Introduction to the Theory of Computation

Lecture 23: $A_{TM}$ is undecidable

Tiago Cogumbreiro
Theorem 4.11

$A_{TM}$ is undecidable
Proof idea

1. Assume solving $A_{TM}$ is decidable and reach a contradiction.
2. Find a program for which it is impossible to decide

```python
def tricky(f):
    return not f(f)

print(tricky(lambda x: True))  # Output?
```
Proof idea

1. Assume solving $A_{TM}$ is decidable and reach a contradiction.
2. Find a program for which it is impossible to decide

```python
def tricky(f):
    return not f(f)

print(tricky(lambda x: True)) # Output?
# False
try:
    print(tricky(tricky)) # Output?
except RecursionError:
    print("could not run: tricky(tricky)")
```
Proof idea

1. Assume solving $A_{TM}$ is decidable and reach a contradiction.
2. Find a program for which it is impossible to decide

```python
def tricky(f):
    return not f(f)

print(tricky(lambda x: True))  # Output?

# False
try:
    print(tricky(tricky))  # Output?
except RecursionError:
    print("could not run: tricky(tricky)")
```

Calling `tricky(tricky)` loops forever.
Proof idea

Let the solver of $A_{TM}$ be `returns_true` which takes a boolean function $f$, an argument $a$, and returns whether $f(a)$ would return true. Function `returns_true` halts for every input.

```python
def tricky_v2(f):
    return not returns_true(f, f)
```

1. What would the result of `tricky_v2(tricky_v2)` be?
Proof idea

Let the solver of $A_{TM}$ be $\text{returns\_true}$ which takes a boolean function $f$, an argument $a$, and returns whether $f(a)$ would return true. Function $\text{returns\_true}$ halts for every input.

```python
def tricky_v2(f):
    return not returns_true(f, f)
```

1. What would the result of $\text{tricky\_v2(tricky\_v2)}$ be?
2. Assume that $\text{tricky\_v2(tricky\_v2)}$ loops
Proof idea

Let the solver of $A_{TM}$ be \texttt{returns\_true} which takes a boolean function $f$, an argument $a$, and returns whether $f(a)$ would return true. Function \texttt{returns\_true} \textbf{halts} for every input.

```python
def tricky_v2(f):
    return not returns_true(f, f)
```

1. What would the result of \texttt{tricky\_v2(tricky\_v2)} be?
2. Assume that \texttt{tricky\_v2(tricky\_v2)} \textbf{loops}
3. \texttt{not return\_true(tricky\_v2, tricky\_v2)} \textbf{loops}
   (replace function call by definition)
Proof idea

Let the solver of $A_{TM}$ be $\text{returns_true}$ which takes a boolean function $f$, an argument $a$, and returns whether $f(a)$ would return true. Function $\text{returns_true}$ halts for every input.

```python
def tricky_v2(f):
    return not returns_true(f, f)
```

1. What would the result of $\text{tricky_v2(tricky_v2)}$ be?
2. Assume that $\text{tricky_v2(tricky_v2)}$ loops
3. not $\text{return_true(tricky_v2, tricky_v2)}$ loops
   (replace function call by definition)
4. not $\text{false}$ loops
   (return_true(tricky_v2, tricky_v2) = false from assumption 2)
Proof idea

Let the solver of $A_{TM}$ be `returns_true` which takes a boolean function $f$, an argument $a$, and returns whether $f(a)$ would return true. Function `returns_true` halts for every input.

```
def tricky_v2(f):
    return not returns_true(f, f)
```

1. What would the result of `tricky_v2(tricky_v2)` be?
2. Assume that `tricky_v2(tricky_v2)` loops
3. `not return_true(tricky_v2, tricky_v2)` loops
   (replace function call by definition)
4. `not false` loops
   (return_true(tricky_v2, tricky_v2) = false from assumption 2)
5. contradiction
Proof idea

