galois

Exercise 1:

SAW file:

```
import "SHA512.cry";

let Sigma0_setup = do {
   x <- llvm_fresh_var "x" (llvm_int 64);
   llvm_execute_func [llvm_term x];
   llvm_return (llvm_term {{ SIGMA_0 x }});
};

let Sigma1_setup = do {
   x <- llvm_fresh_var "x" (llvm_int 64);
   llvm_execute_func [llvm_term x];
   llvm_return (llvm_term {{ SIGMA_1 x }});
};

let sigma0_setup = do {
   x <- llvm_fresh_var "x" (llvm_int 64);
   llvm_execute_func [llvm_term x];
   llvm_return (llvm_term {{ sigma_0 x }});
};

let sigma1_setup = do {
   x <- llvm_fresh_var "x" (llvm_int 64);
   llvm_execute_func [llvm_term x];
   llvm_return (llvm_term {{ sigma_1 x }});
};

let Ch_setup = do {
   x <- llvm_fresh_var "x" (llvm_int 64);
   y <- llvm_fresh_var "y" (llvm_int 64);
   z <- llvm_fresh_var "z" (llvm_int 64);
   llvm_execute_func [llvm_term x, llvm_term y, llvm_term z];
   llvm_return (llvm_term {{ Ch x y z }});
};

let main : TopLevel () = do {
   m <- llvm_load_module "sha512.bc";
   Sigma0_ov <- llvm_verify m "Sigma0" [] true Sigma0_setup z3;
   Sigma1_ov <- llvm_verify m "Sigma1" [] true Sigma1_setup z3;
   sigma0_ov <- llvm_verify m "sigma0" [] true sigma0_setup z3;
   sigma1_ov <- llvm_verify m "sigma1" [] true sigma1_setup z3;
   Ch_ov <- llvm_verify m "Ch" [] true Ch_setup z3;
   print "Done!";
};
```

Running saw on the above file:

```
[16:28:31.471] Loading file ".../s1.saw"
[16:28:31.711] Verifying Sigma0 ...
[16:28:31.728] Simulating Sigma0 ...
[16:28:31.735] Checking proof obligations Sigma0 ...
[16:28:31.806] Proof succeeded! Sigma0
[16:28:31.860] Verifying Sigma1 ...
[16:28:31.876] Simulating Sigma1 ...
[16:28:31.882] Checking proof obligations Sigma1 ...
```

```
[16:28:32.000] Proof succeeded! Sigma1
[16:28:32.053] Verifying sigma0 ...
[16:28:32.072] Simulating sigma0 ...
[16:28:32.077] Checking proof obligations sigma0 ...
[16:28:32.152] Proof succeeded! sigma0
[16:28:32.207] Verifying sigma1 ...
[16:28:32.225] Simulating sigma1 ...
[16:28:32.230] Checking proof obligations sigma1 ...
[16:28:32.310] Proof succeeded! sigma1
[16:28:32.366] Verifying Ch ...
[16:28:32.384] Simulating Ch ...
[16:28:32.388] Checking proof obligations Ch ...
[16:28:32.420] Proof succeeded! Ch
[16:28:32.420] Done!
```

Exercise 2:

```
[prompt]$ make sha-256
cc sha-256.c -o sha-256
[prompt]$ sha-256 "Hello World Folks"
d14155c5fb4dbbb2f8d1d3ade275982a610bc50ff85389a1093875b85993cfeb
[prompt]$
```

Exercise 3:

```
null_digest = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

all_chars msg = y ! 0
 where
   y = [True]#[ z /\ (if (x < 32) \/ (x > 127) then False else True)
              | x <- msg | z <- y ]

digest_in_bytes : {i} (fin i, 64 >= width (8*i)) => [i][8] -> [32][8]
digest_in_bytes msg = if all_chars msg then split(SHA256 msg) else
                      null_digest
```

```
 ╔═╗╦═╗╦ ╦╔═╗╔╦╗╔═╗╦
 ║  ╠╦╝╚╦╝╠═╝ ║ ║ ║║
 ╚═╝╩╚═ ╩ ╩   ╩ ╚═╝╩═╝
version 2.12.0
https://cryptol.net  :? for help

Loading module Cryptol
Cryptol> :l SHA256.cry
Loading module Cryptol
Loading module SHA256
SHA256> digest_in_bytes "Hello World Folks"
[0xd1, 0x41, 0x55, 0xc5, 0xfb, 0x4d, 0xbb, 0xb2, 0xf8, 0xd1, 0xd3,
 0xad, 0xe2, 0x75, 0x98, 0x2a, 0x61, 0x0b, 0xc5, 0x0f, 0xf8, 0x53,
 0x89, 0xa1, 0x09, 0x38, 0x75, 0xb8, 0x59, 0x93, 0xcf, 0xeb]
SHA256>
```

Exercise 4:

```
void SHA256_Buf_Wrapper(uint8_t hash[SIZE_OF_SHA_256_HASH], const void *input) {
   size_t len = strlen((char*)input);
```

```
   calc_sha_256(hash, input, len);
}

[franco@franco lab5E]$ make sha-256
cc sha-256.c -o sha-256
[franco@franco lab5E]$ sha-256 "Hello World Folks"
d14155c5fb4dbbb2f8d1d3ade275982a610bc50ff85389a1093875b85993cfeb
[franco@franco lab5E]$
```

Exercise 5:

```
import "SHA256.cry";

let sha256_setup n = do {
   digest <- llvm_fresh_var "digest" (llvm_array 32 (llvm_int 8));
   pdigest <- llvm_alloc (llvm_array 32 (llvm_int 8));
   llvm_points_to pdigest (llvm_term digest);

   buffer <- llvm_fresh_var "buf" (llvm_array n (llvm_int 8));
   pbuffer <- llvm_alloc (llvm_array n (llvm_int 8));
   llvm_points_to pbuffer (llvm_term buffer);

   llvm_execute_func [ pdigest, pbuffer ];

   llvm_points_to pbuffer (llvm_term {{ digest_in_bytes buffer }});
};

let main = do {
   mm <- llvm_load_module "sha-256.bc";
   sha256 <- llvm_verify mm "SHA256_Buf_Wrapper" [] false (sha256_setup 32) z3;
   print "Done!";
};
```