

Computing the Uncomputable Rado Sigma Function

An Automated, Symbolic Induction Prover for non-halting Turing Machines

Joachim Hertel

50 Main St, STE 1000
White Plains, NY, 10606
jhertel@h-star.com

It is a classic result of computability theory that there exist uncomputable functions.

A prime example is Tibor Rado's [1] function $\Sigma[n]$ (a.k.a. the Busy Beaver function) that measures the productivity of n -state, binary Turing Machines. Despite being uncomputable, one knows [6] exact values of $\Sigma[n]$ for $n=1,2,3,4$ and a high lower bound for $n=5$: $\Sigma[5] \geq 4098$ [7].

To compute the exact value of $\Sigma[n]$ one has to decide the halting question for $(4n+1)^{2^n}$ n -state binary Turing Machines. Fortunately, this huge number can be reduced by applying well-known techniques like Tree Normalization, Back Tracking and Simple Loop Detection [6].

In this talk we report about a new tool that is successfully used in an ongoing project to compute $\Sigma[5]$: an automated Symbolic Induction Prover (SIP). The SIP tool is written in *Mathematica* and provides a unified way to prove that large groups of Turing Machines are non-halters. In a way, SIP enable certain Turing Machines to provide their own proof of being a non-halter.

With few undecided TMs, we now have strong evidence that indeed $\Sigma[5]=4098$. Final results will be reported in a forthcoming report [8].

■ Introduction

□ About Rado's Uncomputable Sigma Function

There are too many functions from positive integers to positive integers for them all to be (Turing) computable. A Cantor diagonalization argument [2] shows, that the set of all such functions is nonenumerable, whereas the set of all Turing machines is denumerable [2]. Hence, there must exist functions that are uncomputable.

In 1962, Tibor Rado [1] presented the uncomputable function Σ (aka the Busy Beaver function). Roughly speaking, $\Sigma(n)$ is the largest number of 1's left on the tape by a halting binary n -state Turing machine when started on an initially all 0-tape. The Σ function is uncomputable, because otherwise it would solve the Halting problem [2], which is known to be undecidable [2]. Even more, Rado [1] proved, that Σ grows faster than any computable function for large enough values of n , i.e. for any computable function $f: \mathbb{N} \rightarrow \mathbb{N}$, it holds that $\exists k_0 \forall k \geq k_0 f(k) < \Sigma(k)$.

In 1964, Green [11] established general lower bounds for the Σ function for larger values of n by explicitly constructing so-called Class M Turing machines, that generate long blocks of 1's. Julstrom [12] showed, that Green's Class M Turing machines are related to the Ackermann function [2], a function that is computable but not primitive recursive.

Various computational-based approaches to making progress on the Σ function have been taken: brute force searches, genetic algorithms, heuristic behavior analysis and many more [13] and the Σ function has become a classic topic in the theory of computing [13]. See [16] for a comprehensive list of references on Rado's Sigma function.

However, so far, exact values of $\Sigma[n]$ have only been computed for $n=1,2,3,4$ (see [6,13]) : $\Sigma[1]=1$, $\Sigma[2]=4$, $\Sigma[3]=6$, $\Sigma[4]=13$. These values are the first elements of the A028444 sequence of integers in the On-Line Encyclopedia of Integers [14].

In 1990, Marxen & Buntrock [7] established the lower bound $\Sigma[5] \geq 4098$, by publishing a record holding binary 5-state Turing machine, that halts after 47,176,870 steps and leaves 4098 1's on the tape.

Michel [5], shows a connection of most of the known low n record holding Turing machines and the Collatz $3x+1$ Problem.

There are three other related uncomputable functions associated with binary n -state Turing Machines:

Shift[n]: The maximum number of moves of a halting binary n -state Turing Machine started on an all 0-tape.

Num[n]: The largest unary number that can be created by a halting binary n -state Turing Machine started on an all 0-tape.

Space[n]: The largest number of different tape cells scanned by a halting binary n -state Turing Machine started on an all 0-tape.

Ben-Amram et al., [15] prove that $\text{Num}[n] \leq \Sigma[n] \leq \text{Space}[n] \leq \text{Shift}[n]$, $\forall n$.

