CS420

Introduction to the Theory of Computation

Lecture 23: Undecidable problems

Tiago Cogumbreiro
Today we will learn...

Decidability of
  - The Halting Problem
  - Emptiness for TM
  - Regularity
  - Equality

Section 5.1
Recap

Decidable languages:

• $A_{DFA}$, $A_{REX}$, $A_{NFA}$, $A_{CFG}$

```python
def A_DFA(D, w):
    return D accepts w
```

$A_{DFA} = \{ \langle D, w \rangle \mid D \text{ accepts } w \}$

• $E_{DFA}$, $E_{CFG}$

```python
def E_DFA(D):
    return L(D) = {}
```

$E_{DFA} = \{ \langle D \rangle \mid L(D) = \emptyset \}$

• $EQ_{DFA}$

```python
def EQ_DFA(D1, D2):
    return L(D1) = L(D2)
```

$EQ_{DFA} = \{ \langle N_1, N_2 \rangle \mid L(N_1) = L(N_2) \}$
Exercise 1

Prove or falsify the following statement: $EQ_{REX}$ is undecidable.
Exercise 1

Prove or falsify the following statement: $\text{EQ}_{REX}$ is undecidable.

**Proof.** False. $\text{EQ}_{REX}$ is decidable, as given by the following pseudo code, where $\text{EQ}_{DFA}$ is the decider of $\text{EQ}_{DFA}$ and $\text{REX}_\text{TO}_\text{DFA}$ is the conversion from a regular expression into a DFA.

```python
def EQ_REX(R1, R2):
    return EQ_DFA(REX_TO_DFA(R1), REX_TO_DFA(R2))
```
Exercise 2

Let $D$ be the DFA below

![DFA Diagram]

- Exercise 2.1: Is $\langle D, 0100 \rangle \in A_{DFA}$?
- Exercise 2.2: Is $\langle D, 101 \rangle \in A_{DFA}$?
- Exercise 2.3: Is $\langle D \rangle \in A_{DFA}$?
- Exercise 2.4: Is $\langle D, 101 \rangle \in A_{REX}$?
- Exercise 2.5: Is $\langle D \rangle \in E_{DFA}$?
- Exercise 2.6: Is $\langle D, D \rangle \in E_{Q_{DFA}}$?
- Exercise 2.7: Is $101 \in A_{REX}$?

```python
def A_DFA(D, w):
    return D accept w
def E_DFA(D):
    return L(D) == {}
def EQ_DFA(D1, D2):
    return L(D1) == L(D2)
```
Exercise 3

Recall that DFAs are closed under $\cap$. Prove the following statement.

If $A$ is regular, then $X_A$ decidable.

\[ X_A = \{ \langle D \rangle \mid D \text{ is a DFA} \land L(D) \cap A \neq \emptyset \} \]
Exercise 3

Recall that DFAs are closed under \( \cap \). Prove the following statement.

If \( A \) is regular, then \( X_A \) decidable.

\[
X_A = \{ \langle D \rangle \mid D \text{ is a DFA} \land L(D) \cap A \neq \emptyset \}
\]

**Proof.** If \( A \) is regular, then let \( C \) be the DFA that recognizes \( A \). Let intersect be the implementation of \( \cap \) and \( E_{DFA} \) the decider of \( E_{DFA} \). The following is the decider of \( X_A \).

```python
def X_A(D):
    return not E_DFA(intersect(C, D))
```
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 $p3$

```
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

```
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.
```

4. Show that $\text{par\_run } M1 \ M2$ recognizes $L$ and is a decider.
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 $L(w)$
- 1.1. $\text{par\_run } M_1 \ M_2$ accepts $w$, then $w \in L$
- 1.2. $w \in L$, then $\text{par\_run } M_1 \ M_2$ accepts $w$ case analysis on run $M_2$ 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.
```

- $M_1$ recognizes $L$
- $M_2$ recognizes $\overline{L}$
- Lemma $\text{par\_mach\_lang}$
Part 3. If \( L \) recognizable and \( \bar{L} \) recognizable, then \( L \) decidable.

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

1. Show that \( \text{par\_run } M1 \ M2 \) recognizes \( L \): \( \text{par\_run } M1 \ M2 \) accepts \( w \) iff \( L(w) \)
   1. If \( \text{par\_run } M1 \ M2 \) accepts \( w \), then \( w \in L \) by case analysis on \( \text{Call } M1 \ w \ \backslash \backslash \ \text{Call } M2 \ w \):
      - \( M1 \) 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 \( w \in L \).
   2. \( w \in L \), then \( \text{par\_run } M1 \ M2 \) accepts \( w \). \( M1 \) accepts \( w \). Case analysis call \( M2 \) with \( w \).
      - \( M2 \) accept \( w \): both cannot accept, contradiction.
      - \( M2 \) reject \( w \): par-call yields \( \text{pboth} \ true \ false \), returns \( \text{Accept} \).
      - \( M2 \) loops \( w \): par-call yields \( \text{bleft} \ true \), returns \( \text{Accept} \).

