The Recursion Theorem

CS154
Chris Pollett
May 8, 2006.

Outline

- The Recursion Theorem

Towards The Recursion Theorem

- One interesting property of living things is that they can reproduce – that is, they can produce "exact" copies of themselves.
- Can machines do this? As a first step:

Lemma. There is a computable function q: $\Sigma^* \rightarrow \Sigma^*$, where if w is any string $q(w)$ is the description of a Turing Machine P_w that prints out w and then halts.

Proof.

Q ="On input w:

1. Construct the following TM P_w.

   P_w = " On any input:

   1. Erase the input.
   2. Write w on the tape.
   3. Halt."

2. Output $< P_w >$."

SELF

- Using *Q* of the last slide we next describe a machine *SELF* which ignores its own input and prints it TM description to the output.
- *SELF* consists of two parts *A* and *B*.
- *A* runs first and then passes control to *B*. *A* is the machine $P_{<B>}$ where *B* is:

"On input $<M>$, where *M* is a portion of a TM:

1. Compute $q(<M>)$.
2. Combine the result with $<M>$ to make a complete TM.
3. Print the description of this TM and halt."

- So once *A* is done the tape has $<B>$ on it.
- *B* then computes $q(<M>) = <P_{<B>}> = <A>$ and concatenates $<B>$ with some extra state to make this into a whole machine. This give $<AB> = <SELF>$ back.
- One way to see this construction works is to consider the following English sentence:

Print the next phrase in quotes twice the second time in quotes: "Print the next phrase in quotes twice the second time in quotes:"

- *B* is like the phrase: Print the next phrase in quotes twice the second time in quotes: ; *A* is the phrase with quotes around it.

The Recursion Theorem

- We can generalize the above argument to allow machines to compute with their own descriptions

Theorem. Let T be a TM that computes a function $t{:}\Sigma^*$ x Σ^* --> Σ^*. There is a Turing machine R that computes a function r: Σ^* --> Σ^*, where for every w,

$$r(w) = t(<R>, w).$$

Proof. The proof is like the construction of *SELF* except now we have the machine T besides A and B, and R will be a combined machine ABT. In the current construction $A{=}P'_{<BT>}$. Here $P'_{<BT>}$ is like $P_{<BT>}$ except it prints <BT> after the input and a #. We design a q' so it looks for a #, sees the string v that follows it and, and appends $<P'_{<v>}>$ to the input. So after A runs the tape has w#$<BT>$ on it. Now B applies q' to the output of A to get $w\#<BT><P'_{<BT>}> = w\#<BT><A>$, and then reformats this as $<<ABT>$, w>. It then starts T. Notice $<R> = <ABT>$.

Applications of the Recursion Theorem

- The recursion theorem allows one to design TM subroutines of the form “ obtain your own description”.
- For instance, *SELF* could be rewritten as:

SELF= “On any input:

1. Obtain, via the recursion theorem, own description <*SELF*>.
2. Print <*SELF*>.”

- The obtain your own description statement is implemented by first writing the machine:

T= “On any input <*M, w*>:

1. Print <M> and halt.”

- Then the recursion theorem says how to get a machine *R* which on input *w* acts like *T* on input <*R, w*>.

More Applications

- We can use the recursion theorem to give an alternative proof that A_{TM} is undecidable:
- First, suppose A_{TM} were decidable by machine H. Then consider the machine B:

B=“ On input w:

1. Obtain, via the recursion theorem, own description $\langle B \rangle$.
2. Run H on $\langle B, w \rangle$.
3. Do the opposite of what H does.”

So running B on input w does the opposite of what H on $\langle B, w \rangle$ does.
So H can’t decide A_{TM}.

The Fixed-Point Theorem

- A **fixed point** of a function f is a value x such that $f(x)=x$.

Theorem. Let t: Σ^* --> Σ^* be a computable function. Then there is a machine F for which L(t(<F>)) = L(F). Here we are assuming that if a string isn't a proper TM, then it describes the empty language.

Proof.

F = "On input w:

1. Obtain via the recursion theorem, own description <F>.
2. Compute t(<F>) to obtain a TM description G.
3. Simulate G on w."