Week 8 at a glance

Textbook reading: Chapter 4, Section 5.3
For Monday, "An undecidable language", Sipser pages 207-209.

For Wednesday, Definition 5.20 and figure 5.21 (page 236) of mapping reduction.

For Friday, Example 5.24 (page 236).

For Monday of Week 9: Example 5.26 (page 237)

We will be learning and practicing to:

- Clearly and unambiguously communicate computational ideas using appropriate formalism. Translate across levels of abstraction.
  - Give examples of sets that are regular, context-free, decidable, or recognizable (and prove that they are).
    - **Define and explain the definitions of the computational problem A_{TM}**
    - **Define and explain the definitions of the computational problem $HALT_{TM}$**
- Know, select and apply appropriate computing knowledge and problem-solving techniques. Reason about computation and systems.
  - Use diagonalization to prove that there are 'hard' languages relative to certain models of computation.
    - **Trace the argument that proves A_{TM} is undecidable and explain why it works.**
  - Use mapping reduction to deduce the complexity of a language by comparing to the complexity of another.
    - **Define computable functions, and use them to give mapping reductions between computational problems**
    - **Build and analyze mapping reductions between computational problems**
    - **Deduce the decidability or undecidability of a computational problem given mapping reductions between it and other computational problems, or explain when this is not possible.**
  - Classify the computational complexity of a set of strings by determining whether it is regular, context-free, decidable, or recognizable.
    - **State, prove, and use theorems relating decidability, recognizability, and co-recognizability.**
    - **Prove that a language is decidable or recognizable by defining and analyzing a Turing machines with appropriate properties.**

TODO:

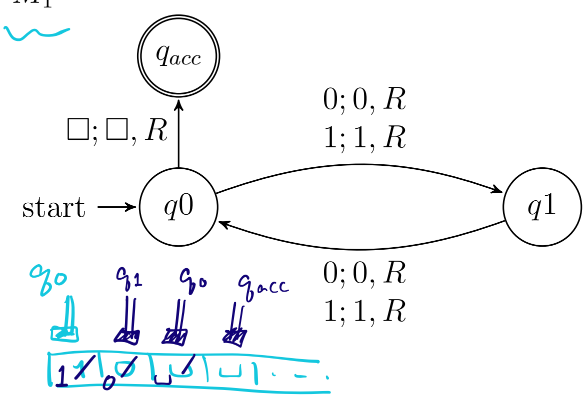
Homework 5 submitted via Gradescope (https://www.gradescope.com/), due Tuesday 11/19/2024

Review Quiz 8 on PrairieLearn (http://us.prairielearn.com), complete by Sunday 11/25/2024

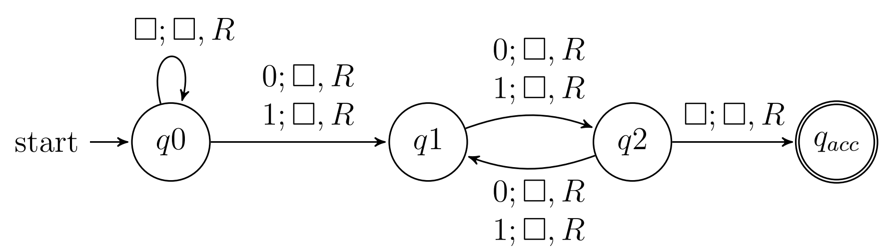
Monday: A_{TM} is recognizable but undecidable

Acceptance problem for Turing machines	A_{TM}	$\{\langle M, w\rangle \mid M \text{ is a Turing machine that accepts input string } w\}$
Language emptiness testing for Turing machines	E_{TM}	$\{\langle M\rangle \mid M \text{ is a Turing machine and } L(M) = \emptyset\}$
Language equality testing for Turing machines	EQ_{TM}	$\{\langle M_1, M_2\rangle \mid M_1 \text{ and } M_2 \text{ are Turing machines and } L(M_1) = L(M_2)\}$

