Revision Sequences and Computers with an Infinite Amount of Time

BENEDIKT LÖWE, Rheinische Friedrich–Wilhelms–Universität Bonn, Mathematisches Institut, Beringstraße 6, D — 53115 Bonn, Germany.
E-mail: loewe@math.uni-bonn.de

Abstract

The author establishes a connection between Revision Theory of Truth and Infinite Time Turing Machines as developed by Hamkins and Kidder.

The ideas from this paper have incited Welch to solve the limit rule problem of revision theory.

Keywords: Revision theory of truth, infinite time Turing machines, definability.

1 Introduction

First-order definitions and even inductive definitions using the second-order induction scheme are neither sufficient for scientific discourse in general nor for the technical applications using computers to check properties which seem to be simple from a heuristic standpoint.¹ Many natural-language notions use some degree of circularity or impredicativity in their definitions. One of these notions is the notion of truth, if you consider TRUE to be a predicate of sentences (i.e., natural numbers via Gödelization): If your language contains a truth predicate you will necessarily run into sentences asserting truth of sentences, so to determine their truth values you have to already know the extension of the truth predicate.

Consequently, building on the theory of inductive definitions and the work of Martin, Woodruff and Kripke,² Herzberger developed a theory called ‘Naive Semantics’ that went beyond inductive definitions.³ This theory was further refined by Gupta and Belnap in [7], [3], and [8], where they call the system ‘Revision Theory of Truth’ and develop their semantic systems $S^*$ and $S^#$ that allow an analysis of definitions that would be logically illicit in classical definability theory. In fact, the aim of this enterprise is even more aspiring:

‘The key to the proper resolution of the problem of truth and paradox lies, in our view, in the theory of definitions. [8, p. 113].’

Their focus on the question of truth leads to a certain disrespect towards the definability aspect of their work, although the definability aspect is a natural feature of the theory (being a generalization of inductive definability with its extensive literature on definability and complexity questions). In this paper we shall stress the question ‘Which sets of natural numbers can be defined using a Gupta–Belnap definition?’ (as opposed to the question ‘Which sentences are true in a Gupta–Belnap semantic system?’).

¹See, e.g. the Ehrenfeucht–Fraïssé proof that the notion of connectedness for finite graphs is not first-order definable, cf. [6, §1].
²[16] and [15].
³[11] and [12], especially [11, p. 485 and 489] where the reader can find pictures of what was to become the Revision Sequences of [8] and Definition 2.3.
Roughly speaking (we shall make this more precise in subsection 2.2) a property $P$ of natural numbers is said to be \textit{revision theoretically definable} if there is a formula $\Phi$ (possibly containing a symbol for $P$) such that a natural number $n$ has the property $P$ if and only if $n$ is in every reasonable approximation for the extension of $\Phi$, where 'reasonable' means that the approximation $h$ reoccurs in a transfinite sequence of iterated applications of $\Phi$ starting from $h$.

We will show that the Gupta–Belnap revision sequences are deeply connected with the Infinite Time Turing Machines defined by Hamkins and Kidder.\footnote{Cf. [9].} This fact is intuitively plausible: with Turing machines consisting of a finite program (corresponding to the formula $\Phi$ in the description of revision theoretic definability) being applied iteratively, their generalization to transfinite computation sequences of ordinal length looks just like the 'transfinite sequence of iterated applications of $\Phi$' in Revision Theory. We will add some substance to this intuition in Section 4.

This connection and the fact that there is a nice structure theory of Infinite Time Turing Computability give rise to possible applications:

The profound analysis of the lengths of Infinite Time Turing Computation gives hopes that the lengths of (the relevant part of) revision sequences can also be understood.

Even more so, Welch’s theorem that Infinite Time Turing machine architecture is somewhat robust with respect to changing the limit rule (Theorem 3.7) seems to point into the direction of the limit rule problem of Revision Theory: there has been a discussion\footnote{Cf. [4] and [22].} whether changing the limit rule for revision sequences might change the semantic content of the definability notions in a relevant manner.

When sections 1 to 5 of this paper were prepared (Spring 1999), the author already suspected that a refinement of the methods of Section 4 might lead to a solution of the limit rule problem. When the paper was presented as a talk at the workshop “Nicht-klassische Formen der Logik” at the XVIII Deutscher Kongreß für Philosophie in Konstanz (October 1999), the problem had been solved by Philip Welch. The newest results are briefly mentioned in Section 6.

For the paper to be understandable to readers from both Revision Theory and Computability Theory we try to give a rather long introduction to both areas omitting almost all proofs.

The main part of the paper is the construction of the Turing machine $M_{\Phi,n}$ in Section 4 in which we tried to suppress most of the tedious details of storing and copying information.

\section{The Gupta–Belnap systems}

\subsection{Revision sequences}

The Revision Theory of Truth as laid out in [8] is very general as it allows definitions of the extensions of arbitrary predicates over arbitrary models (of arbitrary languages), and uses arbitrary many-valued logics. To simplify notation (and to stress the similarity to Turing machines which are normally used to compute subsets of the natural numbers), we shall restrict ourselves to the most basic case throughout this paper; the ground model will be $\mathbb{N}$, the set of natural numbers, our predicate $\dot{x}$ will be unary, and the logic will be classical two-valued logic. Thus the extension we want to define will be a subset of $\mathbb{N}$. As usual in set
theory, we call these objects ‘real numbers’, although they are not exactly real numbers in the sense of real analysis (but rather elements of the Cantor space).

