# The Halting Problem is Soluble in Malament-Hogarth Spacetimes

Mike Stannett
University of Sheffield, UK

September 13, 2023

**Abstract**

We provide an Isabelle verification that the (Turing) Halting Problem can be solved in Malament-Hogarth (MH) spacetimes. Our proof is quite general – rather than assume the full machinery of general relativity, we simply assume the existence of a reachability relation, $p \rightsquigarrow q$, defined on an abstract space of *locations*; this captures the idea that a user (or signal) can travel from one location to another in finite proper time. An MH spacetime can then be described as a space in which there exists an unboundedly long path *mhline*, and a location *mhpoint* which is reachable from all points on *mhline*. Likewise, we use a very general notion of *computation* - the 'current state' of a computation is assumed to be representable as a machine *configuration* containing all the information required to determine how the system changes with the execution of each ensuing instruction. To specify a computation you provide the initial configuration, and the 'operating system' (the action of which is modeled via an assumed function, *getNextConfig*, then computes successor configurations one by one. The program is deemed to halt if the system enters a configuration which is left unchanged under *getNextConfig*. Since this situation is generally detectable by an operating system, we can use its occurrence to trigger events that exploit the nature of MH spacetimes, thereby enabling us to detect whether or not halting will eventually have occurred.

Our verification follows existing arguments in the literature, albeit translated into this more general setting.

## Contents

# 1 MHComputation

In this theory we define five locales

- Computation
- ReachabilitySpace
- Time
- MHSpacetime
- MHComputation

and use them to verify that the Turing halting problem (HP) can be solved if we are allowed to exploit the physical properties of so-called Malament-Hogarth spacetimes.

This verification generalises our earlier proof outline [3], which was itself based on the seminal results of Hogarth [2] and Németi & Etesi [1].

**theory** *MHComputation*
**imports** *Main*
**begin**

## 1.1 locale: Computation

We think of a computing machine as being placed in an initial configuration, which includes all of the required details of the program to be run and its inputs. The machine is equipped with an operating system which "knows" how to execute one instruction at a time, thereby moving the system from one configuration to another as time (measured in discrete "ticks") passes.

We generally refer to the initial configuration using the variable name *spec* (for "specification"). The function *configAtTick* which takes two arguments , the initial configuration and the tick number $n$, and yields the configuration of the corresponding system at completion of the $n$'th tick. This is computed by recursively iterating calls to *getNextConfig*.

We say that the program *halts* if there are two consecutive ticks at which the configurations are identical.

**locale** *Computation* =
**fixes** *halts* :: *'machineConfig* ⇒ *bool*
**and** *getNextConfig* :: *'machineConfig* ⇒ *'machineConfig*
**and** *configAtTick* :: *'machineConfig* ⇒ *nat* ⇒ *'machineConfig*
**assumes**
*halting*: *halts spec* ≡ ∃ *n* . (*configAtTick spec n* = *configAtTick spec* (*n*+*1*))
**and** *configs*: *configAtTick spec n* =

```
                (if n = 0 then spec else getNextConfig (configAtTick
spec (n−1)))
begin

abbreviation haltingTick :: 'machineConfig ⇒ nat option
  where haltingTick spec ≡
        (if (halts spec)
          then Some (Min { n . configAtTick spec n = configAtTick
spec (n+1) })
         else None)

end
```

## 1.2 locale: ReachabilitySpace

Although we think of computations as taking place in a special type of spacetime, this interpretation is far more constraining that required for the proof to work. All we need to know is whether there is a traversible path from one spacetime location to another. We do not specify what we mean by a "location", but we can think of locations as points in a (3+1)-dimensional spacetime, with traversibility indicating the existence of a time-like curve from one location to another.

```
locale ReachabilitySpace =
  fixes   reachable :: 'location ⇒ 'location ⇒ bool (- ⇝ -)
begin
end
```

