CISS362: Introduction to Automata Theory, Languages, and Computation
Assignment a03

The objective of this assignment is to write proofs by induction. I'm choosing strings as the subject, which are objects that are used throughout in automata theory. The facts to prove are simple and obvious. These are for instructional purposes only. In real life, when doing research, many simple and obvious facts might not be explicitly proven.

We will prove something about languages.

First let me fix (formally) various concepts. We fix an alphabet Σ throughout. For the following x, y, z are words in Σ^*. The following are some basic facts about the word concatenation operator. There are not axioms but are facts which can be proven. For this assignment, we assume that they are already proven and we just use them.

C1 xy is a word in Σ^*
C2 $(xy)z$ can be rewritten as $x(yz)$
C3 $x(yz)$ can be rewritten as $(xy)z$
C4 ϵx can be rewritten as x
C5 x can be rewritten as ϵx
C6 $x\epsilon$ can be rewritten as x
C7 x can be rewritten as $x\epsilon$
C8 If $x \in \Sigma^*$ is not ϵ, then $x = x'x''$ for some $x' \in \Sigma$ and $x'' \in \Sigma^*$.
[This is formalizing the idea that "a nonempty word always begins with a symbol from the alphabet". It's extremely important to be able to speak the formal mathematical language and yet be able to translate the formal syntax into something simple.]

The fact C2 and C3 is usually written as

$$x(yz) = (xy)z$$

This equation means "if you see $x(yz)$ in a proof, then you can replace that with $(xy)z$" and "if you see $(xy)z$ in a proof, then you can replace that with $x(yz)$". These are really two facts. That's why we have C2 and C3. In general whenever you have an equation, you have two facts: LHS can be replaced by RHS and RHS can be replaced by LHS.

It's only in some of my assignments that I replace the single rule

$$x(yz) = (xy)z$$

by the two rules. This is just for instructional purposes for this courses to make sure that you really get it. Other books will only use the above single rule.

In the same way C4-C7 is usually stated as

$$\epsilon x = x = x\epsilon$$

C4-C7 is more precise.

For this assignment, you will be completing several proofs. There are gaps in some

of the proofs. These are denoted by "?". You will need to replace the "?".

These "?" are in the `answerbox{}`s in the LaTeX files, `q1.tex`, `q2.tex`, `q3.tex`. For `q5.tex`, you'll write a complete proof.

Here we go ...

First we define formally the length function. Let x be a word over Σ^*.

L1 If $x = \epsilon$, then we define
$$|x| = 0$$

L2 If $x \neq \epsilon$, then $x = x' \cdot x''$ where $x' \in \Sigma$, i.e. x' is a symbol in our alphabet and $x'' \in \Sigma^*$, i.e. x'' is a word over Σ (this is because of C8). We define
$$|x| = 1 + |x''|$$

As an example, let me compute $|aba|$.

$$\begin{aligned}
|aba| &= |a \cdot ba| \\
&= 1 + |ba| && \text{by L2} \\
&= 1 + |b \cdot a| \\
&= 1 + 1 + |a| && \text{by L2} \\
&= 1 + 1 + |a \cdot \epsilon| \\
&= 1 + 1 + 1 + |\epsilon| && \text{by L2} \\
&= 1 + 1 + 1 + 0 && \text{by L1} \\
&= 3
\end{aligned}$$

You may assume following properties about the length function:

L3 $|x| \geq 0$
L4 $|x|$ is an integer

Note that the "other definition" of length

$$|x| = \text{count of the number of symbols in } x$$

although easier to understand (for human beings) is not as precise: "the number of symbols in x" is not strictly mathematical. To illustrate this very clearly, the definition above

$$|x| = \begin{cases} 0 & \text{if } x = \epsilon \\ 1 + |x''| & \text{if } x = x'x'' \text{ for } x' \in \Sigma,\ x'' \in \Sigma^* \end{cases}$$

is recursive (because $|x|$ depends on $|x''|$) and hence can be programmed immediately in C++:

```
#include <iostream>
```

```
int len(char * p)
{
    if (p[0] == '\0') // BASE CASE
    {
        return 0;
    }
    else // RECURSIVE CASE
    {
        return 1 + len(p + 1);
    }
}

int main()
{
    char x[] = "abc";
    std::cout << len(x) << std::endl;
    return 0;
}
```

This is the reason why recursive thinking is so important. A recursive fact can be proven to be absolutely true using Math because of the presence of mathematical induction. After it's been proven, you can immediately program it using recursion.

