Monday

Friday: ATM is recognizable.

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

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

L(MATM) = ATM = {<M,w> | M TM, w string, w ∈ L(M)}

By assumption, for every Turing machine M and every string w and MATM always halts

- 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.

whether M halts on w or not!

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

D "diagonal"
self reference
disagreement
"devious"

type check.

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

1. Run M_{ATM} on $\langle M, \langle M\rangle\rangle$. subroutine — a string
2. If M_{ATM} accepts, reject; if M_{ATM} rejects, accept."

disagree.

Is D a Turing machine?

Yes! Given by high level description, using MATM as a subroutine.

Is D a decider? Yes.

For string x, running D on x means first type check, which takes finitely many steps. Then step 1, running one computation of MATM. This computation halts in finitely many steps b/c MATM is assumed to be a decider. Step 2 also takes only finitely many steps. So D guaranteed to halt in finitely many steps!

What is the result of the computation of D on $\langle D\rangle$?

TM — string.

Type check ✓
step 1: Run MATM on <D, <D>>

Case ① MATM accepts <D, <D>>

By assumption on MATM <D, <D>> ∈ ATM i.e.
D accepts <D>
But step 2 of D tells us to reject <D> when MATM accepts <D, <D>>
→←

Case ② MATM rejects <D, <D>>

By assumption on MATM, <D, <D>> ∉ ATM.
i.e. <D> ∉ L(D) i.e. D does not accept <D>.
But, step 2 of D, when MATM rejected <D, <D>>, D accepted <D>! →←.

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

Proof, first direction: Suppose language L is Turing-decidable. WTS that both it and its complement are Turing-recognizable.

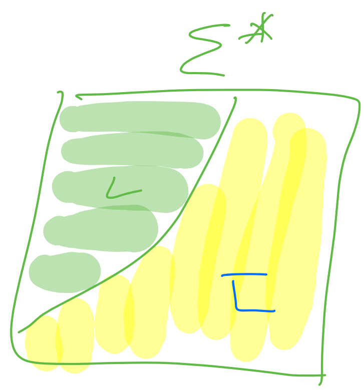
Why is L recognizable?
Use the same Turing machine that decides L to recognize it.

Why is $\overline{L}$ recognizable?
Let M_L be decider for L. Define
$M =$ "On input x
1. Run M_L on x. finite subroutine
2. If M_L accepts, reject; if M_L rejects, accept".

$L(M) = \overline{L}$ and M is decider (b/c M_L is).
M decides $\overline{L}$ and thus recognizes it ✓.

Proof, second direction: Suppose language L is Turing-recognizable, and so is its complement. WTS that L is Turing-decidable.

Given TM M_L that recognizes L.
Given TM M_C that recognizes $\overline{L}$.
Need to build a new TM that (1) is a decider and (2) recognizes L.



Define $D =$ "On input w
1. Run M_L on w and M_C on w, alternating one step at a time ("in parallel", saving configurations of computations)
2. If M_L halts and accepts, accept.
If M_L halts and rejects, reject.
If M_C halts and accepts, rejects.
If M_C halts and rejects, accept. "

Claim: D satisfies goals (1) and (2). Pf....

Give an example of a **decidable** set: language A_{DFA}, $\emptyset$, $\{x \in \Sigma^* \mid |x| \bmod 2 = 0\}$, $REP(L)$ where L is decidable.

Give an example of a **recognizable undecidable** set: language A_{TM}.

Give an example of an **unrecognizable** set: language $\overline{A_{TM}}$

Class of decidable languages is closed under complement.

If $\overline{A_{TM}}$ recognizable then Thm above says A_{TM} decidable

True or **False**: The class of Turing-decidable languages is closed under complementation?

But the class of Turing-recognizable languages is not closed under complement.

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.

Notation: The complement of a set X is denoted with a superscript c, X^c, or an overline, $\overline{X}$.

Review: Week 8 Monday

Recall: Review quizzes based on class material are assigned each day. These quizzes will help you track and confirm your understanding of the concepts and examples we work in class. Quizzes can be submitted on Gradescope as many times (with no penalty) as you like until the quiz deadline: the three quizzes each week are all due on Friday (with no penalty late submission open until Sunday).

Please complete the review quiz questions on Gradescope about undecidability.

