

Problems for Week Ten

Self-Reference

Self-reference is one of the trickier topics from the tail end of the quarter. This problem explores self-reference through a series of questions about a variety of different programs.

i. What does the following program do?

```
int main() {
    string input = getInput();
    string me = mySource();

    if (input == me) accept();
    else reject();
}
```

In the proof from lecture we did that A_{TM} is undecidable, we began by assuming that A_{TM} was decidable. That meant there must be some decider for A_{TM} , which, in software, we represent as a method

```
bool willAccept(string program, string input)
```

that takes as input the source code of a program and an input string, then returns true if the specified program will accept the specified input and returns false otherwise.

ii. Consider the following program:

```
int main() {
    string input = getInput();

    if (input == "") accept();
    else if (input[0] == input[input.length() - 1]) accept();
    else reject();
}
```

What happens if we run the above program with input *abba*? Why?

iii. Let *p1* be a string containing the source of the above program. What will happen if we call *willAccept(p1, "abba")*? Why?

iv. Consider this program:

```
int main() {
    string input = getInput();
    string target = "";

    while (target != input) target += "a";
    accept();
}
```

What happens if we run the above program with input *abba*? Why?

v. Let *p2* be a string containing the source of the above program. What will happen if we call *willAccept(p2, "abba")*? Why?

Self-reference for decidability can be a tricky topic. If you haven't yet done so, you should pause and go read the Guide to Self-Reference on the course website.

Let's look at the self-referential program we wrote in lecture that we used to show A_{TM} was undecidable:

```
int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        reject();
    } else {
        accept();
    }
}
```

Let's go look at this code in some more detail.

vi. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the program accepts *abba*, then the program does not accept *abba*.

vii. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the does not accept *abba*, then it does accept *abba*.

viii. Explain why your answers to parts (vi) and (vii) collectively result in a contradiction that shows that A_{TM} is undecidable.

In the Guide to Self-Reference, we showed another self-referential program we could have written that would also help us see that A_{TM} is undecidable:

```
int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        while (true) {
            // Do nothing
        }
    } else {
        accept();
    }
}
```

Let's go look at this code in some more detail.

ix. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the program accepts *abba*, then it does not accept *abba*.

x. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the program does not accept *abba*, then it does accept *abba*.

xi. Explain why your answers to parts (ix) and (x) collectively result in a contradiction that shows that A_{TM} is undecidable.

In the Guide to Self-Reference, we showed a third self-referential program related to A_{TM} :

```
int main() {
    string me = mySource();
    string input = getInput();

    if (willAccept(me, input)) {
        accept();
    } else {
        reject();
    }
}
```

Let's go look at this code in some more detail.

xii. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the program accepts *abba*, then it does accept *abba*.

xiii. Suppose we feed the string *abba* as input to the above program. Explain why if *willAccept* says that the program does not accept *abba*, then it does not accept *abba*.

xiv. Explain why your answers to parts (xii) and (xiii) do ***not*** prove that A_{TM} is decidable.

Self-Reference and Decidability

Consider the language $L = \{ \langle M \rangle \mid M \text{ is a TM that accepts at least one string } \}$. This language is undecidable. Let's go see why this is.

i. Suppose for the sake of contradiction that $L \in \mathbf{R}$. This means that we could write a function

```
bool acceptsAtLeastOneString(string program)
```

that accepts as input the source code of a program, then returns true if the program accepts at least one string and returns false otherwise. Write a self-referential program that uses this function to obtain a contradiction. As a hint, recall the general template for these sorts of programs: have the program ask whether it accepts at least one string, then have it do the opposite of whatever it determines it's supposed to do.

ii. Formalize your reasoning from part (ii) by writing a formal proof that $L \notin \mathbf{R}$. To do so, follow the proof template from lecture: assume that $L \in \mathbf{R}$, describe what that assumption entails, write a program that causes a contradiction, then explain why you get a contradiction in all cases.

iii. Although L is undecidable, it turns out to be recognizable. Prove that $L \in \mathbf{RE}$ by writing pseudocode for either a recognizer or a verifier for L. Either way, make sure you explain why your program meets the formal requirements for a recognizer/verifier.