M_1



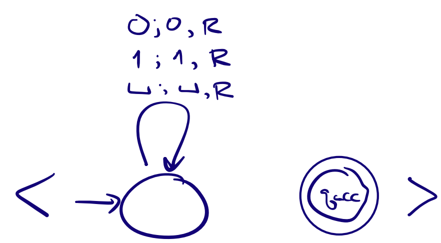
M_2



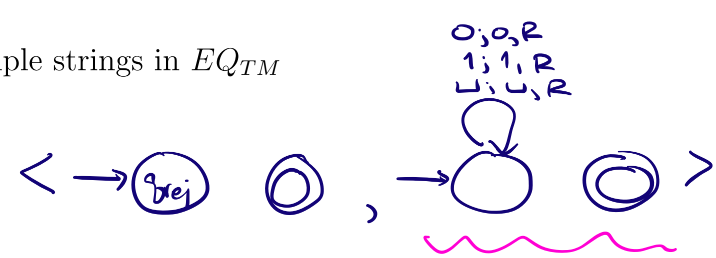
Example strings in A_{TM}

$\langle M_1, 10\rangle$ $\langle M_1, \varepsilon\rangle$ $\langle M_2, 01\rangle$

Example strings in E_{TM}



Example strings in EQ_{TM}



$\langle M_1, M_1\rangle$

Theorem: A_{TM} is Turing-recognizable.

Strategy: To prove this theorem, we need to define a Turing machine R_{ATM} such that $L(R_{ATM}) = A_{TM}$.

Define R_{ATM} = " On input x

high level description

0. Type check: if $x \neq \langle M, w \rangle$ for M any Turing machine and w a string then reject.
1. Let $x = \langle M, w \rangle$ where M is a TM, w string.
2. Simulate M on w.
3. If M accepts w, accept x.
4. If M rejects w, reject x. "

Proof of correctness:

WTS for all strings x, if $x \in A_{TM}$ then R_{ATM} accepts x and if $x \notin A_{TM}$ then R_{ATM} does not accept x.

Let x be arbitrary:

Case 1: $x \neq \langle M, w \rangle$ for any TM M or string w. Then $x \notin A_{TM}$ by definition of A_{TM}. Tracing R_{ATM} on x, in step 0, x fails typecheck so R_{ATM} reject x ✓.

Case 2: $x = \langle M, w \rangle$ for some TM M and string w

Case 2a: M accepts w

By definition of A_{TM}, $x \in A_{TM}$. WTS R_{ATM} accepts x. Tracing def of R_{ATM}, x passes type check in step 0 and then run M on w. By case assumption, computation halts and accepts so R_{ATM} accepts x ✓

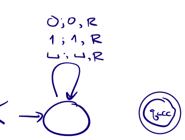
Case 2b: M rejects w

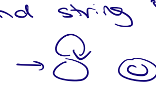
By definition of A_{TM}, $x \notin A_{TM}$. WTS R_{ATM} does not accept x. Tracing def of R_{ATM}, x passes type check in step 0 and then run M on w. By case assumption, computation halts and rejects so R_{ATM} rejects x. ✓

Case 2c: M loops on w

By definition of A_{TM}, $x \notin A_{TM}$. WTS R_{ATM} does not accept x. Tracing def of R_{ATM}, x passes type check in step 0 and then run M on w. By case assumption computation doesn't halt so R_{ATM} loops on x. ✓

Notice: R_{ATM} is not a decider, e.g. if M is a TM that loops on w.

x = <  , 0 > as input to R_{ATM}

- pass the type check in step 0
- parse x into TM and string in step 1
- in step 2 we'd run  on 0 scanning the tape left to right forever never halt!

We will show that A_{TM} is undecidable. *First, let's explore what that means.*

To prove that a computational problem is **decidable**, we find/ build a Turing machine that recognizes the language encoding the computational problem, and that is a decider.

How do we prove a specific problem is **not decidable**?

How would we even find such a computational problem?

Counting arguments for the existence of an undecidable language:

- The set of all Turing machines is countably infinite.
- Each recognizable language has at least one Turing machine that recognizes it (by definition), so there can be no more Turing-recognizable languages than there are Turing machines.
- Since there are infinitely many Turing-recognizable languages (think of the singleton sets), there are countably infinitely many Turing-recognizable languages.
- Such the set of Turing-decidable languages is an infinite subset of the set of Turing-recognizable languages, the set of Turing-decidable languages is also countably infinite.

Since there are uncountably many languages (because $\mathcal{P}(\Sigma^*)$ is uncountable), there are uncountably many unrecognizable languages and there are uncountably many undecidable languages.

