# FRIEDBERG-MUCHNIK THEOREM

PATRICK STEVENS, WITH TIP OF THE HAT TO DR THOMAS FORSTER

https://www.patrickstevens.co.uk/misc/FriedbergMuchnik/FriedbergMuchnik.pdf

## 1. Introduction

We consider Turing machines which query an oracle $O$: that is, they come equipped with an extra instruction "call the oracle with this input", where a call to the oracle is simply a test for membership in $O$.

We may supply different oracles to the same Turing machine, and potentially get different results: for example, the Turing machine which has as its only instruction "output the result of calling the oracle on my input" precisely mimics the oracle.

Recall that a set $A$ is *semi-decidable* if there is a Turing machine $T$ such that for all $n \in \mathbb{N}$, $T(n)$ halts iff $n \in A$.

**Theorem 1** (Friedberg-Muchnik theorem)**.** *There are semidecidable sets $A$ and $B$ such that for all Turing machines $T$ which may query an oracle, the following fail to be equivalent:*

(1) *$T$-querying-$B$ doesn't halt with output* 0
(2) *$T$-querying-$B$ has input in $A$*

*and the following fail to be equivalent:*

(1) *$T$-querying-$A$ doesn't halt with output* 0
(2) *$T$-querying-$A$ has input in $B$*

That is, we can find two semidecidable sets $A$ and $B$ such that neither allows a Turing machine to decide the other, where by "$T$ decides $A$" we mean "$T$ halts and outputs 0 iff its input is not in $A$, and it halts and outputs 1 iff its input is in $A$". (Equivalently, $T$ is a machine which computes the characteristic function of $A$.)

## 2. Proof

We can enumerate all the Turing machines which call an oracle; write $[n]^X$ for the $n$th Turing machine in the enumeration, calling oracle $X$.

What would it mean for the Friedberg-Muchnik theorem to hold? We would be able to find $e_n$ (resp. $f_n$) that witness in turn that the $n$th Turing machine doesn't manage to decide $A$ in the presence of $B$ (resp. $B$ in the presence of $A$).

That is, it would be enough to show that:

*Date*: 5th February 2016.

**Theorem 2.** *There are semidecidable sets $A, B$ such that for all $n \in \mathbb{N}$:*

- *there is $e_n \in \mathbb{N}$ such that*
  - *$e_n \in A$ but $[n]^B(e_n)$ halts with output $0$, or*
  - *$e_n \notin A$ but $[n]^B(e_n)$ fails to halt, or halts at something other than $0$*
- *there is $f_n \in \mathbb{N}$ such that*
  - *$f_n \in B$ but $[n]^A(f_n)$ halts with output $0$, or*
  - *$f_n \notin B$ but $[n]^A(f_n)$ fails to halt, or halts at something other than $0$*

The way we are going to do this is as follows. We'll construct our $A$ and $B$ iteratively, starting from the empty set and only ever adding things in to our current attempts.

For each $n \in \mathbb{N}$, we can make an infinite list of "possible" witnesses: numbers which might eventually be our choice for $e_n$. We don't care what these guesses are at the moment, but we just insist that they be disjoint and sorted into increasing order. Write

$$G_i = \{g_1^{(i)}, g_2^{(i)}, \dots\}$$

for the set of possibilities for $e_i$, and

$$H_i = \{h_1^{(i)}, h_2^{(i)}, \dots\}$$

for the set of possibilities for $f_i$. (I emphasise again that we don't assume any properties of these numbers, other than that $g_m^{(i)}$ and $h_m^{(i)}$ are increasing with $m$, and that no $g_m^{(i)}, g_n^{(j)}, h_p^{(k)}, h_q^{(l)}$ are equal.)

Now, at time-step 0, we have no information about what's going to be in $A$ and $B$, so let

$$A_0 = B_0 = \emptyset$$

Every $G_i$ and $H_i$ is looking for a witness among its members. We assign a priority order to them:

$$G_1 > H_1 > G_2 > H_2 > \dots$$

The idea is that the high-priority sets quickly decide on their witness, and the lower-priority sets get to choose their witness subject to not being allowed to mess up any higher-priority set's decision.

At time-step $t$, we've already built $A_{t-1}, B_{t-1}$ as our best guesses at $A$ and $B$. We seek an $e_i$ for any $G_i$ which hasn't got one (for $1 \leq i \leq t$), and then we can work on finding $f_i$ for the $H_i$ next.

Run the machines $[i]^{B_{t-1}}$ for $i = 1, 2, \dots, t$, for $t$ steps each, each on input $g_1^{(i)}$. This will approximate $[i]^B$, because $B_{t-1} \subseteq B = \cup_{t=1}^{\infty} B_t$, but it is by no means exactly what we need yet.

- If our machine $[i]^{B_{t-1}}$ ever attempts to query its oracle on a value greater than $\max(B_{t-1})$, we declare that the machine crashes for this round, and sits out. Indeed, $B_{t-1}$ is incomplete at this point as a reflection of $B$ - we only have information about it up to $\max(B_{t-1})$ - so it would be useless to try and infer information about $B$ from parts of $B_{t-1}$ which are even bigger than that maximum.

