Theorem proving assignment 2020/2021 : 16

Jorge Paz Ruza
j.ruza@udc.es
Daniel Quintillán Quintillán
daniel.quintillan@udc.es

I. EXERCISE 1

A. Formalisation details

In the foremost exercise we are asked to formalize the concept of *Binary Relations* over a Set, and some properties of certain types of binary relations: reflexivity, transitivity, symmetry, and antisymmetry. To achieve this, we created an auxiliary theory, called *Properties*, where we generalize these properties to relations of any given type, for ease of use.

B. Proving process

We need to prove that a certain binary relation over $\mathbb{N}$ is an equivalence relation. Logically, we'll need to navigate through three subgoals to achieve this:

- **Reflexivity**: We must prove that every element of $\mathbb{N}$ is related with itself. Since the given binary relation is simple $(a, b \in \mathbb{N}, k \in \mathbb{Z}, \exists k : a = b + 3 * k)$, proving this requires a fairly trivial instantiation after a skolemization step.
- **Symmetry**: We must prove that $a\ R\ b \implies b\ R\ a$. This proof is mostly straightforward until the following step:

```
{-1}  EXISTS k: a!1 = b!1 + 3 * k
|-------
{1}   EXISTS k: b!1 = a!1 + 3 * k
```

  The fastest strategy here is first skolemizing the EXISTS in the antecedent, since then we can freely instantiate our EXISTS in the sequent. our instantiation for k will be *-k!1*, leaving:

```
[-1]  a!1 = b!1 + 3 * k!1
|-------
{1}   b!1 = a!1 + 3 * -k!1
```

  which is trivially true.
- **Transitivity**: Similarly to the previous subgoal, only the end of the proof is non-trivial. After skolemizing EXISTS in the antecedent, we end up with a need to perform a correct intantiation:

```
[-1]  a!1 = b!1 + 3 * k!1
{-2}  b!1 = c!1 + 3 * k!2
|-------
[1]   EXISTS k: a!1 = c!1 + 3 * k
```

  This instantiation will be k = *k!1+k!2*, which leaves us with the following situation.

```
[-1]  a!1 = b!1 + 3 * k!1
[-2]  b!1 = c!1 + 3 * k!2
|-------
{1}   a!1 = c!1 + 3 * (k!1 + k!2)
```

  As expected through basic algebraic manipulation, an *assert* command completes the proof.

II. EXERCISE 2

A. Formalisation details

In Exercise 2, we're asked to formalize the ideas of *surjective*,*injective* and *bijective* functions, alongside *function composition*, and prove certain properties about some specific given applications. Following the process applied in Exercise 1, we create auxiliary Theories *functs* and *cmp* to define these properties.

B. Proving process

1. 2 a

The first subsection demands proof that if the composition of two functions h and f is surjective and h is injective, f is surjective.

```
{-1}  FORALL (y: Z): EXISTS (x_1: X): h(f(x_1)) = y
{-2}  FORALL (x1: Y), (x2: Y): (h(x1) = h(x2) IMPLIES (x1 = x2))
|-------
{1}   FORALL (y: Y): EXISTS (x: X): f(x) = y
```

After expanding definitions and swapping statements within the sequent, we want to derive {1} from {-1} and {-2}. The first step is to skolemize the output of f(x) in {1} and instantiate the output of h(f(x)) to h(y!1) in {-1}. An existential quantifier is still preventing further operations involving {-1}, so a skolemization of the input x is required.The Skolem constant x!1 then instantiates the input of f in {1} to get to this point

```
[-1]  h(f(x!1)) = h(y!1)
[-2]  FORALL (x1: Y), (x2: Y): (h(x1) = h(x2) IMPLIES (x1 = x2))
  |-------z
[1]   f(x!1) = y!1
```

Now it's just a matter of instantiating {-2} correctly so it can contribute with {-1} in the inferral of {1}.
The reasoning starts at {-1}, which states that applying h to y!1 is equivalent to applying it to f(x!1). From this, we want to conclude that f(x!1)=y!1 applying {-2}, i.e. the injectivity of h. Ultimately, {2} needs to be instantiated with the two items whose equality we need to prove, y!1 and f(x!1).

```
[-1]  h(f(x!1)) = h(y!1)
{-2}  (h(y!1) = h(f(x!1)) IMPLIES (y!1 = f(x!1)))
  |-------
[1]   f(x!1) = y!1
```

From here, the proof can be trivially completed through *split flatten*, and *assert* commands.