We fix a base language $\mathcal{L}$, and let $\mathcal{L}^*$ be the language $\mathcal{L}$ augmented by the additional unary predicate symbol $\dot{x}$ whose extension we want to define.

Gupta–Belnap definitions can be seen as a generalization of inductive definitions; in an inductive definition we construct a monotone operator $M : \mathbb{R} \to \mathbb{R}$ derived from a formula in the augmented language $\mathcal{L}^*$ and define the extension of $\dot{x}$ to be the least fixed point of $M$ (starting with the empty extension) which must exist since $M$ was monotone.

In the Gupta–Belnap approach we discard the assumption of monotonicity and have to replace it by a closer analysis of the starting extensions.

As in the inductive case, let $\Phi(v_0)$ be a formula of $\mathcal{L}^*$. Then we shall say that a function $\delta_\Phi : \mathbb{R} \to \mathbb{R}$ is the revision operation determined by $\Phi$ if

$$k \in \delta_\Phi(z) \iff \langle \mathbb{N}, z \rangle \models \Phi[k]$$

for all real numbers $z$. (Again, note that elements of $\mathbb{R}$ are identified with subsets of $\mathbb{N}$.)

For revision operations $\delta : \mathbb{R} \to \mathbb{R}$ we use the obvious notation for the iterated application of $\delta$. In the following we will consider sequences of real numbers $\bar{s} = \langle s_\alpha ; \alpha \in \eta \rangle$ where $\eta$ is either a limit ordinal or the class of all ordinals. These sequences are interpreted as approximations to our final interpretation of the predicate $\dot{x}$.

**Definition 2.1**
Let $\bar{s}$ be a sequence of real numbers of length $\eta$ ($\eta$ might be the class of all ordinals), and let $d$ be a natural number. We shall say that ‘$d \in \dot{x}$’ is $\bar{s}$-stably true if there is a $\beta$ such that for all $\alpha \geq \beta$ we have $d \in s_\alpha$.

Likewise, we shall say that ‘$d \in \dot{x}$’ is $\bar{s}$-stably false if there is a $\beta$ such that for all $\alpha \geq \beta$ we have $d \notin s_\alpha$.

**Definition 2.2**
Let $\bar{s}$ be a sequence of reals. Then a real $h$ is said to be $\bar{s}$-coherent (in symbols: $\text{Coh}(h, \bar{s})$) if for all natural numbers $d$ the following hold:

1. If ‘$d \in \dot{x}$’ is $\bar{s}$-stably true, then $d \in h$.
2. If ‘$d \in \dot{x}$’ is $\bar{s}$-stably false, then $d \notin h$.

**Definition 2.3**
Let $\bar{s}$ be a sequence of reals and $\Phi$ a formula of $\mathcal{L}^*$. We shall call $\bar{s}$ a $\Phi$-revision sequence if

1. For all ordinals $\alpha$, we have $s_{\alpha+1} = \delta_\Phi(s_\alpha)$.
2. For every limit ordinal $\lambda$, we have that $s_\lambda$ is $\bar{s}|\lambda$-coherent.

In light of the question of the limit rule (discussed later), a revision sequence of the defined kind might be called an unrestricted revision sequence. To prepare for the limit rule problem, we introduce restricted revision sequences.

Following [4, p. 400], we shall call a function $\gamma$ assigning to a sequence of reals $\bar{s}$ of limit length $\lambda$ a real $\gamma(\bar{s})$ that is $\bar{s}$-coherent a bootstrapping policy. If $\Gamma$ is a class of bootstrapping policies, then we shall say that $\bar{s}$ is a $(\Phi, \Gamma)$-revision sequence if it’s a $\Phi$-revision sequence and there is a $\gamma \in \Gamma$ such that for each limit ordinal $\lambda$ we have $s_\lambda = \gamma(\bar{s}|\lambda)$.

Cf. [17].

Of course, revision sequences of successor length can be easily extended to revision sequences of limit length, so these are the only ones we have to think about.
The most interesting cases are the class \( \Gamma_\infty \) of all bootstrapping policies (being a \( \Phi \)-revision sequence and being a \( \langle \Phi, \Gamma_\infty \rangle \)-revision sequence are the same), and the one-element classes where we have one particular limit rule applied in all limit stages.

At this point we single out the class \( \Gamma_0 := \{ \gamma_0 \} \) where \( \gamma_0 \) is the liminf rule (i.e., exactly the stably true formulae hold in the limit stage). The limit rule \( \gamma_0 \) is the minimal bootstrapping policy in the sense that all others let more formulae be true in the limit. \( \Gamma_0 \) will play a decisive rôle in the construction of Section 4.8.

**DEFINITION 2.4**

Let \( \Phi \) be a formula of \( \mathcal{L}^* \).

1. We shall call a real \( h \langle \Phi, \Gamma \rangle \)-recurring if \( h \) occurs cofinally often in some \( \langle \Phi, \Gamma \rangle \)-revision sequence \( \bar{s} \) of length \( \text{Ord} \). The set of \( \langle \Phi, \Gamma \rangle \)-recurring reals will be denoted by \( \text{Rec}_{\Gamma}(\Phi) \).
2. A real \( h \) is called \( \langle \Phi, \Gamma \rangle \)-reflexive if there is some ordinal \( \alpha > 0 \) and some \( \langle \Phi, \Gamma \rangle \)-revision sequence \( \bar{s} \) such that \( s_\alpha = s_\alpha = h \). We shall call the least such \( \alpha \) the \( \langle \Phi, \Gamma \rangle \)-reflexivity of \( h \), in symbols \( \text{refl}_{\Phi, \Gamma}(h) \).

