CS 121 Section 7:
Universal TMs, Uncomputability

Sahana Srinivasan

Today's Topics

1. Universality
2. Existence of an Uncomputable Function
3. Uncomputability of HALT
4. Reductions
5. Example Uncomputability Proof

Universality (Circuits)

1. Applied to Boolean circuits and straight-line programs
2. Circuit to evaluate all other circuits with a caveat*
3. *Needed a universal circuit of >s gates to evaluate a circuit of s gates

Universality (TMs)

1. Applies to TMs and their equivalents (e.g. NAND-TM)
2. Turing machine U to evaluate all other Turing machines
3. Can evaluate Turing Machines that are more complex than U using U

Formal Definition of Universal TM

Theorem 9.1: Universal Turing Machine

There exists a Turing machine U such that on every string M which represents a Turing machine and $x \in \{0,1\}^*, U(M,x) = M(x)$

Alternatively, U outputs what TM M would on input x, given a first argument M and second argument x. If the M on input *x* does not output anything (does not halt), neither will *U(M, x)*.

Proved by construction.

Existence of Uncomputable Functions

Uncomputable: **function** that cannot be computed by any **Turing machine.**

- Extra important now to differentiate functions vs. programs/Turing machines

Theorem 9.5: Uncomputable Functions
There exists a function $F^* : \{0,1\}^* \to \{0,1\}$ that is not computable by any Turing machine.

Proof Idea: Cantor's proof that the reals are uncountable.

Prove that a function F^* is uncomputable

Define $G(x)$

- If x represents a valid TM M and $M(x)$ halts, $G(x)$ = first bit of $M(x)$
- In any other case of x, $G(x) = 1$

$F^* = 1 - G(x)$

Proof Idea: contradiction. If F* were computable, there is a M that computes it and halts on all inputs. Let x be the string representation of M. $G(x)$ = the first bit of $M(x)$ by construction. So, $F^*(x) = 1$ - first bit of $M(x)$, again by construction. But we just said M computes F^*. How can their outputs differ in the first bit on input x? Contradiction.

What is HALT?

***HALT(M, x)* = 1** if Turing machine *M* halts on the input *x*

***HALT(M, x)* = 0** otherwise.

(You might object that you can compute HALT(*M*, *x*) by simulating *M*. But, if you tried to write such a program P, after what point could P definitively say *M* will never finish? (Never.))

Uncomputability of *HALT*

Proof Idea: Use the fact that F^* is uncomputable.

1. We know F^* is uncomputable.
2. Assume, toward a contradiction, that *HALT* is computable.
3. **Show that the program that computes *HALT*, which exists since we assumed *HALT* is computable, enables us to compute F^*.**
4. But #1 says F^* is uncomputable. So we have a contradiction :(
5. Our assumption in #2 therefore must be wrong. *HALT* is uncomputable.

Step 3 in Detail

Key Idea: If program P that computed *HALT* existed, it would help us build a program P* that computes *F**.

$F^* : \{0,1\}^* \rightarrow \{0,1\}$

- If x represents a valid TM *M* and *M(x)* halts, *F*(x)* = 1- first bit of *M(x)*
- In any other case of x, *F*(x) = 0*

$HALT : \{0,1\}^* \rightarrow \{0,1\}$

- *1 iff M* halts on input *x*

Input: $x \in \{0,1\}^*$
Output: $F^*(x)$
$PHALT$ is the program that computes $HALT$
U is the universal TM

```
def PFSTAR(x):
    if (PHALT(x, x) = 0):
        return 0
    if (U(x, x) = 0):
        return 1
    return 0
```

Proving Uncomputability via Reduction

Main Idea: Use the fact that function A is uncomputable to prove B is uncomputable.

