, from formal developments carried out within mechanized environments.

As far as we know, the most recent contributions are [10, 2, 1, 13]. Norrish [10] develops a proof of equivalence between the recursive functions and the $\lambda$-calculus computational models, and formalizes some computability theory results in the `HOL4` system. The two works most related to the present one are those focusing on TMs. Asperti and Ricciotti [1] develop computability theory up to the existence of a universal machine, by carrying out their effort from a perspective oriented to complexity theory in `Matita`. Xu, Zhang and Urban [13] prove the undecidability of the halting problem and relate TMs to register machines and recursive functions by formalizing a universal TM in `Isabelle/HOL`.

Actually, TMs form an object system which is challenging in several respects. First, TMs are *non-structured*. Second, the *tape*, used by TMs as workspace for computing, is infinite in both directions. Third, the evaluation of TMs may give rise to *diverging* computations. Therefore, TMs provide with a typical scenario where the user is required to define and reason about *infinite* objects and concepts. To address formally such an object system, in this paper we settle within *Intuitionistic Type Theory*. This framework makes available *coinductive types*, *i.e.*, types that have been conceived to provide finite representations of infinite structures. In particular, a handy technique for dealing with *corecursive* definitions and *coinductive* proofs in $\mathrm{CC}^{(\mathrm{Co})\mathrm{Ind}}$ was introduced by Coquand [4] and refined by Giménez [6]. Such an approach is particularly appealing, because proofs carried out by coinduction are accommodated as any other infinite, corecursively defined object. This technique is mechanized in the `Coq` system [11].

The present work is in fact a departure from the two cited formalizations of TMs, due to the following reasons. On the one hand, we adopt corecursion as *definition* principle and coinduction as *proof* principle (while the alternative contributions do not employ coinductive tools). On the other hand, inspired by our previous effort on unlimited register machines [2], we encode TMs and their operational semantics from the perspective of *program certification*: *i.e.*, we introduce and justify a methodology to prove the correctness of concrete TMs.

The motivations to carry out our formalization of TMs in `Coq` are the following. As it is well-known, traditional papers and textbooks about TMs treat the topic at a more superficial level of detail, and in particular the arguments why individual TMs are correct are often left out. Therefore, the mechanization effort in a proof assistant, besides offering the possibility to discover errors, may typically improve the confidence on the subject (*e.g.*, the correctness proofs for concrete TMs in [13], developed to formalize the undecidability of the halting problem, are acknowledged as the most important contribution). Besides being intellectually stimulating, our work has also the educational objective of popularizing corecursion and coinduction, an aim which is pursued by justifying the formalization methodology in an analytical way and via suggestive examples.

We have used, as starting point for our development, the textbooks by Cutland [5] and by Hopcroft et al. [7]. As an effort towards a broader audience, we display rarely `Coq` code in this paper, but present the encoding at a more abstract level (however, the formalization is available as a web appendix [3]).

*Synopsis.* In the next section we recall TMs, then in the two following sections we introduce their formalization and illustrate the implementation of coinduction in `Coq`. In the two central sections 5 and 6 we define a big-step operational semantics for TMs and address its adequacy via a small-step semantics, respectively. In the core Section 7 we prove the correctness of three sample TMs, then we state final remarks and discuss related and future work.

## 2 Turing Machines

Turing Machines (TMs), one among the frameworks proposed to set up a formal characterization of the intuitive ideas of computability and decidability, perform algorithms as carried out by a human agent using paper and pencil. In this work we address *deterministic*, single tape TMs, as introduced by Cutland [5].

**Alphabet and tape.** TMs operate on a *paper tape*, which is *infinite* in both directions and is divided into single squares along its length. Each square is either blank or contains a symbol from a *finite* set of symbols $s_0, s_1, \ldots, s_n$, named the *alphabet* $\mathcal{A}$ (in fact, the "blank" $B$ is counted as the first symbol $s_0$).

**Specification and computation.** At any time, TMs both scan a single square of the tape (via a *reading/writing head*) and are in one of a *finite* number of *states* $q_1, \ldots, q_m$. Depending on the current state $q_i$ and the symbol being scanned $s_h$, TMs take *actions*, as indicated by a *specification*[1], *i.e.* a *finite* col-

[1] As said above, we deal with deterministic TMs, *i.e.*, *non-ambiguous* specifications: for every pair $q_i, s_h$ there is at most one quadruple of the form $\langle q_i, s_h, x, q_j\rangle$.

lection of quadruples $\langle q_i, s_h, x, q_j \rangle$, where $i, j{\in}[1..m]$, $h{\in}[0..n]$, $x{\in}\{R, L\}{\cup}\mathcal{A}$:

$$\langle q_i, s_h, x, q_j \rangle \triangleq \begin{array}{l} 1)\ \textit{if } x{=}R \textit{ then} \text{ move the head one square to the right} \\ \quad\ \textit{else if } x{=}L \textit{ then} \text{ move the head one square to the left} \\ \quad\ \textit{else if } x{=}s_k\ (k{\in}[0..n]) \textit{ then} \text{ replace } s_h \text{ with } s_k \\ 2)\ \text{change the state from } q_i \text{ into } q_j \end{array}$$

When provided with a tape, a specification becomes an *individual TM*, which is capable to perform a *computation*: it keeps carrying out actions by starting from the initial state $q_1$ and the symbol scanned by the initial position of the head.

Such a computation is said to *converge* if and only if, at some given time, there is no action specified for the current state $q_i$ and the current symbol $s_h$ (that is, there is no quadruple telling what to do next). On the other hand, if this never happens, such a computation is said to *diverge*.

**Computable functions.** TMs may be regarded as devices for computing numerical *functions*, according to the following conventions. A natural number $m$ is represented on a tape by an amount of $m{+}1$ *consecutive* occurrences of the "tally" symbol 1 (in such a way, the representation of the $0{\in}\mathbb{N}$ is distinguished from the blank tape). Then, a machine $M$ computes the *partial* function $f\colon \mathbb{N}{\rightharpoonup}\mathbb{N}$ when, for every $a, b{\in}\mathbb{N}$, the computation under $M$, starting from its initial state and the leftmost 1 of the $a$ representation, stops with a tape that contains a *total* of $b$ symbols 1 (not necessarily consecutive) *if and only if* $a{\in}dom(f)$ and $f(a){=}b$ (therefore $f$ is undefined on all inputs $a$ that make the computation diverge). $n$-ary partial functions $g\colon \mathbb{N}^n{\rightharpoonup}\mathbb{N}$ are computed in a similar way, where the representations of the $n$ inputs are separated by single blank squares.

