

# The Busy Beaver Upper-Bound Principle

Arthur Freitas Ramos  
David Barros Hulak  
Ruy J. G. B. de Queiroz

June 1, 2026

## Abstract

This development formalizes the Busy Beaver upper-bound principle and its computability consequences for the Turing machines from AFP's Universal Turing Machine entry. The abstract layer works over machine models with finite size classes and unique exact halting times: it defines the time Busy Beaver function and proves that any total upper bound for this function decides zero-input halting. The development then connects the abstract theory to AFP's Universal Turing Machine formalization by defining exact numeric-result halting times, a finite size measure for Turing programs, a concrete Turing-machine Busy Beaver function, a code-indexed Busy Beaver function whose upper bounds decide AFP's special halting problem *K1*, and a strengthened program/input-pair Busy Beaver function whose upper bounds decide AFP's two-argument halting problem *H1*. Using AFP's Turing-computability interface, it further proves that no upper-bound-induced *H1* decider set has a Turing-computable total characteristic function, with a specialization to the decider induced by the concrete pair-indexed Busy Beaver function itself.

## Contents

<b>1</b>	<b>The Busy Beaver Upper-Bound Principle</b>	<b>2</b>
1.1	Base model and Busy Beaver time . . . . .	3
1.2	Computability consequences . . . . .	5
<b>2</b>	<b>Busy Beaver Upper Bounds for Universal Turing Machines</b>	<b>6</b>
2.1	Exact halting times for numeric results . . . . .	7
2.2	A finite size measure for Turing programs . . . . .	8
2.3	The code-indexed Busy Beaver function and AFP's special halting problem . . . . .	10
2.4	A code-indexed Busy Beaver function for the two-argument halting problem . . . . .	11

## Main Results

The formalization separates a reusable Busy Beaver upper-bound principle from its concrete Turing-machine applications.

- Any total upper bound for the abstract function *BB-time* decides zero-input halting for the corresponding model.
- The concrete function *Turing-BB-time* bounds exact numeric-result halting times for bounded-size Turing programs.
- In AFP's *hpk* locale, code-indexed upper bounds decide the diagonal halting problem *K1*.
- Pair-indexed upper bounds decide the two-argument halting problem *H1*, and the induced *H1* characteristic functions are not Turing-computable total.

```
theory Busy-Beaver-Base
imports Main
begin
```

## 1 The Busy Beaver Upper-Bound Principle

Radó's Busy Beaver function measures the largest halting time among machines of bounded size [1]. This theory formalizes the Busy Beaver upper-bound principle over a model-parametric notion of programs with a finite size measure and unique exact halting times.

The base locale defines the Busy Beaver time function for zero-input runs and proves that any total upper bound for it decides the corresponding fixed-input halting predicate. A second locale adds a generic computability interface: noncomputability follows from an assumed absence of computable fixed-input halting deciders, and eventual domination is derived under an explicit witness assumption that packages the usual input-to-program compilation argument. A separate theory instantiates the base locale with AFP's Turing machines and, by treating program/input pairs as Busy Beaver objects, obtains a concrete upper-bound decider for AFP's two-argument halting problem together with an explicit Turing-noncomputability consequence for the induced characteristic functions.

## 1.1 Base model and Busy Beaver time

**locale** *busy-beaver-base* =  
**fixes** *size* :: 'prog ⇒ nat  
**and** *halts-in* :: 'prog ⇒ nat ⇒ nat ⇒ bool  
**assumes** *finite-size-le*: finite {*p*. size *p* ≤ *n*}  
**and** *halting-time-unique*:  
*halts-in p x t* ⇒ *halts-in p x u* ⇒ *t* = *u*  
**begin**