Wednesday

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.

$w \in X$?

"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".

$f(w) \in Y$?

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^*,

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

well-defined

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.

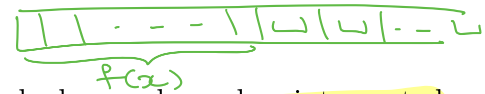
To do
1. What is a computable function?
2. Prove that mapping reductions help.

Turing machine M computes the function $f : \Sigma^* \to \Sigma^*$ means for every string $x \in \Sigma^*$, computation of M on x halts with $f(x)$ in leftmost cells of tape, rest of the tape is blank.

Computable functions

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)

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

multiply by 2 in binary!

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

High-level description

Examples:

$f_1(0) = 00$

$f_1(10) = 100$

$f_1(\varepsilon) = 0$

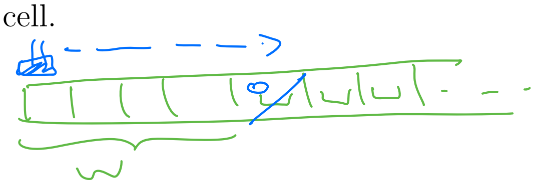
"On input w

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

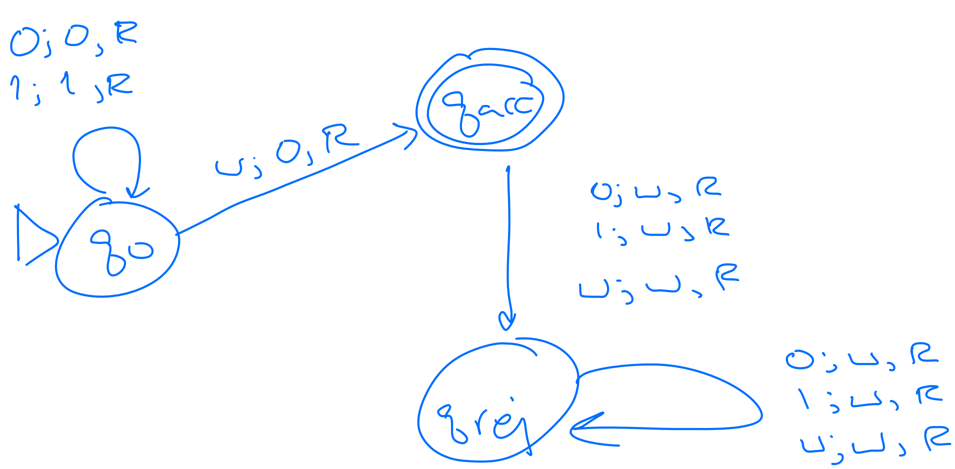
Implementation-level description

"On input w

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



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



Turing machine for identity function on $\Sigma^* = \{0,1\}^*$

implementation level

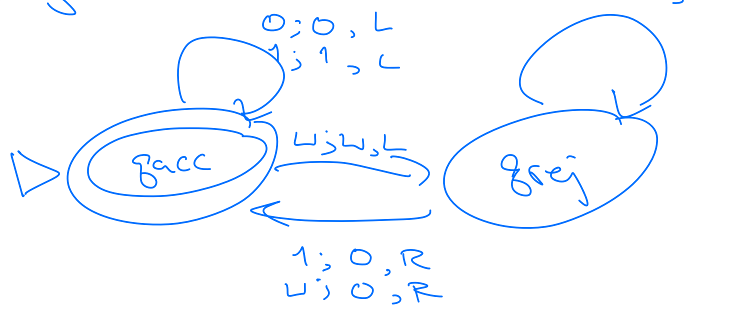
M_1 = "On input x

1. Scan R to first blank.
2. Halt"

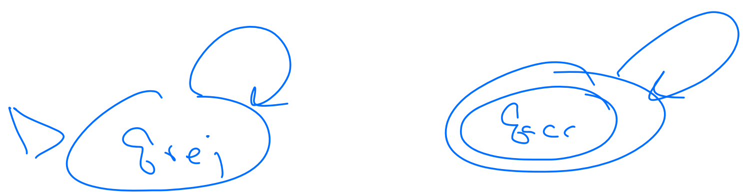
M_2 = "On input x

1. Halt"

State diagram for M_2:



OR



Note: computable functions must be well-defined (may or may not be 1-1, onto)

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

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

High level description of Turing machine computing f_2:

"On input x

1. output xx"

Bonus example: the function that maps a string that is not a code of a NFA to the empty string and maps codes of NFA to the code of the equivalent DFA is computed by the Turing machine

" On input x

1. if $x = \langle N \rangle$ for some NFA N
2. Use the algorithm from Ch 1 to define D_N, the DFA with $L(D_N) = L(N)$ using power set construction
3. Output $\langle D_N \rangle$.
4. Otherwise, output ε "

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.

adds a new state

new transition function

$$f_3 : \Sigma^* \to \Sigma^* \qquad f_3(x) = \begin{cases} \varepsilon & \text{if } x \text{ is not the code of a TM} \\ \underbrace{\langle (Q \cup \{q_{trap}\}, \Sigma, \Gamma, \delta', q_0, q_{acc}, q_{rej}) \rangle}_{\text{code of a TM}} & \text{if } x = \langle (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej}) \rangle \end{cases}$$

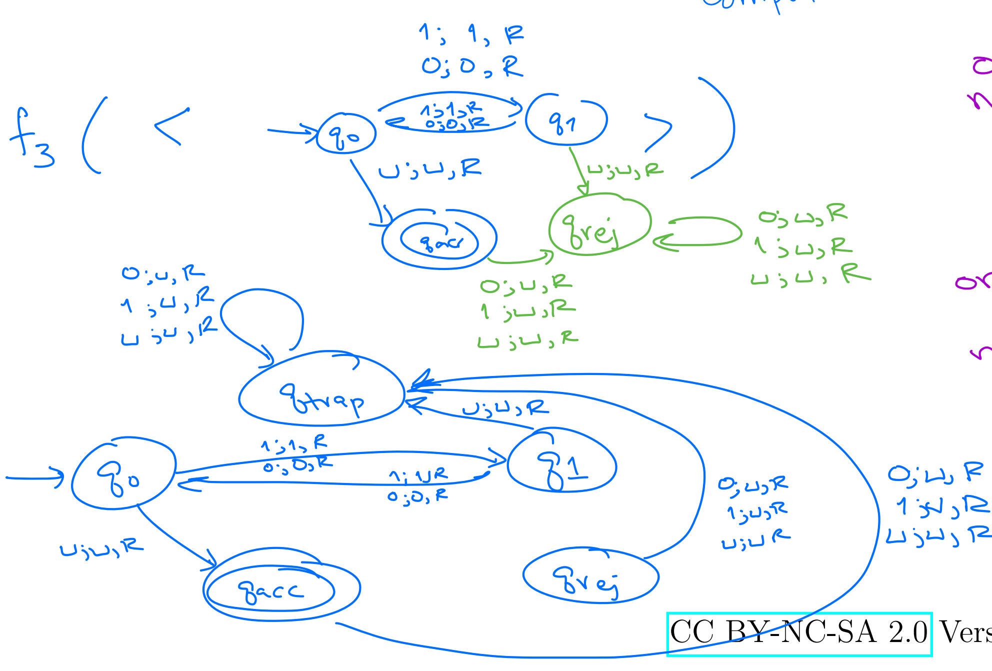
where $q_{trap} \notin Q$ and

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

Manipulate transition function

Compute f_3 (< >) = ε

Compute f_3 (<



>)

= <

original machine and new machine accept 00.

original machines rejects 0
new machine loops on 0

$L(\text{old machine}) = L(\text{new machine})$

The function that maps strings that are not the codes of CFGs to the empty string and that maps strings that code CFGs to the code of a PDA that recognizes the language generated by the CFG.

extra practice

Other examples?

Review: Week 8 Wednesday

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

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

Please complete the review quiz questions on Gradescope about mapping reductions.

Pre class reading for next time: Theorem 5.21 (page 236)

Friday

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: $A_{TM} \leq_m A_{TM}$ witnessing computable function is...

identity function works!

why? (1) computable, as seen with "On input x. & output x" a TM that always halts and outputs the image according to this function.

(2) translation? consider arbitrary string x

$x \in A_{TM}$ iff $x = f(x) \in A_{TM}$?