Consequently, computability theory can be developed via TMs, leading to the well-known characterization of the class of effectively computable functions.

## 3 Turing Machines in `Coq`

As described in the previous section, TMs are formed by two components: the specification and the tape, whose content in fact instantiates the former, making it executable. Specifications and tapes actually work together, but are evidently independent of each other from the point of view of the formalization matter.

Our encoding of TMs in `Coq` reflects such an independence: in the present work we are mainly interested in the formal treatment of the tape, which is more problematic and particularly delicate; conversely, we do not pursue the specification-component management (*automata* are actually supported by `Coq`'s library), thus keeping that part of the formalization down to a minimum.

*Specification and tape.* Concerning the *specification* part, we represent states via natural numbers (reserving the 0 for the halting state, for which no transition is provided), while alphabet symbols and operations performed by the head are finite collections of elements (we fix the alphabet by adding the "mark" symbol 0 to the "blank" $B$ and the "tally" 1 of previous section). Finally, specifications are finite sequences (*i.e.*, lists) of actions (*i.e.*, quadruples)[2]:

$$\begin{array}{llllll}
\mathit{State}: & p,q,i & \in & \mathbb{N}=\{0,1,2,\ldots\} & \text{state} \\
\mathit{Sym}\ : & a,b & \in & \{B,1,0\} & \text{alphabet symbol} \\
\mathit{Head}: & x,y & \in & \{R,L,W(a)\} & \text{head operation} \\
\mathit{Act}\ : & \alpha & \in & \mathit{State}\times\mathit{Sym}\times\mathit{State}\times\mathit{Head} & \text{action} \\
\mathit{Spec}\ : & T,U,V & ::= & (\iota\mapsto\alpha_\iota)^{\iota\in[0..n]}\ (n\in\mathbb{N}) & \text{specification}
\end{array}$$

To formalize the *tape*, whose squares are scanned by the head and contain the alphabet symbols, we adopt a pair of *streams* (*a.k.a.* infinite sequences), a datatype borrowed from the `Haskell` community, where is named "zipper":

$$\begin{array}{llll}
\mathit{HTape}: & l,r & ::= (\iota\mapsto a_\iota)^{\iota\in[0..\infty]} & \text{half tape (stream)} \\
\mathit{Tape}\ : & s,t,u & ::= \langle\!\langle\, l,r \,\rangle\!\rangle & \text{full tape (zipper)}
\end{array}$$

The intended meaning of this encoding is that the second stream ($r = r_0{:}r_1{:}\ldots$) models the infiniteness of the tape towards the right, while the first stream ($l = l_0{:}l_1{:}\ldots$) is infinite towards the left. At any time, the head "$\Downarrow$" will be scrutinizing the first symbol of $r$, which corresponds physically to:

$$\begin{array}{c}
\qquad\qquad\Downarrow \\
\cdots \mid l_1 \mid l_0 \mid r_0 \mid r_1 \mid \cdots
\end{array}$$

This representation allows for a direct access to the content of the tape, an operation which has therefore constant complexity (see the next section).

*Transitions.* To make specifications concretely compute, it is necessary, given the current state and tape symbol, to extract from such lists the corresponding target state and head operation. In our encoding, this task is carried out by a *transition* function $tr$: $\mathit{Spec} \to \mathit{State} \to \mathit{Sym} \to (\mathit{State} * \mathit{Head})$.

In fact, we delegate to this transition function the responsibility to guarantee the *determinism* of TMs. We implement $tr$ as a recursive function that scans a list-like specification $T$: given an input pair $(p,a)$, the target state and head operation are obtained from the *first* quadruple of shape $\langle p,a,q,x\rangle$ found in $T$ (no matter if there are other ones with form $\langle p,a,i,y\rangle$); if, on the other hand, there is no corresponding quadruple in $T$, $tr$ returns an "halting" output:

```
Parameter halt: (State * Head).
```

The motivation for this naïve encoding of determinism is, as said at the beginning of the section, to keep the formalization as minimal as possible, being the modelling and the management of the tape the focus of our investigation.

## 4 Coinduction in `Coq`

The proof assistant `Coq` supports the formal treatment of circular, infinite data and relations by means of the mechanism of *coinductive types.*

[2] The middle columns display the metavariables and the datatypes they range over.

First of all, one may formalize concrete, infinite *objects* (*i.e.*, data) as elements of coinductive *sets*[3], which are fully described by a set of *constructors*. From a pure logical point of view, the constructors can be seen as *introduction rules*; these are interpreted coinductively, that is, they are applied infinitely many times, hence the type being defined is inhabited by infinite objects:

$$\frac{a \in Sym \quad h \in HTape}{a{:}h \in HTape} \; (HTape)_\infty$$

In this example we have formalized (via the *cons* constructor) infinite sequences, *i.e.*, streams, of symbols in the alphabet $Sym {=} \{B, 1, 0\}$, the coinductive set $HTape$ which we have introduced in the Section 3 to model the tape of TMs.

Once a new coinductive type is defined, the system supplies automatically the *destructors*, that is, an extension of the native pattern-matching capability, to *consume* the elements of the type itself. Therefore, coinductive types can also be viewed as the *largest* collection of objects closed *w.r.t.* the destructors. We use here the standard *match* destructor to extract the *head* and *tail* from streams:

$$head(h) \triangleq \text{match } h \text{ with } a{:}k \Rightarrow a \qquad tail(h) \triangleq \text{match } h \text{ with } a{:}k \Rightarrow k$$

However, the destructors *cannot* be used for defining functions by *recursion* on coinductive types, because it is not possible to consume their elements down to a base case. In fact, the natural way to allow self-reference with coinductive types is the *dual* approach of *building* objects that belong to them. Such a goal is fulfilled by defining *corecursive* functions, like, *e.g.*, the following ones:

$$Bs \triangleq B{:}Bs \qquad same(a) \triangleq a{:}same(a) \qquad blink(a,b) \triangleq a{:}b{:}blink(a,b)$$
$$merge(h,k) \triangleq \text{match } h \text{ with } a{:}h' \Rightarrow \text{match } k \text{ with } b{:}k' \Rightarrow a{:}b{:}merge(h',k')$$

Corecursive functions yield infinite objects and may have any type as domain (notice that in the last definition the two parameters are infinite objects as well). To prevent the evaluation of corecursive functions from infinitely looping, their definition must satisfy a *guardedness condition*: every corecursive call has to be guarded by *at least* one constructor (":" in the definitions above) and by *nothing but* constructors[4]. In fact, corecursive functions are never unfolded in `Coq`, unless their elements are explicitly needed, "on demand", by a destruction operation. This way of regulating the implementation of corecursion is inspired by *lazy* functional languages, where the constructors do not evaluate their arguments.

