2nd Lt David Crow

ENG/20M

CSCE 532 Final Exam

1. Let . Give an implementation-level description of a TM such that

.

“On input such that is a string:

1. Scan the input from left to right to determine whether it is a member of ; if it isn’t, .
2. Return to the left-hand end of the tape.
3. Scan to the right. Mark the first unmarked .
4. Scan to the right. If an unmarked is found, mark it. If one is not found before encountering a blank symbol, .
5. Scan to the right. If an unmarked is found, mark it. If one is not found before encountering a blank symbol, .
6. Scan to the left. If an unmarked is found, go to stage 2.
7. Scan from left to right. If an unmarked is found, ; otherwise, .”
8. Let . Define a TM such that .

This question is a little ambiguous. It’s possible that you want us to give a formal definition of a Turing Machine, but the question doesn’t say that. Sipser says that “every higher-level description is actually just shorthand for its formal counterpart,” so I’m simply going to give a higher-level description.

We can construct in the following way:

“On input such that is a string:

1. Scan the input from left to right to determine whether it is a member of ; if it isn’t, . If only one is encountered, . If exactly two s are encountered, .
2. Return to the left-hand end of the tape.
3. Scan the input from left to right until the first blank symbol is found. While scanning, count the number of s. Replace the blank symbol with this count (for the purposes of this procedure, the count is ).
4. Replace the next blank symbol with .
5. Replace the next blank symbols with the numbers , respectively.
6. Return to the .
7. Scan to the right. If an unmarked number is found, mark it and compute .1 If the answer is equal to zero, ; otherwise, return to stage 6.
8. If in stage 7 no unmarked number is found before encountering a blank symbol, .”

1 We can do this in the following way:

1. Add the string to the tape (at some point after the tape symbol for ).
2. Mark the first unmarked in . Mark the first unmarked in .
3. Repeat stage 2 until every is marked or until every is marked (or both).
   1. If every is marked and every is marked, .
   2. If every is marked and at least one is unmarked, . We don’t care about the exact value (however, it’s easy to compute).
   3. Otherwise, clear every mark from and return to stage 2.
4. Prove that, if is Turing-decidable and is Turing-recognizable, then is

Turing-recognizable.

We know is the language consisting of strings in that are not also in .

Because is decidable, there exists some TM that decides . Because is Turing-recognizable, there exists some TM that recognizes . Let’s construct the following TM:

“On input such that is decidable, is Turing-recognizable, and is a string:

1. Construct a TM such that .
2. Run on input . If rejects, .
3. Construct a TM such that .
4. Run on input . If rejects, . If accepts, .”

Clearly, accepts if and . It rejects if or . In all other cases (that is, where stage 4 does not halt because loops on ), loops. Because either accepts, rejects, or loops on every input, is a recognizer. Because we constructed it to evaluate , we’ve shown that is Turing-recognizable.

1. Let . Without invoking Rice’s Theorem, prove that is undecidable.

We use the undecidability of to prove the undecidability of by reducing to . Let’s assume for the purpose of obtaining a contradiction that TM decides . We construct TM to decide , where operates as follows:

“On input such that is an encoding of a TM and is a string:

1. Construct the following TM:

“On input such that is a string:

1. Run on input .
2. .”
3. Run on input . If accepts, ; otherwise, .”

If accepts or rejects , then accepts. If loops on , then rejects. Clearly, decides , but Theorem 5.1 tells us that is undecidable, so we’ve reached a contradiction. We assumed that decides , so our assumption is false. Thus, is undecidable.