Alexandru Mihai Glontaru

Homework 4

Problem 1:

➔ `fun Euclidean Algorithm(m:int, n:int):int`

Function specifications:
- Precondition: $m \geq 1$ *and* $n \geq 1$
- Postcondition: $1 \leq x \leq min(m, n)$

Loop invariants:
- True, False
- $1 \leq x \leq m$
- $1 \leq x \leq n$

Termination orderings:
- $y = y - x$ or,
- $x = x - y$ or,
- $max(m, n) = max(m, n) - min(m, n)$

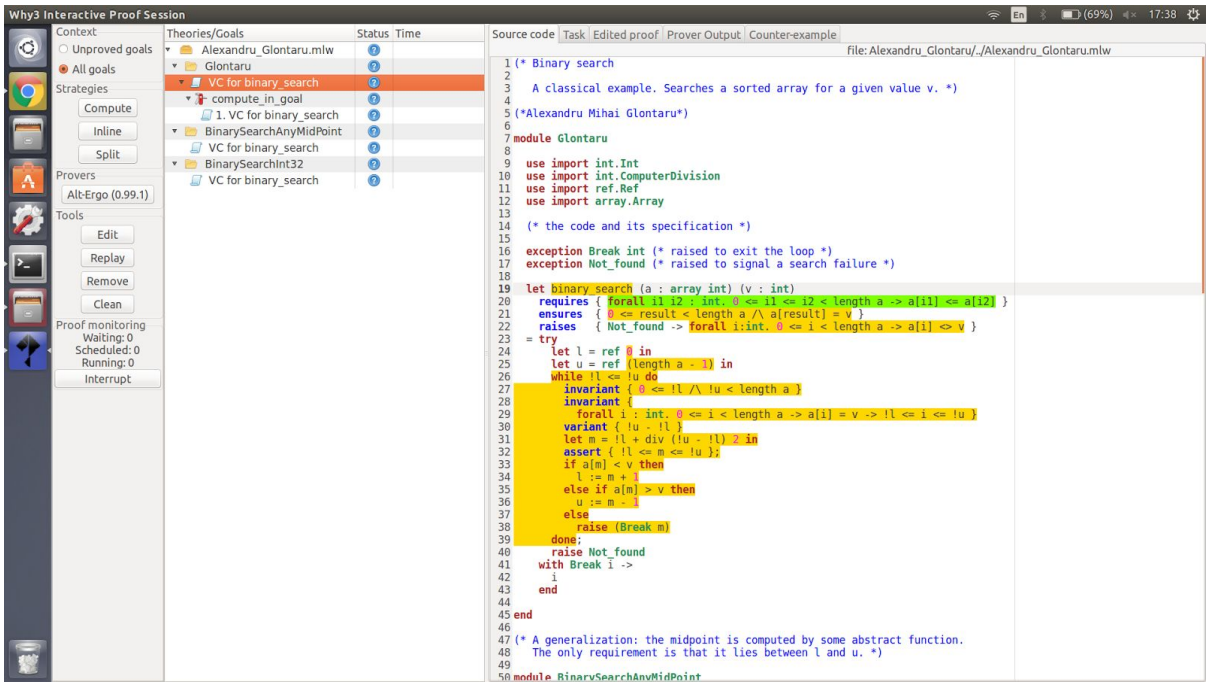
➔ `fun factorial(n:`N`):`N

Function specifications:
- Precondition: $n \geq 0$
- Postcondition: $product \geq 1$ *and* $product = n!$