□ Why Compute Values of Sigma

Chaitin [3] argues, that the Σ function is of considerable metamathematical interest. According to Chaitin [3], let P be a computable predicate of a positive integer n , so that for any specific n it is possible to compute if $P(n)$ is true or false. The Goldbach Conjecture is an example for P . The Σ function provides a crucial upper bound, for if P has algorithmic information content $[3] p$, it suffices to examine the first $\Sigma(p)$ natural numbers to decide whether P is True or False for all natural numbers. Hence, the Σ function enables one to convert a large but finite amount of calculations into a mathematical proof. Calculating $\Sigma(n)$ for specific values of n thus amounts to a systematic effort to settle all finitely refutable mathematical conjectures, that is, to determine all constructive mathematical truth [3].

□ Minds, Machines and Sigma

There is an ongoing philosophical discussion whether the human mind can surpass the Turing Limit, or can be understood as a Turing Machine. The famous logician Kurt Gödel discusses his point of view that the Mind can indeed surpass the Turing Limit in conjunction with his well known Incompleteness Theorem, see [9]. Gödel states [9], that “*although at each stage the number and precision of the abstract terms at our disposal may be finite, both may converge toward infinity in the course of the application of the procedure.*” Computing values of $\Sigma[n]$ for increasing n serves as a quantitative example for the situation Gödel is referring to in [9]. Indeed, Bringsjord et al. [4], present a *New Gödelian Argument for Hypercomputing Minds* that is based on the Σ function. In their argument the Σ function provides a discrete way to measure the achievement of the human mind surpassing the Turing Limit and establishing new constructive mathematical truth. In [4] the core assumption is that if the human mind can compute $\Sigma[n]$ it also can compute $\Sigma[n+1]$, (although that advancement might take quite a long time). Experience from our ongoing project of computing Σ for 5–state binary Turing Machines shows, that the halting question of large subsets of Turing Machines can be decided in a unified way using data mining to identify patterns and by the subsequent use of a *Symbolic Induction Prover* (see next chapter) to prove that these machines are in fact non-halters once they reach a certain pattern when started on an all 0–tape. If we succeed to identify an induction base (i.e., some pattern) we then can use the *Symbolic Induction Prover* to operate as a kind of Meta–Turing Machine, enabling the underlying Turing Machine to create its own proof of being a non–halter. Nothing in that process is restricted to 5–state Turing Machines. Hence we can imagine to climb beyond $n = 5$. The main challenge becomes the data mining part of identifying suitable induction base steps for increasingly complex and “erratic” behavior of Turing Machines.

We now turn in more detail to the computation of $\Sigma[5]$ and to our main computational tool for that task, the *Symbolic Induction Prover* for non–halting Turing Machines.

■ The Computation of Sigma for 5–State, Binary Turing Machines

□ Notations and Definitions

Binary n –state Turing Machines conform to these conditions:

The tape is infinite in both directions

The tape alphabet is $\Sigma = \{0,1\}$

The all 0–tape is called the blank tape Ω

The Turing Machine has n states, labeled $1, \dots, n$ and a HALT state, labeled 0

At each step the machine reads, writes the cell scanned by the Turing head and moves the Turing head one cell to the left/right

Here, we focus on 5–state binary Turing Machines:

Definition: The instruction table of a 5–state binary Turing machine M is a 5–by–2 table such that $M[s,h] = \{ws, mv, ns\}$, with $s \in \{1,2,3,4,5\}$, $h \in \{0,1\}$, $ws \in \{0,1\}$, $mv \in \{L,R\}$, $ns \in \{0,1,2,3,4,5\}$

We call s the current state of M and h the read symbol in the tape cell positioned under the Turing head H . The triple $\{ws, mv, ns\}$ is called a Turing instruction with ws the write symbol being written into the tape cell positioned under the Turing head H , mv the move direction of the Turing head H and ns the next state of M . If $ns = 0$, M stops, otherwise it continues executing instructions.

Without loss of generality we assume the HALT instruction to be $\{1,R,0\}$.

That leaves us with $(2 * 2 * 5 + 1)^{2*5} = 21^{10}$ possible binary 5–state Turing machines. We call this finite set $\mathcal{T}\mathcal{M}_5$.