1. Assume $\text{tricky\_v2(tricky\_v2)} = \text{true}$
Proof idea

1. Assume \textit{tricky\_v2(tricky\_v2)} = true
2. \textit{not return\_true(tricky\_v2, tricky\_v2)} = true
   (replace function call by function body)
Proof idea

1. Assume $\text{tricky\_v2}(\text{tricky\_v2}) = \text{true}$
2. $\neg \text{return\_true}(\text{tricky\_v2}, \text{tricky\_v2}) = \text{true}$
   (replace function call by function body)
3. $\neg \text{true} = \text{true}$
   (since from assumption 2, $\text{return\_true}(\text{tricky\_v2}, \text{tricky\_v2}) = \text{true}$)
Theorem 4.11

Functional view of $A_{TM}$

```python
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

```python
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 that $A_{TM}$ does not terminate, what is the result of $\text{negator}(\text{negator})$?
Theorem 4.11

\( \mathcal{A}_{TM} \) is undecidable

\[ \mathcal{A}_{TM} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts } w \} \]

Lemma no_decides_a_tm: \( \neg \exists m, \text{Decides } m \text{ A_tm}. \)

1. Proof follows by contradiction.
2. Let \( a_{tm} \) be the decider of \( A_{TM} \)
3. Consider the negator machine:

```python
def negator(w):
    # w = <M>
    M = decode_machine w
    b = call a_tm <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 a_tm(f, f)
Theorem 4.11: $A_{TM}$ is undecidable

1. def negator(w):
2. M = decode_machine w
3. b = call D <M, w> # $A_{TM}(M, \langle M \rangle)$?
4. return not b # Return the opposite

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

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**

1. `def negator(w):`
2. `M = decode_machine w`
3. `b = call D <M, w> # $A_{TM}(M <M>)$?`
4. `return not b # Return the opposite`

$A_{TM} = \{(M, w) \mid M \text{ is a TM that accepts } w\}$

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

1. def negator(w):
2.   M = decode_machine w
3.   b = call D <M, w> # A_TM(M, <M>)?
4.   return not b  # Return the opposite

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

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

1. def negator(w):
2. M = decode_machine w
3. b = call D <M, w> # $A_{TM}(M, <M>)$?
4. return not b # Return the opposite

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

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

1. def negator(w):
2.   M = decode_machine w
3.   b = call D <M, w> # M accepts <M>?
4.   return not b # Return the opposite

$A_{TM} = \{<M, w> \mid M \text{ is a TM that accepts } w\}$

7. Case $N$ loops $\langle N \rangle$.
Theorem 4.11: $A_{TM}$ is undecidable

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

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

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!**
The negator

In Python

```python
def negator(i):
    # decode_machine(i) accepts i?
    b = D(decode_machine(i), i)
    return not b  # Return the opposite
```

In Coq

```coq
Definition negator (D:input → prog) :=
    fun i ⇒
        mlet b ← D ([ decode_mach i, i ])
        (* └── Machine ┘ *)
        Ret (neg 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 \( \langle M \rangle \)
- \( \langle M, w \rangle \) is the serialized pair \( M \) and \( w \)
- \( b \) takes the result of calling \( D \) with \( \langle M, w \rangle \)
- 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, $\overline{L} = \Sigma^* - L$
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 $\overline{L}$ 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.

**Theorem 4.22**

$L$ decidable iff $L$ recognizable and $L$ co-recognizable
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. (Proved.)
2. If $L$ decidable, then $L$ is co-recognizable. (Proved.)
3. If $L$ recognizable and $L$ co-recognizable, then $L$ decidable.
Part 3. If $L$ recognizable and $\overline{L}$ recognizable, then $L$ decidable.

We need to extend our mini-language of TMs

\[
\text{plet } b \leftarrow P1 \parallel P2 \text{ in } P3
\]

Runs $P1$ and $P2$ in parallel.