## 1.3 locale: Time

We'll be modelling time using values in a linearly ordered field. However, such fields can include infinite values. We want to ensure that the user can solve the halting problem in a known finite amount of time, so we need some way of saying that a positive value is finite. The details are unimportant. One way would be to note that each natural number can be embedded naturally in the field, and say that a positive value is finite iff it is bounded above by some natural number.

```
locale Time = linordered-field +
  fixes isFinite  :: 'a ⇒ bool
begin

fun embedTick :: nat ⇒ 'a
  where embedTick 0 = zero
  |     embedTick (Suc n) = plus one (embedTick n)
```

**end**

## 1.4 locale: MHSpacetime

A *Malament-Hogarth spacetime* is a spacetime which contains a point *mhpoint* and a timelike curve *mhline*, where *mhline* has infinite proper length and *mhpoint* is reachable from every point on *mhline*. If we arrange for the computer to traverse *mhline*, this ensures that any program that ought to run forever without halting will have "enough time" to do so.

We represent *mhline* as a path comprising locations parameterised by proper time, where proper times are assumed to form a linearly ordered field (in the algebraic sense). Because linearly ordered fields contain unboundedly large values, this ensures that the proper length of *mhline* is infinite.

Since *mhpoint* is reachable from *mhline*(0), there exists some fixed path *basePath* between the two points, which takes some finite proper time *baseTime* to traverse.

**locale** *MHSpacetime* = *ReachabilitySpace* + *Time* +
**fixes** *mhpoint* :: *'location*
**and** *mhline* :: *'b* ⇒ *'location*
**and** *basePath* :: *'b* ⇒ *'location*
**and** *baseTime* :: *'b*
**assumes**
*mhprop*: (*mhline t*) ⇝ *mhpoint*
**and** *baseprop*: (*basePath zero* = *mhline zero*) ∧ (*basePath baseTime* = *mhpoint*)
∧ (*isFinite baseTime*)
**begin**
**end**

## 1.5 locale: MHComputation

This locale combines *Computation* and *MHSpacetime* by assuming that the computer and user follow special paths while the program executes. We think of the user being co-located with the computer at time 0, when some program of interest begins execution on some specific set of inputs (both the program and the inputs are provided by the user in the initial configuration). Our task is to determine (in *finite* – and program-independent – time as measured by the user) whether or not the execution will eventually halt.

To do this, we send the computer along the path *mhline*, while the user travels instead to *mhpoint* via *basePath*. The machine's operating system is equipped with a signalling device, which is

triggered if (and only if) the program is found to have halted.

To determine whether the program eventually halts, all the user has to do is check when they arrive at *mhpoint* whether or not a signal is also present. If so, the operating system must have been been triggered to send it, which means that the program must have halted at some point. If no signal is present, then no step of the program triggered the operating system to send one, which means the program never halted while the computer traversed *mhline*. Since this trajectory provided the computer with enough time to execute an unlimited number of ticks, this means that the program ran forever.

The runtime, *baseTime*, of this procedure is finite, and is the same for all choices of initial configuration, *spec*.

**locale** *MHComputation* = *Computation* + *MHSpacetime* +
  **fixes** *machinePath* :: $'c \Rightarrow 'b$
**and** *userPath* :: $'c \Rightarrow 'b$
**and** *signalSentFrom* :: $'a \Rightarrow 'b \Rightarrow bool$
**and** *signalPresentAt* :: $'a \Rightarrow 'b \Rightarrow bool$
**and** *runtime* :: $'c$
**assumes**
  *pathOfMachine*: *machinePath* = *mhline*
**and** *pathOfUser*: *userPath* = *basePath*
**and** *decisionTime*: *runtime* = *baseTime*
**and** *signalling*: (*signalSentFrom spec pt*) $\longleftrightarrow$
  ($\exists$ *n* . (*haltingTick spec* = *Some n*)
    $\wedge$ (*pt* = *machinePath* (*embedTick n*)) )
**and** *signalReception*: (*signalPresentAt spec pt* $\longleftrightarrow$
  ($\exists$ *pt'* . *signalSentFrom spec pt'* $\wedge$ (*pt'* $\rightsquigarrow$ *pt*)))

**begin**

### 1.5.1 lemma: hpMHDecidable
### The halting problem is decidable in MH-Spacetime

We show that the user can determine whether or not any specified program will eventually halt by checking for the receipt of a signal after a fixed finite runtime. The same runtime works regardless of which program is being examined.