Let $M \in \mathcal{TM}_5$. $M[\Omega]$ means Turing machine M is started on tape Ω .
 Define :

$$\sigma(M) := \begin{cases} k & M[\Omega] \text{ halts and leaves } k \text{ '1's on tape} \\ 0 & \text{otherwise} \end{cases}$$

Example : $M_{MB} = \left(\begin{array}{ll} \{1, R, 2\} & \{1, L, 3\} \\ \{1, R, 3\} & \{1, R, 2\} \\ \{1, R, 4\} & \{0, L, 5\} \\ \{1, L, 1\} & \{1, L, 4\} \\ \{1, R, 0\} & \{0, L, 1\} \end{array} \right)$

M_{MB} is a 5 – state Turing Machine first published by Marxen and Buntrock [7] .

$M_{MB}[\Omega]$ halts after 47, 176, 870 steps and leaves 4098 1 – symbols on the tape, i.e., $\sigma[M_{MB}] = 4$

$$\Sigma[5] := \text{Max}[\{\sigma[M], M \in \mathcal{TM}_5\}]$$

Hence $\Sigma[5] \geq 4098$.

One has to solve the halting question for a finite, but large number of Turing machines to compute the value of $\Sigma(5)$.

Configurations

Let denote Σ^* the set of finite words from alphabet Σ .

A machine configuration c is an expression

$$c = \{s, \{\omega, \{\{x\}, 1\}, h, \{\{y\}, 1\}, \omega\}, s \in \{0, 1, 2, 3, 4, 5\}, h \in \Sigma, x, y \in \Sigma^*\}$$

Note : ω denotes the left / right infinite blank portion of the tape

In this notation $\Omega = \{\omega, \mathbf{0}, \omega\}$.

We use $\{\{x\}, n\} := \left\{ \left\{ \underset{\text{n-times}}{x}, \dots, x \right\}, 1 \right\}$ for all $x \in \Sigma$

Σ^* to define symbolic configurations with multipliers $n \geq 1$.

Example : $c = \{s, \{\omega, \{\{x\}, n\}, h, \{\{y\}, m\}, \omega\}\}$

Example : $c = \{\mathbf{3}, \{\omega, \mathbf{0}, \{\{1, 0, 1\}, k\}, \omega\}\}$

i.e. M is in state **3**, the Turing head scans a 0 – cell, and the sub – tape $\{1, 0, 1\}$ is occurring k – times.

□ General Approach: Discover and Prove

By using well-known techniques [6], such as Tree Normalization, Backtracking, Simple Loop Detection etc., one can decide the Halting question for many of the 5-state binary Turing Machines:

21^{10} possible 5-state binary Turing Machines $\xrightarrow{\text{apply known techniques}}$ $\approx 1,000,000$
 undecided machines ("Holdouts")

For each of the remaining undecided Turing Machines do:

Iterate the holdout machine a "number of times", starting on the blank tape Ω

Save the configuration after each step

Try to discover recurring patterns and store them as machine configurations

Examples:

$$c_L(n) := \{\mathbf{s}, \{\omega, \mathbf{h}, \{(x), a^*n+b\}, \omega\}\}, \quad n \geq 0$$

$$c_R(n) := \{\mathbf{s}, \{\omega, \{(x), a^*n+b\}, \mathbf{h}, \omega\}\}, \quad n \geq 0$$

$$c_L(n) := \{\mathbf{s}, \{\omega, \mathbf{h}, \{(x), 1\}, \{(y), p_n\}, \omega\}\}, \quad n \geq 0$$

$$c_R(n) := \{\mathbf{s}, \{\omega, \{(x), 1\}, \{(y), p_n\}, \mathbf{h}, \omega\}\}, \quad n \geq 0$$

with $x, y \in \Sigma^*$ and the multiplier p_n satisfying a recurrence relation $p_{n+1} = a^*p_n + b$

Here are some actual examples of configurations:

A simple linear configuration: $c(n) = \{\mathbf{2}, \{\omega, \mathbf{0}, \{\{1\}, 4n\}\}, \omega\}$

A simple exponential configuration: $c(n) = \{\mathbf{1}, \{\omega, \mathbf{1}, \{\{0\}, 3^n\}, \{\{1\}, 3\}, \omega\}\}$

A more complicated configuration:

$$c(n) = \{\mathbf{1}, \{\omega, \mathbf{0}, \{\{0, 0, 1\}, 2^{2^{n+1}}\}, \{\{0, 1\}, 3\}, \{\{0, 0, 1\}, 2^{2^{n-1}}\}, \{\{0, 1\}, 3\}, \dots, \{\{0, 0, 1\}, 2^3\}, \{\{0, 1\}, 3\}, \{\{0, 0, 1\}, 2^1\}, \{\{0, 1\}, 3\}, \omega\}\}$$

Once we have identified a reliable hypothesis for a recurring configuration $c(n)$ of a Turing Machine M we want to create an induction proof showing that M does indeed NOT halt.