- If $P1$ and $P2$ loop, the whole computation loops
- If $P1$ halts and $P2$ halts, pass the success of both to $P3$
- If $P1$ halts and $P2$ loops, pass the success of $P1$ to $P3$
- If $P1$ loops and $P2$ halts, pass the success of $P2$ to $P3$

```plaintext
Inductive par_result :=
| pleft: bool → par_result
| pright: bool → par_result
| pboth: bool → bool → par_result.
```
Part 3. If $L$ recognizable and $\overline{L}$ recognizable, then $L$ decidable.

**Proof.**

1. Let $M_1$ recognize $L$ from assumption $L$ recognizable
2. Let $M_2$ recognize $\overline{L}$ from assumption $\overline{L}$ recognizable
3. Build the following machine

\begin{verbatim}
Definition par_run M1 M2 w :=
  plet b ← Call M1 w \ Call M2 w in
  match b with
  | pleft true  ⇒ ACCEPT
  | pboth true _ ⇒ ACCEPT
  | pright false ⇒ ACCEPT
  | _           ⇒ REJECT
  end.
\end{verbatim}

4. Show that $\text{par}_\text{run} M_1 M_2$ recognizes $L$ and is a decider.
Part 3. If $L$ recognizable and $\overline{L}$ recognizable, then $L$ decidable.

Point 4: Show that \texttt{par\_run M1 M2} recognizes $L$ and is a decider.

1. Show that \texttt{par\_run M1 M2} recognizes $L$: \texttt{par\_run M1 M2} accepts $w$ iff $L(w)$
   1.1. \texttt{par\_run M1 M2} accepts $w$, then $w \in L$
   1.2. $w \in L$, then \texttt{par\_run M1 M2} accepts $w$ case analysis on run $M2$ with $w$

\begin{verbatim}
Definition \texttt{par\_run M1 M2 w :=}
p\texttt{let b <- Call M1 w | Call M2 w in}
\texttt{match b with}
  | \texttt{pleft true}
  | \texttt{pright false}
  | \texttt{pboth true _ => ACCEPT}
  | _ => REJECT
\texttt{end.}
\end{verbatim}

- M1 recognizes $L$
- M2 recognizes $\overline{L}$
- Lemma \texttt{par\_mach\_lang}
Part 3. If $L$ recognizable and $\overline{L}$ recognizable, then $L$ decidable.

Point 4: Show that $\text{par\_run } M_1 \ M_2$ recognizes $L$ and is a decider.

1. Show that $\text{par\_run } M_1 \ M_2$ recognizes $L$: $\text{par\_run } M_1 \ M_2$ accepts $w$ iff $L(w)$

1. If $\text{par\_run } M_1 \ M_2$ accepts $w$, then $w \in L$ by case analysis on $\text{Call M1 w } \backslash \backslash \text{Call M2 w}$:
   - $M_1$ halts and $M_2$ loops. $M_1$ must accept, thus $w \in L$.
   - $M_2$ halts and $M_1$ loops. $M_2$ must reject, but both cannot reject (contradiction).
   - $M_1$ and $M_2$ halt. $M_1$ must accept, thus $w \in L$.

2. $w \in L$, then $\text{par\_run } M_1 \ M_2$ accepts $w$. $M_1$ accepts $w$. Case analysis call $M_2$ with $w$:
   - $M_2$ accept $w$: both cannot accept, contradiction.
   - $M_2$ reject $w$: par-call yields $p_{\text{both}} \text{ true false}$, returns Accept.
   - $M_2$ loops $w$: par-call yields $p_{\text{left}} \text{ true}$, returns Accept.

(1) understand execution of a program by observing its output; (2) understand execution by observing its input
Part 3. If $L$ recognizable and $\overline{L}$ recognizable, then $L$ decidable.

Point 4: Show that $\text{par\_run M1 M2}$ recognizes $L$ and is a decider.

2. Show that $\text{par\_run M1 M2}$ decides $L$

(Walk through the proof of $\text{recognizable} \iff \text{co-recognizable} \implies \text{decidable}$...