In the following we state a couple of simple consequences of the basic definitions. Proofs of these easy facts can be found in [8]:

**PROPOSITION 2.5**

Let \( \bar{s} \) be a sequence of real numbers. Then the following are equivalent:

1. \( 'd \in \bar{x}' \) is \( \bar{s} \)-stably true,
2. for all \( h \) occurring cofinally often in \( \bar{s} \) we have \( d \in h \).

**PROPOSITION 2.6**

For any real \( h \) the following are equivalent:

1. \( h \) is \( \langle \Phi, \Gamma_\infty \rangle \)-recurring,
2. \( h \) is \( \langle \Phi, \Gamma_\infty \rangle \)-reflexive.

The use of the Belnap rule \( \Gamma_\infty \) in Proposition 2.6 is important. The conclusion is provably false for some other rules, including the Herzberger rule \( \Gamma_0 \).

**PROPOSITION 2.7**

If \( h \) is \( \langle \Phi, \Gamma \rangle \)-reflexive then \( \text{refl}_{\Phi, \Gamma}(h) < \omega_1 \).

### 2.2 The systems S^# and S^*

Now we define the semantic relation for the Gupta–Belnap systems \( S^# \) and \( S^* \):

\[
\mathbb{N} \models S_{\Phi, \Gamma}^* \varphi \iff \forall h \in \text{Rec}_{\Gamma}(\Phi) \exists n \geq n(\langle \mathbb{N}, \delta^n_\Phi(h) \rangle) \models \varphi,
\]

and

\[
\mathbb{N} \models S_{\Phi, \Gamma}^* \varphi \iff \forall h \in \text{Rec}_{\Gamma}(\Phi)(\langle \mathbb{N}, h \rangle) \models \varphi.
\]

---

[8] The rule \( \Gamma_\infty \) was the original rule used in [8], and it is called the Belnap rule. Herzberger in [11] used the rule \( \Gamma_0 \) whence we call it the Herzberger rule. In [11, p. 487], Herzberger compares his liminf rule to Kripke’s limsup rule of inductive definitions from [15].

[9] [8, p. 175].
We shall say that a real number \( z \) is \( S^\#_\Gamma \)-definable (\( S^*_\Gamma \)-definable) if there is a formula of \( \mathcal{L}^* \) such that for all natural numbers \( n \) the following holds:

\[
    n \in z \iff \mathbb{N} \models S^\#_{\Phi, \Gamma} \cdot 'n \in \dot{x}'
\]

\[
    \left( n \in z \iff \mathbb{N} \models S^*_\Phi \cdot 'n \in \dot{x}' \right).
\]

In the following, we shall omit \( \Gamma \) in the notation if \( \Gamma = \Gamma^\infty \).

Of course, if \( \varphi \) is a sentence not containing \( \dot{x} \) (i.e., a sentence of \( \mathcal{L} \)), then \( \mathbb{N} \models S^\#_{\Phi, \Gamma} \varphi \) if and only if \( \mathbb{N} \models \varphi \), so \( \{ \varphi ; \mathbb{N} \models S^\#_{\Phi, \Gamma} \varphi \} \) does contain the true sentences of arithmetic. Thus, by Gödel’s Incompleteness Theorem it can’t be first-order definable.

Kremer and Antonelli in [14], [1] and [2] progressed further along this line and showed that \( \{ \varphi ; \mathbb{N} \models S^\#_{\Phi, \Gamma} \varphi \} \) is \( \Pi_2 \)-complete.\(^{10}\)

From our definability standpoint, the most interesting open question about the complexity of revision theory is: ‘What reals are \( S^\#_\Gamma \)- (\( S^*_\Gamma \)-)definable?‘ More precisely, this innocuous question is a vast array of questions. Not only can we try to compute the set of \( S^\#_\Gamma \)-definable reals for all sorts of \( \Gamma \), but we can ask the question

Under what circumstances (i.e., what conditions do we have to impose on \( \Gamma \)) is \( S^\#_\Gamma \)-definability equivalent to \( S^\#_\Gamma \)-definability?

This question keeps us very close to the limit rule question: there have been various proposals for the most natural choice of \( \Gamma \), and it is still under discussion for which \( \Gamma \) the system \( S^\#_\Gamma \) is most natural.\(^{11}\)

If we fix \( \Gamma := \Gamma^\infty \), we can (just by counting quantifiers) make the following immediate observation:

**Proposition 2.8**

The sets \( \{ \varphi ; \mathbb{N} \models S^\#_{\Phi, \Gamma^\infty} \varphi \} \) and \( \{ \varphi ; \mathbb{N} \models S^*_\Phi \Gamma^\infty \varphi \} \) are \( \Pi_2 \).