**abbreviation** *decisionAtTime* :: $'a \Rightarrow 'c \Rightarrow bool$
  **where** *decisionAtTime spec t* $\equiv$ *signalPresentAt spec* (*userPath t*)

**lemma** *hpMHDecidable*: (*isFinite runtime*) $\wedge$
  ($\forall$ *spec* . (*decisionAtTime spec runtime* = *True*) $\longleftrightarrow$ *halts spec*)
**proof** −

**have** *part1*: *isFinite runtime* **using** *baseprop decisionTime* **by** *auto*

**moreover have** *part2*: $\forall$ *spec* . (*decisionAtTime spec runtime* = *True*) $\longleftrightarrow$ *halts spec*
**proof** −
**{ fix** *spec*

**{ assume** *case1*: *decisionAtTime spec runtime* = *True*
**hence** *signalPresentAt spec* (*userPath runtime*) **by** *simp*
**then obtain** *pt′* **where** *pt′*: (*signalSentFrom spec pt′*) ∧ (*pt′* $\rightsquigarrow$ (*userPath runtime*))
**using** *signalReception* **by** *auto*
**then obtain** *n* **where** *n*: *haltingTick spec* = *Some n* **using** *signalling* **by** *auto*
**hence** *halts spec* **by** (*meson option.distinct*(*1*))
**}**
**hence** *decisionAtTime spec runtime* = *True* $\longrightarrow$ *halts spec* **by** *auto*

**moreover have** *halts spec* $\longrightarrow$ *decisionAtTime spec runtime* = *True*
**proof** −
**{ assume** *case2*: *halts spec*
**obtain** *m* **where** *m*: *m* = *Min* { *n* . *configAtTick spec n* = *configAtTick spec* (*n*+*1*) }
**by** *auto*
**define** *pt* **where** *pt*: *pt* = *machinePath* (*embedTick m*)
**hence** (*haltingTick spec* = *Some m*) ∧ (*pt* = *machinePath* (*embedTick m*)) **using** *m case2* **by** *simp*
**hence** *signalSentFrom spec pt* **using** *signalling* **by** *auto*
**moreover have** *pt* $\rightsquigarrow$ *mhpoint* **by** (*metis mhprop pathOfMachine pt*)
**ultimately have** *signalPresentAt spec* (*userPath runtime*)
**using** *baseprop decisionTime pathOfUser signalReception* **by** *auto*
**hence** *decisionAtTime spec runtime* = *True* **by** *simp*
**}**
**thus** *?thesis* **by** *auto*
**qed**

**ultimately have** (*decisionAtTime spec runtime* = *True*) $\longleftrightarrow$ *halts spec*
**by** *blast*
**}**
**thus** *?thesis* **by** *blast*
**qed**

**ultimately show** *?thesis* **by** *blast*
**qed**

**end**
**end**

# References


[1] G. Etesi and I. Németi. Non-turing computations via Malament-Hogarth space-times. *Int. J. Theor. Phys.*, 41:341–370, 2002.

[2] M. Hogarth. Deciding arithmetic using SAD computers. *British Journal for the Philosophy of Science*, 55:681–691, 2004.

[3] M. Stannett. Towards formal verification of computations and hypercomputations in relativistic physics. In J. Durand-Lose and B. Nagy, editors, *Machines, Computations, and Universality 2015, 09-11 Sep 2015, Famagusta, North Cyprus*, number 9288 in Lecture Notes in Computer Science. Springer Verlag, 2015.