Given a coinductive set (such as $HTape$ above), no *proof principle* can be automatically generated by the system: actually, proving properties about infinite objects requires the potential of building *proofs* which are infinite too. What is needed is the design of *ad-hoc* coinductive *predicates* (*i.e.*, relations)[5]; these

[3] Coinductive sets are coinductive types whose type is the sort `Set`.

[4] Syntactically, the constructors guard the corecursive call "on the left"; this captures the intuition that infinite objects are built via the repetition of a productive step.

[5] Coinductive predicates are coinductive types whose type is the sort `Prop`.

types are in fact inhabited by infinite *proof terms*. The traditional example is *bisimilarity*, that we define on streams and name $\simeq \subseteq HTape \times HTape$:

$$\frac{a \in Sym \quad h, k \in HTape \quad h \simeq k}{a{:}h \simeq a{:}k}\ (\simeq)_\infty$$

Two streams are bisimilar if we can observe that their heads coincide and, recursively, *i.e.*, *coinductively*, their tails are bisimilar. Once this new predicate is defined, the system provides a corresponding proof principle, to carry out proofs about bisimilarity: such a tool, named "guarded induction" principle [4, 6], is particularly appealing in a context where proofs are managed as any other infinite object. In fact, a bisimilarity proof is just an infinite proof term built by corecursion (hence, it must respect the same guardedness constraint that corecursive functions have to). The guarded induction principle provides a handy technique for building proofs inhabiting coinductive predicates, as such proofs can be carried out *interactively* through the `cofix` tactic[6]. This tactic allows the user to yield proof terms as *infinitely regressive* proofs, by assuming the thesis as an extra hypothesis and using it later with care, *i.e.*, provided its application is guarded by constructors. In this way the user is not required to pick out any bisimulation beforehand, but may build it incrementally, via tactics.

To illustrate the support provided by the `cofix` tactic, we display below the proof of the property $\forall a, b{\in}Sym.\ merge(same(a), same(b)) \simeq blink(a, b)$, in *natural deduction* style[7]. By mimicking `Coq`'s top-down proof practice, first the coinductive hypothesis is assumed among the hypotheses[8]; then, the corecursive functions *same*, *blink* and *merge*, in turn, are unfolded to perform a computation step; finally, the constructor $(\simeq)_\infty$ is applied twice. Hence, the initial goal is reduced to $merge(same(a), same(b)) \simeq blink(a, b)$, *i.e.*, an instance of the coinductive hypothesis. Therefore, the user is eventually allowed to exploit (*i.e.*, discharge) such a hypothesis, whose application is now guarded by the constructor $(\simeq)_\infty$. The application of the coinductive hypothesis in fact completes the proof, and intuitively has the effect of repeating ad infinitum the initial fragment of the proof term, thus realizing the "and so on forever" motto:

$$\dfrac{a, b{\in}Sym \qquad \dfrac{\dfrac{\dfrac{\begin{array}{c}[\forall a, b{\in}Sym.\ merge(same(a), same(b)) \simeq blink(a, b)]_{(1)}\\ \vdots \\ merge(same(a), same(b)) \simeq blink(a, b)\end{array}}{a{:}b{:}merge(same(a), same(b)) \simeq a{:}b{:}blink(a, b)}\ (\simeq)_\infty, twice}{merge(a{:}same(a), b{:}same(b)) \simeq a{:}b{:}blink(a, b)}\ (def\!:\ merge)}{merge(same(a), same(b)) \simeq blink(a, b)}\ (def\!:\ same,\ blink)}{\forall a, b{\in}Sym.\ merge(same(a), same(b)) \simeq blink(a, b)}\ (1), (introduction)$$

[6] A tactic is a command to solve a goal or decompose it into simpler goals.

[7] As usual, local hypotheses are indexed with the rules they are discharged by.

[8] According to Gentzen's notation, we write such an hypothesis (among the leaves of the proof tree) within square brackets, to bear in mind that it can be *discharged*, *i.e.*, cancelled, in the course of a formal proof, as it represents a *local* hypothesis.

## 5 Operational semantics

As stressed in Sections 2 and 3, the semantics of TMs' specifications is parametric *w.r.t.* tapes: computations, induced by specifications, may either converge or diverge, depending on the tape that is coupled to them and the initial position of the head (while the initial state is $1 \in \mathbb{N}$). In Section 3 we have also chosen an encoding for tapes (via a zipper, made of two streams) such that the position of the head is implicit within the tape itself. Therefore, the semantics of TMs may be defined by considering configurations $(T, p, s)$, where $T$ is a specification, $p$ a state, and $s = \langle\!\langle l, r = r_0{:}r_1{:}\ldots \rangle\!\rangle$ a tape. Some configurations make actually a computation stop, because there is no action specified by $T$ for the current state $p$ and symbol $r_0$: these configurations will play the role of the *values* of our semantics. In the following, we will denote with $tr(T, p, s)$ the application of the transition function $tr$, introduced in Section 3: in particular, we will write $tr(T, p, s) = \downarrow$ for `(tr T p r0)=halt`, and $tr(T, p, s) = \langle i, x \rangle$ for `(tr T p r0)=(i,x)`.

In this section we define a *big-step* semantics for TMs, which will play the role of our main tool throughout the rest of the paper. The *potential* divergence of computations provides us with a typical scenario which may benefit from the use of *coinductive* specification and proof principles. In fact, a faithful encoding has to reflect the separation between converging and diverging computations, through two different judgments. Hence, we define the *inductive* predicate $b_* \subseteq Spec \times Tape \times State \times Tape \times State$ to cope with converging evaluations, and the *coinductive* $b_\infty \subseteq Spec \times Tape \times State$ to deal with diverging ones.

