CS 525V: Modeling and Verifying Simple Designs in ACL2

due: March 25, 2001; 11:59pm (electronically)

These exercises are intended to give you some practice modeling designs and specifying properties in ACL2. Turn in a single ACL2 file for your final submission.

Exercise 1 (**Modeling a Banking System**) A simple model of a banking system consists of the set of transactions that have been performed on accounts in the bank. Consider a system which supports three kinds of transactions:

1. deposits, which consist of an account number and the amount to deposit,
2. checks (drawn on an account), which consist of an account number, the amount of the check, and the check number, and
3. fees (assessed to an account), which consist of an account number and the amount of the fee.

From the list of transactions in the system, we can compute the total amount held in each account by adding up the deposit amounts and subtracting the amounts for checks and fees.

Develop an ACL2 program for this model. In addition, write the following two functions over your model:

1. A function *account-value*, which takes an account number and a flat list of transactions and returns the sum of the deposits minus the sum of the checks and fees.
2. A function *sort-accounts*, which takes a list of transactions and sorts the list in increasing order of account number (the order within an account number is irrelevant). [Warning: use plain insertion sort here – the fancier your sorting algorithm, the harder the proofs will be to complete!]

Exercise 2 (**Verifying the Banking System**)
Prove the following properties about your banking system model using ACL2.

1. Prove that if you add a deposit transaction to the transaction list, then the corresponding account value increases by the amount of the deposit.
2. Using the *account-value* function, prove that if you add a fee transaction for an account, then the value in all other accounts remains the same.
3. Define a function *okay-listp*, which takes a list of items (not necessarily transactions) and returns true iff no two items that are equal are separated by a non-equal item in the list. For example:
   - `(okay-listp (list 'a 'b 'c))` → `t`
   - `(okay-listp (list 'a 'a 1 1 1 'b 'c 2 2))` → `t`
   - `(okay-listp (list 1 2 1))` → `nil`

   Create a version of the *sort-accounts* function that sorts a list of numbers instead of a list of transactions (call it *sort-nums*). Prove that the output of *sort-nums* satisfies *okay-listp* (i.e., that *okay-listp* returns true on the output of *sort-nums*).
4. The previous problem is a simple version of the following exercise on the banking system: Develop a predicate *sorted-transactionsp* to recognize the desired output of *sort-accounts*. Prove that your *sort-accounts* function produces outputs accepted by *sorted-transactionsp*. This problem turns out to be harder than the previous problem. Why? (Include a comment containing your answer to this problem.)