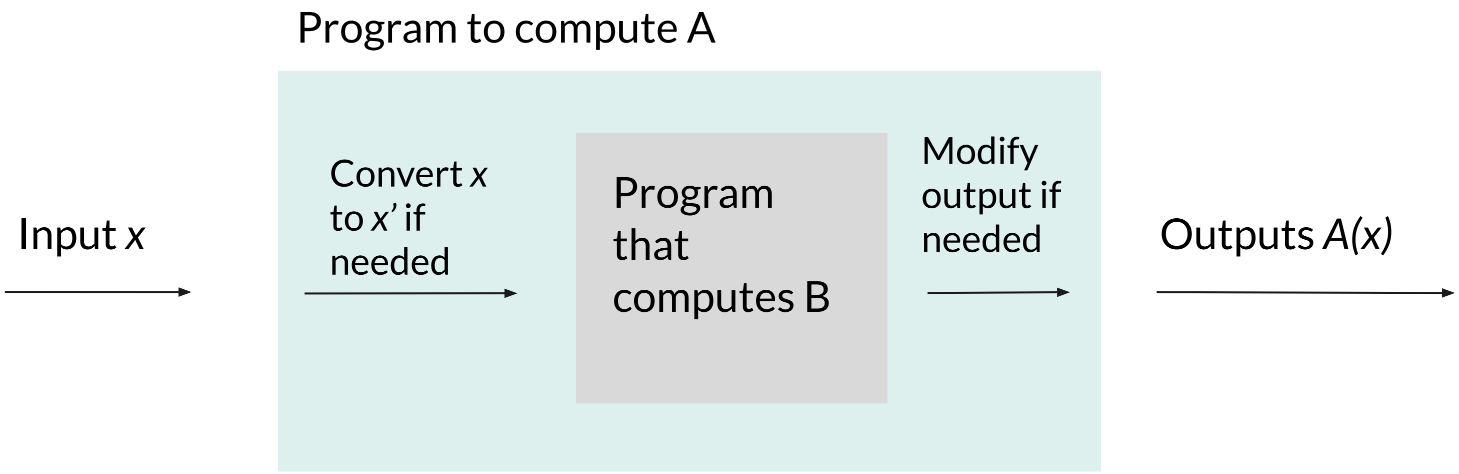
The "reduction": If we could compute B, we can compute A too, since the task of computing A *reduces* to just the task of computing B."

Reductions

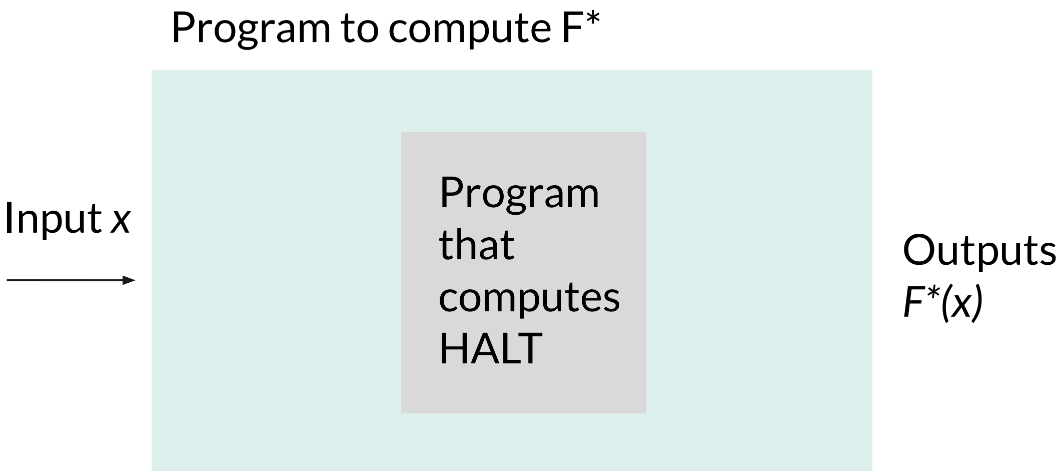
A = function we know is uncomputable. *B* = function we want to prove is uncomputable.

1. We know *A* is uncomputable.
2. Assume, toward a contradiction, that *B* is computable.
3. **Show that the program that computes *B*, which exists since we assumed *B* is computable, enables us to compute *A*. [REDUCTION STEP]**
4. But #1 says *A* is uncomputable. So we have a contradiction :(
5. Our assumption in #2 therefore must be wrong. *B* is uncomputable.

