Problem Set 2

Due: February 20

Reading:

- Chapter 3.6. *Predicate Formulas*,
- Chapter 4. *Mathematical Data Types* through 4.2. *Sequences* in the course textbook.

Problem 1.
A *formula of set theory* is a predicate formula that only uses the predicate "$x \in y$." The domain of discourse is the collection of sets, and "$x \in y$" is interpreted to mean that the set x is one of the elements in the set y.

For example, since x and y are the same set iff they have the same members, here's how we can express equality of x and y with a formula of set theory:

$$(x = y) ::= \forall z. (z \in x \text{ IFF } z \in y). \tag{1}$$

(a) Explain how to write a formula Members(p, a, b) of set theory that means $p = \{a, b\}$.

Hint: Say that everything in p is either a or b. It's OK to use subformulas of the form "$x = y$," since we can regard "$x = y$" as an abbreviation for a genuine set theory formula.

A *pair* (a, b) is simply a sequence of length two whose first item is a and whose second is b. Sequences are a basic mathematical data type we take for granted, but when we're trying to show how all of mathematics can be reduced to set theory, we need a way to represent the ordered pair (a, b) as a set. One way that will work[1] is to represent (a, b) as

$$\text{pair}(a, b) ::= \{a, \{a, b\}\}.$$

(b) Explain how to write a formula Pair(p, a, b), of set theory that means $p = \text{pair}(a, b)$.

Hint: Now it's OK to use subformulas of the form "Members(p, a, b)."

(c) Explain how to write a formula Second(p, b), of set theory that means p is a pair whose second item is b.

Problem 2.
Prove De Morgan's Law for set equality

$$\overline{A \cap B} = \overline{A} \cup \overline{B}. \tag{2}$$

by showing with a chain of IFF's that $x \in$ the left hand side of (2) iff $x \in$ the right hand side. You may assume the propositional version of De Morgan's Law:

$$\text{NOT}(P \text{ AND } Q) \text{ is equivalent to } \overline{P} \text{ OR } \overline{Q}.$$

[1] Some similar ways that don't work are described in problem 7.25 in the course textbook.

Problem 3.
A *binary word* is a finite sequence of `0`'s and `1`'s. For example, (`1`, `1`, `0`) and (`1`) are words of length three and one, respectively. We usually omit the parentheses and commas in the descriptions of words, so the preceding binary words would just be written as `110` and `1`.

The basic operation of placing one word immediately after another is called *concatentation*. For example, the concatentation of `110` and `1` is `1101`, and the concatentation of `110` with itself is `110110`.

We can extend this basic operation on words to an operation on *sets* of words. To emphasize the distinction between a word and a set of words, from now on we'll refer to a set of words as a *language*. Now if R and S are languages, then $R \cdot S$ is the language consisting of all the words you can get by concatenating a word from R with a word from S. That is,

$$R \cdot S ::= \{rs \mid r \in R \text{ AND } s \in S\}.$$

For example,

$$\{\texttt{0}, \texttt{00}\} \cdot \{\texttt{00}, \texttt{000}\} = \{\texttt{000}, \texttt{0000}, \texttt{00000}\}$$

Another example is $D \cdot D$, abbreviated as D^2, where $D ::= \{\texttt{1}, \texttt{0}\}$ is just the two binary digits.

$$D^2 = \{\texttt{00}, \texttt{01}, \texttt{10}, \texttt{11}\}.$$

In other words, D^2 is the language consisting of all the length two words. More generally, D^n will be the language of length n words.

If S is a language, the language you can get by concatenating any number of copies of words in S is called S^*—pronounced "S star." (By convention, the empty word, λ, always included in S^*.) For example, $\{\texttt{0}, \texttt{11}\}^*$ is the language consisting of all the words you can make by stringing together `0`'s and `11`'s. This language could also be described as consisting of the words whose blocks of `1`'s are always of even length. Another example is $(D^2)^*$, which consists of all the even length words. Finally, the language, B, of *all* binary words is just D^*.

A language is called *concatenation-definable* (*c-d*) if it can be constructed by starting with finite languages and then applying the operations of concatenation, union, and complement (relative to B) to these languages a finite number of times.[2] Note that the $*$-operation is *not* allowed. For this reason, the c-d languages are also called the "star-free languages," [32] in the course textbook.

[2] We can assign to each c-d language a *count* which bounds the number of the allowed operations (Union, Concatenation, and Complement) it takes to make it.