**Definition 1.** *(Evaluation) Assume* $T \in Spec$, $s = \langle\!\langle l = l_0{:}l_1{:}\ldots, r = r_0{:}r_1{:}\ldots \rangle\!\rangle$ *and* $t \in Tape$, $p, q, i \in State$. *Then,* $b_*$ *is defined by the following inductive rules:*

$$\frac{tr(T,p,s) = \downarrow}{b_*(T, s, p, s, p)}\ (stop) \qquad \frac{tr(T,p,s) = \langle i, R \rangle \quad b_*(T, \langle\!\langle r_0{:}l, tail(r) \rangle\!\rangle, i, t, q)}{b_*(T, \langle\!\langle l, r \rangle\!\rangle, p, t, q)}\ (right)_*$$

$$\frac{tr(T,p,s) = \langle i, L \rangle \quad b_*(T, \langle\!\langle tail(l), l_0{:}r \rangle\!\rangle, i, t, q)}{b_*(T, \langle\!\langle l, r \rangle\!\rangle, p, t, q)}\ (left)_*$$

$$\frac{tr(T,p,s) = \langle i, W(a) \rangle \quad b_*(T, \langle\!\langle l, a{:}tail(r) \rangle\!\rangle, i, t, q)}{b_*(T, \langle\!\langle l, r \rangle\!\rangle, p, t, q)}\ (write)_*$$

*And* $b_\infty$ *is defined by the following rules, (this time) interpreted coinductively*[9]*:*

$$\frac{tr(T,p,s) = \langle q, R \rangle \quad b_\infty(T, \langle\!\langle r_0{:}l, tail(r) \rangle\!\rangle, q)}{b_\infty(T, \langle\!\langle l, r \rangle\!\rangle, p)}\ (right)_\infty$$

$$\frac{tr(T,p,s) = \langle q, L \rangle \quad b_\infty(T, \langle\!\langle tail(l), l_0{:}r \rangle\!\rangle, q)}{b_\infty(T, \langle\!\langle l, r \rangle\!\rangle, p)}\ (left)_\infty$$

[9] The relation $b_\infty$ is the greatest fixed-point of the above rules, or, equivalently, amounts to the conclusions of infinite derivation trees built from such rules.

$$\frac{tr(T,p,s){=}\langle q, W(a)\rangle \quad b_{\infty}(T,\langle\!\langle l, a{:}tail(r)\rangle\!\rangle, q)}{b_{\infty}(T,\langle\!\langle l,r\rangle\!\rangle,p)}\ (write)_{\infty}$$

*Notice that in the rules above we write $r_0$ and $l_0$ for head$(r)$ and head$(l)$, respectively (see Section 4 for the definitions of the* head *and* tail *functions).* □

In our semantics, given a specification $T$, a tape $s$ and a state $p$, we capture on the one hand the *progress* of both the head and the states transitions, and on the other hand the *effect* of the operations performed by the head itself.

In detail, the intended meaning of $b_*(T,s,p,t,q)$ is that the computation under the specification $T$, by starting from the tape $s$ and the state $p$, *stops* in the state $q$, transforming $s$ into $t$. Conversely, $b_\infty(T,s,p)$ asserts that the computation under $T$, by starting from the tape $s$ and the state $p$, *loops*: *i.e.*, there exist a state $i$ and a pattern-tape $u$ (reachable from $p$ and $s$) such that, afterwards, the computation gets again to the state $i$ with a tape fulfilling $u$ after a non-zero, finite number of actions. Therefore, a *final* tape cannot exist for $b_\infty$, because the initial $s$ is scrutinized (and possibly updated) "ad infinitum".

Since TMs are not structured, we have embedded in the big-step semantics an alternative *structuring criterion*, *i.e.*, the number of evaluation steps implicit amount. In fact, we have defined a base (*i.e.*, non-recursive) rule for $b_*$ (the computation stops because no next action exists) and (co)inductive rules for both $b_*$ and $b_\infty$, to address how moving the head and writing on the tape is carried out within a converging computation and a diverging one, respectively.

We remark again that the benefit of the zipper encoding of tapes (introduced in Section 3) is that every operation of the head may be carried out via basic functions on streams, whose complexity is minimal and constant.

## 6 Adequacy

To argue that our big-step semantics for TMs is appropriate, we introduce here a *small-step* semantics *à la* Leroy [9], and prove that they are equivalent.

We first define a *one-step* reduction concept, to express the three basic actions of TMs (*i.e.*, moving the reading head and writing on the current square). Formally, it is defined as a predicate $\rightarrow \subseteq Spec{\times}Tape{\times}State{\times}Tape{\times}State$, that we write more suggestively as $(T,s,p)\rightarrow(T,t,q)$. Note (again) that, since TMs are not structured, we do not need to define *contextual* reduction rules.

Now we can formalize the small-step semantics as reduction sequences: *finite* reductions $\overset{*}{\rightarrow}$, defined by *induction*, are the reflexive transitive closure of $\rightarrow$, while *infinite* reductions $\overset{\infty}{\rightarrow}$, defined by *coinduction*, its transitive closure.

**Definition 2.** *(Reduction) Assume $T{\in}Spec$, $s{=}\langle\!\langle l,r\rangle\!\rangle{\in}Tape$, and $p,q{\in}State$. Then, the* one-step *reduction $\rightarrow$ is defined by the following rules:*

$$\frac{tr(T,p,s){=}\langle q,R\rangle}{(T,\langle\!\langle l,r\rangle\!\rangle,p)\rightarrow(T,\langle\!\langle r_0{:}l, tail(r)\rangle\!\rangle,q)}\ (\rightarrow_R)$$

$$\frac{tr(T,p,s){=}\langle q,L\rangle}{(T,\langle\!\langle l,r\rangle\!\rangle,p)\rightarrow(T,\langle\!\langle tail(l), l_0{:}r\rangle\!\rangle,q)}\ (\rightarrow_L)$$

$$\frac{tr(T,p,s){=}\langle q, W(a)\rangle}{(T, \langle\!\langle l,r\rangle\!\rangle, p) \to (T, \langle\!\langle l, a{:}tail(r)\rangle\!\rangle, q)}\ (\to_W)$$

*For* $t, u{\in}Tape$, $i{\in}State$, finite *reduction* $\overset{*}{\to}$ *is defined by induction, via the rules:*

$$\frac{}{(T,s,p) \overset{*}{\to} (T,s,p)}\ (\overset{*}{\to}_0) \qquad \frac{(T,s,p) \to (T,u,i) \quad (T,u,i) \overset{*}{\to} (T,t,q)}{(T,s,p) \overset{*}{\to} (T,t,q)}\ (\overset{*}{\to}_+)$$

*And* infinite *reduction* $\overset{\infty}{\to}$ *is defined by the following coinductive rule:*

$$\frac{(T,s,p) \to (T,t,q) \quad (T,t,q) \overset{\infty}{\to}}{(T,s,p) \overset{\infty}{\to}}\ (\overset{\infty}{\to}_\infty)$$

We can prove that evaluation and reduction are equivalent concepts, both in their converging and diverging versions. We remark that our proofs are *constructive*, whereas Leroy [9] had to postulate the "excluded middle" for divergence.

