Advanced Programming Assignment 2 - Number Sets in Prolog

Philip Munksgaard & Ronni Elken Lindsgaard

October 8, 2012

1 Code documentation

We have implemented the code in a prolog module file called sets.pl. The predicate `less/2` recursively checks whether `X` $<$ `Y`. `checkset/1` lifts off the first to elements in a list and checks that the first is the lesser one. The second element is pushed onto the list again and recursion is used to run through all elements of the list.

The `ismember/3` predicate assumes that we get a proper natural number set as defined in the assigment, and mainly uses pattern matching in order to decide the result. The logic for each rule is described here

1. If the list is empty, the result should only be true if the third parameter is **no**.
2. If the element in the list matches the element we are looking for, we should return **yes**.
3. If the head of the list is bigger than the element we are looking for, so are all the rest of the elements of the list, since the head is the smallest one (given that the list is a proper representation of a set of natural numbers), thus the third parameter should be **no**.
4. If the head of the list is smaller than the element we are looking for, the element still might be in the list, so we call `ismember/3` recursively on the rest of the list.

If one of the sets s_1 or s_2 are empty, then `union/3` returns the non-empty set. We believe it is bad practice to return a illegitimate set and therefore make a call to `checkset/1`. We recurse through the two lists and makes sure that `XS` always contain the set with the lesser numbers.

`intersection/3` works just like `union/3` except that we only match subjects in t_3 if they are present in both t_1 and t_2.

1.1 The behavior of `ismember(N, [s(z), s(s(s(z)))], A).`

The second rule is matched at first, the head `s(z)` is found and `A = yes`. Then prolog tries to find all possibilites for `N`. The query always succeeds since it is always possible to find a suitable `N` where `A = no`. When `N = s(s(s(z)))` the fourth rule is matched at first which results in a recursive call that finally results in matching of the second rules which makes `A = yes`.

2 Assessment of our solution

The program matches the requirements and we believe that we have implemented a good solution. We have made excessive testing with plunit that covers a wide range of possibilities that should succeed and also what should fail, specifically testing the examples from the assignment text. We believe that we have made reasonable and well founded judgment for behavior that was not specified in the assignment. Our code is comprehensible and clear. Due to the small codebase, and the documentation above, we have omitted inline comments.