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

decidable decidable

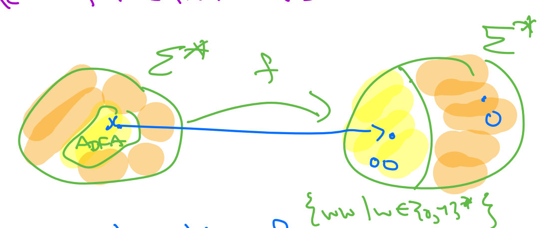
witnessing computable function is

$$f(x) = \begin{cases} 00 & \text{if } x \in A_{DFA} \\ 0 & \text{if } x \notin A_{DFA} \end{cases}$$

(1) "On input x

1. Check if $x \in A_{DFA}$ using decider from last week
2. If so, output 00
3. If not, output 0"

witnesses that f is computable



(2) By construction, for each x, if $x \in A_{DFA}$, $f(x) = 00 \in \{ww \mid w \in \{0,1\}^*\}$; if $x \notin A_{DFA}$ $f(x) = 0 \notin \{ww \mid w \in \{0,1\}^*\}$

Example: $\{0^i 1^j \mid i \geq 0, j \geq 0\} \leq_m A_{TM}$

extra practice

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

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

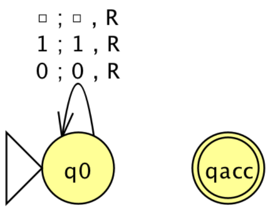
Halting problem

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

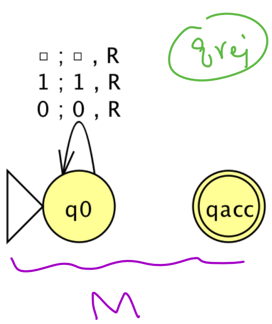
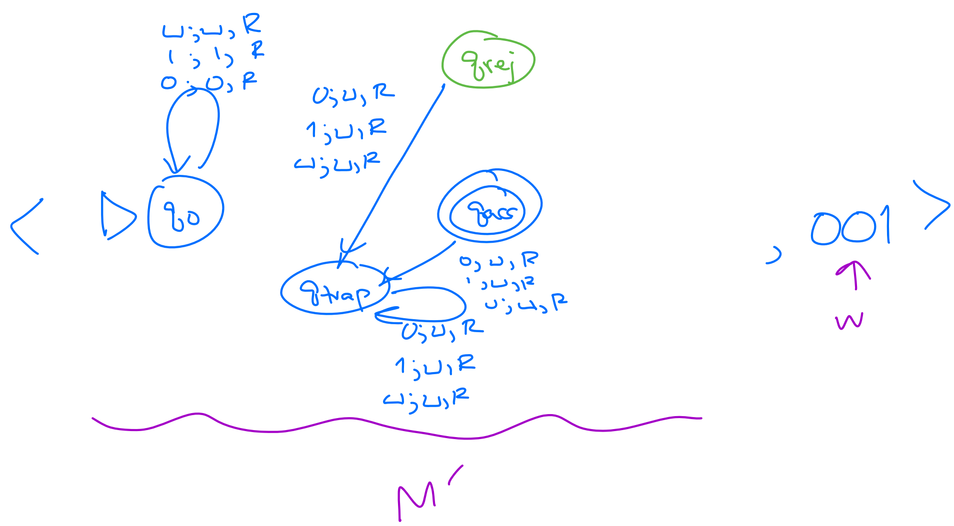
Define $F : \Sigma^* \to \Sigma^*$ by