**definition** *halts* :: 'prog ⇒ nat ⇒ bool **where**  
*halts p x* ⇔ (∃ *t*. *halts-in p x t*)

**definition** *halting-time* :: 'prog ⇒ nat ⇒ nat **where**  
*halting-time p x* = (THE *t*. *halts-in p x t*)

**definition** *time-set* :: nat ⇒ nat set **where**  
*time-set n* = {*t*. ∃ *p*. size *p* ≤ *n* ∧ *halts-in p 0 t*}

**definition** *BB-time* :: nat ⇒ nat **where**  
*BB-time n* = Max (insert 0 (*time-set n*))

**definition** *upper-bound-for-BB* :: (nat ⇒ nat) ⇒ bool **where**  
*upper-bound-for-BB b* ⇔ (∀ *n*. *BB-time n* ≤ *b n*)

**definition** *halting-decider-from* :: (nat ⇒ nat) ⇒ 'prog ⇒ nat ⇒ bool **where**  
*halting-decider-from b p x* ⇔ (∃ *t* ≤ *b (size p)*. *halts-in p x t*)

**lemma** *halting-time-eq*:  
**assumes** *halts-in p x t*  
**shows** *halting-time p x* = *t*  
**unfolding** *halting-time-def*  
**using** *assms halting-time-unique* **by** *blast*

**lemma** *finite-time-set [simp]*: finite (*time-set n*)

**proof** –

**let** *?A* = {*p*. size *p* ≤ *n* ∧ *halts p 0*}  
**have** *A-finite*: finite *?A*  
**by** (*simp add: finite-size-le*)  
**have** *time-set n* ⊆ (λ*p*. *halting-time p 0*) ‘ *?A*

**proof**

**fix** *t*  
**assume** *t* ∈ *time-set n*  
**then obtain** *p* **where** *p*: size *p* ≤ *n* **and** *t*: *halts-in p 0 t*  
**by** (*auto simp: time-set-def*)  
**have** *halts p 0*  
**using** *t* **by** (*auto simp: halts-def*)  
**moreover have** *halting-time p 0* = *t*  
**using** *halting-time-eq[OF t]*.  
**ultimately show** *t* ∈ (λ*p*. *halting-time p 0*) ‘ *?A*

```

    using p by auto
  qed
  moreover have finite (( $\lambda p$ . halting-time p 0) ‘ ?A)
    using A-finite by simp
  ultimately show ?thesis
    by (rule finite-subset)
  qed

lemma finite-insert-time-set [simp]: finite (insert 0 (time-set n))
  by simp

lemma BB-time-ge-time:
  assumes size p  $\leq$  n
    and halts-in p 0 t
  shows t  $\leq$  BB-time n
proof –
  have fin: finite (insert 0 (time-set n))
    by simp
  have mem: t  $\in$  insert 0 (time-set n)
    using assms by (auto simp: time-set-def)
  show ?thesis
    unfolding BB-time-def using Max-ge[OF fin mem] .
  qed

lemma BB-time-upper-bound:
  assumes  $\bigwedge p t$ . size p  $\leq$  n  $\implies$  halts-in p 0 t  $\implies$  t  $\leq$  B
  shows BB-time n  $\leq$  B
proof –
  have fin: finite (insert 0 (time-set n))
    by simp
  have nonempty: insert 0 (time-set n)  $\neq$  {}
    by simp
  have bound:  $\bigwedge t$ . t  $\in$  insert 0 (time-set n)  $\implies$  t  $\leq$  B
    using assms by (auto simp: time-set-def)
  show ?thesis
    by (simp add: BB-time-def bound)
  qed

lemma halting-decider-from-sound:
  assumes halting-decider-from b p x
  shows halts p x
  using assms by (auto simp: halting-decider-from-def halts-def)

lemma halting-decider-from-complete-0:
  assumes ub: upper-bound-for-BB b
  assumes halts p 0
  shows halting-decider-from b p 0
proof –
  obtain t where t: halts-in p 0 t

```

```

    using assms by (auto simp: halts-def)
  have  $t \leq \text{BB-time}$  (size  $p$ )
    using BB-time-ge-time[of  $p$  size  $p$   $t$ ] t by simp
  also have  $\dots \leq b$  (size  $p$ )
    using ub by (simp add: upper-bound-for-BB-def)
  finally show ?thesis
    using t by (auto simp: halting-decider-from-def)
qed

