Using Why

Assuming file f.why contains some program and specification, running why -coq f.why will produce a Coq file f_why.v containing all the verification conditions, while gwhy f.why will open the graphical user interface.

1 McCarthy’s 91 function

McCarthy’s 91 function is the function $f$ from $\mathbb{Z}$ to $\mathbb{Z}$ defined by

$$f(n) = \begin{cases} f(f(n + 11)) & \text{if } n \leq 100 \\ n - 10 & \text{otherwise.} \end{cases}$$

1. Define function $f$ in Why. The Why syntax for a recursive function is

   ```why
   let rec f (n: int) : int = ...
   ```

2. $f(n)$ is 91 when $n \leq 100$ and $n - 10$ otherwise. Annotate $f$ accordingly and prove in Coq the generated VCs.

3. Prove in Coq the termination of $f$ by inserting the following variant

   ```why
   let rec f (n: int) : int { variant max (0, 101 - n) } = ...
   ```

   Since max is not a primitive function, you must introduce it with a logic and axiomatize it with an axiom.

2 Fibonacci function

1. Introduce the Fibonacci function $F$ with a logic and three axioms in Why. We recall that $F(0) = F(1) = 1$ and $F(n) = F(n - 1) + F(n - 2)$ for $n \geq 2$.

2. Define a recursive function $f_1$ computing $F$ (with a naive, i.e. exponential, algorithm). Prove its correctness and termination in Coq.

3. Define a function $f_2$ computing $F$ using a linear algorithm which maintains $F(n - 1)$ and $F(n)$ in two references. Prove its correctness and termination in Coq.

4. Define a third Why function $f_3$ computing $F(n)$, using the same linear algorithm but using a recursive function instead of a loop. Note how the loop invariant is naturally transformed into a precondition.

3 Minimum and maximum of an array

1. Fill the precondition of the following function so that the postcondition holds. Add an invariant and a variant so that the Why function is completely and automatically proved by Alt-Ergo.

   ```why
   let dummy_loop (n : int) =
   let i = ref 0 in while !i < n do i := !i + 2 done; !i
   { result = n or result = n + 1 }
   ```
2. Define an abstract datatype for representing pairs of values with generic types in Why. Realize it in Coq.

3. Fill the body of the following Why function, so that it returns the indexes of the minimum and maximum elements of its array argument. Note: the standard fast algorithm scans two elements at each step, so its complexity is $3n/2$ comparisons for an array of length $n$.

```plaintext
include "arrays.why"
let fast_minmax (t : int array) =
{ array_length(t) >= 1 }
mk_pair 0 0
{ forall i : int. 0 <= i < array_length(t) ->
  t[first(result)] <= t[i] <= t[second(result)]
}
```


The semantic of `for i = e1 to e2 do e3 done` can be specified as: $e_1$ and $e_2$ are evaluated only once (values $v_1$ and $v_2$); if $v_1 > v_2$, the loop is skipped, otherwise $e_3$ is evaluated iteratively with $i = v_1, v_1 + 1, \ldots, v_2$. Note that $i$ is visible only in $e_3$ and it is not writable.

1. Define a Coq function of type $\mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z}$ that is equivalent to the following Caml program. An auxiliary function (inductively defined on $\mathbb{N}$) has to be used.

```plaintext
let f a b =
  let d = ref 1 in
  for i = a to b do d := 19 * !d + i done;
  !d
```

2. Complete the following Hoare rules and explain why they are sound. Note that $a$ and $b$ are not expressions but constant integers.

\[
\frac{
\{(a > \ldots) \land \ldots\} \text{ for } i = a \text{ to } b \text{ do } s \text{ done } \{Q\}
}{
\{a \leq \ldots \land I(\ldots)\} \text{ for } i = a \text{ to } b \text{ do } s \text{ done } \{I(\ldots)\}
}\]

3. An induction principle `for_rec` is needed to prove in Coq programs using `for` loops. It has the following type:

```plaintext
forall (a b:Z), a <= b+1 ->
forall (P : Z -> Set),
  P a -> (forall i, a <= i <= b -> P i -> P (i+1))
  -> P (b+1).
```

Prove `for_rec` or define its value. One can use an auxiliary function inductively defined on $\mathbb{N}$, as was done in the first question.

4. By using `for_rec`, define in Coq a function `sqr` of type

\[
\forall z:Z, z \geq 0 \rightarrow \{ s:Z \mid s = z \times z \}
\]

that matches the following Caml program

```plaintext
let sqr z =
  let s = ref 0 in
  for i = 0 to z-1 do s := !s + 2*i + 1 done;
  !s
```