On the other hand,

$$|x| = \text{count of the number of symbols in } x$$

is more like performing a scan of the characters manually and therefore looks executing a loop:

```
#include <iostream>

int len(char * p)
{
    int count = 0;
    while (p[count] != '\0')
    {
        count++;
    }
    return count;
}

int main()
```

```
{
    char x[] = "abc";
    std::cout << len(x) << std::endl;
    return 0;
}
```

There is no corresponding mathematical concept of loops and therefore no corresponding mathematical proof technique to prove correctness of algorithms in a concise manner.

Of course for something as simple of the above, it's no big deal either way. However for really complex and critical systems, correctness must be proven or at least verified to some extent.

Read the above two definitions and then read their corresponding code. Make sure you really understand the difference between the two. Learn to love recursion.

Now for the reverse function. The **reverse function** $(\cdot)^R$ on a word x is defined as follows:

R1 If $x = \epsilon$, we define
$$\epsilon^R = \epsilon$$

R2 If $x \neq \epsilon$, then $x = x' \cdot x''$ where $x' \in \Sigma$, i.e., x' is a symbol in our alphabet Σ and $x'' \in \Sigma^*$, i.e., x'' is a word over Σ (this is by C8). We define
$$x^R = (x'')^R \cdot x'$$

As an example, let me compute $(abb)^R$.

$$\begin{aligned}
(abb)^R &= (a \cdot bb)^R \\
&= (bb)^R \cdot a && \text{by R2} \\
&= \big((b \cdot b)^R\big) \cdot a \\
&= \big(b^R \cdot b\big) \cdot a && \text{by R2} \\
&= \big((b \cdot \epsilon)^R \cdot b\big) \cdot a \\
&= (\epsilon)^R \cdot b \cdot b \cdot a && \text{by R2} \\
&= \epsilon \cdot b \cdot b \cdot a && \text{by R1} \\
&= bba
\end{aligned}$$

In terms of C++, using the above recursive definition of the reverse function we have this (I'm using C++ strings for this example since cutting and concatenating strings are easier with the C++ string class):

```
#include <iostream>
#include <string>

std::string reverse(const std::string & s)
{
    if (s == "") // BASE CASE
    {
        return "";
    }
    else // RECURSIVE CASE
    {
        char y = s[0];
        std::string z = s.substr(1);
        return reverse(z) + y;
```

```
    }
}

int main()
{
    std::string x("abc");
    std::cout << reverse(x) << std::endl;
    return 0;
}
```

The other definition of reverse, i.e. "the reverse of x is the word which is the same as x but with the symbols in reverse order" is not precise mathematically. The code would look something like this:

```
#include <iostream>
#include <string>

std::string reverse(const std::string & s)
{
    std::string t = "";
    for (size_t i = 0; i < s.length(); i++)
    {
        t.push_back(s[s.length() - i - 1]);
    }
    return t;
}

int main()
{
    std::string x("abc");
    std::cout << reverse(x) << std::endl;

    return 0;
}
```

Let Σ be an alphabet. Let x and y be words in Σ^*. We want to prove that

$$P: \text{ If } x, y \text{ are words in } \Sigma^*, \text{ then } (xy)^R = y^R x^R$$

Instead of proving P directly, we will prove this by mathematical induction, inducting on the length of x. Therefore we let $P(n)$ be this statement:

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ with } |x| = n, \text{ then } (xy)^R = y^R x^R$$

Since the length of x can be any integer $n \geq 0$, our base case is when $n = 0$.