```

```

theorem halting-decider-from-correct-0:
  assumes upper-bound-for-BB  $b$ 
  shows halting-decider-from  $b$   $p$   $0 \iff \text{halts } p$   $0$ 
  using halting-decider-from-sound halting-decider-from-complete-0[OF assms]
  by blast

```

```

lemma BB-time-is-upper-bound: upper-bound-for-BB BB-time
  by (simp add: upper-bound-for-BB-def)

```

end

## 1.2 Computability consequences

The following locale deliberately keeps computability assumptions explicit. In particular, eventual domination is not derived from finite size classes and exact halting times alone; the assumption *computable-has-busy-witness* states the compilation witness needed to turn values of a computed function into sufficiently long zero-input halting computations.

```

locale busy-beaver-model = busy-beaver-base size halts-in
  for size :: 'prog  $\Rightarrow$  nat
  and halts-in :: 'prog  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool +
  fixes computes :: 'prog  $\Rightarrow$  (nat  $\Rightarrow$  nat)  $\Rightarrow$  bool
  assumes compute-halts:
    computes  $p$   $f \implies \exists t. \text{halts-in } p$   $x$   $t$ 
  and computable-has-busy-witness:
    computes  $p$   $f \implies \exists N. \forall n \geq N. \exists q t. \text{size } q \leq n \wedge \text{halts-in } q$   $0$   $t \wedge f$   $n \leq t$ 
begin

```

```

definition computable :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  bool where
  computable  $f \iff (\exists p. \text{computes } p$   $f)$ 

```

```

definition eventually-dominates :: (nat  $\Rightarrow$  nat)  $\Rightarrow$  (nat  $\Rightarrow$  nat)  $\Rightarrow$  bool where
  eventually-dominates  $g$   $f \iff (\exists N. \forall n \geq N. f$   $n \leq g$   $n)$ 

```

```

lemma no-computable-upper-bound-if-halting-undecidable:
  assumes no-decider:
     $\bigwedge b. \text{computable } b \implies \neg (\forall p. \text{halting-decider-from } b$   $p$   $0 \iff \text{halts } p$   $0)$ 

```

```

shows  $\neg (\exists b. \text{computable } b \wedge \text{upper-bound-for-BB } b)$ 
using halting-decider-from-correct-0 no-decider by blast

theorem BB-time-not-computable-if-halting-undecidable:
assumes no-decider:
   $\bigwedge b. \text{computable } b \implies$ 
   $\neg (\forall p. \text{halting-decider-from } b \ p \ 0 \longleftrightarrow \text{halts } p \ 0)$ 
shows  $\neg \text{computable } \text{BB-time}$ 
using BB-time-is-upper-bound no-computable-upper-bound-if-halting-undecidable
no-decider
by blast

lemma BB-time-dominates-computed-function-from-size:
assumes comp: computes p f
shows eventually-dominates BB-time f
proof –
  obtain N where  $\forall n \geq N. \exists q \ t. \text{size } q \leq n \wedge \text{halts-in } q \ 0 \ t \wedge f \ n \leq t$ 
  using computable-has-busy-witness[OF comp] by blast
  then have  $\forall n \geq N. f \ n \leq \text{BB-time } n$ 
  by (meson BB-time-ge-time order.trans)
  then show ?thesis
  by (auto simp: eventually-dominates-def)
qed

theorem BB-time-eventually-dominates-computable:
assumes computable f
shows eventually-dominates BB-time f
using assms BB-time-dominates-computed-function-from-size
by (auto simp: computable-def)

end

end
theory Turing-Busy-Beaver
imports
  Busy-Beaver-Base
  Universal-Turing-Machine.TuringComputable
begin

```

## 2 Busy Beaver Upper Bounds for Universal Turing Machines

This theory connects the abstract Busy Beaver upper-bound principle to the Turing machines from AFP’s Universal Turing Machine entry [2]. We use halting with a numeric result, the halting predicate that occurs in AFP’s formalization of the special halting problems. The first instantiation is a direct Turing-program Busy Beaver function. Inside AFP’s *hpk* locale we

also define code-indexed variants for the one-argument problem  $K1$  and the two-argument problem  $H1$ . Finally, using AFP's Turing-computability interface, we make the noncomputability consequence explicit for the characteristic functions of the induced  $H1$  decider sets.

