# Assignment 2: Number Sets in Prolog
# Advanced Programming

Guðmundur Páll Kjartansson
Jón Gísli Egilsson

October 9, 2012

## 1 The code

For this assignment we implemented a few predicates to work with natural number sets in Prolog. The first predicate we are asked to implement is `less/2`. Our way of implementing this predicate was like so:

```
less(z, s(_)).
less(s(X), s(Y)) :- less(X,Y).
```

Where we can see the base case being `less(z, s(_)).` and to check other cases we check recursively if the first term is less than the second term until we reach the base case where we return `true` or another case where the left term is not the only term which is `z` where `false` is returned.

The second predicate we implemented is `checkset/1`:

```
checkset([_]).
checkset([X1,X2 | XS]) :- less(X1,X2), checkset([X2|XS]).
```

Here we check if the first two elements of the list are in ascending order with our `less` predicate from above and then we recursively check if the rest of the list is a set by calling `checkset` again. The base case being when the list includes only one element, then it's always a set, no matter which natural number it includes.

The the third predicate we had to implement is `ismember/3`:

```
ismember(_,[],X2) :- X2 = no.
ismember(X,[X|_],X2) :- X2 = yes.
ismember(X1,[_|XS],X3) :- ismember(X1,XS,X3).
```

In this predicate we check if the head of a list is the number we in the first parameter. If that is not the case we recursively ask if the number is a member of the rest of the list. The base case here being when we ask if a number is a member of the empty list, which always returns `X2=no`. Here we also demonstrate the following execution:

```
?- ismember(N, [s(z), s(s(s(z)))], A).
N = s(z),
A = yes ;
N = s(s(s(z))),
A = yes ;
A = no.
```

As we see both members `N` are found with `A = yes`.

The fourth predicate we implemented is `union/3`:

```
union([],X,X).
union(X,[],X).
union([X1|X1s],[X2|X2s],[X1|X3s]) :-
  less(X1,X2),
  union(X1s,[X2|X2s],X3s).
union([X1|X1s],[X2|X2s],[X2|X3s]) :-
  less(X2,X1),
  union([X1|X1s],X2s,X3s).
union([X|X1s],[X|X2s],[X|X3s]) :-
  union(X1s,X2s,X3s).
```

Here we take in three lists where the first two are sets and the third one is the union of the two sets. What we do here is check if either of the heads of the two sets is the head of the union. If so we remove the head from the union and the set that contained the head (possibly both sets) and check again. Since the lists are ordered in ascending order we know we can do this. If however neither of the set heads is the head of the union then we know that this union is the wrong one and we return a false. The base cases here are if we have the empty set as one of the sets and `X` the other the union is of course `X`.

The fifth and final predicate we implemented is `intersection/3`:

```
intersection([],_,[]).
intersection(_,[],[]).
intersection([X|X1s],[X|X2s],[X|X3s]) :-
  intersection(X1s,X2s,X3s).
intersection([X1|X1s],[X2|X2s],X3) :-
  less(X1,X2),
  intersection(X1s,[X2|X2s],X3).
intersection([X1|X1s],[X2|X2s],X3) :-
  less(X2,X1),
  intersection([X1|X1s],X2s,X3).
```

This one works more or less like the `union/3` predicate above in the way how we check if the third argument is the intersection of the two given sets. The base cases are where we have the empty set and some other (could be the same) set we know the intersection of those two are the empty set.

## 2 Assessment of the code

Much time was spent developing these predicates so that they are self contained, that is no auxiliary predicates were needed. We think this implementation is quite elegant although some things could be done better. For example we sometimes get the same answer when trying to derive things, as we see here:

```
?-  union(X, [s(z)], [z,s(z)]).
X = [z] ;
X = [z, s(z)] ;
X = [z, s(z)] ;
false.
```

The code correctness is somewhat talked about in the chapter above and also with unit testing. The unit tests can be found in the `natural.pl`.