**Proof.** Using Proposition 2.7 and Proposition 2.6 we can transform the definition equivalently to the form

\[
    \forall h \left( \exists \alpha < \omega_1 (\exists \dot{s} \in (\mathbb{R})^{\alpha+1} ( \forall \beta < \alpha (s_{\beta+1} = \delta_{\Phi}(s_{\beta})) \right.
    \wedge \forall \lambda < \alpha (\text{Lim}(\lambda) \rightarrow \text{Coh}(s_\lambda, s_{\dot{I}}(\lambda)))
    \wedge s_0 = s_\alpha = h))
    \left. \rightarrow (\exists n \forall p \geq n (\langle \mathbb{N}, \delta_{\Phi}^p(h) \rangle \models \varphi) \right)
\]

for \( \# \) and

\[
    \forall h \left( \exists \alpha < \omega_1 (\exists \dot{s} \in (\mathbb{R})^{\alpha+1} ( \forall \beta < \alpha (s_{\beta+1} = \delta_{\Phi}(s_{\beta})) \right.
    \wedge \forall \lambda < \alpha (\text{Lim}(\lambda) \rightarrow \text{Coh}(s_\lambda, s_{\dot{I}}(\lambda)))
    \wedge s_0 = s_\alpha = h))
    \left. \rightarrow (\exists \alpha \models \varphi) \right)
\]

\(^{10}\) For a weaker result with proof, cf. Proposition 2.8. The Kremer–Antonelli result is not only stronger but also much more general since they do not restrict their attention to the standard model \( \mathbb{N} \). Furthermore, Kremer’s completeness result extends to a broader class of revision theories he calls plausible revision theories, cf. [14, p.590sqq]. \( \mathbb{N} \models S^\#_{\Phi, \Gamma} \) is a plausible revision theory for arbitrary \( \Gamma \), hence these relations constitute \( \Pi_2 \)-complete sets.

\(^{11}\) Cf. [4].
for $S^*$. To compute the complexity of these formulae, first note that the function $\delta_A$ doesn’t add complexity since it is first-order definable (via $\Phi$).

Now both formulae are of the form

$$\forall h \left( (\exists \alpha < \omega_1 \exists \bar{\delta} \bar{\Psi}(\alpha, \bar{\delta}, h)) \rightarrow \Psi^*(h) \right),$$

where $\Psi$ is arithmetic in its arguments (already assuming that ordinal quantifiers bounded by $\alpha$ will be natural number quantifiers after coding $(\alpha, \bar{\epsilon})$ as a real). Note that the satisfaction relation in $\langle \mathbb{N}, h \rangle$ is $\Delta_1^1(h)$, so the $\Psi^*$-part doesn’t add complexity either.

Now we code the countable ordinal by a real and receive

$$\forall h \left( (\exists \bar{\gamma}(\text{WO}(\gamma) \land \exists \bar{\delta} \bar{\Psi}(\alpha, \bar{\delta}, h))) \rightarrow \Psi^*(h) \right).$$

Thus the premiss of the implication is $\exists (\Pi_1 \land \Sigma_1^1)$ which is $\Sigma_1^1$, and hence the whole formula is $\Pi_1^1$.

This computation gives an upper bound for the complexity of any given $S^#$-definable real:

**Corollary 2.9**

Every $S^\#_1$-definable real (and every $S^\#_\omega$-definable real) is a $\Pi_1^1$ real.

**Proof.** By definition, a real $x$ is $S^\#_\omega$-definable if and only if there is a formula $\Phi$ of $L^+$ such that for all $n$

$$n \in x \iff \mathbb{N} \models \Phi_{S^\#_\omega} \text{ '}n \in x\text{'.}$$

But by Proposition 2.8, the right-hand side is $\Pi_1^1$.

What is to be said about lower bounds? Using an idea of Gupta, Kremer has proved that every inductively definable real is $S^#$-definable. Since inductive definability over $\mathbb{N}$ and being $\Pi_1^1$ coincide, we have a lower bound for the complexity of $S^#$-definability. It was unknown until very recently whether there were more complicated $S^#$-definable sets (see Section 6).

### 3 Infinite time Turing machines

The notion of an Infinite Time Turing Machine is a natural generalization of the ordinary Turing machines that constitute our basic model of computation. Jeffrey Kidder and Joel Hamkins blended the fields of Recursion Theory (Computability Theory) and Set Theory by allowing their Turing machines to resume their computations after infinitely many steps and start at limit steps in a specified limit stage.

Although probably technically of no importance, the investigation of Infinite Time Turing Machines is motivated by certain philosophical thought-experiments using the Lorentz...
contraction of spacetime to fit a countable sequence of time intervals into a finite time interval (according to the timeline of a different observer), or, more in the language of general relativity, to fit the entire future time cone of one observer into the past time cone of a second observer.

Readers interested in this basic background are referred to the introduction of [9], and the papers [5] and [13].

In the following, we shall assume that the reader is familiar with the basic theory of ordinary Turing machines.\textsuperscript{17}

An Infinite Time Turing Machine works like an ordinary Turing machine—it has a finite program including a specified finite set of states, one of these states is a \texttt{HALT} state telling the program to stop, and in addition to that, it has an infinite tape for output.

The content of the infinite tape of a Turing machine can be seen as a real number. We shall identify the tape with this real number at numerous points in the construction of Section 4. If we have an Infinite Turing Machine Computation of ordinal length \(\alpha\) with a tape labeled \(x\), and \(\beta < \alpha\) then we shall denote the snapshot real at time \(\beta\) with \(x^\beta\). We can (and shall) go even further: If we have an Infinite Time Turing Machine with a tape that is split into countably many infinite components \(\langle x_i; i \in I \rangle\), then we shall denote with \(x^\beta_{i}\) the real that constitutes the snapshot of tape \(x_i\) at the time \(\beta\).

