CS420

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

```
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

```
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)")
```

UMass Boston

Proof idea

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

```
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.

```
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 `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**

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)

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)

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 `tricky_v2(tricky_v2) = true`

Proof idea

1. Assume `tricky_v2(tricky_v2) = true`
2. `not return_true(tricky_v2, tricky_v2) = true`
   (replace function call by function body)

UMass Boston

Proof idea

1. Assume `tricky_v2(tricky_v2) = true`
2. `not return_true(tricky_v2, tricky_v2) = true`
   (replace function call by function body)
3. `not true = true`
   (since from assumption 2, `return_true(tricky_v2, tricky_v2) = true`)

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 `a_tm` be the decider of A_{TM}
3. Consider the `negator` machine:

```
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

```
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
```

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

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!**

The negator

In Python

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

In Coq

```
Definition negator (D:input → prog) :=
  fun i ⇒
    mlet b ← D <[ decode_mach i, i ]> in
                 (* └─ 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 $\langle M \rangle$
- `«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.

---

class: impact

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

`plet b ← P1 \\ P2 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`

```
Inductive par_result :=
| pleft: bool → par_result
| pright: bool → par_result
| pboth: bool → bool → par_result.
```

UMass Boston

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

```
Definition par_run M1 M2 w :=             (* M1 and M2 are parameters of the machine *)
  plet b ← Call M1 w \\ Call M2 w in      (* Call M1 with w and M2 with w in parallel *)
  match b with
  | pleft true    ⇒ ACCEPT
  | pboth true _  ⇒ ACCEPT                (* If M1 accepts w, accept *)
  | pright false ⇒ ACCEPT                 (* If M2 rejects w, accept *)
  | _ ⇒ REJECT                            (* Otherwise, reject *)
  end.
```

4. Show that `par_run M1 M2` recognizes L and is a decider.

UMass Boston

Part 3. If L recognizable and $\overline{L}$ recognizable, then L decidable.

Point 4: Show that `par_run M1 M2` recognizes L and is a decider.

- 1. Show that `par_run M1 M2` recognizes L: `par_run M1 M2` accepts w iff $L(w)$
- 1.1. `par_run M1 M2` accepts w, then $w \in L$
- 1.2. $w \in L$, then `par_run M1 M2` accepts w case analysis on run `M2` with w

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

- `M1` recognizes L
- `M2` recognizes $\overline{L}$
- Lemma `par_mach_lang`

Part 3. If L recognizable and $\overline{L}$ recognizable, then L decidable.

Point 4: Show that `par_run M1 M2` recognizes L and is a decider.

1. Show that `par_run M1 M2` recognizes L: `par_run M1 M2` accepts w iff $L(w)$
   1. If `par_run M1 M2` accepts w, then $w \in L$ by case analysis on `Call M1 w \\ Call M2 w`:
      - `M1` halts and `M2` loops. `M1` must accept, thus $w \in L$
      - `M2` halts and `M1` loops. `M2` must reject, but both cannot reject (contradiction).
      - `M1` and `M2` halt. `M1` must accept, thus $w \n L$.
   2. $w \in L$, then `par_run M1 M2` accepts w. `M1` accepts w. Case analysis call `M2` with w.
      - `M2` accept w: both cannot accept, contradiction.
      - `M2` reject w: par-call yields `pboth true false`, returns `Accept`.
      - `M2` loops w: par-call yields `bleft 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 `par_run M1 M2` recognizes L and is a decider.

2. Show that `par_run M1 M2` decides L

   (Walk through the proof of** `recognizable_co_recognizable_to_decidable`**...)