Three related Busy Beaver notions appear below. The theory *Busy-Beaver-Base* supplies the model-parametric function *BB-time*, whose objects are abstract programs run on input  $0$ . The first concrete interpretation instantiates this as *Turing-BB-time* for AFP Turing programs, still using a direct program-size measure. The code-indexed interpretations are introduced for the halting problems in the Universal Turing Machine entry: the diagonal code-indexed function decides  $K1$ , while the pair-indexed function folds a machine code and input into a single Busy Beaver object so that upper bounds decide  $H1$ .

## 2.1 Exact halting times for numeric results

**definition** *has-num-result-at* ::  $tprog0 \Rightarrow nat \ list \Rightarrow nat \Rightarrow bool$  **where**  
*has-num-result-at*  $tm \ ns \ t \longleftrightarrow$   
*is-final* ( $steps0 \ (1, [], <ns>) \ tm \ t$ )  $\wedge$   
 $(\exists k \ n \ l. \ steps0 \ (1, [], <ns>) \ tm \ t = (0, Bk \uparrow \ k, <n::nat> \ @ \ Bk \uparrow \ l))$

**definition** *tm-num-halts-in* ::  $tprog0 \Rightarrow nat \Rightarrow nat \Rightarrow bool$  **where**  
*tm-num-halts-in*  $tm \ input \ t \longleftrightarrow$   
*has-num-result-at*  $tm \ [input] \ t \wedge (\forall u < t. \neg \text{has-num-result-at } tm \ [input] \ u)$

**lemma** *tm-num-halts-in-unique*:  
**assumes** *tm-num-halts-in*  $tm \ input \ t$   
**and** *tm-num-halts-in*  $tm \ input \ u$   
**shows**  $t = u$   
**by** (*meson* *assms* *nat-neq-iff* *tm-num-halts-in-def*)

**lemma** *TMC-has-num-res-iff-tm-num-halts*:  
*TMC-has-num-res*  $tm \ [input] \longleftrightarrow (\exists t. \text{tm-num-halts-in } tm \ input \ t)$

**proof**

**assume** *TMC-has-num-res*  $tm \ [input]$   
**then obtain**  $t \ k \ n \ l$  **where**  
*final*: *is-final* ( $steps0 \ (1, [], <[input]>) \ tm \ t$ ) **and**  
*result*:  $steps0 \ (1, [], <[input]>) \ tm \ t = (0, Bk \uparrow \ k, <n::nat> \ @ \ Bk \uparrow \ l)$   
**unfolding** *TMC-has-num-res-iff* **by** *blast*  
**have**  $t$ : *has-num-result-at*  $tm \ [input] \ t$   
**unfolding** *has-num-result-at-def* **using** *final* *result* **by** *blast*  
**let**  $?t = LEAST \ u. \ \text{has-num-result-at } tm \ [input] \ u$   
**have**  $ex$ :  $\exists u. \ \text{has-num-result-at } tm \ [input] \ u$   
**using**  $t$  **by** *blast*  
**have**  $least$ : *has-num-result-at*  $tm \ [input] \ ?t$   
**using**  $ex$  **by** (*rule* *LeastI-ex*)  
**have** *none-before*:  $\forall u < ?t. \neg \text{has-num-result-at } tm \ [input] \ u$

```

    by (meson not-less-Least)
  have tm-num-halts-in tm input ?t
    using least not-less-Least tm-num-halts-in-def by auto
  then show  $\exists t. tm\text{-num-halts-in } tm\ input\ t$ 
    by blast
next
assume  $\exists t. tm\text{-num-halts-in } tm\ input\ t$ 
then obtain t where tm-num-halts-in tm input t
  by blast
then have has-num-result-at tm [input] t
  unfolding tm-num-halts-in-def by blast
then show TMC-has-num-res tm [input]
  unfolding TMC-has-num-res-iff
  using has-num-result-at-def by auto
qed

