# The Undecidability of the Generalized Collatz Problem

Stuart A. Kurtz
Department of Computer Science
The University of Chicago

Janos Simon
Department of Computer Science
The University of Chicago

December 26, 2006

**Abstract**

The Collatz problem, widely known as the $3x+1$ problem, asks whether or not a certain simple iterative process halts on all inputs. We build on earlier work by J. H. Conway, and show that a natural generalization of the Collatz problem is recursively undecidable.

## 1 Introduction

Define the function $g: \omega \to \omega$ as follows:

$$g(x) = \begin{cases} x/2, & \text{if } n \text{ is even;} \\ 3x+1, & \text{if } n \text{ is odd.} \end{cases}$$

Let $g^{(i)}$ denote the $i$-th iterate of $g$, i.e.,

$$g^{(i)}(x) = \overbrace{g(g(\ldots g}^{i}(x)\ldots))$$

The Collatz problem asks

**Problem 1.1** *For all integers $x > 0$, is there is an $i$ such that $g^{(i)}(x) = 1$.*

Because of its tantalizingly elementary form, and our inability to settle it the Collatz problem has received substantial attention. Collatz started working on the problem in 1928, but, since he felt he made little progress, only published a history of its origin in 1986 [**?**]. There is a very extensive literature on the many attempts to settle the conjecture, as well as related questions, using an arsenal of technologies from Number Theory, to Dynamical Systems, and Markov Chains: there is a 47-page annotated bibliography [**?**], excellent surveys [**?**], even a monograph dedicated to the Dynamical Systems generalization [**?**]. Erdös offered \$500,00 for a solution, and there are people working on it today [**?**].

We provide a good heuristic explanation for the apparent difficulty of the problem. We consider a generalization, defined by Conway [**?**], [**?**].

m 1.1.

**Definition 1.2** *A function $g$ is called an* Collatz function *if there is an integer $n$ together with rational numbers $\{a_i : \ i < n\}$, $\{b_i : \ i < n\}$ such that whenever $x \cong i \bmod p$, then $g(x) = a_i x + b_i$ is integral.*

Note that this is a natural generalization: in the original problem $p$ is 2, and the two functions are $b_0 = n/2$ and $b_1 = 3n + 1$

The corresponding problem is

**Problem 1.3** *Given a representation for a Collatz function $g$, it can be decided whether for all integers $x > 0$, there is an $i$ such that $g^{(i)}(x) = 1$?*

Our result is

**Theorem 1.4** *Given a Collatz function $g$, it is undecidable whether or not for all integers $x$ there exists an $i$ such that $g^{(i)}(x) = 1$.*

The somewhat stronger version of the result is the following.

Let $g()$ be a Collatz function, and let $range - g() = \{x|\text{there exists an } i$ such that $g^{(i)}(x) = 1\}$.

**Theorem 1.5** *The problem is $range - g() = \omega\}$ is $\Pi_2$complete.*

Our results build on the beautiful results of Conway, who proved

**Theorem 1.6** *Given a Collatz function $g$, it is undecidable whether or not for all integers $x$ of the form $2^k$, there exists an $i$ such that $g^{(i)}(x) = 1$.*

Conway's proof works by constructing for each partial recursive function $\varphi_e$ a Collatz function $g$, such that various integers represent instantaneous descriptions of a register machine computing $\varphi_e$, and (iterates of) $g$ perform the same transformation on integers that the register machine performs on states. Integers of the form $2^k$ represent the configuration consisting of the initial state, with $k$ as the input.

such that $g^{(i)}(2^x) = 1$ if and only if $\varphi_e(x)$ converges in $i$ steps, when $\psi$ is computed on a simple register machine. Theorem 1.6 follows from the recursive undecidability of $\mathsf{tot} = \{e : \varphi_e \text{ is total}\}$.

The difficulty with extending Conway's proof to obtain a proof of the undecidability of Problem 1.3 is that the action of Conway's $g$'s on integers not of the form $2^k$ has nothing to do with the totality $\varphi_e$. Trying to extend Conway's proof requires producing a computational system whose behavior can be controlled when started in an arbitrary state, with garbage on its tape or in its store.