**Proposition 1.** *(Equivalence) Let be* $T{\in}Spec$, $s,t,u{\in}Tape$, *and* $p,q,i{\in}State$.

1. *If* $(T,s,p) \to (T,u,i)$ *and* $b_*(T,u,i,t,q)$, *then* $b_*(T,s,p,t,q)$
2. *If* $(T,s,p) \overset{*}{\to} (T,u,i)$ *and* $b_*(T,u,i,t,q)$, *then* $b_*(T,s,p,t,q)$
3. $b_*(T,s,p,t,q)$ *if and only if* $(T,s,p) \overset{*}{\to} (T,t,q)$ *and* $tr(T,q,t){=}\downarrow$
4. $b_\infty(T,s,p)$ *if and only if* $(T,s,p) \overset{\infty}{\to}$

*Proof. 1) By inversion of the first hypothesis. 2) By structural induction on the derivation of* $(T,s,p) \overset{*}{\to} (T,u,i)$, *and point 1. 3) Both directions are proved by structural induction on the hypothetical derivation, but the direction (⇐) requires also point 1. 4) Both directions by coinduction and hypothesis inversion.* □

The above result points out that the proof practice of reduction and evaluation is very similar in `Coq`. In fact, the small-step predicate $\overset{*}{\to}$ is slightly less handy, because, to perform a TM action, the user is required to exhibit the witness tape, besides the target state; obviously, the small-step version lacks the "halting" concept (*i.e.*, $tr(T,q,t){=}\downarrow$), which is internalized by the big-step judgment.

*Streams vs. lists.* We complete this section with a digression about a different encoding for tapes, that we pursued in a preliminary phase of our research.

Even if streams are a datatype which captures promptly and naturally the infiniteness of tapes, a formalization approach via (finite) *lists* may also be developed: in this case, the empty list is intended to represent an infinite sequence of blanks. The choice of lists makes explicit the assumption about TMs that, when a computation starts, only a finite number of squares can contain non-blank symbols (in fact, the representation of numerical functions in Cutland's setting, that we have adopted at the end of Section 2, respects such a constraint).

Therefore, we proceed by encoding the tape through a pair of lists:

| | | |
|---|---|---|
| $HTape_L$ | $: ll, rl ::= (\iota{\mapsto}a_\iota)^{\iota\in[0..n]}$ | half tape (list, $n{\in}\mathbb{N}$) |
| $Tape_L$ | $: sl, tl ::= \langle\!\langle\ ll, rl\ \rangle\!\rangle$ | full tape (list-pair) |

Afterwards, big-step semantics predicates, playing the role of the ones that deal with streams in Section 5, can be introduced. However, since lists (conversely to streams) might be empty, such predicates must take into consideration this extra pattern and manage it via additional rules. Without going into the full details (for lack of space), we display here the rules for the move-R action[10]:

$$\frac{bL_*(T, \langle\!\langle\, B{:}ll, [\,]\,\rangle\!\rangle, i, t, q)}{bL_*(T, \langle\!\langle\, ll, [\,]\,\rangle\!\rangle, p, t, q)}\ (r_{[\,]})_* \qquad \frac{bL_*(T, \langle\!\langle\, a{:}ll, rl\,\rangle\!\rangle, i, t, q)}{bL_*(T, \langle\!\langle\, ll, a{:}rl\,\rangle\!\rangle, p, t, q)}\ (r_L)_*$$

The inductive convergence predicate $bL_* \subseteq Spec{\times}Tape_L{\times}State{\times}Tape_L{\times}State$ has the same intended meaning of $b_*$. The coinductive divergence predicate $bL_\infty \subseteq Spec{\times}Tape_L{\times}State$, corresponding to $b_\infty$, is defined analogously.

By using the predicates $bL_*$ and $bL_\infty$, we can prove that the semantics with streams may mimic that with lists, and a limited form of the opposite result (in the Proposition below we denote with $Bs$ the stream of blank symbols and with "::" a recursive function that appends a list in front of a stream).

**Proposition 2.** *(Tape) Let be $T{\in}Spec$, $ll, rl, ll', rl'{\in}HTape_L$, and $p, q{\in}State$.*

1. *If $bL_*(T, \langle\!\langle\, ll, rl\,\rangle\!\rangle, p, \langle\!\langle\, ll', rl'\,\rangle\!\rangle, q)$,*
   *then $b_*(T, \langle\!\langle\, ll{::}Bs, rl{::}Bs\,\rangle\!\rangle, p, \langle\!\langle\, ll'{::}Bs, rl'{::}Bs\,\rangle\!\rangle, q)$*
2. *$bL_\infty(T, \langle\!\langle\, ll,\ rl\,\rangle\!\rangle, p)$ if and only if $b_\infty(T, \langle\!\langle\, ll{::}Bs,\ rl{::}Bs\,\rangle\!\rangle, p)$*

*Proof. 1) By structural induction on the hypothetical derivation. 2) Both the directions are proved by coinduction and hypothesis inversion.* □

The difficulty of proving the reverse implication of point 1 above depends on the fact that the representation of the tape through lists is not unique, because one may append to any list blank symbols at will; hence, it is necessary to introduce an *equivalence* relation on list-tapes to develop their metatheory. For this reason (and because lists demand to double the length of proofs, as their predicates have two constructors for any action), we prefer working with streams.

## 7 Certification

In this section we use the big-step predicates $b_*$ and $b_\infty$, introduced in Section 5 and justified in Section 6, to address the *certification* of the partial functions computed by *individual* TMs. This "algorithmic" approach, which exploits corecursion and coinduction in an involved setting, is significant as it provides a foundation methodology for the formal development of computability theory.

The divergence of TMs may be caused by different kinds of behavior. Clearly, it is easy to manage the scenario where a finite portion of the tape is scanned. The interesting case is when TMs scrutinize an infinite area of it; this may happen by moving the head infinitely either just in one direction or in both directions. In this section we address one example for each pattern of behavior, to convey to the reader the confidence that we can master all of them.

[10] We omit from both the rules the transition conditions, that is, the premise $tr(T, p, \langle\!\langle\, ll, [\,]\,\rangle\!\rangle){=}\langle i, R\rangle$ from $(r_{[\,]})_*$ and $tr(T, p, \langle\!\langle\, ll, a{:}rl\,\rangle\!\rangle){=}\langle i, R\rangle$ from $(r_L)_*$.