```

## 2.2 A finite size measure for Turing programs

```

fun action-rank :: action  $\Rightarrow$  nat where

```

```

  action-rank WB = 0
| action-rank WO = 1
| action-rank L = 2
| action-rank R = 3
| action-rank Nop = 4

```

```

definition instr-size :: instr  $\Rightarrow$  nat where

```

```

  instr-size i = Suc (action-rank (fst i) + snd i)

```

```

definition tm-size :: tprog0  $\Rightarrow$  nat where

```

```

  tm-size tm = length tm + sum-list (map instr-size tm)

```

```

lemma finite-instr-size-le: finite {i::instr. instr-size i  $\leq$  n}

```

```

proof -

```

```

  have subset: {i::instr. instr-size i  $\leq$  n}  $\subseteq$  (UNIV::action set)  $\times$  {s. s  $\leq$  n}

```

```

    by (auto simp: instr-size-def)

```

```

  have finite (UNIV::action set)

```

```

proof -

```

```

  have eq: (UNIV::action set) = {WB, WO, L, R, Nop}

```

```

    by (auto intro: action.exhaust)

```

```

  have finite ({WB, WO, L, R, Nop} :: action set)

```

```

    by simp

```

```

  then show ?thesis

```

```

    using eq by simp

```

```

qed

```

```

  have fn: finite ((UNIV::action set)  $\times$  {s. s  $\leq$  n})

```

```

    by (simp add:  $\langle$ finite (UNIV::action set) $\rangle$ )

```

```

  show ?thesis

```

```

    by (rule finite-subset[OF subset fn])

```

```

qed

```

```

lemma length-le-tm-size:
  length tm ≤ tm-size tm
  by (simp add: tm-size-def)

lemma instr-size-le-tm-size:
  assumes i ∈ set tm
  shows instr-size i ≤ tm-size tm
proof –
  have instr-size i ∈ set (map instr-size tm)
    using assms by auto
  then have instr-size i ≤ sum-list (map instr-size tm)
    by (simp add: member-le-sum-list)
  also have ... ≤ tm-size tm
    by (simp add: tm-size-def)
  finally show ?thesis .
qed

lemma finite-tm-size-le: finite {tm::tprog0. tm-size tm ≤ n}
proof –
  let ?I = {i::instr. instr-size i ≤ n}
  have subset: {tm::tprog0. tm-size tm ≤ n}
     $\subseteq$  {tm. set tm  $\subseteq$  ?I  $\wedge$  length tm  $\leq$  n}
  proof
    fix tm :: tprog0
    assume tm: tm ∈ {tm. tm-size tm ≤ n}
    then have len: length tm ≤ n
      using length-le-tm-size[of tm] by simp
    have set tm  $\subseteq$  ?I
      using instr-size-le-tm-size mem-Collect-eq order-trans subset-code(1) tm
      by fastforce
    then show tm ∈ {tm. set tm  $\subseteq$  ?I  $\wedge$  length tm  $\leq$  n}
      using len by simp
  qed
  have fin: finite {tm::tprog0. set tm  $\subseteq$  ?I  $\wedge$  length tm  $\leq$  n}
    using finite-instr-size-le by (rule finite-lists-length-le)
  show ?thesis
    by (rule finite-subset[OF subset fin])
qed

interpretation turing-busy-beaver: busy-beaver-base tm-size tm-num-halts-in
proof
  fix n
  show finite {p. tm-size p ≤ n}
    by (rule finite-tm-size-le)
next
  fix p x t u
  assume tm-num-halts-in p x t and tm-num-halts-in p x u
  then show t = u