Reductions (Generally)



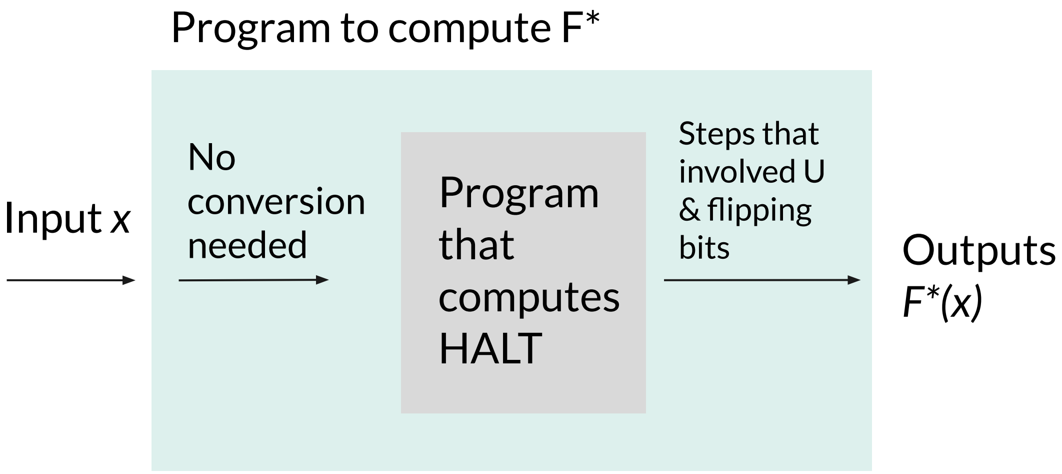
Reduction (HALT uncomputability)



Input: $x \in \{0,1\}^*$
Output: $F^*(x)$
$PHALT$ is the program that computes $HALT$
U is the universal TM

```
def PFSTAR(x):
    if (PHALT(x, x) = 0):
        return 0
    if (U(x, x) = 0):
        return 1
    return 0
```

Reduction (HALT uncomputability)



Input: $x \in \{0,1\}^*$
Output: $F^*(x)$
$PHALT$ is the program that computes $HALT$
U is the universal TM

```
def PFSTAR(x):
    if (PHALT(x, x) = 0):
        return 0
    if (U(x, x) = 0):
        return 1
    return 0
```

Reduction from

A to B

We can write a program to compute this function if we can write a program to compute this function.

This is the function that we know is uncomputable, and this is the function we're proving is uncomputable.

We want to know how to compute/solve this, and we know how to compute/solve this.

We reduce the problem of computing this function to the problem of computing this function.

Example 1 F* HALT

Example 2 HALT HALTONZERO

Example: $HALTONZERO : \{0,1\}^* \to \{0,1\}$

- *HOZ(M) = 1* iff Turing machine represented by *M* halts on input 0

How can we prove uncomputability?

- Intuition: feels similar to *HALT*
- Use *HALT*'s uncomputability to prove *HOZ* is uncomputable

HALT

- Takes in *M* and *x*
- 1 iff M(x) halts

HOZ

- Takes in just *M*
- 1 iff M(0) halts

Example Uncomputability Proofs

Input: TM M, input x
Output: $HALT(M, x)$
$PHOZ$ is the program that computes HOZ

```
def PHALT(M, x):
    M' = description of the TM that takes an input, ignores it, and runs EVAL(M, x)

    return PHOZ(M')
```

Reduction (HOZ uncomputability)

Program
that
computes
HOZ
P(M')

Reduction (HOZ uncomputability)