Thus, there's at least one undecidable language!

What's a specific example of a language that is unrecognizable or undecidable?

To prove that a language is undecidable, we need to prove that there is no Turing machine that decides it.

Key idea: proof by contradiction relying on self-referential disagreement.

Theorem: A_{TM} is not Turing-decidable.

Proof: Suppose **towards a contradiction** that there is a Turing machine that decides A_{TM}. We call this presumed machine M_{ATM}.

witness

By assumption, for every Turing machine M and every string w

$\langle M,w\rangle \in A_{TM}$

- If $w \in L(M)$, then the computation of M_{ATM} on $\langle M, w\rangle$ halts and accepts.
- If $w \notin L(M)$, then the computation of M_{ATM} on $\langle M, w\rangle$ halts and rejects.

$\langle M,w\rangle \notin A_{TM}$

Define a **new** Turing machine using the high-level description:

implicit type check

$D =$" On input $\langle M\rangle$, where M is a Turing machine:

1. Run M_{ATM} on $\langle M, \langle M\rangle\rangle$. self-reference
2. If M_{ATM} accepts, reject; if M_{ATM} rejects, accept."

disagree

type check: finitely many steps

M_{ATM} guaranteed to halt in finitely many steps b/c decider

conditional on boolean: finitely many steps

Is D a Turing machine? Yes!

Is D a decider? Yes! (because M_{ATM} is)

What is the result of the computation of D on $\underbrace{\langle D \rangle}_{\text{string}}$?

Case ① $\overset{M}{D}$ accept $\overset{w}{\langle D \rangle}$

This means $\langle D, \langle D \rangle \rangle \in A_{TM}$

so M_{ATM} accepts $\langle \overset{M}{D}, \overset{w}{\langle D \rangle} \rangle$

Running D: step 2 D rejects $\langle D \rangle$

contradiction!

Case ② D rejects $\langle D \rangle$

This means $\langle D, \langle D \rangle \rangle \notin A_{TM}$

so M_{ATM} rejects $\langle \overset{M}{D}, \overset{w}{\langle D \rangle} \rangle$

Running D: step 2 D accepts $\langle D \rangle$

contradiction!

↯

Summarizing:

$A_{TM} = \{ \langle M, w \rangle \mid M \text{ TM}, w \text{ string}, w \in L(M) \}$

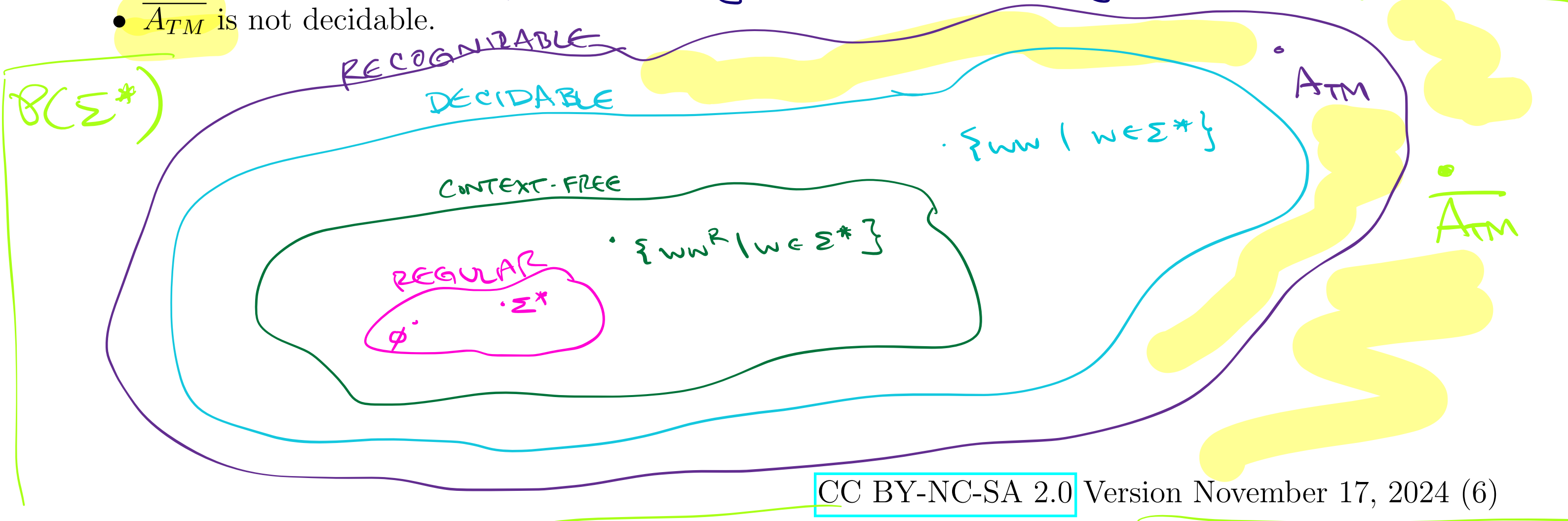
- A_{TM} is recognizable.
- A_{TM} is not decidable.