The general scheme is as follows:

Step 1: Establish the Induction Proposition: $M:\Omega \Rightarrow c(n), \forall n \geq 0$

(i.e. M transforms the blank tape into $c(n)$ in a countable number of steps)

Step 2: Verify the Base Case, i.e. check that $M:\Omega \Rightarrow c(n), n = 0$ (or any finite number of n)

Step 3: Formulate the Induction Hypothesis, i.e. assume that $M:\Omega \Rightarrow c(k), \forall 0 \leq k \leq n$

Step 4: Prove the Induction Step, i.e. $M:\Omega \Rightarrow c(n+1)$, or equivalently $M:c(n) \Rightarrow c(n+1)$

Note: It is the Induction Step (Step 4) that involves the handling of symbolic configurations because we have to show $M:c(n) \Rightarrow c(n+1)$. This cannot be done by using the Turing Machine's instruction table as it is, because the Turing instructions are defined for numeric tape configurations only, not for symbolic configurations. For this we need the Symbolic Induction Prover (SIP), a kind of meta interpreter, that enables

the Turing Machine to produce it's own induction proof of being a non-halter.

Note: If the induction proof is successful we know that the Turing Machine M does not halt and hence $\sigma(M) = 0$.

Note: the induction proof might fail because:

The hypothesis is wrong (it's just based on a finite amount of tape data)

The proof does not terminate within the pre-specified amount of CPU time (i.e. it might go through if we allow more CPU time)

A situation is encountered beyond the implemented induction logic (we need to enhance the prover)

A generalized Collatz $(3x+1)$ problem is encountered (see below)

We now turn to a more detailed discussion of the Symbolic Induction Prover (SIP).

■ The Symbolic Induction Prover

□ Meta-Instructions

The underlying idea is a unified and simple principle: analyse the Turing Machine's instruction table and extract rules which describe meta-transactions on maximal invariant sub-tapes.

Example: Let $M =$

s / h	0	1
1	*	*
2	(0, R, 5)	*
3	(1, L, 3)	*
4	*	*
5	*	(0, R, 2)

Example : 1st order Meta - Instruction

$$\boxed{\{3, \{gl, \{0, n\}, \mathbf{0}, gr\}\} \rightarrow \{3, \{gl, \mathbf{0}, \{1, n\}, gr\}\}} \forall n \geq 1$$

Here:

gl symbol for **general left** tape

gr symbol for **general right** tape

If at time t , M has configuration

$$c_t(n) = \{3, \{\omega, \{x, 1\}, \{0, n\}, \mathbf{0}, \{y, 1\}, \omega\}\} \text{ for some words } x, y \in \Sigma^*$$

we can apply the meta-instruction to get

$$c_{t+1}(n) = \{3, \{\omega, \{x, 1\}, \mathbf{0}, \{1, n\}, \{y, 1\}, \omega\}\}$$

Example : 2nd order Meta - Instruction

$$\boxed{\{2, \{gl, \mathbf{0}, \{1, 0, n\}, gr\}\} \rightarrow \{2, \{gl, \{0, 2n\}, \mathbf{0}, gr\}\}} \forall n \geq 1$$

Note: A meta-instruction modifies an arbitrarily large countable portion of the tape.

Induction Schemes

SIP has a build-in library to support several induction proof schemes.

Induction schemes are used in conjunction with meta-instructions to produce an induction proof for a non-halting Turing Machine.

If necessary, the library can be enhanced with new induction schemes.

We discuss some of the implemented schemes.

Scheme: Commutation Relations at the Tape Boundary

Assume Turing Machine M exhibits the symbolic machine configuration

$c(k) = \{s, \{\omega, h, \{x, k\}, \{q, 1\}, \omega\}\}$ for some $x, q \in \Sigma^*$

Verify M: $\Omega \implies c(k)$ for $k=0$.

Assume: M: $c(k-1) \implies c(k)$.

Prove by Induction: M: $c(k) \implies c(k+1)$

If true, M does not HALT and hence $\sigma[M]=0$.

