# CS420

## Introduction to the Theory of Computation

### Lecture 22: Undecidability

Tiago Cogumbreiro

# Today we will learn...

- Turing Machine theory in Coq
- Undecidability
- Unrecognizability

Section 4.2

# Turing Machine theory in Coq

# Turing Machine theory in Coq

- **What?** I am implementing the Sipser book in Coq.
- **Why?**
  - So that we can dive into any proof at any level of detail.
  - So that you can inspect any proof and step through it on your own.
  - So that you can ask why and immediately have the answer.

Do you want to help out?

# Why is proving important to CS?

- **Generality is important.**
  Whenever we implement a program, we are implicitly proving some notion of correctness in our minds (the program is the proof).
- **Rigour is important.**
  The importance of having precise definitions. Fight ambiguity!
- **Assume nothing and question everything.**
  In formal proofs, we are pushed to ask why? And we have a framework to understand why.
- **Models are important.**
  The basis of formal work is abstraction (or models), e.g., Turing machines as models of computers; REGEX vs DFAs vs NFAs.

What follows is a description of our Coq implementation

# Turing Machine Theory in Coq

## Unspecified input/machines

For the remainder of this module we leave the input (string) and a Turing Machine unspecified.

```
Variable input: Type.
Variable machine: Type.
```

# Turing Machine Theory in Coq

## Unspecified input/machines

For the remainder of this module we leave the input (string) and a Turing Machine unspecified.

```
Variable input: Type.
Variable machine: Type.
```

## Running a TM

We can run any Turing Machine given an input and know whether or not it accepts, rejects, or loops on a given input. We leave running a Turing Machine unspecified.

```
Inductive result := Accept | Reject | Loop.

Variable run: machine → input → result.
```

# What is a language?

A language is a predicate: a formula parameterized on the input.

```
Definition lang := input → Prop.
```

## Defining a set/language

### Set builder notation

$$L = \{x \mid P(x)\}$$

### Functional encoding

$$L(x) \stackrel{\texttt{def}}{=} P(x)$$

## Defining membership

### Set membership

$$x \in L$$

### Functional encoding

$$L(x)$$

# Example

## Set builder example

$$L = \{a^n b^n \mid n \geq 0\}$$

## Functional encoding

$$L(x) \stackrel{\texttt{def}}{=} \exists n, x = a^n b^n$$

# The language of a TM

## Set builder notation

The language of a TM can be defined as:

$$L(M) = \{w \mid M \text{ accepts } w\}$$

## Functional encoding

$$L_M(w) \stackrel{\texttt{def}}{=} M \text{ accepts } w$$

### In Coq

```
Definition Lang (m:machine) : lang := fun w ⇒ run m w = Accept.
```

# Recognizes

We give a formal definition of recognizing a language. We say that $M$ recognizes $L$ if, and only if, $M$ accepts $w$ whenever $w \in L$.

```
Definition Recognizes (m:machine) (L:lang) := forall w, run m w = Accept <-> L w.
```

## Examples

- Saying $M$ recognizes $L = \{a^n b^n \mid n \geq 0\}$ is showing that there exist a proof that shows that all inputs in language $L$ are accepted by $M$ and vice-versa.
- Trivially, $M$ recognizes $L(M)$.

# We will prove 4 theorems

- Theorem 4.11 $A_{TM}$ is undecidable
- Theorem 4.22 $L$ is decidable if, and only if, $L$ is recognizable ***and*** co-recognizable
- Corollary 4.23 $\overline{A}_{TM}$ is unrecognizable
- Corollary 4.18 Some languages are unrecognizable

## Why?

- We will learn that we cannot write a program that decides if a TM accepts a string
- We can define decidability in terms of recognizability+complement
- There are languages that cannot be recognized by some program

# Theorem 4.11

## $A_{TM}$ is undecidable

# Theorem 4.11

## Functional view of $A_{TM}$

```
def A_TM(M, w):
  return M accepts w
```

## Theorem 4.11: $A_{TM}$ is undecidable

Show that `A_TM` loops for **some** input.

**Proof idea:** Given a Turing machine

```
def negator(w):     # w = <M>
  M = decode_machine w
  b = A_TM(M, w) # Decider D checks if M accepts <M>
  return not b      # Return the opposite
```

Given tht `A_TM` does not terminate, what is the result of `negator(negator)`?

# Theorem 4.11

## $A_{TM}$ is undecidable

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

```
Lemma no_decides_a_tm: ~ exists m, Decides m A_tm.
```

1. Proof follows by contradiction.
2. Let $D$ be the decider of $A_{TM}$
3. Consider the negator machine:

```
def negator(w):      # w = <M>
  M = decode_machine w
  b = call D <M, w> # Same as: A_TM(M, <M>)
  return not b      # Return the opposite
```

```
# If we expand D and
# ignore decoding we get:
def negator(f):
  return not f(f)
```

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # A_TM(M, <M>)?
    return not b      # Return the opposite
```

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$

4. Let negator be $N$. Case analysis on the result of running N with $\langle N\rangle$ **reach contradiction**.
5. Case $N$ accepts $\langle N\rangle$, or negator(negator).

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # A_TM(M, <M>)?
    return not b      # Return the opposite
```