*First example: R moves.* The first partial function that we work out computes the half of *even* natural numbers, and is not defined on *odd* ones:

$$div2(n) \triangleq \begin{cases} n/2 & \text{if } n \in \mathbb{E} \\ \uparrow & \text{if } n \in \mathbb{O} \end{cases}$$

One algorithm that implements the *div*2 function is conceived as follows. Erase the first "1" (which occurs by definition) and move the head to the right; then try to find pairs of consecutive "1": if this succeeds, erase the second "1" and restart the cycle, otherwise (a single "1" is found) move indefinitely to the right.

Such an algorithm can be realized, *e.g.*, by the following specification $T$:

$$\{\langle 1,1,W(B),1\rangle, \langle 1,B,R,2\rangle, \langle 2,1,R,3\rangle, \langle 3,B,R,3\rangle, \langle 3,1,W(B),4\rangle, \langle 4,B,R,2\rangle\}$$

This implementation of the *div*2 function is certified through the predicates $b_*$ and $b_\infty$; the computation starts from the state 1 and the following tape[11]:

$$- \mid B \mid \overset{\Downarrow}{1} \mid \underbrace{1 \mid - \mid 1}_{\text{n}} \mid B \mid - \tag{1}$$

which is formalized as $\forall n.\ \langle\!\langle Bs,\ 1{:}ones(n){::}Bs\rangle\!\rangle$, where $Bs$ is the stream of blank symbols, $ones(n)$ a list of $n$ consecutive "1" symbols, "::" a recursive function that appends a list in front of a stream, and ":" the *cons* constructor on streams.

To fulfill our goal we carry out, via tactics, a top-down formal development that simulates the computation of the TM at hand. First, we perform a write-B and a move-R action from the starting *configuration*[12] (state 1 and tape (1), that represents the input $n$), thus reaching the state 2 with the tape:

$$- \mid B \mid \underbrace{\overset{\Downarrow}{1} \mid - \mid 1}_{\text{n}} \mid B \mid - \tag{2}$$

Proving the *divergence* requires a combination of coinductive and inductive reasoning. The core property is the divergence when proceeding from the state 3 and a right-hand blank tape, a lemma which is proved by coinduction[13]:

$$\dfrac{l\in HTape \qquad \dfrac{\dfrac{tr(T,3,\langle\!\langle l,B{:}Bs\rangle\!\rangle)=\langle 3,R\rangle \qquad \begin{array}{c}[\forall\, l\in HTape.\ b_\infty(T,\langle\!\langle l,Bs\rangle\!\rangle,3)]_{(1)}\\ \vdots\\ b_\infty(T,\langle\!\langle B{:}l,Bs\rangle\!\rangle,3)\end{array}}{b_\infty(T,\langle\!\langle l,B{:}Bs\rangle\!\rangle,3)}\,(right)_\infty}{b_\infty(T,\langle\!\langle l,Bs\rangle\!\rangle,3)}\,(def\colon Bs)}{\forall\, l\in HTape.\ b_\infty(T,\langle\!\langle l,Bs\rangle\!\rangle,3)}\,(1),(introduction) \tag{3}$$

[11] From now on, we will use "$a \mid -$" to represent an infinite amount of "$a$" symbols.
[12] Given a specification $T$, a configuration will be a pair $\langle state, tape\rangle$ from now on.
[13] Like at the end of Section 4, we display coinductive proofs in natural deduction-style: the coinductive hypothesis is indexed with the rule it is discharged by.

If $n$ is *odd*, we prove by induction on $k$ that the tape (2) leads to divergence:

$$\forall\, l \in HTape.\ b_\infty(T, \langle\!\langle\, l,\ ones(2k{+}1)::Bs \,\rangle\!\rangle, 2)$$

If $k{=}0$, carry out a move-R and apply the lemma (3) above; if $k{=}h{+}1$, complete a cycle (by erasing the second “1”) and conclude via the induction hypothesis.

We address the *convergence* in the complementary scenario (an *even* input $n$ in (2)) by proving the following property, again by induction on $k$:

$$\forall\, l \in HTape.\ b_*(T, \langle\!\langle\, l,\ ones(2k)::Bs \,\rangle\!\rangle, 2, \langle\!\langle\, rpt(k)::l,\ Bs \,\rangle\!\rangle, 2)$$

where $rpt(k)$ in the final tape stands for a list of $k$ consecutive pairs “$B$:1”. $\square$

*Second example: R and L moves.* The second sample function that we choose is partially defined on input *pairs*, and may be named “partial minus”:

$$pminus(m, n) \triangleq \begin{cases} m - n & \text{if } m \geq n \\ \uparrow & \text{if } m < n \end{cases}$$

To compute it, we devise the following algorithm. First scan the tape towards the right till reaching the $B$ that separates the two inputs; then erase the leftmost “1” from the representation of $n$ and the rightmost “1” from that of $m$ (both the “1s” must occur) by replacing them, respectively, with a mark symbol “0” (on the right, for $n$) and a $B$ (on the left). The core of the computation is repeating this cycle, which leads to one of two possible situations: if the end of $n$ is reached (*i.e.*, we are scanning the first $B$ on the right of a 0-block), then stop; on the other hand, replacing $m$ with $B$ symbols may cause that the head (looking for “1$s$”) moves indefinitely on the left. The specification is the following:

$$\begin{aligned} U \triangleq \{ & \langle 1, 1, R, 1\rangle, \langle 1, B, R, 2\rangle, \langle 2, 0, R, 2\rangle, \langle 2, 1, W(0), 3\rangle, \langle 3, 0, L, 3\rangle, \\ & \langle 3, B, L, 4\rangle, \langle 4, B, L, 4\rangle, \langle 4, 1, W(B), 5\rangle, \langle 5, B, R, 5\rangle, \langle 5, 0, R, 2\rangle \} \end{aligned}$$

The initial part of the formal development (erasing the first pair of “1s”, so moving from state 1 to 5) is common to the divergence and convergence cases[14]:

$$- \mid B \mid \underbrace{\overset{\Downarrow}{1} \mid - \mid 1}_{\text{m+1}} \mid B \mid \underbrace{1 \mid - \mid 1}_{\text{n+1}} \mid B \mid - \overset{*}{\Longrightarrow} - \mid B \mid 1^m \mid B \mid B \mid \overset{\Downarrow}{0} \mid 1^n \mid B \mid -$$

At this point of the proof, the key pattern to be mastered is shaped as follows:

$$- \mid B \mid \underbrace{1 \mid - \mid 1}_{\text{m}} \mid \underbrace{B \mid - \mid B}_{\text{k+2}} \mid \underbrace{\overset{\Downarrow}{0} \mid - \mid 0}_{\text{k+1}} \mid \underbrace{1 \mid - \mid 1}_{\text{n}} \mid B \mid - \qquad (4)$$

[14] Informally, we represent with $\overset{*}{\Longrightarrow}$ the effect of a finite number of actions on a tape. Moreover, we denote with $1^m$ a block of $m$ consecutive squares with the “1” symbol.