We get around this difficulty by running a *double* simulation of a register machine. One simulation, the so-called *outer simulation* is run forward. This simulation will halt if and only if the original register machine halted. The other simulation, the so-called *inner simulation* checks the outer simulation for consistency. When the inner simulation agrees that the outer simulation is consistent, only then is the outer simulation permitted to advance. At this point, a new inner simulation is started. We furthermore arrange that the inner simulation is always guaranteed to halt.

## 2 Register Machines

A register machine is a simple mathematical model of a digital computer. Register machines are equipped with finitely many registers, each of which can contain any natural number. Control is handled by a finite sequence of instructions. There are three kinds of instructions: increment a register, decrement a register, and halt.

An increment instruction indicates which register is to be incremented, and the index of the next instruction to be executed. A decrement instruction indicates which register is to decremented, and two indices, one which will be the next instruction to be executed if the decrement succeeded, i.e., if the register to be decremented was originally nonzero, and the other which will be the next instruction if the decrement failed, i.e., if the register was originally zero.

Each register machine $M$ defines a function $\psi_M$, where $\psi_M(i) = j$ if and only if when $i$ is placed in register 0, and the machine is started in state 0, it halts with $j$ in register 0.

**Theorem 2.1** *A function $\varphi$ is partial recursive if and only if there is a register machine $M$ such that $\varphi = \psi_M$.*

Research on this particular formalism was initiated by Minski [Min61], who proved:

**Theorem 2.2** *It suffices to consider register machines having only two registers, i.e., every register machine can be effectively converted into a register machine computing the same function which has only two registers.*

We will not use Theorem 2.2 in this paper, although it would slightly simply the presentation of our main results. Our contribution begins with the following:

**Theorem 2.3** *A register machine $M$ can be effectively converted into a register machine $M'$ such that $M$ is total if and only if $M'$ reaches a halting configuration from* every *configuration.*

**Proof:** Fix a register machine $M$.

Rather than providing an explicit instruction sequence for $M'$, we will describe its behavior using certain higher level instructions and control structures which can be easily compiled into the language of register machines. This technique for register machine construction, as well as most of our high level primatives, is extensively justified in [**?**].

The machine $M'$ has the following registers:

- for every register of $M$, we give $M'$ *two* registers, one for the *inner simulation*, and one for the *outer simulation*;
- a register called the *input* register;
- two registers called the *inner-clock*, and the *outer-clock*; and
- several temporary registers used to implement higher level instructions.

The machine $M'$ will also have two local variables:

- an *inner-state* and an *outer-state*.

```
begin
    outer-state := initial state of $M$                                (1)
```

Figure 1: The simulation for Theorem 2.3.

The range these variables consists of the states of $M$. As this range is finite, these variables will actually be compiled into the state of $M'$.

We believe the intent of each of the lines in Figure 1 are self-explanatory, with the possible exception of the "advance" commands at lines (5) and (11). Let us consider line (5) specifically. The intent of the command "advance outer simulation" is that $M'$, using the notion of the state of $M$ that it stored in the variable "outer-state", is to do with the outer-registers whatever $M$ does to its registers, and then to update "outer-state" to reflect the next state of $M$.

It remains only to see that the simulation does what it purports to do. Assume that $M'$ diverges when begun in some configuration $C$. Observe that each line in the code requires only finitely much time to execute. Therefore, the only way for $M'$ to diverge is to have either the while loop beginning at line (4) require infinitely many iterations, or to have some invocation of the while loop beginning at line (10) require infinitely many iterations. This later case can never occur, as the variable "inner-clock" is decremented by one each time through the loop, and so eventually reaches the value 0, causing termination of the loop.

□

## 3 The Main Result

## References

[col]

[cona]

[conb]

[Laga]

[lagb]

[mat]

[Min61] Marvin L. Minski. Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, November 1961.

[wir]