Q1. We know that if $x = \epsilon$, then $|x| = 0$. Now for the converse:

Prove the following: Let x be a word in Σ^*. If $|x| = 0$, then $x = \epsilon$.

SOLUTION. We will prove that if $x \neq \epsilon$, then $|x| \neq 0$.

$$\begin{aligned}
& x \neq \epsilon \\
\therefore\ & x = x'x'' \text{ for some } x' \in \Sigma, x'' \in \Sigma^* && \text{by } \boxed{?} \\
\therefore\ & |x| = |x'x''| \text{ for some } x' \in \Sigma, x'' \in \Sigma^* \\
\therefore\ & |x| = 1 + |x''| \text{ for some } x' \in \Sigma, x'' \in \Sigma^* && \text{by } \boxed{?} \\
\therefore\ & |x| \geq 1 \text{ for some } x' \in \Sigma, x'' \in \Sigma^*
\end{aligned}$$

This implies that $|x| \geq 1$ and therefore $|x| \neq 0$.

Therefore we conclude that if $|x| = 0$, then $x = \epsilon$. □

Note. The less formal (but equally rigorous) way to write the proof is as follows:

Suppose $x \neq \epsilon$, then by $\boxed{?}$, $x = x'x''$ for some $x' \in \Sigma$ and $x'' \in \Sigma^*$. In that case

$$\begin{aligned}
|x| &= |x' \cdot x''| \\
&= 1 + |x''| && \textit{by } \boxed{?} \\
&\geq 1
\end{aligned}$$

which implies that $|x| \neq 0$.

Note. Some basic facts are implicitly used.

1. If $x = y$ then $|x| = |y|$.
2. If $n \geq 0$ then $n + 1 \geq 1$.
3. If $n \geq 1$ then $n \neq 0$.
4. $\exists x(P)$ is the same as P if P does not contain variables. For instance "there exists an x such that the sky is blue" is the same as "the sky is blue". (Check existential generalization in your discrete math book.)

Note. Note that in this case instead of proving $P \implies Q$, I prefer to prove $\neg Q \implies \neg P$. Of course the two statements are the same, i.e.;

$$(P \implies Q) \equiv (\neg Q \implies \neg P)$$

How do you decide which one to prove: the left or the right?

To prove $P \implies Q$ directly means that I have to start with P. I would then look at all the facts that begins with P.

If I want to prove $\neg Q \implies \neg P$, I would have to look at all the facts that I know that begins with $\neg Q$.

I would compare the two above and see which one gives me more facts to work with. Sometimes it's obvious. Sometimes it's not. You might have a red herring where one gives you more facts, but they end up (after a few steps of deduction) leading up to a deadend. Sometimes either way work fine.

For the above case, it comes down to which of the following gives me more tools to work with:

1. $|x| = 0$, or
2. $x \neq \epsilon$

Q2. Let x be a word in Σ^*. Then

$$|x| \neq 0 \iff x \neq \epsilon$$

SOLUTION. We already know that $|x| = 0 \iff x = \epsilon$ (see Q1). Hence

$$|x| \neq 0 \iff x \neq \epsilon$$

(You don't have to prove anything here. I've done everything.)

Let's get back to the main problem. Recall that we are trying to prove

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ with } |x| = n, \text{ then } (xy)^R = y^R x^R$$

Let's us go for the base case.

Q3. Prove that $P(0)$ is true.