```

by (rule *tm-num-halts-in-unique*)  
qed

**abbreviation** *Turing-BB-time* :: nat  $\Rightarrow$  nat **where**  
*Turing-BB-time*  $\equiv$  *turing-busy-beaver.BB-time*

**theorem** *Turing-BB-time-ge*:  
assumes *tm-size*  $tm \leq n$   
and *tm-num-halts-in*  $tm\ 0\ t$   
shows  $t \leq \textit{Turing-BB-time}\ n$   
using *turing-busy-beaver.BB-time-ge-time* *assms* by *blast*

## 2.3 The code-indexed Busy Beaver function and AFP's special halting problem

**context** *hpk*  
**begin**

The next predicate is deliberately diagonal: the formal locale input is not used because the special halting problem *K1* asks whether the machine coded by *c* halts on input *c*.

**definition** *coded-diagonal-num-halts-in* :: nat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
*coded-diagonal-num-halts-in*  $c\ \textit{input}\ t \iff \textit{tm-num-halts-in}\ (c2t\ c)\ c\ t$

**interpretation** *coded-busy-beaver*: *busy-beaver-base*  $\lambda c::\textit{nat}.\ c\ \textit{coded-diagonal-num-halts-in}$

**proof**

fix *n*  
show *finite*  $\{p::\textit{nat}.\ p \leq n\}$   
by *simp*  
**next**  
fix *p x t u*  
assume *coded-diagonal-num-halts-in*  $p\ x\ t$  **and** *coded-diagonal-num-halts-in*  $p\ x\ u$   
then show  $t = u$   
by (*auto simp: coded-diagonal-num-halts-in-def* *dest: tm-num-halts-in-unique*)  
qed

**abbreviation** *Coded-BB-time* :: nat  $\Rightarrow$  nat **where**  
*Coded-BB-time*  $\equiv$  *coded-busy-beaver.BB-time*

**lemma** *coded-halts-iff-K1*:

*coded-busy-beaver.halts*  $c\ 0 \iff [c] \in K1$

**proof**

assume *coded-busy-beaver.halts*  $c\ 0$   
then have  $\exists t.\ \textit{tm-num-halts-in}\ (c2t\ c)\ c\ t$   
unfolding *coded-busy-beaver.halts-def* *coded-diagonal-num-halts-in-def* by *blast*  
then have *TMC-has-num-res*  $(c2t\ c)\ [c]$   
using *TMC-has-num-res-iff-tm-num-halts* by *blast*  
then show  $[c] \in K1$

```

    unfolding K1-def by blast
next
assume [c] ∈ K1
then have TMC-has-num-res (c2t c) [c]
  unfolding K1-def by auto
then have ∃ t. tm-num-halts-in (c2t c) c t
  using TMC-has-num-res-iff-tm-num-halts by blast
then show coded-busy-beaver.halts c 0
  unfolding coded-busy-beaver.halts-def coded-diagonal-num-halts-in-def by blast
qed

```

```

theorem coded-BB-upper-bound-decides-K1:
  assumes coded-busy-beaver.upper-bound-for-BB b
  shows coded-busy-beaver.halting-decider-from b c 0 ⟷ [c] ∈ K1
  using coded-busy-beaver.halting-decider-from-correct-0[OF assms, of c]
  by (simp add: coded-halts-iff-K1)

```

```

corollary K1-not-turing-decidable-again: ¬ turing-decidable K1
  by (rule not-Turing-decidable-K1)

```

## 2.4 A code-indexed Busy Beaver function for the two-argument halting problem

The previous code-indexed function targets the diagonal/special problem  $K1$ . To cover AFP's two-argument halting problem  $H1$ , we fold the input into the Busy Beaver object itself: an object is a pair  $(c, m)$  consisting of a machine code and an input. The size measure  $c + m$  has finite bounded classes, and an upper bound for the resulting Busy Beaver function decides membership in  $H1$ .

```

definition coded-pair-num-halts-in :: (nat × nat) ⇒ nat ⇒ nat ⇒ bool where
  coded-pair-num-halts-in cm input t ⟷
    tm-num-halts-in (c2t (fst cm)) (snd cm) t