Starting from this tape and the state 5, we can discriminate between divergence and convergence by distinguishing the case $m<n$ from $m\geq n$. Notice that we have introduced the variable $k$ to obtain a more general induction hypothesis.

When we come to the state 5 and an instance (for $k{=}1$) of the above tape (4) we prove the *divergence*, under the hypothesis $m<n$, by nested induction on $n$ and $m$. This proof requires auxiliary lemmas, to scan 0-blocks and $B$-blocks (by induction on $k$) and for assuring the divergence from the state 4 with the tape $Bs$ towards the left. One key point is that we can use the predicate $b_\infty$ in a *compositional* way: *i.e.*, when carrying out a divergence proof in top-down fashion, we can perform a preliminary finite number of actions, thus reducing to a different goal. In fact, this amounts to split a divergent computation into a convergent one, easily provable, plus another divergent one, which becomes our goal; *e.g.*, we scan, by moving the head to the right, a 0-block (of length $k$, formalized by the *blanks* function) via the lemma (proved by induction on $k$):

$$\forall\, k{\in}\mathbb{N}, \forall\, l, r{\in}HTape.\ b_\infty(U, \langle\!\langle\, blanks(k){::}l,\ r\,\rangle\!\rangle, 5) \Rightarrow b_\infty(U, \langle\!\langle\, l,\ blanks(k){::}r\,\rangle\!\rangle, 5)$$

Conversely, it is *not* possible to use the predicate $b_*$ in a compositional way to manage the *convergence* scenario. The problem is that $b_*$ requires to exhibit the final tape, but in this case, due to the complexity of the proof, we cannot master it *tout-court* as we have done in the first example. Therefore, we need an extra tool to accomplish the convergence. Actually, such a tool is provided by the *small-step* predicate $\overset{*}{\rightarrow}$: by applying the Proposition 1.2, we may decompose a convergent computation and address separately the intermediate steps. In the end, we carry out the proof from (4), under the hypothesis $m\geq n$, by nested induction on $n$ and $m$, and by means of lemmas similar to those used for $b_\infty$. $\square$

*Third example: R and L moves, infinitely.* In this example we consider the unary function $f_\emptyset$, undefined on every input, for which we devise an implementation that points out a problem that involves the mechanization of coinduction.

In fact, our algorithm to compute $f_\emptyset$ is very simple: first scan the 1-block towards the right and replace the first blank with a "1"; then move the head towards the left till reaching the first blank and replace it again with a "1"; proceed infinitely in the same way. The specification we pick out is minimal:

$$V \triangleq \{\langle 1,1,R,1\rangle, \langle 1,B,W(1),2\rangle, \langle 2,1,L,2\rangle, \langle 2,B,W(1),1\rangle\}$$

The idea beneath the formal divergence proof is nesting a couple of inductions inside the main coinduction; that is, by using the notation introduced in the previous example to display the modification of the tape, we want to perform the two computations (passing to state 2 and then coming back to state 1):

$$- \mid B \mid \underbrace{\overset{\Downarrow}{1} \mid - \mid 1}_{\text{n+1}} \mid B \mid - \overset{*}{\Longrightarrow} - \mid B \mid \underbrace{1 \mid - \mid \overset{\Downarrow}{1}}_{\text{n+2}} \mid B \mid - \overset{*}{\Longrightarrow} - \mid B \mid \underbrace{\overset{\Downarrow}{1} \mid - \mid 1}_{\text{n+3}} \mid B \mid -$$

It is apparent that, to accommodate this proof, we may assume the coinductive hypothesis for the initial configuration (state 1 and leftmost tape above) and then carry out two finite computations, thus reducing to a configuration (state 1 and rightmost tape) which is an instance of the coinductive hypothesis itself.

Nevertheless, the application of the coinductive hypothesis is *not* allowed by `Coq`, because the whole proof (*i.e.*, the proof term built interactively through tactics, and mainly via `cofix`) is recognized as *non-guarded* by constructors. Essentially, this is caused by the fact that the syntactic check does not accept an induction (*i.e.*, a lemma) nested inside the coinductive development[15].

To circumvent the problem, we introduce here a new small-step divergence predicate. The idea is very direct: divergence may be characterized as the *coinductive* transitive closure of the *inductive* non-reflexive transitive closure of $\to$.

**Definition 3.** *(Extra reduction) Assume $T \in Spec$, $s,t,u \in Tape$, $p,q,i \in State$. Then,* finite positive *reduction $\overset{+}{\to}$ is defined by induction, via the rules:*

$$\frac{(T,s,p) \to (T,t,q)}{(T,s,p) \overset{+}{\to} (T,t,q)}\ (\overset{+}{\to}_1) \qquad \frac{(T,s,p) \to (T,u,i) \quad (T,u,i) \overset{+}{\to} (T,t,q)}{(T,s,p) \overset{+}{\to} (T,t,q)}\ (\overset{+}{\to}_+)$$

*And* infinite split reduction $\overset{\infty}{\Rightarrow}$ *is defined by the following coinductive rule:*

$$\frac{(T,s,p) \overset{+}{\to} (T,t,q) \quad (T,t,q) \overset{\infty}{\Rightarrow}}{(T,s,p) \overset{\infty}{\Rightarrow}}\ (\overset{\infty}{\Rightarrow}_\infty)$$

**Proposition 3.** *(Equivalence, bis) Let be $T \in Spec$, $s \in Tape$, and $p \in State$.*

1. *If $(T,s,p) \overset{+}{\to} (T,u,i)$ and $(T,u,i) \overset{+}{\to} (T,t,q)$, then $(T,s,p) \overset{+}{\to} (T,t,q)$*
2. *If $(T,s,p) \overset{\infty}{\Rightarrow}$, then $(T,s,p) \overset{\infty}{\to}$*
3. *$b_\infty(T,s,p)$ if and only if $(T,s,p) \overset{\infty}{\Rightarrow}$*

*Proof. 1) By structural induction on the derivation of $(T,s,p) \overset{+}{\to} (T,u,i)$. 2) By coinduction and hypothesis inversion. 3) ($\Rightarrow$) By coinduction and hypothesis inversion. ($\Leftarrow$) By Proposition 1.4 and point 2.* □