SOLUTION. Let x, y be words in Σ^* with $|x| = 0$. We have the following:

$$
\begin{aligned}
& |x| = 0 && \\
\therefore\ & x = \boxed{?} && \text{by Q1} && \text{(a)} \\
\therefore\ & (xy)^R = \left(\boxed{?}y\right)^R && \\
\therefore\ & (xy)^R = y^R && \text{by } \boxed{?} \\
\therefore\ & (xy)^R = y^R\epsilon && \text{by } \boxed{?} \\
\therefore\ & (xy)^R = y^R\boxed{?}^R && \text{by } \boxed{?} \\
\therefore\ & (xy)^R = \boxed{?}^R\boxed{?}^R && \text{by (a)}
\end{aligned}
$$

Hence $P(0)$ is true. □

One note about proper math writing. If you want to write

$$
\begin{aligned}
& x = 1 + 2 + 3 \\
\therefore\ & x = 3 + 3 \\
\therefore\ & x = 6
\end{aligned}
$$

then you should write this instead:

$$
\begin{aligned}
x &= 1 + 2 + 3 \\
&= 3 + 3 \\
&= 6
\end{aligned}
$$

The first reads as "x is $1 + 2 + 3$, therefore x is $3 + 3$, therefore x is 6". The second reads as "x is $1 + 2 + 3$, which is $3 + 3$, which is 6."

Recall that we are trying to prove

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ with } |x| = n, \text{ then } (xy)^R = y^R x^R$$

We are done with the base case. The only thing left is the inductive case. Note that there are two forms of mathematical induction. To prove that $P(n)$ is true for all $n \geq 0$. You can prove the following two statesments hold:

1. $P(0)$ is true
2. If $P(n)$ is true, then $P(n+1)$ is true.

Or you can prove the following holds:

1. $P(0)$ is true
2. If $P(0), P(1), ..., P(n)$ are true, then $P(n+1)$ is true.

The second form assumes that $P(0), P(1), ..., P(n)$ are all true so that you have more information to work with. That's why the second form of mathematical induction is called the strong form of induction. The first form is called the weak form. It turns out that they are equally powerful/valid. We will be using the strong form for the proof of our inductive case.

Q4. Recall that

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ with } |x| = n, \text{ then } (xy)^R = y^R x^R$$

Let $n \geq 0$. Assume that $P(k)$ is true for $k = 0, 1, \ldots, n$. Prove that $P(n+1)$ is true.

SOLUTION. Let x, y be words in Σ^* with $|x| = n + 1$.

We have the following:

$$\begin{aligned}
& |x| = n + 1 && && \text{(a)} \\
\therefore\ & |x| \geq 1 \\
\therefore\ & |x| \neq 0 \\
\therefore\ & x \neq \boxed{?} && \text{by Q2} \\
\therefore\ & x = x'x'' \text{ for some } x' \in \Sigma,\ x'' \in \Sigma^* && \text{by } \boxed{?} && \text{(b)} \\
\therefore\ & |x| = 1 + |x''| \text{ for some } x'' \in \Sigma^* && \text{by } \boxed{?} \\
\therefore\ & n + 1 = 1 + |x''| \text{ for some } x'' \in \Sigma^* && \text{by (a)} \\
\therefore\ & |x''| = n \text{ for some } x'' \in \Sigma^*
\end{aligned}$$

Therefore for some $x' \in \Sigma$ and $x'' \in \Sigma^*$ we have the following:

$$\begin{aligned}
(xy)^R &= \left(\boxed{?}x''y\right)^R && \text{by (b)} \\
&= (\boxed{?} \cdot x''y)^R \\
&= (x''y)^R \cdot (\boxed{?})^R && \text{by } P(1) \\
&= \left(y^R \cdot (x'')^R\right) \cdot (\boxed{?})^R && \text{by } P(n) \\
&= y^R \cdot \left((x'')^R \cdot (\boxed{?})^R\right) && \text{by } \boxed{?} \\
&= y^R \cdot \left(\boxed{?}x''\right)^R && \text{by } \boxed{?} \\
&= \boxed{?}^R \cdot \boxed{?}^R && \text{by (b)}
\end{aligned}$$

Hence $P(n+1)$ holds. □

Note. The intuition behind the proof is that given x of length $n + 1$, we cut it up into x' and x'' of lengths 1 and n. We then using the inductive hypothesis $P(1)$ and $P(n)$. Read over the proof again and make sure the see the strategy in the proof. Writing proofs formally is one thing. Understanding the strategy in a proof is another. Only by studying lots of proofs and understanding their strategy then will you really

understand how to construct convincing proofs of your own.