Induction Proof

Let M be in configuration

$c(k) = \{s, \{\omega, h, \{x, k\}, \{q, 1\}, \omega\}\}$

SIP searches for maximal invariant boundary conditions and commutation relations:

Check if

M: $\{s, \{\omega, h, \{x, 0\}, \{\tilde{q}, 1\}, gr\}\} \implies \{s, \{\omega, h, \{x, 1\}, \{\tilde{q}, 1\}, gr\}\}$, for some $\tilde{q} \in \Sigma^*$, such that $\tilde{q} =$

Take[q, Length[\tilde{q}]]

If True, check further if $x || \tilde{q} = \tilde{q} || \tilde{x}$ for some $\tilde{x} \in \Sigma^*$

If True, modify the Induction Assumption to read:

Extended Induction Hypothesis

M: $\{s, \{\omega, h, \{x, k-1\}, \{\tilde{q}, 1\}, gr\}\} \implies \{s, \{\omega, h, \{x, k\}, \{\tilde{q}, 1\}, gr\}\}$

SIP proves this extended induction assumptions as follows:

$\{s, \{\omega, h, \{x, k\}, \{\tilde{q}, 1\}, gr\}\}$

↓

$\{s, \{\omega, h, \{x, k-1\}, \{x, 1\}, \{\tilde{q}, 1\}, gr\}\}$

↓

$\{s, \{\omega, h, \{x, k-1\}, \{\tilde{q}, 1\}, \{\tilde{x}, 1\}, gr\}\}$ (using the commutation relation)

↓

$\{s, \{\omega, h, \{x, k\}, \{\tilde{q}, 1\}, \{\tilde{x}, 1\}, gr\}\}$ (using the extended induction hyp for

$(k-1) \mapsto (k)$ on any gr

↓

$\{s, \{\omega, h, \{x, k\}, \{x, 1\}, \{\tilde{q}, 1\}, gr\}\}$ (using commutation relation)

↓

$\{s, \{\omega, h, \{x, k+1\}, \{\tilde{q}, 1\}, gr\}\}$ ■

Hence M: $c(k) \implies c(k+1), \forall k \geq 0$ and M does NOT halt, hence $\sigma[M]=0$.

Scheme: Cyclic Conditions at the Tape Boundary

Assume Turing Machine M exhibits the symbolic machine configuration

$c(k) = \{s, \{\omega, h, \{x, k\}, \{q, 1\}, \omega\}\}$, for some $x, q \in \Sigma^*$

Induction Base:

Verify $M: \{s, \{\omega, \mathbf{h}, \{x, 1\}, \text{gr}\}\} \Rightarrow M: \{s, \{\omega, \{y, 1\}, \mathbf{h}, \text{gr}\}\}$

Induction Assumption

$M: \{s, \{\omega, \mathbf{h}, \{x, j\}, \text{gr}\}\} \Rightarrow M: \{s, \{\omega, \{y, j\}, \mathbf{h}, \text{gr}\}\}, \forall 1 \leq j \leq k-1$

Assume further, that the instruction table of M results in meta-instructions:

$M1: \{s, \{\omega, \{y, n\}, \mathbf{h}, \{c_i, 1\}, \text{gr}\}\} \rightarrow \{s, \{\omega, \mathbf{h}, \{x, n\}, \{c_{i+1}, 1\}, \text{gr}\}\}, y, c_0, \dots, c_m \in \Sigma^*, \forall n \geq 0, 0 \leq i < m$

$M2: \{s, \{\omega, \{y, n\}, \mathbf{h}, \{c_m, 1\}, \text{gr}\}\} \rightarrow \{s, \{\omega, \{y, n+1\}, \mathbf{h}, \text{gr}\}\}$

Note: c_0, \dots, c_m are the cyclic boundary conditions, with $c_0 = \{\}$

Induction proof:

$\{s, \{\omega, \mathbf{h}, \{x, k\}, \{q, 1\}, w\}\}$	
$\{s, \{\{\omega, \mathbf{h}, \{x, k-1\}, \{x, 1\}, \{q, 1\}, w\}\}\}$	
$\{s, \{\{\omega, \mathbf{h}, \{x, k-1\}, \text{gr}\}\}\}$	(replacing $\{\{x, 1\}, \{q, 1\}, w\}$ by gr)
$\{s, \{\omega, \{y, k-1\}, \mathbf{h}, \text{gr}\}\}$	(1 st use of induction hyp)
$\{s, \{\omega, \mathbf{h}, \{x, k-1\}, \{c_1, 1\}, \text{gr}\}\}$	(using meta-instruction M1)
$\{s, \{\omega, \{y, k-1\}, \mathbf{h}, \{c_1, 1\}, \text{gr}\}\}$	(2 nd use of induction hyp)
$\{s, \{\omega, \mathbf{h}, \{x, k-1\}, \{c_2, 1\}, \text{gr}\}\}$	(using meta-instruction M1)
⋮	
$\{s, \{\omega, \{y, k-1\}, \mathbf{h}, \{c_{m-1}, 1\}, \text{gr}\}\}$	(($m-1$) th use of induction hyp)
$\{s, \{\omega, \mathbf{h}, \{x, k-1\}, \{c_m, 1\}, \text{gr}\}\}$	(using meta-instruction M1)
$\{s, \{\omega, \{y, k\}, \mathbf{h}, \text{gr}\}\}$	(use of induction hyp and use of M2)
$\{s, \{\omega, \mathbf{h}, \{x, k\}, \{c_1, 1\}, \text{gr}\}\}$	(using meta-instruction)
$\{s, \{\omega, \mathbf{h}, \{x, k\}, \{c_1, 1\}\}\{x, 1\}, \{q, 1\}, w\}$	(plugging back-in the explicit right tape portion)
↓	
$\{s, \{\omega, \mathbf{h}, \{x, k+1\}, \{p, 1\}\}\{q, 1\}, w\}$	(this step might vary in other cases)
↓	
$\{s, \{\omega, \mathbf{h}, \{x, k+1\}, \{q, 1\}\}\{0, p\}, w\}$	for some $p > 0$
$\{s, \{\omega, \mathbf{h}, \{x, k+1\}, \{q, 1\}, w\}\}$	(subsuming the finite 0's into ω)■

Scheme: Decreasing Cell Sequence at the Tape Boundary

Assume Turing Machine M exhibits:

$\{s, \{\omega, \mathbf{h}, \{A, n\}, \{B, 1\}, \{R, k\}, \text{gr}\}\}$	$k \geq 0$ Integer
⋮	
$\{s, \{\omega, \mathbf{h}, \{A, n\}, \{B, 2\}, \{R, k-1\}, \text{gr}\}\}$	
⋮	
$\{s, \{\omega, \mathbf{h}, \{A, n\}, \{B, j\}, \{R, k-j\}, \text{gr}\}\}$	

Induction Base:

Verify, that for some function f :

$M: \{s, \{\omega, \mathbf{h}, \{A, n\}, \{B, 1\}, \{R, 1\}, \text{gr}\}\} \rightarrow \{s, \{\omega, \mathbf{h}, \{A, n + f(1)\}, \{B, 2\}, \text{gr}\}\}$

Verify the Swap Meta Instruction

$$\{s, \{\omega, \mathbf{h}, \{A\}, n\}, \{B\}, k, \{R\}, 1, \text{gr}\} \longrightarrow \\ \{s, \{\omega, \mathbf{h}, \{A\}, n+k_0\}, \{B\}, 1, \{R\}, k-1, \{B\}, 1, \text{gr}\}$$

Induction Assumption:

$$M: \{s, \{\omega, \mathbf{h}, \{A\}, n\}, \{B\}, k-1, \{R\}, 1, \text{gr}\} \longrightarrow \{s, \{\omega, \mathbf{h}, \{A\}, n+f[k-1]\}, \{B\}, k, \text{gr}\},$$

for some function f

Induction Proof:

$$\{s, \{\omega, \mathbf{h}, \{A\}, n\}, \{B\}, k, \{R\}, 1, \text{gr}\}$$

↓

(apply Swap Meta

Instruction)

$$\{s, \{\omega, \mathbf{h}, \{A\}, n+k_0\}, \{B\}, 1, \{R\}, k-1, \{B\}, 1, \text{gr}\}$$

↓

(apply (k-1)-times

Induction Assumption)

$$\{s, \{\omega, \mathbf{h}, \{A\}, n+k_0 + \sum_{i=1}^{k-1} f[i]\}, \{B\}, k, \{R\}, 0, \{B\}, 1, \text{gr}\}$$

↓

$$\{s, \{\omega, \mathbf{h}, \{A\}, n+k_0 + \sum_{i=1}^{k-1} f[i]\}, \{B\}, k+1, \text{gr}\}$$