Recall definition: A language L over an alphabet Σ is called **co-recognizable** if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.

and Recall Theorem (Sipser Theorem 4.22): A language is Turing-decidable if and only if both it and its complement are Turing-recognizable.

Recall also:
Class of decidable languages is closed under complementation.
Class of undecidable languages is closed under complementation.

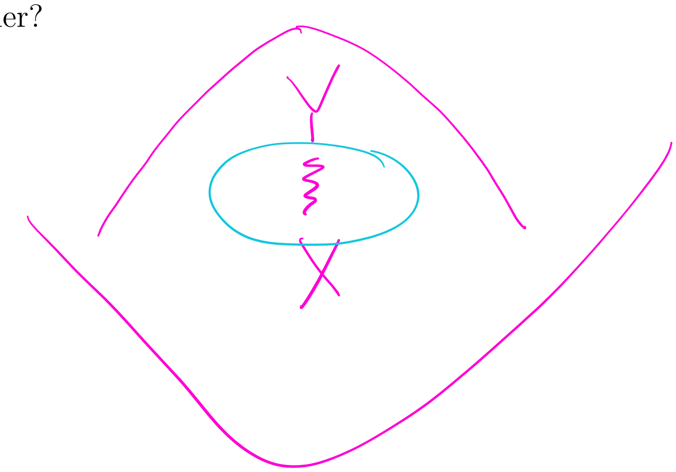
- A_{TM} is recognizable.
- A_{TM} is not decidable.
- $\overline{A_{TM}}$ is not recognizable. because if it were then A_{TM} would be co-recognizable and recognizable hence decidable
- $\overline{A_{TM}}$ is not decidable.



Wednesday: Computable functions and mapping reduction

Mapping reduction

Motivation: Proving that A_{TM} is undecidable was hard. How can we leverage that work? Can we relate the decidability / undecidability of one problem to another?



★

If problem X is **no harder than** problem Y

…and if Y is easy,

…then X must be easy too.

If problem X is **no harder than** problem Y

…and if X is hard,

…then Y must be hard too.

"Problem X is no harder than problem Y" means "Can answer questions about membership in X by converting them to questions about membership in Y".

For all languages A, B

Definition: A is **mapping reducible to** B means $\boxed{\text{there}}$ is a computable function $f : \Sigma^* \to \Sigma^*$ such that *for all* strings x in Σ^*,

well-defined

$$x \in A \qquad \text{if and only if} \qquad f(x) \in B.$$

Notation: when A is mapping reducible to B, we write $A \leq_m B$.

Intuition: $A \leq_m B$ means A is no harder than B, i.e. that the level of difficulty of A is less than or equal the level of difficulty of B.

To do:

1. Define what is means for a function to be computable?
2. Is mapping reduction enough to allow us to reason as in ★?

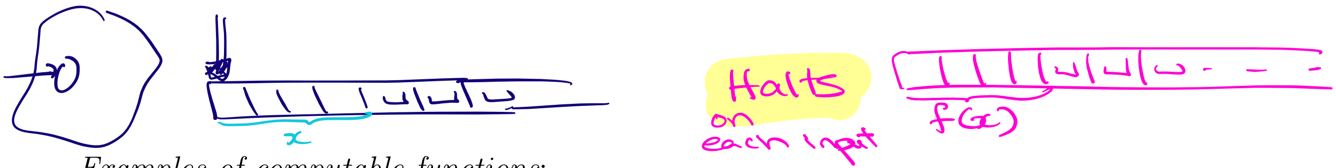
TODO

1. What is a computable function?
2. How do mapping reductions help establish the computational difficulty of languages?

Computable functions

by a Turing machine

Definition: A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** means there is some Turing machine such that, for each x, on input x the Turing machine halts with exactly $f(x)$ followed by all blanks on the tape