There is no difference between Infinite Time Turing machines and ordinary Turing machines in the part of the Infinite Time Turing machines described up to now. The only difference from ordinary Turing machines is the Infinite Time Turing machine’s \texttt{LIMIT} state, and the fact that if the computation doesn’t reach the \texttt{HALT} state at any given \(\beta < \lambda\) for a limit ordinal \(\lambda\), then it will go into the \texttt{LIMIT} state and set all cells according to the limsup rule: every cell gets the \textit{limes superior} of the cell values at the stages below \(\lambda\); in other words: if a cell contains a \(1\) cofinally often, it has \(1\) in the limit.

As in the standard case we can define Infinite Time Turing Computability as follows:

\textbf{Definition 3.1}

A real \(x\) is \textbf{Infinite Time Turing Computable} if there is an Infinite Time Turing Machine that produces, starting from the empty input, the real \(x\) on the tape at the time it reaches the \texttt{HALT} state.

This definition immediately leads to Relative Infinite Time Turing Computability (denoted by \(\leq^\infty_T\)) and a degree structure as for the ordinary Turing degrees.

At present, the structure theory of Infinite Time Turing degrees is only of marginal importance to the results of Section 4. But since the main aim of this paper is to open up a possible area of applications of the field of Infinite Time Computability theory to Revision Theory, it seems reasonable to give the reader a faint idea about what is known in the former area. Thus we briefly describe what is known without any proofs.

Hamkins and Lewis\textsuperscript{18} and (following a visit of Joel Hamkins to Kobe in the year 1998) Philip Welch provided us with an abundance of structure theoretic results about Infinite Time Turing Computability and the derived degree structure.

It is easily seen that arithmetic truth is Infinite Time Turing Computable, since you can just

\textsuperscript{17}As is to be found in any good textbook of Recursion Theory; e.g., [18].

\textsuperscript{18}[9].
use $\omega$ steps to check every possible witness for an existential quantifier.\(^{19}\) The class of Infinite Time Turing computable reals forms a subclass of the $\Delta^1_2$-reals as can be seen in Figure 1.

The analysis of the Infinite Time Halting problem is an important component of the structure theory of Infinite Time Turing Machines: The Infinite Time Halting Problem (i.e., the set of machines halting given the empty input) is not Infinite Time Turing Computable. The halting problem and its boldface version (the set of reals coding a machine $M$ and a real $x$ such that $M$ halts given the input $x$) yield Infinite Time Turing jump operations $x \mapsto x^\vee$ and $x \mapsto x^\triangledown$ of which the following can be shown \cite[Theorems 5.1 and 5.6]{}:\(^{19}\)

**THEOREM 3.2**

1. $x <^\infty_T x^\vee <^\infty_T x^\triangledown$.
2. $\Delta^1_2$ is closed under $x \mapsto x^\vee$ and $x \mapsto x^\triangledown$.

Hamkins and Lewis\(^{20}\) have looked much closer at the connections between these jump

\(^{19}\)[9, Theorem 2.1 and Theorem 2.6].

\(^{20}\)[Cf. [10].]
operations and the Infinite Time degree structure, finding situations both symmetrical and asymmetrical with respect to classical Turing degree theory.

Infinite Time Turing Machines are connected to a couple of interesting classes of ordinals. For the following, we fix a coding of countable ordinals as real numbers.

**Definition 3.3**

Let $\alpha$ be an ordinal.

1. $\alpha$ is called **writable** if there is a real $x$ coding $\alpha$ such that $x$ is Infinite Time Turing Computable,

2. $\alpha$ is called **clockable** if there is an Infinite Time Turing Machine which reaches its $\text{HALT}$ state after exactly $\alpha$ steps of computation, and

3. $\alpha$ is called **eventually writable** if there is an Infinite Time Turing Machine and an ordinal $\eta$ such that after step $\eta$ the machine just has a fixed code for $\alpha$ on the tape.

We shall write $\Lambda_\infty$ for the supremum of the writable ordinals, $\Upsilon_\infty$ for the supremum of the clockable ordinals, and $Z_\infty$ for the supremum of the eventually writable ordinals. These ordinals reach quite far into the hierarchy of countable ordinals [9, Corollary 8.2 and Corollary 8.6]:

**Theorem 3.4**

$\Lambda_\infty$ and $Z_\infty$ are admissible ordinals.

Philip Welch [20, Theorem 1.1] was able to prove the following connection between the length of Infinite Time Turing Computations and the possible outputs and then to actually compute $Z_\infty$.

**Theorem 3.5**

1. $\Lambda_\infty = \Upsilon_\infty$, and

2. $Z_\infty = \min \{ \xi : \text{L}_\xi \text{ has a transitive } \Sigma_2 \text{-end extension} \}$.

The computation of Theorem 3.5 gives a very interesting result about the actual power of the Infinite Time Turing Machine architecture [19, Theorem 2.6]. At limit stages the original Hamkins–Kidder machine used the limsup rule: if a cell contains the value 1 cofinally often, the limit value is also 1. If we look at machine architectures that use different rules, then the computational power does not change (provided the different rule is simple enough). To make this more precise, let us fix the notation: let $z_{\beta, y}^p$ be the real containing the snapshot of the computation of the Infinite Time Turing Machine with the code $p$ on input $y$ at time $\beta$.