Now, require that

$$k_0 + \sum_{i=1}^{k-1} f[i] = f[k] \text{ and hence } f[k] = k_0 \cdot 2^k$$

Similar schema are also supported:

$$\{s, \{\omega, \{A\}, n\}, \{B\}, k, \mathbf{h}, \{R\}, 1, \text{gr}\}$$

↓

(apply Swap Meta

Instruction)

$$\{s, \{\omega, \{A\}, n+k_0\}, \{B\}, 1, \mathbf{h}, \{R\}, k-1, \{X\}, 1, \text{gr}\}$$

and

$$\{s, \{\omega, \{A\}, n\}, \mathbf{h}, \{B\}, k, \{R\}, 1, \text{gr}\}$$

↓

(apply Swap Meta

Instruction)

$$\{s, \{\omega, \{A\}, n+k_0\}, \mathbf{h}, \{B\}, 1, \{R\}, k-1, \{X\}, 1, \text{gr}\}$$

Modulo Arithmetic

Often, meta-instructions are subject to some modulo condition, see e.g., the annotated sample proof

Example:

$$M: \{s, \{\text{gl}, \{1, n\}, \mathbf{h}, \text{gr}\}\} \longrightarrow \{s, \{\text{gl}, \{1\}, \text{Mod}[n, 3]\}, \mathbf{h}, \{0, 1, 0\}, 3 * \text{Floor}[\frac{n}{3}], \text{gr}\}$$

If possible, SIP calculates $\text{Mod}[n, p]$ explicitly by using structural information of n .

Example:

If the variable is of the form $2n+1$ and $p = 2$ we know that $\text{Mod}[2n+1, 2] = 1$.

In general SIP uses modulo arithmetic in the finite ring \mathbb{Z}_n , including

Fermat's Little Theorem: $\text{Mod}[a^p, p] = a, \forall a \in \text{Integers}, \forall p \in \text{Primes}$

and **Euler's generalization** $\text{Mod}[a^{\varphi(n)}, n] = 1, \forall a, n \in \text{Integers}$

(Euler's Phi function $\varphi[n]$ gives the number of positive integers less than or equal to n)

which are relatively prime to n).

If nothing specific can be computed, SIP picks a value $\text{Mod}[n,p] = v \in \{0, \dots, p-1\}$ and generates p sub-proofs, one for each possible v value at each decision point in the general induction proof.

□ The SIP Package

SIP includes these major functional packages to support meta instructions and symbolic induction proofs for Turing Machine configurations:

Create and Execute Meta Instructions

metaservices \Rightarrow creates rules that represent Meta Instructions

swapservices \Rightarrow handles all orders of meta instructions

shiftservices \Rightarrow shifts configurations into Normal Form to make them comparable

Handling of Induction Proofs

inductionservices \Rightarrow the main package for induction proof schemes

tracebufferservices \Rightarrow detects emerging schemes by tracing the steps of an induction proof, producing induction assumptions, invokes SIP recursively for proof

Modulo Arithmetic for Symbolic Variables

getgaugedvarservice \Rightarrow handles modulo arithmetic for symbolic variables

Handling Boundary Conditions

extremepointservices \Rightarrow handles logic associated with maximal invariant tape boundaries

Technical Services

semaphoresupport \Rightarrow provides services to serialize induction schemes to prevent "vicious circles"

environmentservices \Rightarrow specifies the Proof Environment (global parameters etc)

□ Annotated Sample Proof

Consider this Turing Machine

$$M = \left(\begin{array}{cc} \{1, R, 2\} & \{0, L, 1\} \\ \{1, R, 3\} & \{1, L, 2\} \\ \{1, R, 4\} & \{1, L, 1\} \\ \{1, L, 5\} & \{0, R, 5\} \\ \{1, R, 0\} & \{0, L, 2\} \end{array} \right)$$

Assumed configuration:

$$c(n) = \{4, \{w, \{1\}, 5+2n\}, 0, \{1\}, 1+n, w\}$$

Base Case: $M:\Omega \Rightarrow c(n)$ verified for $n=0,2,4,\dots$

Induction Step

$M:c[n] \rightarrow c[n+2]$

NotebookOpen["r://sip.nb"]; (* CD drive *)

□ Results

Start with \mathcal{TM}_5 , $\text{card}(\mathcal{TM}_5) = 21^{10}$

Applying well-known [6] and fast techniques (such as Tree Normal Form, BackTrack, Simple Loop Detection etc) leaves about 1,000,000 Turing Machines undecided.