(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 $\text{par\_run } M_1 \ M_2$ recognizes $L$ and is a decider.

2. Show that $\text{par\_run } M_1 \ M_2$ decides $L$

(Walk through the proof of recognizable_co_recognizable_to_decidable...)}
Homework 7 tutorial
Basic definitions
Run, recognizes

Running a Turing Machine

Use run to let a Turing machine execute input $i$. Returns a result.

```
Inductive result := Accept | Reject | Loop.
```
Run, recognizes

Running a Turing Machine

Use run to let a Turing machine \( m \) execute input \( i \). Returns a result.

\[
\text{Inductive} \quad \text{result} := \text{Accept} \mid \text{Reject} \mid \text{Loop}.
\]

Recognizes

A Turing machine \( m \) recognizes a language \( L \) if \( m \) accepts the same inputs as those in language \( L \).

\[
\text{Definition} \quad \text{Recognizes} \ m \ L := \forall i, \ \text{run} \ m \ i = \text{Accept} \iff L \ i.
\]

- Use constructor \text{recognizes}\_\text{def} to build \text{Recognizes} \ m \ L
Recognizable

Definition 3.5: Recognizable

Call a language Turing-recognizable if some Turing machine recognizes it.

Definition Recognizable \( L := \exists m, \text{Recognizes } m \ L \).

- Use constructor recognizable_def to build Recognizable \( L \).
Decides

A Turing machine $m$ decides a language $L$ if:

1. $m$ recognizes $L$
2. $m$ is a decider

**Definition** \( \text{Decides } m \ L := \text{Recognizes } m \ L \land \text{Decider } m. \)

- Use \texttt{decides_def} to build \texttt{Decides } m \ L
Decider

A Turing machine that never loops for all possible inputs.

**Definition** Decider $m := \forall i, \text{run } d i \not\leftrightarrow \text{Loop}.$

- Use `decider_def` to build `Decider m`
Decidable

Definition 3.6

Call a language Turing-decidable or simply decidable if some Turing machine decides it.

Definition Decidable \( L \) := \( \exists m, \text{Decides } m \ L \).

- Use \texttt{decidable_def} to build \texttt{Decidable } \( L \).
## Summary

<table>
<thead>
<tr>
<th>Term</th>
<th>Usage</th>
<th>Coq</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run</td>
<td>run a TM with a given input i</td>
<td>run m i</td>
<td>N/A</td>
</tr>
<tr>
<td>Recognizes</td>
<td>a TM recognizes a language</td>
<td>Recognizes m L</td>
<td>recognizes_def</td>
</tr>
<tr>
<td>Recognizable</td>
<td>a language is recognizable</td>
<td>Recognizable L</td>
<td>recognizable_def</td>
</tr>
<tr>
<td>Decides</td>
<td>a TM decides a language</td>
<td>Decides m L</td>
<td>decides_def</td>
</tr>
<tr>
<td>Decider</td>
<td>a TM is a decider</td>
<td>Decider m</td>
<td>decider_def</td>
</tr>
<tr>
<td>Decidable</td>
<td>a language is decidable</td>
<td>Decidable L</td>
<td>decidable_def</td>
</tr>
</tbody>
</table>
Prog

A DSL for composing Turing Machines
Specifying TMs with Prog

- Prog is a **domain-specific** language (DSL) that allow us to compose Turing machines.
- Prog gives an unique opportunity for CS420 students to study complex Theoretical Computer Science problems in a (hopefully) intuitive framework.
- All theorems studied in this course are fully proved; students can see all details at their own time, interactively.
- The proofs follow the structure of the book as close as possible.

**Did you know?**

- [gitlab.com/cogumbreiro/turing](http://gitlab.com/cogumbreiro/turing) is a **research project** that stemmed from trying to teach CS420 in a more compelling way (project-based, + interactive, + student-autonomous).
- This semester we are pushing the state-of-the-art of teaching Theoretical Computer Science.
- Your input matters!
Turing programs \( \text{Prog} \)

Inductive \( \text{Prog} := \)

\[
\text{Seq} : \text{Prog} \to (\text{bool} \to \text{Prog}) \to \text{Prog} \\
\text{Call} : \text{machine} \to \text{input} \to \text{Prog} \\
\text{Ret} : \text{result} \to \text{Prog}.
\]

- Seq combines two programs
- Call runs a Turing machine on a given input
- Ret loops/rejects/accepts (pick one) for all inputs
Turing programs Prog

Notations

We use 3 notations to write shorter programs:

\[ \text{mlet } x \leftarrow p_1 \text{ in } p_2 = \text{Seq } p_1 (\text{fun } x \Rightarrow p_2) \]

\[ \text{ACCEPT } = \text{Ret Accept} \]
\[ \text{REJECT } = \text{Ret Reject} \]
\[ \text{LOOP } = \text{Ret Loop} \]
1. Rule \( \text{run}_{\text{ret}} \): the result of returning \( r \) (with \( \text{Ret} \ r \)) is \( r \)

\[
\begin{align*}
\text{Run} \ (\text{Ret} \ r) & = r \\
\end{align*}
\]