**Definition 3.6**

Let $\Gamma : (2^\omega)^{\text{Ord}} \rightarrow 2^\omega$ be a function. Then $\Gamma$ is called a **suitable $\Sigma_2$ operator** if for every machine using $\Gamma$ in the limit stages there is a $\Sigma_2$ formula $\Psi(v_0, v_1, v_2)$ such that

$$\text{L}_y [y] = z_{\beta, y}^p(i) = 0 \leftrightarrow \Psi[i, p, y].$$

**Theorem 3.7**

Let $\gamma$ be a suitable $\Sigma_2$ operator. Then a real is computable by a Hamkins–Kidder machine (an Infinite Time Turing Machine as described above) if it is computable by a machine using the limit rule $\gamma$. 
4 Revision sequences modelled by Infinite Time Turing Machines

By now, it should be clear to the reader that there is some connection between Infinite Time Turing Machines and revision sequences: both concepts move stepwise, taking the information given in the previous step and computing the next one with a finite amount of programming; both concepts are about moving on throughout the ordinals and in both cases we can use the fact that we can run for an infinite amount of time to check arithmetic truth. The following two quotes highlight this similarity:

A central idea in our approach is to view the function $\delta_{D,M}$ that is supplied by a circular definition $D$ as a rule of revision. Its application to a hypothetical extension $X$ results in a set $\delta_{D,M}(X)$ that is a better candidate for the extension of $G$, according to the definition, than $X$. [8, p. 121]

The Infinite Time Turing Machines have something in the nature of a non-monotonic inductive operator: a set of integers may be input, and at various stages there is a set of integers present on the output tape. [19]

In this section, we shall assume that we have a $(\Phi, \Gamma_0)$-revision sequence of writable ordinal length $\alpha$, and construct an Infinite Time Turing Machine that in its course of computation constructs the revision sequence on its tape. Our Infinite Time Turing Machine will not use the limsup rule as the standard Hamkins–Kidder machines did, but it will use the liminf rule. This doesn’t change the computational power of the machine by Theorem 3.7.

For $\eta < \alpha$ we will denote by $\varrho_\eta$ the stage of the computation of our machine where $s_\eta$ appears on the tape for the first time. Our machine will be constructed such that for limit ordinals $\lambda$ we have $\varrho_\lambda = \bigcup_{\beta < \lambda} \varrho_\beta$.

Now we fix a first-order formula $\Phi$ and a writable ordinal $\alpha$. The Infinite Time Turing Machine we shall construct will have a couple of states that are needed for counting and checking arithmetic truth (we will try and suppress all this detail in the description of the Turing Machine architecture), the START, the HALT and the LIMIT state, and three distinguished states COMPUTE, PREd and NOPREd.

We now describe the action of our machine.

Case 1: If the machine is in the state START, it splits its tape into six countable parts $x_0, x_1, x_2, x_3, x_4$ and $x_5$. It copies the previous content of the tape into $x_3$, then it initializes $x_0, x_1, x_4$ and $x_5$ with an infinite sequence of 0s, and $x_2$ with an infinite sequence of 1s.

The machine will fill $x_0$ with a code for the ordinal $\alpha$, $x_2$ will be an array telling the machine what natural numbers it has already checked, $x_3$ will be the storage of initial segments of the new real (i.e., the next real in the revision sequence), and $x_1$ will be the list of ordinals already handled.

The part $x_5$ will be used by the machine to do its computations and other details like storing the natural number and the ordinal $\beta < \alpha$ it is working on at the moment. This is the part of the action of the machine that we will suppress almost completely in our account. The bits $x_5(0)$ and $x_5(1)$ will have a special significance. $x_5(0)$ will be called the SEQUENCE bit, it contains the information whether we are done writing the next element of $\bar{s}$ onto $x_3$. 
$x_0(1)$ will be called the **CHECKING** bit, and it contains the information whether we are finished with checking an arithmetic truth for some fixed natural number.

$x_4$ will be split up into countably many pieces and will contain the whole revision sequence in the end. The partitioning of the tape can be seen in Figure 2.

After the initialization, the machine writes the ordinal $\alpha$ (i.e., a code for it) onto the tape $x_1$. Then it splits up the tape $x_4$ into a countable sequence of tapes $\langle y_\beta; \beta < \alpha \rangle$. The content of tape $x_1$ tells us the order of the countable sequence of tapes. Then it copies the content of $x_3$ to every tape $y_\beta$ and goes to the **COMPUTE** state.

**Case 2**: If the machine is in the **COMPUTE** state, it browses through $x_1$ in the order given by the code $x_0$ to find the (in the order of $x_0$) least $\beta$ such that $x_1(\beta) = 0$.

If this $\beta$ has a predecessor $\zeta$, the machine moves to the **PRED** state.

If this $\beta$ has no predecessor, it moves to the **NOPRED** state.

**Case 3**: If the machine is in the **NOPRED** state, it just sets $x_1(\beta) := 1$ and starts the **COMPUTE** state case all over again.

**Case 4**: If the machine is in the **PRED** state, the machine sets the **CHECKING** bit to 1, the **SEQUENCE** bit to 0, initializes $x_3$ with an infinite sequence of 0s and starts checking $\Phi$ with input $y_\zeta$ at the natural number 0.

**Case 5**: If the machine is in the **LIMIT** state, its head moves to the **SEQUENCE** bit. If this bit contains a 1, the machine writes the content of $x_3$ to all $y_\eta$ such that $x_1(\eta) = 0$, afterwards.