Loop invariants:
- True, False
- $product \geq 1$
- $factor \geq 1$

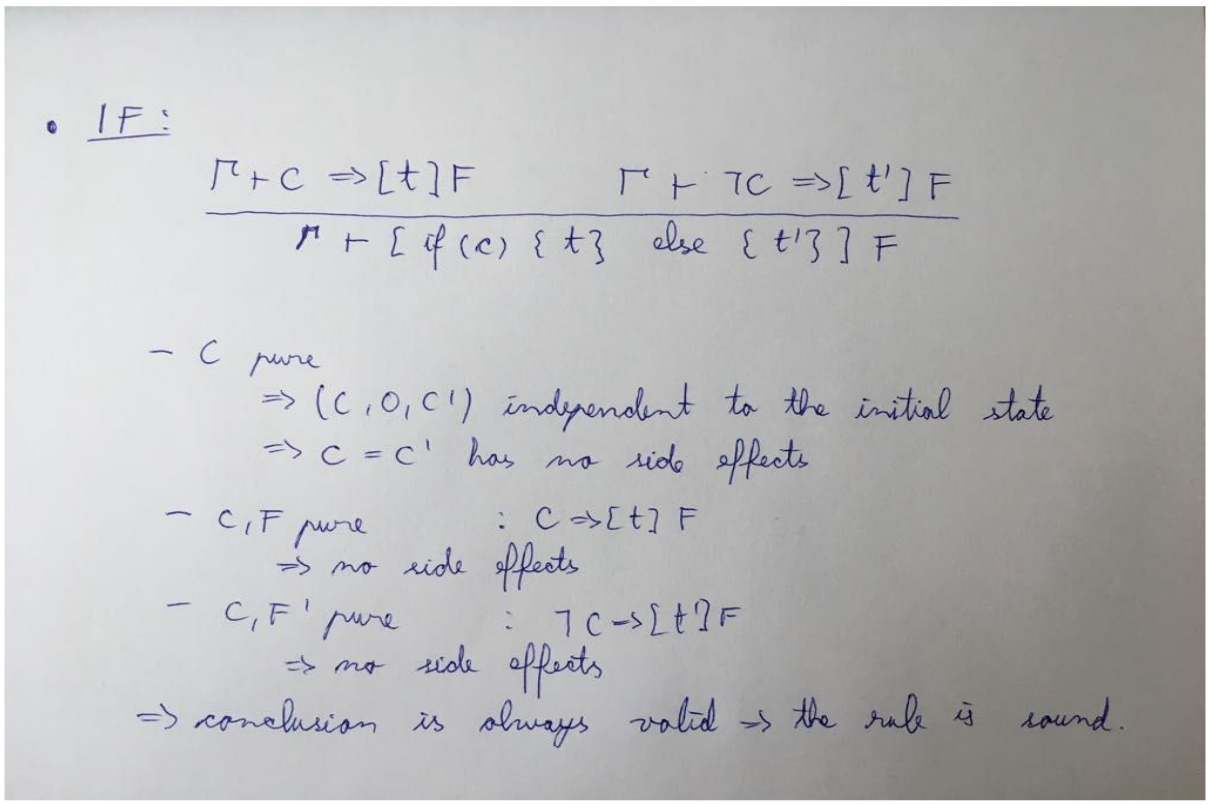
Termination orderings:
- $n - factor$

Problem 2:


Why3 Interactive Proof Session
Context
Unproved goals
All goals
Strategies
Compute
Inline
Split
Provers
Alt-Ergo (0.99.1)
Tools
Edit
Replay
Remove
Clean
Proof monitoring
Waiting: 0
Scheduled: 0
Running: 0
Interrupt
Theories/Goals
Status
Time
Alexandru_Glontaru.mlw
Glontaru
VC for binary_search
compute_in_goal
1. VC for binary_search
BinarySearchAnyMidPoint
VC for binary_search
BinarySearchInt32
VC for binary_search
Source code
Task
Edited proof
Prover Output
Counter-example
file: Alexandru_Glontaru/../Alexandru_Glontaru.mlw
(* Binary search
A classical example. Searches a sorted array for a given value v. *)
(*Alexandru Mihai Glontaru*)
module Glontaru
use import int.Int
use import int.ComputerDivision
use import ref.Ref
use import array.Array
(* the code and its specification *)
exception Break int (* raised to exit the loop *)
exception Not_found (* raised to signal a search failure *)
let binary_search (a : array int) (v : int)
requires { forall i1 i2 : int. 0 <= i1 <= i2 < length a -> a[i1] <= a[i2] }
ensures { 0 <= result < length a /\ a[result] = v }
raises { Not_found -> forall i:int. 0 <= i < length a -> a[i] <> v }
= try
let l = ref 0 in
let u = ref (length a - 1) in
while !l <= !u do
invariant { 0 <= !l /\ !u < length a }
invariant {
forall i : int. 0 <= i < length a -> a[i] = v -> !l <= i <= !u }
variant { !u - !l }
let m = !l + div (!u - !l) 2 in
assert { !l <= m <= !u };
if a[m] < v then
l := m + 1
else if a[m] > v then
u := m - 1
else
raise (Break m)
done;
raise Not_found
with Break i ->
i
end
end
(* A generalization: the midpoint is computed by some abstract function.
The only requirement is that it lies between l and u. *)
module BinarySearchAnyMidPoint

Problem 3:

- IF rule:



- WHILE :