Examples of computable functions:

The function that maps a string to a string which is one character longer and whose value, when interpreted as a fixed-width binary representation of a nonnegative integer is twice the value of the input string (when interpreted as a fixed-width binary representation of a non-negative integer)

$$f_1 : \Sigma^* \to \Sigma^* \qquad f_1(x) = x0$$

To prove f_1 is computable function, we define a Turing machine computing it.

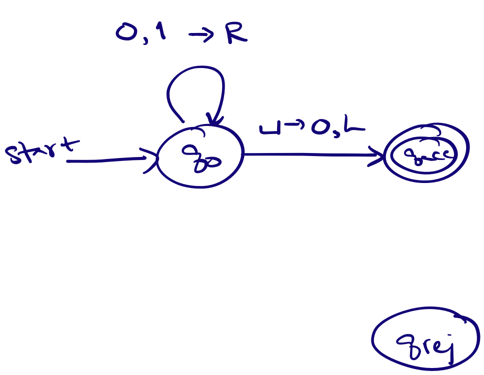
High-level description

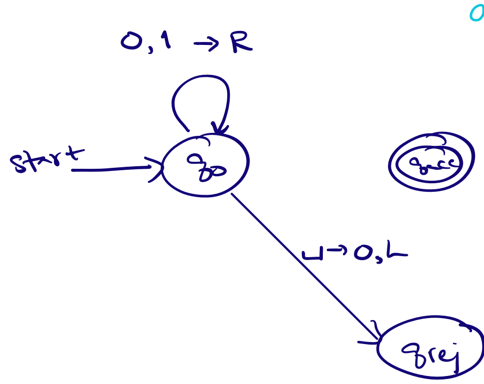
"On input w
1. Append 0 to w.
2. Halt."

Implementation-level description

"On input w
1. Sweep read-write head to the right until find first blank cell.
2. Write 0.
3. Halt."

Formal definition $(\{q0, qacc, qrej\}, \{0, 1\}, \{0, 1, \sqcup\}, \delta, q0, qacc, qrej)$ where δ is specified by the state diagram:





assume all missing arrows direct to qrej

The function that maps a string to the result of repeating the string twice.

$$f_2 : \Sigma^* \to \Sigma^* \qquad f_2(x) = xx$$

"On input x

1. Output xx"

extra practice

The function that maps strings that are not the codes of NFAs to the empty string and that maps strings that code NFAs to the code of a DFA that recognizes the language recognized by the NFA produced by the macro-state construction from Chapter 1.

extra practice

The function that maps strings that are not the codes of Turing machines to the empty string and that maps strings that code Turing machines to the code of the related Turing machine that acts like the Turing machine coded by the input, except that if this Turing machine coded by the input tries to reject, the new machine will go into a loop.

$$f_4 : \Sigma^* \to \Sigma^* \qquad f_4(x) = \begin{cases} \varepsilon & \text{if } x \text{ is not the code of a TM} \\ \langle (Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej}) \rangle & \text{if } x = \langle (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}) \rangle \end{cases}$$

one additional state in M_{new} input TM

where $q_{trap} \notin Q$ and

$$\delta'((q, x)) = \begin{cases} (r, y, d) & \text{if } q \in Q,\ x \in \Gamma,\ \delta((q, x)) = (r, y, d), \text{ and } r \neq q_{rej} \\ (q_{trap}, \sqcup, R) & \text{otherwise} \end{cases}$$

simulate input machine

except if told to go to q_{reject}, go to q_{trap} instead.

F = "On input x

Type check

1. If $x \neq <M>$ for any TM, output ε
2. If $x = <M>$ TM, build a new TM,

M_{new} = "On input w,

1. Simulate M on w
2. If M accepts w, accept.
3. If M rejects w, go to 4.
4. Go to 4."

3. Output $<M_{new}>$"

$A \leq_m B$

Definition: A is **mapping reducible to** B means there is a computable function $f : \Sigma^* \to \Sigma^*$ such that *for all* strings x in Σ^*,

$$x \in A \qquad \text{if and only if} \qquad f(x) \in B.$$

In which case we say the function f witnesses that $A \leq_m B$

Making intutition precise ...