2. 2 b

In this second section of the exercise we are asked to proof that the function f(x,y)=(2y,-x) is bijective. Logically, this means we will obtain two subgoals: one to proof surjectivity, and one to prove injectivity.

- **Injectivity**: in the process of proving the injectivity of the function, we apply basic logic manipulation until obtaining the following state:

  ```
  {-1}  2 * x1!1`2 = 2 * x2!1`2
  {-2}  -x1!1`1 = -x2!1`1
  |-------
  {1}   (x1!1 = x2!1)
  ```

which we can interpretate as a need to proof that, if the result of applying *f* is the same for two input (x,y) pairs, those pairs must be the same, which is of course the definition of injectivity.

To achieve this, we use the *decompose-equality* command, that lets us, just like the name says, decompose the equality in {1} as the separate equalities of each of the components of the pair, therefore trivially completing the proof of injectivity.

- **Surjectivity**: To prove the surjectivity of *f*, we obtain the following subgoal after expanding the definition:

```
|-------
{1}   FORALL (y_1: [real, real]):
EXISTS (x_1: [real, real]): (2 * x_1‘2, -x_1‘1) = y_1
```

Since we cannot work directly with x_1 and y_1 as bound vars, as their decomposition will be needed to carry the proof, we execute the command *(detuple-boundvars)*, leaving us, after an additional skolemization, with:

```
|-------
{1}   EXISTS (x_2: real), (x_3: real): (2 * x_3, -x_2) = (y!1, y!2)
```

Here we only need to "undo" the output of our function when doing the instiation. In this case, it's as simple as instantiating (x_2,x_3) to (-y!2,y!1/2), thus obtaining:

```
|-------
{1}   (2 * (y!1 / 2), --y!2) = (y!1, y!2)
```

which lets us end the proof by executing one last *assert*.

III. EXERCISE 3

A. Formalisation details

In the final exercise, we create an auxiliar *parity* theory containing the following formalisations:

```
even(x : nat) : bool =(EXISTS k : x =2*k)
odd(x : nat): bool = (EXISTS k: x = 2*k+1)
```

These functions are used to prove three statements about the relationship between even and odd numbers.

B. Proving process

All of the sections in this exercise were proven by induction.

1. 3 a

To prove that, for every natural number n, either itself or its successor is odd, we apply weak induction over the natural numbers, splitting our problem into two subgoals: the base case and the induction step.

```
|-------
{1}   odd(0) OR odd(0 + 1)
```

The base case (n=0) can be proven by expanding *odd*, which automatically "discards" odd(0). As for the expansion of odd(0+1), we just have to instantiate it with k=0.
The induction step

```
|-------
{1}   FORALL j: odd(j) OR odd(j + 1) IMPLIES odd(j + 1) OR odd(j + 1 + 1)
```

is first skolemized and then manipulated with *flatten* and *split* commands until two subgoals emerge from it. One of them is trivially true, so the other one will be discussed instead. After expandind "odd" we obtain the subgoal:

```
{-1}  EXISTS k: j!1 = 1 + 2 * k
  |-------
{1}   EXISTS k: j!1 = 2 * k
{2}   EXISTS k: 2 + j!1 = 1 + 2 * k
```

For {2} to be equivalent (and thus derived) from {-1}, k needs to be one unit more in {2} than it is in {-1}. To achieve this, we skolemize {-1} (which generates the skolem variable k!1) and instantiate k in {2} as k!1+1.. After that, the proof is trivially completed through the basic algebraic manipulation of *assert*.

2. 3 b and 3 c

The last subsections of exercise 3 are also solved by induction. However, since they generate a vast number of subgoals and the strategies required to approach them are simmilar to those applied in exercise 3 a, they will not be discussed in this report.