```

```

definition coded-pair-size :: nat × nat ⇒ nat where
  coded-pair-size cm = fst cm + snd cm

```

```

lemma finite-coded-pair-size-le:
  finite {cm::nat × nat. coded-pair-size cm ≤ n}

```

```

proof -
  have subset:
    {cm::nat × nat. coded-pair-size cm ≤ n} ⊆ {c. c ≤ n} × {m. m ≤ n}
  by (auto simp: coded-pair-size-def)
  have fin: finite ({c. c ≤ n} × {m. m ≤ n})
  by simp
  show ?thesis
  by (rule finite-subset[OF subset fin])
qed

```

**interpretation** *pair-busy-beaver*: *busy-beaver-base coded-pair-size coded-pair-num-halts-in*  
**proof**

```

  fix n
  show finite {p. coded-pair-size p ≤ n}
    by (rule finite-coded-pair-size-le)
next
  fix p x t u
  assume coded-pair-num-halts-in p x t and coded-pair-num-halts-in p x u
  then show t = u
    by (auto simp: coded-pair-num-halts-in-def dest: tm-num-halts-in-unique)
qed

```

**abbreviation** *Pair-BB-time* :: *nat ⇒ nat* **where**  
*Pair-BB-time* ≡ *pair-busy-beaver.BB-time*

**lemma** *coded-pair-halts-iff-H1*:

*pair-busy-beaver.halts* (c, m) 0 ↔ [c, m] ∈ *H1*

**proof**

```

  assume pair-busy-beaver.halts (c, m) 0
  then have ∃ t. tm-num-halts-in (c2t c) m t
    unfolding pair-busy-beaver.halts-def coded-pair-num-halts-in-def by simp
  then have TMC-has-num-res (c2t c) [m]
    using TMC-has-num-res-iff-tm-num-halts by blast
  then show [c, m] ∈ H1
    unfolding H1-def by blast
next
  assume [c, m] ∈ H1
  then have TMC-has-num-res (c2t c) [m]
    unfolding H1-def by auto
  then have ∃ t. tm-num-halts-in (c2t c) m t
    using TMC-has-num-res-iff-tm-num-halts by blast
  then show pair-busy-beaver.halts (c, m) 0
    unfolding pair-busy-beaver.halts-def coded-pair-num-halts-in-def by simp
qed

```

**theorem** *pair-BB-upper-bound-decides-H1-pair*:

**assumes** *pair-busy-beaver.upper-bound-for-BB* b  
**shows** *pair-busy-beaver.halting-decider-from* b (c, m) 0 ↔ [c, m] ∈ *H1*  
**using** *pair-busy-beaver.halting-decider-from-correct-0*[*OF* *assms*, *of* (c, m)]  
**by** (*simp add: coded-pair-halts-iff-H1*)

**definition** *H1-decider-from* :: (*nat ⇒ nat*) ⇒ *nat list ⇒ bool* **where**

```

H1-decider-from b nl ↔
  (case nl of
    [c, m] ⇒ pair-busy-beaver.halting-decider-from b (c, m) 0
  | - ⇒ False)

```

**theorem** *H1-decider-from-correct*:

**assumes** *pair-busy-beaver.upper-bound-for-BB* b

```

shows H1-decider-from b nl  $\longleftrightarrow$  nl  $\in$  H1
proof (cases nl)
  case Nil
  then show ?thesis
    by (simp add: H1-decider-from-def H1-def)
next
  case (Cons c xs)
  note nl-Cons = Cons
  then show ?thesis
  proof (cases xs)
    case Nil
    then show ?thesis
      using nl-Cons by (simp add: H1-decider-from-def H1-def)
    next
    case (Cons m ys)
    note xs-Cons = Cons
    then show ?thesis
  proof (cases ys)
    case Nil
    have nl-eq: nl = [c, m]
      using nl-Cons xs-Cons Nil by simp
    then show ?thesis
      using pair-BB-upper-bound-decides-H1-pair[OF assms, of c m] nl-eq
      by (simp add: H1-decider-from-def)
    next
    case (Cons k zs)
    then show ?thesis
      using nl-Cons xs-Cons by (auto simp: H1-decider-from-def H1-def)
  qed
qed
qed