Since the reduction predicate $\overset{\infty}{\Rightarrow}$ turns out to be equivalent to $b_\infty$, we adopt the former to carry out our divergence proof. Actually, $\overset{\infty}{\Rightarrow}$ does not suffer from the non-guardedness problem, as it is apparent from the following proof tree[16]:

$$\dfrac{n \in \mathbb{N} \qquad \dfrac{(V,s,1) \overset{+}{\to} (V,t,1) \qquad \begin{array}{c}[\forall\, n \in \mathbb{N}.\ (V,s,1) \overset{\infty}{\Rightarrow}]_{(1)} \\ \vdots \\ (V,t,1) \overset{\infty}{\Rightarrow}\end{array}}{(V,s,1) \overset{\infty}{\Rightarrow}}\ (\overset{\infty}{\Rightarrow}_\infty)}{\forall\, n \in \mathbb{N}.\ (V,s,1) \overset{\infty}{\Rightarrow}}\ (1), (introduction)$$

The proof of the premise $(V,s,1) \overset{+}{\to} (V,t,1)$ relies on the transitivity of $\overset{+}{\to}$ (Proposition 3.1) and on two auxiliary lemmas, argued by induction on $n$. □

[15] See [8] for a recent proposal of an alternative, *semantic* guardedness checking.

[16] We write $s$ for $\langle\!\langle\, Bs,\ ones(n{+}1){::}Bs \,\rangle\!\rangle$ and $t$ for $\langle\!\langle\, Bs,\ ones(n{+}3){::}Bs \,\rangle\!\rangle$.

## 8 Conclusion

In the present contribution we have formalized TMs and their (big-step and small-step) operational semantics in the `Coq` proof assistant. Our key choices are the encoding of tapes as pairs of *streams* (managed by means of corecursion) and a clear distinction between *converging* computations (modeled via inductive predicates) and *diverging* ones (formalized through coinductive predicates). In the previous, core section we have pointed out the potential of our machinery, by proving the correctness of representative TMs (that is, by certifying the implementation of the partial functions computed by them).

Our encoding provides a completely mechanized management of the transitions (via the `auto` tactic), with the benefit that we may concentrate on the formal treatment of the tape and the logic of proofs. *Divergence* can be proved very often in a compositional way, via the sole big-step coinductive predicate. When "non-guardedness" complications arise (essentially because induction is nested inside coinduction), alternative, equivalent small-step coinductive predicates may be employed, by taking advantage of their close relationship with the main big-step predicate. On the other hand, it is not always possible to master *convergence* proofs by compositionality. When this is not feasible (due to the difficulty of the proof at hand), the small-step semantics predicates may be used again as an auxiliary tool, to perform intermediate computation steps.

We note also that, in order to carry out either divergence or convergence proofs, often the user has the responsibility to figure out how to decompose the main goal. As usual, it is sometimes necessary to generalize the statements to obtain sufficiently powerful (co)inductive hypotheses. Moreover, some proofs require a subtle combination of inductive and coinductive reasoning.

*Related work.* The contributions of the literature most related to the present one are those by Asperti and Ricciotti in `Matita` [1], Xu, Zhang and Urban in `Isabelle/HOL` [13], and Leroy in `Coq` [9]. Both the first two works address TMs, achieving the ambitious goals we have reported in Section 1.

Asperti and Ricciotti formalize the tape as a triple, made of two lists plus the square currently scrutinized. The non-termination is managed by requiring that the total computation function returns an optional value, when it meets an upper bound of iterations without reaching a final state. The semantics is defined through a relation between tapes, (weakly) "realized" by TMs.

Xu, Zhang and Urban represent the tape via a pair of lists. They handle the non-termination in a similar way, *i.e.*, via the condition that there is no transition into a halting state. The semantics is defined by means of Hoare-rules.

None of the above two works makes use of coinductive tools (that we have exploited to deal with stream-tapes and divergence); from this perspective, our paper is more related to that of Leroy [9], who adopts coinduction in `Coq` to capture infinite evaluations and reductions of a call-by-value $\lambda$-calculus.

*Future work.* We believe that the main result achieved by our work (*i.e.*, the development of a technology for proving the correctness of concrete TMs, via several versions of big-step and small-step semantics) is a promising tool to pursue more advanced goals which are outside the scope of the present paper.

In particular, our effort may be seen as a first step towards the development of computability theory, as the construction of "brick" TMs and their composition at higher-levels of abstraction is the natural progress of this contribution.

It would be also stimulating to relate the present formalization to that of unlimited register machines, that we have addressed in a previous work [2].

*Acknowledgments.* The author is very grateful to the anonymous referees for their helpful, constructive reviews.

## References

1. A. Asperti and W. Ricciotti. Formalizing Turing Machines. In C.-H. L. Ong and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 7456 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 2012.
2. A. Ciaffaglione. A coinductive semantics of the Unlimited Register Machine. In F. Yu and C. Wang, editors, *INFINITY*, volume 73 of *Electronic Proceedings in Theoretical Computer Science*, pages 49–63, 2011.
3. A. Ciaffaglione. *The Web Appendix of this paper.* Università di Udine, Italia, 2014. Available at `http://users.dimi.uniud.it/~alberto.ciaffaglione/Turing/`.
4. T. Coquand. Infinite objects in Type Theory. In H. Barendregt and T. Nipkow, editors, *TYPES*, volume 806 of *Lecture Notes in Computer Science*, pages 62–78. Springer, 1993.
5. N. J. Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.
6. E. Giménez. Codifying guarded definitions with recursive schemes. In P. Dybjer, B. Nordström, and J. M. Smith, editors, *TYPES*, volume 996 of *Lecture Notes in Computer Science*, pages 39–59. Springer, 1994.
7. J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, 2003.
8. C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In R. Giacobazzi and R. Cousot, editors, *POPL*, pages 193–206. ACM, 2013.
9. X. Leroy. Coinductive big-step operational semantics. In P. Sestoft, editor, *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2006.
10. M. Norrish. Mechanised Computability Theory. In M. C. J. D. van Eekelen, H. Geuvers, J. Schmaltz, and F. Wiedijk, editors, *ITP*, volume 6898 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2011.
11. The Coq Development Team. *The Coq Proof Assistant, version 8.4.* INRIA, 2012. Available at `http://coq.inria.fr`.
12. A. M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. Lond. Math. Soc.*, 42, 1936.
13. J. Xu, X. Zhang, and C. Urban. Mechanising Turing Machines and Computability Theory in Isabelle/HOL. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 2013.