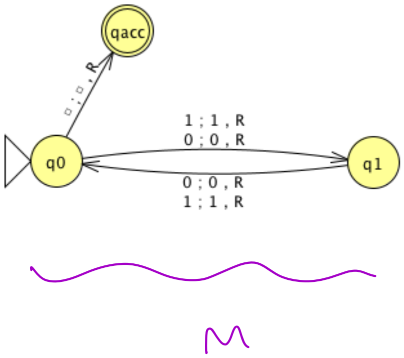
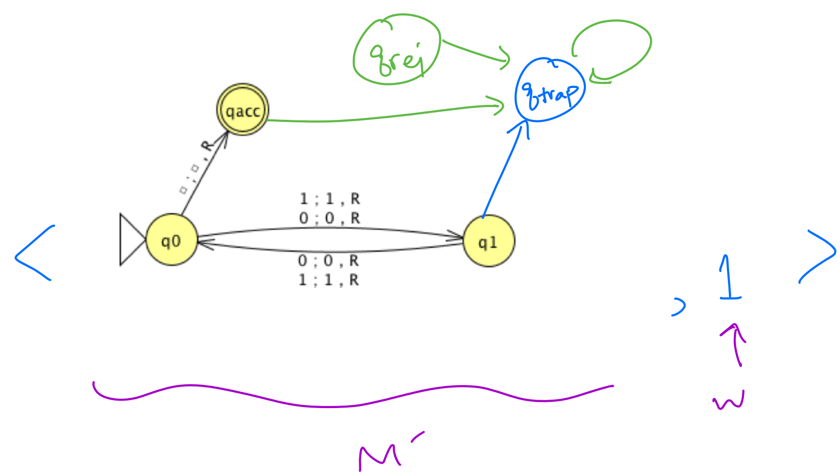
$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w\rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M', w\rangle & \text{if } x = \langle M, w\rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet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

input string

where $const_{out} = \langle$  $, \varepsilon\rangle$ and M' is a Turing machine that computes like M except, if the computation ever were to go to a reject state, M' loops instead.

(just like f_3 from before)

$F(\langle$  $, 001\rangle) =$ 

$F(\langle$  $, 1\rangle) =$ 

So $HALT_{TM}$ is undecidable!

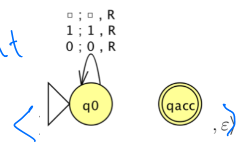
To use this function to prove that $A_{TM} \leq_m HALT_{TM}$, we need two claims:

Claim (1): F is computable

Pf: Need Turing machine that computes it

"On input x

1. Type check: if $x = \langle M, w \rangle$ where M TM, w string, then continue to step 2. If not output

$\langle$ 

$, \varepsilon \rangle$

2. Build M' (using f_3)
3. Output $\langle M', w \rangle$"

(guaranteed to halt and output F).

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

Pf Idea: Case ① $x \in A_{TM}$ ... WTS $F(x) \in HALT_{TM}$

Case ② $x \notin A_{TM}$ ... WTS $F(x) \notin HALT_{TM}$.

Pf: Consider $x \in \Sigma^*$.

Case ① Suppose $x \in A_{TM}$. Then $x = \langle M, w \rangle$ for some Turing machine M and string and M accepts w. By definition $F(x) = \langle M', w \rangle$ where M' accepts all (and only) strings that M does. Thus M' halts on w so $F(x) \in HALT_{TM}$, as required.

Case ② Suppose $x \notin A_{TM}$. There are three subcases:

Case (2a) $x \neq \langle M, w \rangle$ for any TM M, string w

Case (2b) $x = \langle M, w \rangle$ M rejects w.

Case (2c) $x = \langle M, w \rangle$ M loops on w.

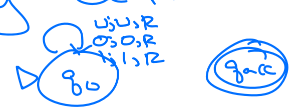
CONTINUED BELOW

Review: Week 8 Friday

Please complete the review quiz questions on Gradescope about the relationship between A_{TM} and $HALT_{TM}$

Pre class reading for next time: Example 5.30.

Case 2a Assume $x \neq \langle M, w \rangle$ for any Turing machine M or string w. By definition $F(x) = const_{out}$ where $const_{out} = \langle$



$, \varepsilon \rangle$
The TM encoded in the first component is a TM that loops on each input string so $F(x) \notin HALT_{TM}$, as required.

Case 2b Assume $x = \langle M, w \rangle$ for some Turing machine M and string w where M rejects w. By definition $F(x) = \langle M', w \rangle$ where M' is the TM that simulates M but loops when M would reject. Since M rejects w, M' loops on w so $F(x) \notin HALT_{TM}$.

Case 2c Assume $x = \langle M, w \rangle$ for some Turing machine M and string w where M loops on w. By definition $F(x) = \langle M', w \rangle$ where M' is the TM that simulates M but loops when M would reject. Since M loops on w, M' loops on w too. So $F(x) \notin HALT_{TM}$.

In each case, $x \in A_{TM}$ iff $F(x) \in HALT_{TM}$ ∎