B
$\vee^m$
A

Theorem (Sipser 5.22): If $A \leq_m B$ and B is decidable, then A is decidable.

Pf: Let A and B be languages
Assume $A \leq_m B$.
By def, there is f computable function
So that for each string $x \in \Sigma^*$, $x \in A$ iff $f(x) \in B$.

WTS if B is decidable so is A.

Assume B is decidable.
By definition, there is a TM M_B — decider; $L(M_B) = B$
WTS A is decidable.

Define M_A = " On input x.
1 Compute $f(x)$ — finitely many steps b/c f is computable
2. Run M_B on $f(x)$. — finitely many steps b/c M_B decider.
3. If accepts, accept; if rejects, reject "

To show M_A decides A, consider arbitrary string x
Case 1: Assume $x \in A$. By def of $f(x)$, $f(x) \in B$. Running M_A on x, step 1 takes finitely long to compute $f(x)$ and M_B on $f(x)$ halts and accepts so M_A accepts x in step 3 ✓

Case 2: Assume $x \notin A$. By def of $f(x)$, $f(x) \notin B$. Running M_A on x, step 1 takes finitely long to compute $f(x)$ and M_B on $f(x)$ halts and rejects so M_A rejects x in step 3 ✓

Theorem (Sipser 5.23): If $A \leq_m B$ and A is undecidable, then B is undecidable.

Pf: Let A and B be languages and assume (towards a contradiction that
① $A \leq_m B$ and ② A undecidable and ③ B decidable.
By Theorem 5.23, ① + ③ guarantee that
A is decidable, contradicting ②.

Friday: The Halting problem

Recall definition: A is **mapping reducible to** B means there is a computable function $f : \Sigma^* \to \Sigma^*$ such that *for all* strings x in Σ^*,

$$x \in A \qquad \text{if and only if} \qquad f(x) \in B.$$

Notation: when A is mapping reducible to B, we write $A \leq_m B$.

Intuition: $A \leq_m B$ means A is no harder than B, i.e. that the level of difficulty of A is less than or equal the level of difficulty of B.

Example: $A_{TM} \leq_m A_{TM}$

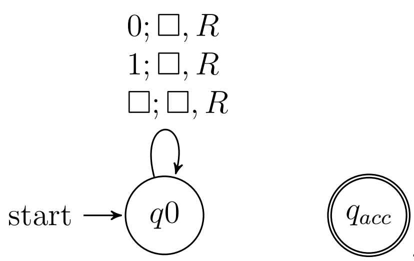
Example: $A_{DFA} \leq_m \{ww \mid w \in \{0,1\}^*\}$

Halting problem

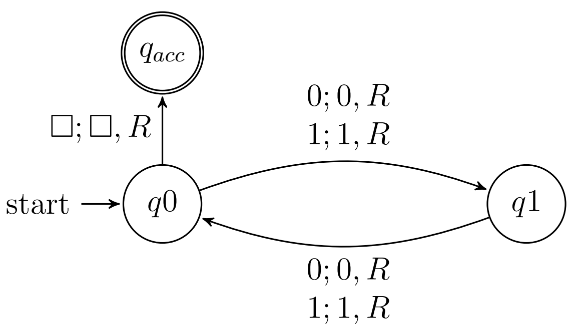
$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\}$$

Define $F : \Sigma^* \to \Sigma^*$ by

$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w\rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M'_x, w\rangle & \text{if } x = \langle M, w\rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet of } M. \end{cases}$$

where $const_{out} = \langle$ 

$, \varepsilon\rangle$ and M'_x is a Turing machine that computes like M except, if the computation of M ever were to go to a reject state, M'_x loops instead.

$F(\langle$ 

$, \varepsilon\rangle) =$

To use this function to prove that $A_{TM} \leq_m HALT_{TM}$, we need two claims:

Claim (1): F is computable

Claim (2): for every x, $x \in A_{TM}$ iff $F(x) \in HALT_{TM}$.