Since finite languages are given to be c-d, they are the 0-count languages. For example,

- {00, 111},
- the words of length $\leq 10^{10}$, and
- the empty language, ∅,

are all 0-count.

finite languages

$R \to (R_1 - R_2 \cdots\cdots R_n)$ n n

$S \to (S_1 - S_2 \cdots\cdots S_n)$

We get a 1-count language by applying one of the operations to a 0-count language. So applying the complement operation to each of the above 0-count languages gives the following 1-count languages:

- $\overline{\{00, 111\}}$, the language of all binary words except `00` and `111`,
- the words of length $> 10^{10}$, and
- the language B of all words.

These languages are all infinite, so none of them are 0-count.

Notice that you don't get anything new by using the Union operation to combine two 0-count languages, since the union of finite sets is finite. Likewise, you don't get anything new by concatenating two 0-count languages because the Concatenation of two finite languages is finite—if R and S are finite languages respectively containing n and m words, then $R \cdot S$ contains at most mn words. (Exercise, give an example where $R \cdot S$ contains *fewer* than mn words.)

So the 1-count languages that are not 0-count are precisely those that come from complementing a finite language. That is, they are the languages that include all but a finite number of words.

We can apply Concatenation to a 0-count and a 1-count language to get a 2-count language. For example,

$$\{\texttt{00}, \texttt{111}\} \cdot B$$

Lots of interesting languages turn out to be concatenation-definable, but some very simple languages are not. This problem ends with the conclusion that the language $\{\texttt{00}\}^*$ of even length words whose bits are all $\texttt{0}$'s is not a c-d language.

(a) Show that if R and S are c-d, then so is $R \cap S$.

Now we can show that the set B of all binary words is c-d as follows. Let u and v be any two different binary words. Then $\{u\} \cap \{v\}$ equals the empty set. But $\{u\}$ and $\{v\}$ are c-d by definition, so by part (a), the empty set is c-d and therefore so is $\overline{\emptyset} = B$.

Now a more interesting example of a c-d set is language of all binary words that include three consecutive $\texttt{1}$'s:

$$B\texttt{111}B.$$

Notice that the proper expression here is "$B \cdot \{\texttt{111}\} \cdot B$." But it causes no confusion and helps readability to omit the dots in concatenations and the curly braces for sets with one element.

(b) Show that the language consisting of the binary words that start with $\texttt{0}$ and end with $\texttt{1}$ is c-d.

(c) Show that $\texttt{0}^*$ is c-d.

(d) Show that $\{\texttt{01}\}^*$ is c-d.

Let's say a language S is $\texttt{0}$-*finite* when it includes only a finite number of words whose bits are all $\texttt{0}$'s, that is, when $S \cap \texttt{0}^*$ is a finite set of words. A langauge S is $\texttt{0}$-*boring*—boring, for short—when either S or $\overline{S}$ is $\texttt{0}$-finite.

(e) Explain why $\{\texttt{00}\}^*$ is not boring.

(f) Verify that if R and S are boring, then so is $R \cup S$.

(g) Verify that if R and S are boring, then so is $R \cdot S$.

Hint: By cases: whether R and S are both $\texttt{0}$-finite, whether R or S contains no all-$\texttt{0}$ words at all (including the empty word λ), and whether neither of these cases hold.

(h) Explain why all c-d languages are boring.

So we have proved that the set $(\texttt{00})^*$ of even length all-$\texttt{0}$ words is not a c-d language.

---

is a 2-count language consisting of all the words that start with either 00 or 111. Notice that this language is not 0-count or 1-count, since both it and its complement are infinite.

Doing a concatenation of the 1-count language B with this 2-count language, gives a $1 + 1 + 2 = 4$-count language

$$B \cdot \{00, 111\} \cdot B$$

which consists of all the words that have either two consecutive $\texttt{0}$'s or three consecutive $\texttt{1}$'s. We don't know at this point whether this language is also 3-count, or even 2-count, because we haven't ruled out the possibility that it could be built using fewer than 4 operations (though we don't think it can).

Now doing a complement of this 4-count language give a 5-count language consisting of all the words in which

- every occurrence of $\texttt{0}$ is followed by a $\texttt{1}$, except for a possible $\texttt{0}$ at the end of the word, and also
- every occurrence of$\texttt{11}$ is followed by a $\texttt{0}$, except for a possible $\texttt{11}$ at the end of the word.

The c-d languages are precisely the languages that are n-count for some nonnegative integer n.