| galois |

Exercise 1:

add.bc:

```
clang -g -O0 -c -emit-llvm add.c -o add.bc
```

add.saw:

```
m <- llvm_load_module "add.bc";
let add_spec = do {
    x <- llvm_fresh_var "x" (llvm_int 32);
    y <- llvm_fresh_var "y" (llvm_int 32);
    llvm_execute_func [llvm_term x, llvm_term y];
    ret <- llvm_fresh_var "ret" (llvm_int 32);
    llvm_return (llvm_term ret);
};
add_ov <- llvm_verify m "add" [] true add_spec z3;
```

running add.saw:

```
saw add.saw
[16:22:28.263] Loading file "<path-to-add.saw>/add.saw"
[16:22:28.334] Verifying add ...
[16:22:28.336] Simulating add ...
[16:22:28.339] Checking proof obligations add ...
[16:22:28.406] Proof succeeded! add
```

Exercise 2:

add_ptr.bc:

```
clang -g -O0 -c -emit-llvm add_ptr.c -o add_ptr.bc
```

add_ptr.saw:

```
m <- llvm_load_module "add_ptr.bc";
let ptr_to_fresh(name : String) (type : LLVMType) = do {
   x <- llvm_fresh_var name type;
   p <- llvm_alloc type;
   llvm_points_to p (llvm_term x);
   return (x, p);
};

let add_in_spec = do {
   (x,p) <- ptr_to_fresh "x" (llvm_int 32);
   y <- llvm_fresh_var "y" (llvm_int 32);
   llvm_execute_func [p, llvm_term y];
   llvm_return (llvm_term {{ x+y : [32]}});
};

add_in_ov <- llvm_verify m "add_in" [] true add_in_spec z3;
```

running add.saw:

saw add_ptr.saw
[20:21:06.429] <path-to-add_ptr.saw>/add_ptr.saw"
[20:21:06.501] Verifying add_ptr ...
[20:21:06.503] Simulating add_ptr ...
[20:21:06.506] Checking proof obligations add_ptr ...
[20:21:06.512] Proof succeeded! add_ptr

Exercise 3:

rotl.bc:

```
clang -g -O0 -c -emit-llvm rotl.c -o rotl.bc
```

rotl.saw:

```
// Rotational Left Shift
m <- llvm_load_module "rotl.bc";

// Declare the specification
let rotl_spec = do {

   // Initialise our variables
   x <- llvm_fresh_var "x" (llvm_int 32);
   r <- llvm_fresh_var "r" (llvm_int 8);

   // Add in the preconditions!
   llvm_precond {{ 0 < r }};
   llvm_precond {{ r < 32}};

   // Call the function and return the value
   llvm_execute_func [llvm_term x, llvm_term r];
   llvm_return (llvm_term {{(x >> (32-(r % 32))) || (x << (r % 32)) }});
};

// Verify the proof
rotl_ov <- llvm_verify m "ROTL" [] true rotl_spec z3;
```

running rotl.saw:

```
saw rotl.saw
[17:13:26.359] Loading file "<path-to-saw-file>/rotl.saw"
[17:13:26.498] Verifying ROTL ...
[17:13:26.508] Simulating ROTL ...
[17:13:26.513] Checking proof obligations ROTL ...
[17:13:26.539] Proof succeeded! ROTL
```

note:

If the preconditions are removed the following error occurs:

```
[18:00:10.683] Subgoal failed: ROTL safety assertion:
rotl.c:7:25: error: in ROTL
Undefined behavior encountered
Details:
```

```
Poison value created
The second operand of `lshr` was equal to or greater than the number of
bits in the first operand

[18:00:10.683] SolverStats {solverStatsSolvers = fromList ["SBV->Z3"],
solverStatsGoalSize = 379}
[18:00:10.684] ----------Counterexample----------
[18:00:10.684]   r: 0
[18:00:10.684] ----------------------------------
[18:00:10.684] Stack trace:
"llvm_verify" (/home/franco/Downloads/A/rotl.saw:21:15-21:26):
Proof failed.
```

Even though `r % 32` is written into the `llvm` specification.