Exercise 6: Advanced Types and Collection Types

1. Create a new PVS theory named "adv_types" with `N: posint` as a theory parameter.

2. Declare a type `S` which consists of even integers greater than or equal to 3*N. Note: the predicate `even?` is predefined in the PVS prelude. (You can view the prelude with `M-x vpf`.)

3. Develop a lemma in PVS that means "if s is of type S then there is an integer i with 2*i=s and i is greater than or equal to floor(N*3/2)." Prove this lemma. These commands will probably be used: `(typepred "s!1")`, `(expand "even?")` and `(inst?)` which guesses an instantiation.

4. Declare two parameterized types, `S3(cc,dd)` and `S4(cc,dd)` with `S3(cc,dd)` containing all the real numbers between (and including) cc and dd, and `S4(cc,dd)` containing all the real numbers between (but excluding) cc and dd.

5. Add a judgement expressing the fact that `S4(cc,dd)` is a subtype of `S3(cc,dd)` Issue `M-x show-tccs` to see if the obligation generated by this judgement. Issue `M-x typecheck-prove` to see if PVS can prove this obligation automatically (hint: it should).

6. Add `T: TYPE` as a theory parameter.

7. Develop a lemma in PVS that means "if A and C are disjoint sets of type T then

$$(A \cup B) \cap (C \cup B) = B.$$

   Hint: the prelude contains a predicate `disjoint?`

8. Prove the lemma in number 7. Hint: try using `(decompose-equality)`, `(install-rewrites :defs t)`, and `(iff)`.

9. Develop a lemma for

$$A \subseteq B \ \text{ IFF } \ B = A \cup (B \setminus A)$$

   Prove this formula. Hint: first split it into two subgoals with `(prop)`.

10. Issue `M-x spt` to make sure everything has been proved. Notice that as you added new text to the `.pvs` file, previously proved lemmas become `unfinished`. Issue `M-x prt` to reprove all of the unfinished lemmas.

11. Write a lemma that states that if you add an element to a finite set (not already in it) and then remove that element from the set, the cardinality of the resulting set is the same as the original set. Prove this lemma using lemmas `card_remove` and `card_add` in the `finite_sets` theory in the prelude. Do not expand `remove` or `add`.

12. Write a lemma that states that the cardinality of the intersection of two finite sets is less than or equal to the cardinality of their union. Prove this using lemmas available in the theory `finite_sets`.

Note. It is usually not a good idea to put lemmas in a theory that do not use all of the theory parameters. For example the lemma created in step 7 does not depend upon the N parameter. Therefore, it should be in a separate theory. To simplify this exercise, we just threw them all in one theory.

If you finish early,

1. Develop and prove a lemma that means "If x is in type `S3(c,d)` and $y \leq c$ and $d \leq z$ then `x` is in the set `{t: S3(y,z)| true}`" Note: x in `{t: S3(y,z)| true}` can be abbreviated as `fullset[S3(y,z)](x)`

2. Assuming `A`, `B` and `C` are sets of type `T` prove:

```
LEMMA A = C IMPLIES
         intersection(union(A,B),union(C,B)) = union(A,B)
```

3. Assuming `FA` and `FB` are finite sets of type `T` and `k` is a natural number prove:

```
LEMMA card(union(FA,FB)) = k AND card(intersection(FA,FB)) = k
          IMPLIES card(FA) = k AND card(FB) = k
```

4. Add the following to your theory. (This type is available in the `below_arrays` library, but for simplicity we will incorporate it directly.)

```
below_array: TYPE = [below(N) -> T]
```

5. Define a recursive function with the following signature

```
builder(AA: below_array, n:below(N)): RECURSIVE set[T] =
```

   which creates a set out of the `0` thru `n` elements of the array `AA`.

6. Prove `builder_lem` using `(induct "n")`:

```
n: VAR below(N)
AA: VAR below_array
builder_lem: LEMMA FORALL (i: upto(n)): builder(AA,n)(AA(i))
```

7. Define a function `build(AA: below_array): set[T]` in terms of `builder` that constructs a set from all N elements of the array AA.

8. Prove:

```
build_lem: LEMMA FORALL (i: below(N)): build(AA)(AA(i))
```

Advanced students might want to try:

1. Prove (Hint: The lemma `emptyset_is_empty?` in the prelude may be useful.):

```
x: VAR T

sl5: LEMMA subset?(C, singleton(x))  % **** tricky ****
                  IMPLIES C = singleton(x) OR  C = emptyset[T]
```

2. This is an exercise is understanding PVS syntax. Create a new type `letters: TYPE = {a,b,c,d,e,f}` and define the set of the type letters: `{a,b}`. Hint: `s: VAR set = {a,b}` will not typecheck!

3. Develop a PVS lemma (but don't prove yet) for

$$\{choose(\{a,b\})\} \cup rest(\{a,b\}) = \{a,b\}$$

   Hint: you will need to use `singleton` to construct a singleton set from the value returned by choose.

4. Prove this formula using the `(decompose-equality)` and `(grind)` commands and nothing else!

5. Reprove the lemma without using `(grind)`. Hint: you can use the lemma `choose_rest_or` in the prelude. You will have to provide the appropriate type to the `(lemma ...)` command.