---

21 From this point onwards, we shall confuse $\alpha$ and the ordering of $\mathbb{N}$ isomorphic to $\alpha$ given by $x_0$. To be exact, we should speak of $\langle y_n; n \in \mathbb{N} \rangle$ ordered by $y_n \prec y_m$ if and only if $\pi_{x_0}(n) < \pi_{x_0}(m)$ where $\pi_{x_0}$ is the bijection between $\mathbb{N}$ and $\alpha$ coded by $x_0$. But we don’t.
sets \( x_1(\beta) := 1 \) and moves into the state \texttt{Compute}.

If the \texttt{Sequence} bit contains a 0, the head moves on to the \texttt{Checking} bit. If the \texttt{Checking} bit is 1, the machine continues with its checking job. If the \texttt{Checking} bit is 0, the machine writes 1 into the \texttt{Sequence} bit, runs along \( x_2 \) until it meets the first 1 in \( x_2 \). At that point it overwrites the \texttt{Sequence} bit with 0 and the \texttt{Checking} bit with 1, and starts the checking of \( \Phi \) at the next natural number (with the same input).

As soon as the machine determines whether \( \Phi \) holds at that natural number, the machine writes the outcome of the check into \( x_3 \) and sets the appropriate bit of \( x_2 \) and the \texttt{Checking} bit to 0.

This completes the definition of our Infinite Time Turing Machine. We shall denote it by \( M_{\Phi, \alpha} \). The working schematics of the machine can be seen in the flow diagram, Figure 3.

Note that the construction is somewhat flexible (remember that we are standing on the shoulders of Theorem 3.7). If \( \Gamma = \{ \gamma \} \) is a singleton with a sufficiently simply definable \( \gamma \) and \( \bar{s} \) is a \( \langle \Phi, \Gamma \rangle \)-revision sequence, we could be a little bit more careful in our construction and receive a machine that computes \( \bar{s} \).

**Theorem 4.1**

Let \( \alpha \) be a writable ordinal, \( \Phi \) be any given formula of \( \mathcal{L}^* \), and \( \bar{s} = (s_\beta; \beta < \alpha) \) a sequence of reals. Then the following are equivalent:

1. \( \bar{s} \) is a \( \langle \Phi, \Gamma_\alpha \rangle \)-revision sequence, and
2. \( M_{\alpha, \alpha} \) eventually computes \( \vec{\sigma} \) on the \( x_4 \) tape, given \( s_0 \) as the input.

**Proof.** We describe what the machine does to see that it does exactly what it’s supposed to do.

We feed \( s_0 \) to the machine in the **START** state. The machine creates six copies of its tape, writing \( s_0 \) on \( x_3 \) and erasing all information from the other copies.

The machine then writes the ordinal \( \alpha \) on \( x_0 \) (which is possible since \( \alpha \) was writable). Then it copies the content of \( x_3 \) (which is the original input \( s_0 \)) to every component of \( y_\eta \) of \( x_4 \).

In the **COMPUTE** state, it finds that we have \( x_1(\eta) = 0 \) for all ordinals \( \eta \), thus \( 0 \) is the least such ordinal. But \( 0 \) has no predecessor, so the machine moves to the **NOPRED** state, writes \( 1 \) to \( x_1(0) \) and goes back into the **COMPUTE** state. Now the least \( \eta \) such that \( x_1(\eta) = 0 \) is 1 which is the successor of 0.

The machine takes \( y_0 \) as the input and starts checking whether \( \Phi(y_0, 0) \) is true. If it meets a limit stage during this check, the **CHECKING** bit is 1, so it resumes its checking until it manages to determine whether \( \Phi(y_0, 0) \) holds. If \( \Phi(y_0, 0) \) holds, then the machine sets \( x_3(0) := 1 \), otherwise it writes 0. Afterwards the machine defines \( x_2(0) \) to be 0, in order to say that it has checked the number 0. Now the **CHECKING** bit is set to 0, and the machine can move on: it runs along \( x_2 \), meets the first 1 at the first cell and starts to check \( \Phi(y_0, 1) \) (after reinitializing the **CHECKING** bit to 1).

After the machine has checked all natural numbers with the same input, it finds itself in the **LIMIT** state. At this moment, the **SEQUENCE** bit is still 0, but every cell on \( x_2 \) is filled with a 0. So when the head runs along \( x_2 \) if never encounters a 1 and thus never changes the **SEQUENCE** bit; by the liminf rule, the **SEQUENCE** bit contains a 1 in the next limit step and hence the machine writes the just computed real (the content of \( x_3 \), which is by construction \( \delta_\alpha(s_0) = s_1 \)) into every \( y_\eta \) with \( \eta > 0 \) (note that 0 is still the only ordinal such that \( x_1(0) = 1 \)). Then it sets \( x_1(1) := 1 \) and moves in the **COMPUTE** state. Now 2 is the least ordinal \( \eta \) such that \( x_1(\eta) = 0 \). As the predecessor of 2 is 1, the computation is started all over again with \( y_1 \) as the input.

This describes in fact how that machine computes all successor steps of the revision sequence \( \vec{\sigma} \).

What about the limit stages in the sequence \( \vec{\sigma} \)?

Let \( \lambda \) be a limit ordinal and \( \varnothing := \bigcup_{\eta < \lambda} \theta_\eta \).