Review/Potpourri

Below is a selection of problems based on the topics you wanted more practice with. Feel free to jump around and complete whichever problems will be most helpful to you :)

Graphs

i. A *complete graph* is a graph G = (V, E) such that $\forall x \in V. \forall y \in V. (x \neq y \rightarrow \{x,y\} \in E)$. Prove by induction that a complete graph on n vertices has $n(n-1)/2$ edges, for any $n \in \mathbb{N}$.

ii. Prove by induction that for any graph G = (V, E) and vertices $x,y \in V$, if there is a path from x to y then there is a *simple* path from x to y. (Hint: try inducting on some property of the path, rather than the graph itself.)

Regular Languages

Prove that each of the following languages is regular.

i. $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and $L = \{ w \in \Sigma^* \mid$ all of the **a**'s in w appear in groups of 2 or more $\}$.

ii. $\Sigma = \{\mathbf{a}, \mathbf{b}\}$ and $L = \{ w \in \Sigma^* \mid w$ does not contain the substring **ab** $\}$.

iii. $\Sigma = \{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ and $L = \{ w \in \Sigma^* \mid w$ does not contain the substring **ab** $\}$.

Context-Free Languages

Consider the alphabet $\Sigma = \{ \mathbf{a}, \mathbf{b}, \boldsymbol{\varepsilon}, \boldsymbol{\emptyset}, \boldsymbol{\cup}, \boldsymbol{*}, \boldsymbol{(}, \boldsymbol{)} \}$ and the following language, which we'll call REGEXP:

$$\{ w \in \Sigma^* \mid w \text{ is a valid regular expression over } \{\mathbf{a},\mathbf{b}\} \}.$$

You may wish to refer back to the formal definition of regular expressions from lecture 17.

i. Give three examples of strings in REGEXP, and three examples of strings *not* in REGEXP.

ii. Prove or disprove: the language REGEXP defined above is regular.

iii. Write a context-free grammar for REGEXP.

Consider the alphabet $\Sigma = \{ \mathbf{a}, \boldsymbol{\in}, \boldsymbol{,}, \boldsymbol{\{}, \boldsymbol{\}} \}$ and the following language, which we'll call ELEMOF:

$$\{ w\boldsymbol{\in\{}\ell\boldsymbol{\}} \mid w \in \{\mathbf{a}\}^+ \text{ and } \ell \text{ is a comma-separated list of strings in } \{\mathbf{a}\}^+\text{, at least one of which equals } w \}.$$

Essentially this language contains all true "element-of" relations for finite sets of non-empty strings.

iv. Give three examples of strings in ELEMOF, and three examples of strings *not* in ELEMOF.

v. Prove or disprove: the language ELEMOF defined above is regular.

vi. Write a context-free grammar for ELEMOF.

R vs. RE

For each of the following languages, determine whether it is (a) decidable, (b) recognizable but NOT decidable, or (c) not recognizable, and prove that your choice is correct. (Hint: we haven't seen very many proofs of non-recognizability, but see if you can adapt the ideas from the Guide to Self-Reference.)

i. $L = \{ \langle M_1, M_2 \rangle \mid M_1, M_2$ are Turing machines and $\mathcal{L}(M_1) \cap \mathcal{L}(M_2) \neq \emptyset \}$

ii. $L = \{ \langle M \rangle \mid M$ writes at least one non-blank symbol on the tape when given input $\varepsilon \}$

iii. $L = \{ \langle M \rangle \mid M$ is a Turing machine but NOT a decider $\}$

Closure Properties of RE

i. Let L_1 and L_2 be recognizable languages over the same alphabet Σ. Prove that $L_1 \cap L_2$ is also recognizable. To do so, suppose that you have Turing machines M_1 and M_2 such that $\mathcal{L}(M_1) = L_1$ and $\mathcal{L}(M_2) = L_2$, then write pseudocode for recognizing whether a given input string is in $L_1 \cap L_2$. Then, briefly justify why your construction is correct.

ii. Repeat part (i), except proving that the **RE** languages are closed under union.

Verifiers

In class, we proved that the diagonal language L_D is not recognizable. Explain why the following program is NOT a verifier for L_D.

```
bool checkLD(tm M, int c) {
    run M on ⟨M⟩ for c steps;
    if (M is in a rejecting state) accept();
    else reject();
}
```

Complexity Theory

For each of the following statements, determine whether it is true or false and justify your answer.

i. For every language L, if $L \in$ **P** then $L \in$ **REG**.

ii. For every language L, if $L \in$ **P** then $L \in$ **R**.

iii. For every language L, if $L \in$ **P** then $L \in$ **RE**.

iv. For languages A and B, if $A \leq_p B$ and $B \in$ **P** then $A \in$ **NP**.