| galois |

Exercise 1:

saf.bc:

```
clang-12 -g -O0 -c -emit-llvm saf.c -o saf.bc
```

saf.saw:

```
import "saf.cry";

let safe_setup = do {
   arr <- llvm_fresh_var "array" (llvm_array 10 (llvm_int 8));
   parr <- llvm_alloc (llvm_array 10 (llvm_int 8));
   llvm_points_to parr (llvm_term arr);

   idx <- llvm_fresh_var "index" (llvm_array 1 (llvm_int 8));
   pidx <- llvm_alloc (llvm_array 1 (llvm_int 8));
   llvm_points_to pidx (llvm_term idx);

   llvm_execute_func [ pidx, parr ];

   llvm_return (llvm_term {{ saf idx arr }});
};

let main : TopLevel () = do {
   m <- llvm_load_module "saf.bc";
   saf_proof <- llvm_verify m "saf" [] false safe_setup yices;
   print "Done!";
};
```

running saf.saw:

```
saw saf.saw
[17:59:08.348] Verifying saf ...
[17:59:08.349] Simulating saf ...
[17:59:08.352] Checking proof obligations saf ...
[17:59:08.372] Proof succeeded! saf
[17:59:08.372] Done!
```

Exercise 2:

sat.bc:

```
clang-12 -g -O0 -emit-llvm -c saf.c -o saf.bc
```

sat.saw:

```
import "saf.cry";

let safe_setup = do {
   arr <- llvm_fresh_var "array" (llvm_array 10 (llvm_int 8));
   parr <- llvm_alloc (llvm_array 10 (llvm_int 8));
   llvm_points_to parr (llvm_term arr);

   idx <- llvm_fresh_var "index" (llvm_int 8);

   llvm_execute_func [ llvm_term idx, parr ];

   llvm_return (llvm_term {{ saf idx arr }});
};
```

```
let main : TopLevel () = do {
   m <- llvm_load_module "saf.bc";
   saf_proof <- llvm_verify m "saf" [] false safe_setup yices;
   print "Done!";
};
```

running saf.saw:

```
saw saf.saw
[18:48:52.259] Verifying saf ...
[18:48:52.259] Simulating saf ...
[18:48:52.262] Checking proof obligations saf ...
[18:48:52.282] Proof succeeded! saf
[18:48:52.282] Done!
```

Exercise 3:

popcount.bc:

```
clang-12 -g -O0 -c -emit-llvm popcount.c -o popcount.bc
```

popcount.cry:

```
popcount : [32] -> [32]
popcount n = z!0
  where
    z = [0]#[ if x==1 then y+1 else y | x <- n | y <- z ]
```

popcount.saw:

```
import "popcount.cry";

popmod <- llvm_load_module "popcount.bc";

let pop_cryptol_check = do {
   x <- llvm_fresh_var "x" (llvm_int 32);
   llvm_execute_func [ llvm_term x ];
   llvm_return (llvm_term {{ popcount x }});
};

// same verification against Cryptol spec
llvm_verify popmod "pop_count" [] true pop_cryptol_check yices;

// Begin Cryptol additional verifications
// another tricky implementation
llvm_verify popmod "pop_count_mul" [] true pop_cryptol_check yices;

// verify the while loop version
llvm_verify popmod "pop_count_sparse" [] true pop_cryptol_check yices;
```

running popcount.saw:

```
saw popcount.saw
[21:28:40.206] Verifying pop_count ...
[21:28:40.208] Simulating pop_count ...
[21:28:40.212] Checking proof obligations pop_count ...
[21:28:41.044] Proof succeeded! pop_count
[21:28:41.079] Verifying pop_count_mul ...
```

```
  [21:28:41.081] Simulating pop_count_mul ...
  [21:28:41.084] Checking proof obligations pop_count_mul ...
  [21:28:41.630] Proof succeeded! pop_count_mul
  [21:28:41.665] Verifying pop_count_sparse ...
  [21:28:41.666] Simulating pop_count_sparse ...
  [21:28:42.425] Checking proof obligations pop_count_sparse ...
  [21:29:18.740] Proof succeeded! pop_count_sparse
```

note:
true instead of false in verify statements results in faster execution