Altogether for the following statement:

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ with } |x| = n, \text{ then } (xy)^R = y^R x^R$$

we have shown that

1. $P(0)$ is true
2. If $P(0), P(1), \ldots, P(n)$ are true, then $P(n+1)$ is true.

By mathematical induction, $P(n)$ must be true for all $n \geq 0$. Therefore the following must be true:

If x, y are words in Σ^*, then

$$(xy)^R = y^R x^R$$

QED.

How to "see" proof by induction hiding in a statement

Sometimes when you are given a statement to prove, it's not always clear that the proof method to be used in induction. (In fact the statement should be proven using another method. Or the statement can be proven in many different ways, including induction.) Of course you have seen the proof of the statement

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

(using induction) so many times, that you stop thinking about it altogether. In fact, in the above, the "n" is staring at you. So you would think about proof by induction right away.

Now if you look at the statement that we are proving in this assignment:

$$\text{If } x, y \text{ are words in } \Sigma^*, \text{ then } (xy)^R = y^R x^R$$

you actually don't see any n at all!!! In this case, you have to change the above to

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ and } |x| = n, \text{ then } (xy)^R = y^R x^R$$

Note that you can also use this proposition instead:

$$P(n): \text{ If } x, y \text{ are words in } \Sigma^* \text{ and } |y| = n, \text{ then } (xy)^R = y^R x^R$$

(You should try to prove this by induction.)

Here's another example where you need to convert the problem into a form that make proof by induction possible. Consider the Euler formula:

$$\text{If } G \text{ is a connected planar graph, then } v_G - e_G + f_G = 2$$

Here v_G and e_G are the number of vertices and edges of G and f_G is the number of faces in the planar drawing of G. Again, you don't see any "n". Well actually there are (at least) *two* integers in the above statement. See it?

Here's one way of rephrasing the above:

$$P(n): \text{If } G \text{ is a connected planar graph and } v_G = n, \text{ then } v_G - e_G + f_G = 2$$

And here's another:

$$P(n): \text{If } G \text{ is a connected planar graph and } e_G = n, \text{ then } v_G - e_G + f_G = 2$$

In general you have at least two choices on how to prove a statement on graphs. Of course in the above case, since the graph G is planar, it also has an f_G.

In the case of Euler's formula, which is better? Induct on number of vertices or induct on number of edges? You should try both.

A string $x \in \Sigma^*$ is a **palindrome** (or it is **palindromic**) if

$$x = x^R$$

Q5. Prove that $x \in \Sigma^*$ is palindromic iff one of the following holds:

(a) $|x| = 0$
(b) $|x| = 1$
(c) $x = x'x''x'$ where $x' \in \Sigma$ and $x'' \in \Sigma^*$ is palindromic (this is when $|x| \geq 2$).

Prove this by induction.

SOLUTION.

Some help: Let $Q(n)$ be the statement $x = x^R, |x| = n$ and $R(n)$ be the statement if $n = 0$, then $|x| = 0$ or if $n = 1$, then $|x| = 1$ or if $n \geq 2$, then $x = x'x''x'$ where $x' \in \Sigma$ and $x'' \in \Sigma^*$. Now define $P(n)$ to be the statement

$$P(n) = [Q(n) \longleftrightarrow R(n)]$$

Prove that $P(n)$ holds for $n \geq 0$. You might want to show $P(0)$ and $P(1)$ are both true first.

Q6. The following question is a DIY question (i.e., you don't turn it in).

1. For writing a `bool is_palindrome(const std::string &)` function, which is better, the definition or the equivalent (a)-(b)-(c) formulation in Q5? Why?
2. Implement `is_palindrome(const std::string &)` using the better palindrome check. (Hint: The better check when converted to code will use recursion.)