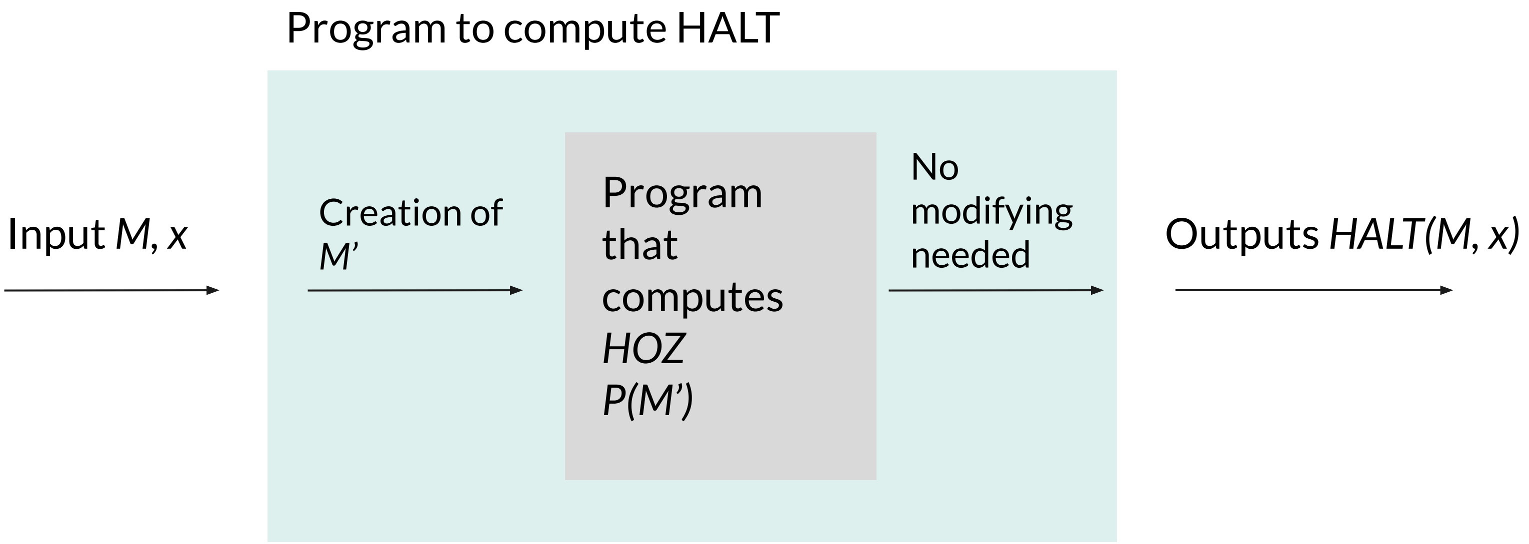
Program to compute HALT

Input *M*, *x*

Program that computes *HOZ P(M')*

Outputs *HALT(M, x)*

Reduction (HOZ uncomputability)



Uncomputability of *HOZ*

Proof Idea: Use the fact that *HALT* is uncomputable.

1. We know *HALT* is uncomputable.
2. Assume, toward a contradiction, that *HOZ* is computable.
3. **Show that the program that computes *HOZ*, which exists since we assumed *HOZ* is computable, enables us to compute *HALT*.**
4. But #1 says *HALT* is uncomputable. So we have a contradiction :(
5. Our assumption in #2 therefore must be wrong. *HOZ* is uncomputable.

Section Problems!

Let $TIMEDHALT : \{0,1\}^* \to \{0,1\}$ be the function that on input (a string representing) a triple (M, x, t), $TIMEDHALT(M, x, t) = 1$ iff the Turing machine M, on input x, halts within at most t steps (where a step is defined as one sequence of reading a symbol from the tape, updating the state, and writing a new symbol and (potentially) moving the head.)

Prove that $TIMEDHALT$ is computable.

Section Problems!

Let $\mathit{IS\text{-}TM\text{-}ONE}$: $\{0,1\}^* \to \{0,1\}$ be the function that takes as input a string representation of a Turing machine M and outputs 1 iff $M(x) = 1$ for every $x \in \{0,1\}^*$. Prove or disprove: $\mathit{IS\text{-}TM\text{-}ONE}$ is computable.

Section Problems!

Prove that the following function is uncomputable:

$$COMPUTES-PARITY(P) = \begin{cases} 1 & \text{P computes the parity function*} \\ 0 & \text{otherwise} \end{cases}$$

*Parity(x) = 1 iff x has an odd number of 1s.