Then \( \varnothing \) is a limit, since \( \langle \theta_\eta; \eta < \lambda \rangle \) is a strictly increasing sequence of limit length. But if we look at \( y_\lambda \) in the course of computation up to \( \varnothing \), \( y_\lambda \) contains every element of \( \vec{\sigma}[\lambda] \). So the following holds for all natural numbers \( n \):

\[ \exists \zeta < \varnothing \forall \xi \geq \zeta \ (g_\xi(\zeta, n) = 1) \iff \exists \zeta < \lambda \forall \xi \geq \zeta \ (n \in s_\xi) \]

Thus by the liminf rule and by the fact that \( s_\lambda = \gamma_\varnothing(\vec{\sigma}[\lambda]) \), we have \( g_\lambda(\varnothing, \zeta) = s_\lambda \). In fact, this is true for all \( g_\xi \) with \( \zeta \geq \lambda \).

Now below \( \varnothing \) the machine has run through the **COMPUTE** case cofinally often. Thus, the **SEQUENCE** bit has been reset to 0 cofinally often and by the liminf rule this means that it is 0 at \( \varnothing \).

The **CHECKING** bit is also 0 since the machine has successfully checked a given natural number cofinally often below \( \varnothing \).

With the same argument, \( x_2 \) at the stage \( \varnothing \) is constantly 0 (every natural number has been checked cofinally often), so the machine runs along \( x_2 \) and in the step \( \varnothing + \omega \), the **SEQUENCE** bit is at 1.
Consequently, the machine moves to the COMPUTE state, realizes that \( \lambda \) is the least ordinal such that \( x_1(\lambda) = 0 \). Since \( \lambda \) does not have a predecessor, it writes \( x_1(\lambda) = 1 \) and moves on to compute \( y_{\lambda+1} \).

5 The limit rule and other applications

Using the correspondence between revision sequences and Infinite Time Turing Machines we head towards possible applications (and point out possible complications and obstacles).

We have mentioned the limit problem of Revision Theory a couple of times before:

‘If we let \( \Gamma \) vary, what happens to the notion of \( S_1^{\#} \)-definability?\(^{22}\)

The apparent freedom of choosing the limit rule for Infinite Time Turing Machines should give us some way of tackling the limit problem for Revision Theory. In order to use that, we would have to get a uniform version of Theorem 4.1: suppose that \( h \) is \( S_1^{\#} \)-definable. Then for every natural number \( n \) such that \( n \notin h \), we have a revision sequence \( s_h^n \) witnessing this (i.e., \( n \notin s_h^n \) and \( s_h^n \) recurs in \( s_h^n \)). One of the problems in applying Theorem 4.1 is that we don’t know anything about the lengths of the witnessing sequences \( s_h^n \), and our theorem works only for sequences of writable ordinal length.

This problem naturally leads us into a second possible field of applications:

The fact that we have a good understanding what ordinals are writable and where the boundaries of writability lie, poses several interesting related questions about Revision Theory:

What can we say about the length of relevant parts of revision sequences? (i.e., how large can \( \text{refl}_{\psi, \Gamma}(h) \) become?)

A useful solution of this question would also give us the needed uniform version of Theorem 4.1.

In solving the limit rule problem, there are some additional (somewhat deeper) problems with unrestricted revision sequences (i.e., if the class \( \Gamma \) is large), though, since the mere existence of a revision sequence doesn’t say anything about the complexity of choices from \( \Gamma \) made on the way to \( \text{refl}_{\psi, \Gamma}(s_0) \).

A third area of possible applications concerns the extent of revision theoretic definability. It might be possible to use the stratification of \( \Delta^1_1 \) by the relation \( \leq_T^\varphi \) to get results for the complexity of revision theoretically definable reals.

Concluding, there is a clear connection between Revision Theory and Infinite Time Turing machines, and it seems very promising, but there is still a lot of technical work to be done to get to a point where we can exploit this connection.

\(^{22}\)[4, p.403]: ‘What category a proposition is going to belong to depends on the kinds of bootstrapping policies admitted.’
6  Aftermath

In the months between the preparation of the first version and the submission of the final version of this article, Philip Welch has used the methods of this paper in his [21] to solve the questions mentioned in Section 5.

THEOREM 6.1 (Welch)
Every $\Pi^0_2$ real is $S^#_\Gamma$-definable and $S^*_\Gamma$-definable for a class of $\Gamma$ including $\Gamma_0$ and $\Gamma_\infty$.

Theorem 6.1 together with Corollary 2.9 gives an exact computation of the class of $S^#_{\Gamma,\infty}$-definable reals. They are exactly the $\Pi^0_2$ reals.

THEOREM 6.2 (Welch)
For arithmetic formulae $\Phi$, simple classes $\Gamma$ of bootstrapping policies and any $\langle \Phi, \Gamma \rangle$-reflexive real number $h$, we have $\text{refl}_{\Phi,\Gamma}(h) < \Upsilon_{\Gamma}(h)$, where $\Upsilon_{\Gamma}(h)$ is the supremum of the ordinals clockable relative to $h$.

Acknowledgements

The author is indebted to Kai-Uwe Kühnberger (Bloomington IN & Tübingen) who introduced him to the problem discussed in this paper, to Philip Welch (Kobe, Bristol & Berkeley CA) for discussions and advice, and to the anonymous referee. Part of this work was completed while the author held a grant of the Studienstiftung des deutschen Volkes. The author wishes to extend his special gratitude towards the Association of Symbolic Logic for a travel grant to Logic Colloquium 1998 in Prague.

References


Received June 1999