```

**corollary** *H1-not-turing-decidable-again:  $\neg$  turing-decidable H1*  
**by** (*rule not-Turing-decidable-H1*)

For every upper bound, the Boolean decider above defines the same set as *H1*. Hence AFP's existing undecidability theorem for *H1* immediately yields a concrete noncomputability consequence: the characteristic function of any such upper-bound-induced decider set is not Turing-computable, even in the total sense. In particular, this applies to the decider induced by *Pair-BB-time* itself.

**definition** *H1-decider-set-from* ::  $(nat \Rightarrow nat) \Rightarrow nat$  list set **where**  
*H1-decider-set-from b = {nl. H1-decider-from b nl}*

**lemma** *H1-decider-set-from-eq-H1*:  
**assumes** *pair-busy-beaver.upper-bound-for-BB b*  
**shows** *H1-decider-set-from b = H1*  
**using** *H1-decider-from-correct[OF assms]*  
**by** (*auto simp: H1-decider-set-from-def*)

**theorem** *no-turing-decidable-H1-decider-set-from*:  
**assumes** *pair-busy-beaver.upper-bound-for-BB b*  
**shows**  $\neg$  *turing-decidable (H1-decider-set-from b)*  
**using** *H1-decider-set-from-eq-H1 [OF assms] not-Turing-decidable-H1*  
**by** *simp*

**theorem** *no-turing-computable-partial-H1-decider-from*:  
**assumes** *pair-busy-beaver.upper-bound-for-BB b*  
**shows**  $\neg$  *turing-computable-partial (chi-fun (H1-decider-set-from b))*  
**using** *no-turing-decidable-H1-decider-set-from [OF assms]*  
*turing-computable-partial-imp-turing-decidable*  
**by** *blast*

**theorem** *no-turing-computable-total-H1-decider-from*:  
**assumes** *pair-busy-beaver.upper-bound-for-BB b*  
**shows**  $\neg$  *turing-computable-total (chi-fun (H1-decider-set-from b))*  
**using** *no-turing-computable-partial-H1-decider-from [OF assms]*  
*turing-computable-total-imp-turing-computable-partial*  
**by** *blast*

**theorem** *no-turing-computable-total-H1-deciding-upper-bound*:  
 $\neg$  ( $\exists b$ . *pair-busy-beaver.upper-bound-for-BB b*  $\wedge$   
*turing-computable-total (chi-fun (H1-decider-set-from b))*)  
**using** *no-turing-computable-total-H1-decider-from* **by** *blast*

**corollary** *no-turing-computable-total-Pair-BB-time-decider*:  
 $\neg$  *turing-computable-total (chi-fun (H1-decider-set-from Pair-BB-time))*  
**using** *no-turing-computable-total-H1-decider-from*  
*pair-busy-beaver.BB-time-is-upper-bound*  
**by** *blast*

**end**

**end**

**theory** *Busy-Beaver*

**imports**

*Busy-Beaver-Base*

*Turing-Busy-Beaver*

**begin**

### 3 The Busy Beaver Entry

This theory is the AFP entry point. It collects the abstract Busy Beaver upper-bound principle and its instantiation for the Turing machines from the AFP Universal Turing Machine development.

**end**

## AI Assistance

AI assistance was used for proof engineering. The final definitions, statements, and proofs are checked by Isabelle.

## References

- [1] T. Radó. On non-computable functions. *Bell System Technical Journal*, 41(3):877–884, 1962.
- [2] 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, *Interactive Theorem Proving*, volume 7998 of *Lecture Notes in Computer Science*, pages 147–162, Berlin, Heidelberg, 2013. Springer.