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$

4. Let negator be $N$. Case analysis on the result of running N with $\langle N\rangle$ **reach contradiction**.
5. Case $N$ accepts $\langle N\rangle$, or `negator(negator)`.
    1. If $N$ accepts $\langle N\rangle$, then $D$ rejects $\langle N, \langle N\rangle\rangle$
    2. By the definition of $D$ (via $A_{TM}$), then $N$ rejects $\langle N\rangle$. **Contradiction!**

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # A_TM(M, <M>)?
    return not b      # Return the opposite
```

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$

4. Let negator be $N$. Case analysis on the result of running N with $\langle N\rangle$ **reach contradiction**.
5. Case $N$ accepts $\langle N\rangle$, or `negator(negator)`.
   1. If $N$ accepts $\langle N\rangle$, then $D$ rejects $\langle N, \langle N\rangle\rangle$
   2. By the definition of $D$ (via $A_{TM}$), then $N$ rejects $\langle N\rangle$. **Contradiction!**
6. Case $N$ rejects $\langle N\rangle$.

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # A_TM(M, <M>)?
    return not b      # Return the opposite
```

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$

4. Let `negator` be $N$. Case analysis on the result of running N with $\langle N\rangle$ **reach contradiction**.
5. Case $N$ accepts $\langle N\rangle$, or `negator(negator)`.
   1. If $N$ accepts $\langle N\rangle$, then $D$ rejects $\langle N, \langle N\rangle\rangle$
   2. By the definition of $D$ (via $A_{TM}$), then $N$ rejects $\langle N\rangle$. **Contradiction!**
6. Case $N$ rejects $\langle N\rangle$.
   1. If $N$ rejects $\langle N\rangle$, then $D$ accepts $\langle N, \langle N\rangle\rangle$
   2. Thus, by definition of $D$ (via $A_{TM}$), then $N$ accepts $\langle N\rangle$. **Contradiction!**

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # M accepts <M>?
    return not b      # Return the opposite
```

$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$

7. Case $N$ loops $\langle N\rangle$.

# Theorem 4.11: $A_{TM}$ is undecidable

```
def negator(w):
    M = decode_machine w
    b = call D <M, w> # M accepts <M>?
    return not b      # Return the opposite
```

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM that accepts } w\}$$

7. Case $N$ loops $\langle N\rangle$.
    1. If $N$ loops $\langle N\rangle$, then $D$ accepts $\langle N, \langle N\rangle\rangle$
    2. Thus, by definition of $D$ (via $A_{TM}$), then $N$ accepts $\langle N\rangle$. **Contradiction!**

# Understanding the Coq formalism

## Pseudo-code as a mini-language

1. `Call M w`
   Use the Universal Turing machine to call a machine $M$ with input $w$,
   Returns whatever $M$ returns by processing $w$
2. `mlet x ← P1 in P2`
   Runs pseudo-program `P1`; if `P1` halts, passes a boolean with the result of acceptance to `P2`. If `P1` loops, then the whole pseudo-program loops.
3. `Ret r`
   A Turing Machine that returns whatever is in r.
   ***Abbreviations:*** `Ret Accept = ACCEPT`, `Ret Reject = REJECT`, and `Ret Loop = LOOP`.

> This language is enough to prove the results in Section 4.2.

# The negator

## In Python

```
def negator(w):
  M = decode_machine w
  b = call D <M, w> # M accepts <M>?
  return not b      # Return the opposite
```

## In Coq

```
Definition negator D w :=
  let M := decode_machine w in
  mlet b ← Call D « M, w» in
  halt_with (negb b).
```

- `D` is a parameter of a Turing machine, given $\langle M, w\rangle$ decides if $M$ accepts $w$
- `w` is a serialized Turing machine $\langle M\rangle$
- `«M, w»` is the serialized pair `M` and `w`
- `b` takes the result of calling `D` with `«M,w»`
- halt the machine with negation of `b`

# Theorem 4.22

$L$ decidable iff $L$ is recognizable + co-recognizable

# Theorem 4.22

$L$ decidable iff $L$ recognizable and $L$ co-recognizable

> Recall that $L$ co-recognizable is $\overline{L}$.

## Complement

$$\overline{L} = \{w \mid w \notin L\}$$

Or, $\overline{L} = \Sigma^\star - L$

# Theorem 4.22

## $L$ decidable iff $L$ recognizable and $L$ co-recognizable

**Proof.** We can divide the above theorem in the following three results.

1. If $L$ decidable, then $L$ is recognizable.
2. If $L$ decidable, then $L$ is co-recognizable.
3. If $L$ recognizable and $L$ co-recognizable, then $L$ decidable.

# Part 1. If $L$ decidable, then $L$ is recognizable.

**Proof.**

# Part 1. If $L$ decidable, then $L$ is recognizable.

**Proof.**

Unpacking the definition that $L$ is decidable, we get that $L$ is recognizable by some Turing machine $M$ and $M$ is a decider. Thus, we apply the assumption that $L$ is recognizable.

# Part 2: If $L$ decidable, then $L$ is co-recognizable.

**Proof.**

# Part 2: If $L$ decidable, then $L$ is co-recognizable.

**Proof.**

1. We must show that if $L$ is decidable, then $\overline{L}$ is decidable. †
2. Since $\overline{L}$ is decidable, then $\overline{L}$ is recognizable.

---

†: Why? We prove in the next lesson.