- If $[i]^{B_{t-1}}$ halts at something other than 0, then it's no use to us: it can't possibly be a witness we can add to $B$, because such witnesses $e_i$ must satisfy "$[i]^B(e_i)$ halts at 0". So $G_i$ will sit out of this round.
- If $[i]^{B_{t-1}}$ fails to halt in the allotted $t$ steps, it is likewise not something we can add to $B$, because we can't (yet) even prove that the machine *halts* on this input, let alone that it halts on 0. So $G_i$ will again sit out of this round.
- But if $[i]^{B_{t-1}}$ halts and outputs 0 in the allotted $t$ steps, we're in business: for some collection of $i$, we have found some things $(g_1^{(i)})$ that might serve as witnesses. Throw each of these into $B_{t-1}$ to make $B_t$.

OK. Now $G_i$ is happy, but remember we might have had a side-effect here, because if (for the sake of argument) $G_1$ had already decided on its witness during time-step $t-1$, it made that decision with reference to $B_{t-1}$ and not with reference to $B_t$. The fact that $g_1^{(i)}$ is now in our $B$-guess may alter the computation that $[1]^{B_{t-1}}$ performed to decide on its witness. (This is because $[1]^{B_{t-1}}(e_1)$ is not in general equal to $[1]^{B_t}(e_1)$.)

How can we ensure that in fact $G_1$'s witness isn't broken? Well, [1] is a finite machine which we have run for a finite time, so it can only have asked the oracle for values up to some finite number $\beta_1$ before it halted. So if we can make sure we only ever added $g_n^{(i)}$ to $B_{t-1}$ if it was above $\beta_1$, then we haven't actually changed $B$ from the point of view of [1]. Even after $B_{t-1}$ becomes $B_t$, the computation $[1]^{B_{t-1}}$ performs is exactly the same as the computation $[1]^{B_t}$ performs, because the oracles are the same on all points [1] might query.

Therefore, after adding something from $G_i$ to make $B_t$, we need to delete all numbers below $\beta_i$ from all lower-priority $G_j$ and $H_j$. (This is easy to do because of our stipulation that the $G_j$ be listed with elements in ascending order.) That way, no $G_j$ will never even consider any element that breaks a higher-priority $G_i$.

Once we've found the $G$-witnesses at time-step $t$, we can find the $H$-witnesses in exactly the same way; and finally, we move on to time-step $t+1$.

2.1. **Problem.** This procedure works pretty well, but there's a problem with it. You might like to meditate on this for a few minutes, because it's revealed in the next paragraph.

The problem is simply that while no lower-priority entry can break a higher-priority one, the reverse might happen! It might be that $G_1, G_2, G_3$ take ten steps of execution before halting, while $G_4$ halts after just one step and so decides on its witness immediately (that is, at time $t = 4$, as opposed to $G_1$'s $t = 10$). Subsequently, $G_1$ will decide on its witness, and the act of throwing its witness into $B$ might break $G_4$'s choice. Remember that $G_4$ only eliminated breaking-values from *lower*-priority $G_j$, not the higher-priority $G_1$. (Allowing it to eliminate breaking-values from higher-priority sets could cause the entire protocol to enter an infinite loop, with $G_1$ and $G_4$ each invalidating the results of the other on successive time-steps.)

However, this isn't actually too much of a problem. Since $G_1$ can only ever decide on its witness once (being the highest-priority), that means $H_1$ will only ever need to decide twice; $G_2$ only ever needs to decide at most four times (it could be first to pick its witness, then $H_1$ overrules it, then it picks again, then $G_1$ overrules it and $H_1$, then it

picks again, then $H_1$ overrules it, and finally it picks again). In general, the $i$th element of the priority order can only be overruled $2^i - 1$ times.

So if $G_i$ is overruled, we can just keep churning through the procedure, chucking more and more elements into $B$; $G_i$ can only be overruled finitely many times, and it has infinitely many elements in its list to play with, so eventually it will work its way into a position when it can never be overruled.

2.2. **Final problem.** OK, this works fine if every $G_i$ eventually finds a witness. But there's another case: $G_1$ may never find a witness. For example, $[1]^{B_t}(g_1^{(1)})$ may never halt, or it may halt but output the value 1 instead of 0 (so the protocol sees it as "uninteresting" and just repeatedly tells $G_1$ to sit out of the round).

But remember that we're trying to construct a witness that a certain equivalence fails, and so far we've been constructing witnesses that it fails in one of its directions. (Remember: the equivalence we want to fail is that $T^B$ halts with 0 iff it has input not in $A$.) We could still win by finding $e_1$ such that $[1]^B(e_1)$ fails to halt at 0 despite not being in $A$. And look! We've precisely got one of those elements, and it's $g_1^{(1)}$.

## 3. Summary

The output of this procedure is a pair of sets

$$A = \cup_{t=1}^{\infty} A_t, B = \cup_{t=1}^{\infty} B_t$$

which are semi-decidable (because we built them as a union of finite sets $A_t, B_t$). For each Turing machine $[i]^X$, we have a witness $e_n$, such that:

- either $[i]^B(e_n)$ halts with output 0 and $e_n \in A$, or
- $[i]^B(e_n)$ fails to halt, or halts with output not equal to 0, and $e_n \notin A$.

(This can be proved by induction: if $e_n$ is a witness at time $t$ and it is never overruled, then it remains a witness when we pass to $B$, because by construction its computation doesn't change on passing to $B$.)

That is, no Turing machine $[i]^B$ decides $A$.

Likewise, no Turing machine $[i]^A$ decides $B$.