2. The result of calling a TM \( m \) is given by calling \( \text{run} \ m \ i \).

\[
\begin{align*}
\text{run}(m, i) & = r \\
\text{Run}(\text{Call} \ m \ i) & = r \\
\end{align*}
\]
3. If we run program \( p \) and get a result \( r_1 \) and \( p \) terminates with \( b \) and we run \((p \ b)\) and get a result \( r_2 \), then sequencing \( p \) with \( q \) returns result \( r_2 \)

\[
\begin{align*}
\text{Run } p & \quad r_1 \\
\text{Dec } r_1 & \quad b \\
\text{Run (q b)} & \quad r_2 \\
\hline
\text{Run (Seq p q)} & \quad r_2
\end{align*}
\]

4. If program \( p \) loops, then running \( p \) followed by \( q \) also loops:

\[
\begin{align*}
\text{Run } p & \quad \text{Loop} \\
\hline
\text{Run (Seq p q)} & \quad \text{Loop}
\end{align*}
\]
P-run in Coq

**Inductive** Run: Prog → result → Prop :=

- **run_ret:**
  - \( \forall r, \) Run (Ret r) r

- **run_call:**
  - Run (Call m i) (run m i)

- **run_seq_cont:**
  - \( \forall p q b r1 r2, \) Run p r1 →
    - Dec r1 b →
    - Run (q b) r2 →
    - Run (Seq p q) r2

- **run_seq_loop:**
  - \( \forall p q, \) Run p Loop →
    - Run (Seq p q) Loop
Why do we need P-run?

- Because Prog is inductively defined, we can reason about all possible ways in which we can declare a program (induction proofs).
- Because Run is inductively defined, we can also reason about all possible ways in which we can run a program.
- Prog is already being informally used in the book, we are just making the meta-theory more formal!
- Proofs are easier (homework assignments have less technicalities/distractions).
P-Recognizes

Program $p$ P-recognizes a language $L$ if $p$ accepts the same inputs as those in language $L$.

Definition $\text{PRecognizes } p \ L := \forall i, \text{Run}(p \ i) \text{ Accept} \leftrightarrow L \ i$

- Use $p\text{-recognizes\_def}$ to build $\text{PRecognizes } p \ L$
P-Recognizable

Call a language P-recognizable if some Prog recognizes it.

- There is no definition PRecognizable! We use Recognizable still.
- Use p_recognizable_def to build Recognizable L with a program!
P-Decides

A program $p$ P-decides a language $L$ if:

1. $p$ P-recognizes $L$
2. $p$ is a P-decider

**Definition** $\text{PDecides } p \ L := \text{PRecognizes } p \ L \lor \text{PDecider } p$.

- Use $p_{-decides\_def}$ to build $\text{PDecides } p \ L$
P-Decider

A program that never loops for all possible inputs.

\textbf{Definition} \ PDecider \ p := \forall i, \ PHalts \ (p \ i).

- Use \texttt{p\_decider\_def} to build \texttt{PDecider p}

P-Halts

\textbf{Definition} \ PHalts \ p := \exists r : \text{result}, \ \text{Run} \ p \ r \ \land \ r \neq \text{Loop}

- Use \texttt{p\_halts\_def} to build \texttt{PHalts p}.
P-Decidable

Call a program P-decidable or simply decidable if some program decides it.

- There is no definition PDecidable! We use Decidable still.
- Use p_decidable_def to build Decidable L
## Summary

<table>
<thead>
<tr>
<th>Term</th>
<th>Usage</th>
<th>Coq</th>
<th>Constructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-Run</td>
<td>run a program with a given input ( i ) and result ( r )</td>
<td>Run ( p \ i \ r )</td>
<td>Print Run.</td>
</tr>
<tr>
<td>P-Recognizes</td>
<td>a program <strong>recognizes</strong> a language</td>
<td>( P\text{Recognizes} \ p \ L )</td>
<td>( p_\text{recognizes_def} )</td>
</tr>
<tr>
<td>P-Recognizable</td>
<td>a language is <strong>recognizable</strong></td>
<td>( \text{Recognizable} \ L )</td>
<td>( p_\text{recognizable_def} )</td>
</tr>
<tr>
<td>P-Decides</td>
<td>a program <strong>decides</strong> a language</td>
<td>( P\text{Decides} \ p \ L )</td>
<td>( p_\text{decides_def} )</td>
</tr>
<tr>
<td>P-Decider</td>
<td>a program is a <strong>decider</strong></td>
<td>( P\text{Decider} \ p )</td>
<td>( p_\text{decider_def} )</td>
</tr>
<tr>
<td>P-Decidable</td>
<td>a language is <strong>decidable</strong></td>
<td>( \text{Decidable} \ L )</td>
<td>( p_\text{decidable_def} )</td>
</tr>
</tbody>
</table>