

Pete Manolios, CSU290 HWK2, Due Mon Feb 4 2008

---

Homework 2:

- You *have to* work in pairs for this homework, so find a partner right away. Try to work with someone you don't know.
- If you get stuck, get help right away.
- Even if you can't solve a problem, you can assume you have a solution for subsequent problems.
- Submit your homework via blackboard. Your submission should be a text file with the ACL2s code (including comments, test cases, etc.). The format of the file should be: a comment indicating the start of the code for a problem, and then your solution. In particular, do not include the text of this assignment in your homework submission.

You are developing a social networking Web site and your users are clamoring for new functionality. They want to use their friends to meet people they don't directly know. For example, Julia wants to meet Allan, but she doesn't know him. She'd like to know if it possible for one of her friends to introduce her to one of their friends, who introduces her to one of their friends, ..., until she gets introduced to Allan. In fact, she wants to know how many people she can meet this way and who they are. In this homework assignment we will develop functions to do this.

For those of you interested in a challenge, try to solve the above problem without reading further. Then, compare what you get with the solution this homework leads you to. (Of course, you will still have to do this homework assignment.)

This will essentially involve defining functions to manipulate sets and relations. For example, we will define functions to extract information from the most important information your social networking site maintains, the friend-of relation.

All the homework problems involve the definition of ACL2 functions. Use ACL2s in programming mode and use the design recipe described in class. Remember that this is the design recipe we saw in 211 extended the guidelines, checks, and recursive schemes we studied in class.

We start by developing functions for manipulating finite sets. A finite set is just a list of elements. So, it is a true-listp who elements satisfy allp. That is, it is a true-listp.

#### PROBLEM 1

Write the recognizer setp: a true-listp (all of whose elements satisfy allp).

END PROBLEM 1

Next, we define set membership. An element,  $e$ , is in a set,  $s$ , if  $e$  is one of the objects in the list  $s$ . For example, the elements of (list 3 2 1 3) are 1, 2, and 3, so 1 is in (list 3 2 1 3); similarly (list 1 2) is in (list 1 2 (list 1 2)), but (list 1 2) is not in (list 1 2) (since the only members of this list are 1, 2).

#### PROBLEM 2

Define the function set-memberp: it takes as input any object in the ACL2 universe and a set and checks if the object is in the

set.  
END PROBLEM 2

Next we define the notion of set equality. Note that set equality differs from =. For example, (list 1 1 2 2) represents the same set as (list 2 1), so they are set equal, but not =. Remember that sets are equal iff they have exactly the same members.

PROBLEM 3  
Define the function set-subsetp. Set-subsetp takes two sets as input, say x and y, and returns t if x is a subset of y, i.e., if every element in x is also in y.  
END PROBLEM 3

PROBLEM 4  
Define the function set-equalp. Set-equalp takes two sets as input and returns t if they have the same elements and nil otherwise. Hint: there is a simple, non-recursive definition.  
END PROBLEM 4

PROBLEM 5  
Define the function set-union: it takes two sets as input and returns a set corresponding to their union. That is, an element is in (set-union x y) iff it is in x or in y.  
END PROBLEM 5

PROBLEM 6  
Define the function set-size: it takes a set as input and returns a natural number corresponding to the size of the set. The size of a set is the number of distinct elements it has. For example (list "John" "Bob" "Alice" "Allen" "Bob" "Alice") has 4 distinct elements, so its size is 4.  
END PROBLEM 6

Notice that the above functions work on sets of any type. Often we are interested in sets of type X. A set of type X is a set, all of whose elements are of type X. For example, a set of type stringp is a set, all of whose elements satisfy the stringp predicate. Recall that stringp is the primitive function that recognizes strings.

PROBLEM 7  
Write the recognizer string-setp: a set of strings.  
END PROBLEM 7

We will now define relations. A relation is a set of conses. There are no restrictions on the cons pairs, i.e., each cons pair in a relation has a car and cdr that satisfy allp. For example (list (cons "a" "b") (cons "a" "c") (cons "b" "c") (cons "a" "c")) is the relation {<"a", "c">, <"b", "c">, <"a", "b">}

PROBLEM 8  
Define the recognizer relationp that recognizes relations.  
END PROBLEM 8

PROBLEM 9  
Define the recognizer string-relationp, which recognizes relations on strings. That is, each cons in a relation has a car that satisfies stringp and has a cdr that satisfies stringp.  
END PROBLEM 9

Remember, that you are developing a social networking Web site and your users are clamoring for the following functionality: they want to use their friends to meet people they don't directly

know. For example, Julia wants to meet Allan, but she doesn't know him. She'd like to know if it possible for one of her friends to introduce her to one of their friends, who introduces her to one of their friends, ..., until she gets introduced to Allan. In fact, she wants to know how many people she can meet this way and who they are. The goal in the remaining problems is to develop functions to do exactly this.

**\*\*PROBLEM 10**

Write a function that given the friend-of relation for your social networking Web site (your function should work for any relation on strings), and a set of users returns the set of users that they can meet via a sequence of introductions, as per the above discussion. This set should include the initial users. Here is one way of doing this. Define the image of a set,  $s$ , under relation  $r$  to be the set  $\{ y \mid (x . y) \text{ is in } r \text{ and } x \text{ is in } s \}$ , that is, it is the set of elements that are related to some element of  $s$ . (Hint: you might want to define a function to do this.) We'll denote the image of  $s$  under  $r$  as  $r(s)$ , and we'll use  $\cup$  to denote set union. Now, start with the initial set, let's call that  $s_0$ , and keep taking images under  $r$ , to obtain  $s_1 = s_0 \cup r(s_0)$ ,  $s_2 = s_1 \cup r(s_1)$ , ...,  $s_n = s_{(n-1)} \cup r(s_{(n-1)})$ . Note that  $s_0$  is a subset of  $s_1$ , which is a subset of  $s_2$ , ... . So, the sequence is increasing and we should stop when the sets stop growing. This kind of calculation is called a "fixpoint" calculation. Also, this is actually a common operation on relations and it has a name: it is called the reflexive, transitive closure of a relation, so call the function `relation-trans`. The stars next to the problem number indicate that this is more challenging than the previous problems.

**\*\*END PROBLEM 10**

The solutions to our original problems now fall out rather easily. It is worth pointing out that as you progress through your undergraduate computer science careers, you should be able to think about problems at a higher level of abstraction, and this exercise should seem trivial: you should be able to read the initial description and say: I can solve that problem by just computing the reflexive, transitive closure of the friend-of relation.

**PROBLEM 11**

Write the function `possible-friends`: it is given as input the name of a user, a string (say "Julia") and the friend-of relation (your function should work on any relation over strings). The function should return the set of users that Julia can meet via a sequence of introductions, as per the above discussion. Run it on a relation with at least 10 people. Write up to a few paragraphs explaining what you observed when you tested your function. What, if any, changes to your code are suggested? Show how you would modify your solutions above to address any issues. Hint: given the solution to problem 12, this should be an easy, non-recursive definition.

**END PROBLEM 11****PROBLEM 12**

It is now easy to solve Julia's first request. Define a function, `num-of-possible-friends` that given a string (the user name) and a relation (the friend-of relation) returns the number of people the user can meet via a sequence of introductions amongst friends. This is just the set-size of the `possible-friends` function applied to user, say "Julia". Try it on a few examples with at least 10 people.

**END PROBLEM 12**