→ **850,000** Turing Machines of a linear type, e.g. $\{s, \{\omega, h, \{x, a \cdot n + b\}, \omega\}\}$ for some $x \in \Sigma^*$, or slightly more complex, **proved by SIP to not HALT**

→ **143,000** Turing Machines of polynomial or exponential configurations, **proved by SIP to not HALT**

→ 1,000 Turing Machines currently too complex for the Symbolic Prover

→ → **900** of those are **manually proven to not HALT**

→ → **100** cases are still **HOLDOUTS** and are currently under consideration, but there is **strong evidence that they do not halt**

→ **6,000** Turing Machines **HALT within 50 Mio Steps**

Given that, we find that for the number of 1-Symbols (Σ) left on the tape, the number of cells scanned (Space) and the number of moves (Shift) the Marxen-Buntrock [7] machine is the champion

$$M_{\text{MB}} = \begin{pmatrix} \{1, R, 2\} & \{1, L, 3\} \\ \{1, R, 3\} & \{1, R, 2\} \\ \{1, R, 4\} & \{0, L, 5\} \\ \{1, L, 1\} & \{1, L, 4\} \\ \{1, R, 0\} & \{0, L, 1\} \end{pmatrix}$$

with:

$$\Sigma[5] = 4098$$

$$\text{Shift}[5] = 47,176,870$$

$$\text{Space}[5] = 12,289$$

The largest **unary** number (i.e. empty tape with a single block of consecutive 1's) produced by any halting binary 5-state Turing Machine is

$$\text{Num}[5] = 165$$

The NUM-Champion is:

$$M_{\text{NUM-Champ}} = \begin{pmatrix} \{1, R, 2\} & \{1, L, 1\} \\ \{1, R, 3\} & \{1, L, 5\} \\ \{1, R, 4\} & \{1, R, 5\} \\ \{0, L, 1\} & \{1, R, 3\} \\ \{1, R, 0\} & \{0, L, 2\} \end{pmatrix}$$

Note: There exists the possibility that the halting question for one or more of the remaining holdouts is linked to the Collatz $3x+1$ problem (see also [5]).

This is currently under investigation and the final results will be reported in a forthcoming paper [8].

■ References

- [1] T.Rado, Bell System Technical Journal, **41**, 1962, 877–884
- [2] G.S.Boalos, R.C.Jeffrey Computability and Logic, Cambridge, 1989
- [3] G.J.Chaitin, Computing the Busy Beaver Function
in: T.M. Cover, B. Gopinath, Open Problems in Communication and Computation,
Springer, 1987
- [4] S.Bringsjord et al., A New Gödelian Argument for Hypercomputing Minds Based on
the Busy Beaver Problem
To be published in: Applied Mathematics and Computation
- [5] P.Michel, Arch. Math. Logic, **32**, 1993, 351–367
- [6] R.Machlin, Q.Strout, Physica D, **42**, 1990, 85–98, and references therein
- [7] H.Marxen, J.Buntrock, Bull. EATACS, **40**, 1990, 247–251
- [8] J.Hertel, The Computation of the Value of Rado's Non-Computable Ω Function for
5-State Turing
Machines, work in progress
- [9] K.Gödel, Some Remarks on the Undecidability Results, in [10, pp 305–306], 1972
- [10] S.Fefferman et al. (eds.), Kurt Gödel Collected Works Vol. II, Oxford, 1990
- [11] M.W.Green: Proceedings of the 5th IEEE Annual Symposium on Switching Circuit
Theory and Logical Design, 1964, 91–94
- [12] B.A.Julstrom, ACM SIGACT, **23**, 1992, 100–106
- [13] E. W. Weisstein, Busy Beaver.From MathWorld—A Wolfram Web Resource.
<http://mathworld.wolfram.com/BusyBeaver.html>
- [14] On-Line Encyclopedia of Integer Sequences, Sequence Id=A028444
<http://www.research.att.com/~njas/sequences/index.html>
- [15] A.M.Ben-Amrein, B.A.Julstrom, K.Zwick, Math. Systems Theory, **29**, 1996, 375–386
- [16] H.J.M.Wijers, Bibliography on the Busy Beaver Problem, 2004
<http://www.win.tue.nl/~wijers/bbbibl.pdf>