Homework 8
Homework 8

Exercises $E_{\text{tm\_red\_EQ\_tm\_1}}$ and $E_{\text{tm\_red\_EQ\_tm\_2}}$ are not difficult, as long as you use constructor (_inv) and destructor (_def) theorems.

- When should you use _inv?
- When should you use _def?
- Simplify assumptions run (Build _) with run_simpl_all.
- $E_{TM}$ is a unary predicate; $EQ_{TM}$ is a binary predicate.
  F3 maps one to the other.

See Example 5.26 (pp 237):
Let $<\text{M}> = p$.

Function F3 (defined below) maps the input $<\text{M}>$ to the output $<\text{M}, \text{M1}>$, where $\text{M1}$ is the machine that rejects all inputs.
Homework 8

E_tm_red_EQ_tm_1 and E_tm_red_EQ_tm_2

- For E_tm and EQ_tm constructors: when supplying a turing machine specified as a program P, you need to write Build (fun i ⇒ P)
- For E_tm and EQ_tm constructors: if you don't know the turing machine, just provide an arbitrary one, the first goal will help you find the right answer.
- If you have trouble applying E_tm_def to your goal, then assert it: assert (Hx := E_tm_def Machine Word). If you don't know what to provide play with the parameters until the pre-conditions are trivial.
5.6 Suppose $A \leq_m B$ and $B \leq_m C$. Then there are computable functions $f$ and $g$ such that $x \in A \iff f(x) \in B$ and $y \in B \iff g(y) \in C$. Consider the composition function $h(x) = g(f(x))$. We can build a TM that computes $h$ as follows: First, simulate a TM for $f$ (such a TM exists because we assumed that $f$ is computable) on input $x$ and call the output $y$. Then simulate a TM for $g$ on $y$. The output is $h(x) = g(f(x))$. Therefore, $h$ is a computable function. Moreover, $x \in A \iff h(x) \in C$. Hence $A \leq_m C$ via the reduction function $h$.

5.7 Suppose that $A \leq_m \overline{A}$. Then $\overline{A} \leq_m A$ via the same mapping reduction. Because $A$ is Turing-recognizable, Theorem 5.28 implies that $\overline{A}$ is Turing-recognizable, and then Theorem 4.22 implies that $A$ is decidable.
Homework 8

Textbook solution for Theorem 5.22

**THEOREM 5.22**

If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**PROOF** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{"On input } w:\n1. \text{ Compute } f(w). \n2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs."

Clearly, if $w \in A$, then $f(w) \in B$ because $f$ is a reduction from $A$ to $B$. Thus, $M$ accepts $f(w)$ whenever $w \in A$. Therefore, $N$ works as desired.
Homework 8

ex_5_22_a

- Unfold Reduction before you start.

ex_5_22_b

- Search \(_ \leq m \_\). is your friend
Homework 7
Homework 7

ex1 and ex2

- Look at sample code for inspiration
- The table below is important

<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 recognizes a language</td>
<td>PRecognizes p L</td>
<td>p_recognizes_def</td>
</tr>
<tr>
<td>P-Recognizable</td>
<td>a language is recognizable</td>
<td>Recognizable L</td>
<td>p_recognizable_def</td>
</tr>
<tr>
<td>P-Decides</td>
<td>a program decides a language</td>
<td>PDecides p L</td>
<td>p_decides_def</td>
</tr>
<tr>
<td>P-Decider</td>
<td>a program is a decider</td>
<td>PDecider p</td>
<td>p_decider_def</td>
</tr>
<tr>
<td>P-Decidable</td>
<td>a language is decidable</td>
<td>Decidable L</td>
<td>p_decidable_def</td>
</tr>
</tbody>
</table>
Homework 7

ex3

- Recall the operators we have learned
- Look at the examples we have written so far

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Prog</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call m w</td>
<td>mlet x &lt;- p1 in p2</td>
<td>Calls a turing machine</td>
</tr>
<tr>
<td></td>
<td>Seq p1 (fun x =&gt; p2)</td>
<td>Sequence of two progs</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>Ret Accept</td>
<td>Accepts</td>
</tr>
<tr>
<td>REJECT</td>
<td>Ret Reject</td>
<td>Rejects</td>
</tr>
<tr>
<td>LOOP</td>
<td>Ret Loop</td>
<td>Loops</td>
</tr>
<tr>
<td>halt_with b</td>
<td>Ret (if b then Accept else Reject)</td>
<td>Rejects/accepts according to bool</td>
</tr>
</tbody>
</table>
Homework 7

ex3

- A prog is simply a composition of calls.
- Progs can only manipulate bools/nats, but not Props.

This machine loops if is less than equal 7, otherwise calls a turing machine m with input w.

```plaintext
if Nat.leb x 7 then
  LOOP
else
  Call m w
```
Homework 7

ex4 and ex5

- Print Run (Lecture 24)
- You will need to use these constructors

```plaintext
Inductive Run: Prog → result → Prop :=
| run_ret:
  ..
| run_call:
  ...
| run_seq_cont:
  ...
| run_seq_loop:
  ...
```
Homework 7

ex6_1

- Use `p_recognizes_def`
- Use `run_simpl_all` to clean up assumptions.
- In the second branch of `p_recognizes_def` you want to use `destruct (run_exists (p i))` as `(r, Hr)`. 