| galois |

Lab: Introduction to SAW

The Software Analysis Workbench (SAW) is a tool for constructing mathematical models of the computational behavior of software, transforming these models, and proving properties about them. The models take the form of typed functional programs. Various external proof tools, including a variety of SAT and SMT solvers, can be used to prove properties about the functional models. SAW can construct models from arbitrary Cryptol programs, and from C and Java programs that have fixed-size inputs and outputs and that terminate after a fixed number of iterations of any loop (or a fixed number of recursive calls).

In many cases, such as C programs, SAW uses the Intermediate Representation (IR) llvm to provide the means to verify properties given a specification. The acronym llvm stands for low level virtual machine. Figure 1 below shows what the llvm project can do. The tool clang compiles C code to llvm IR. Then the llvm compiler can be used to continue compilation down to some architecture. As Figure 1 shows, there are tools for other languages that do the same thing. Thus, at the llvm level, the same function expressed in different languages, and even specifications, may be checked for equivalence.

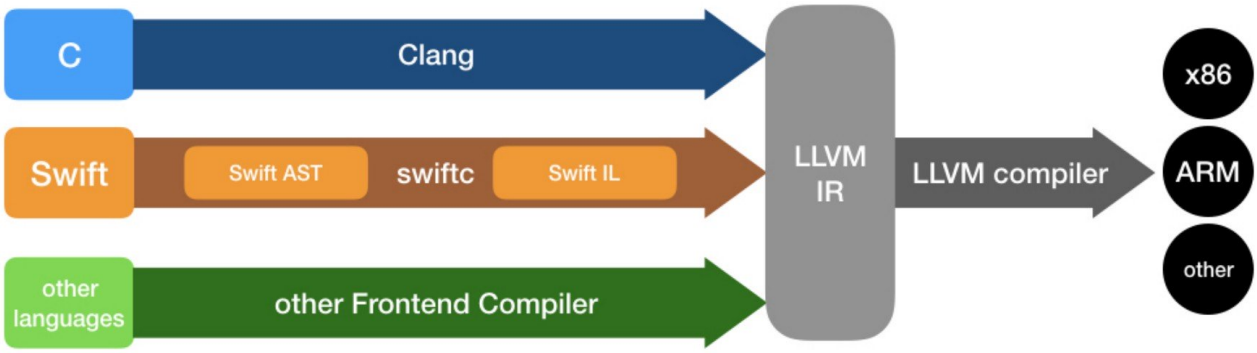

Figure 1: LLVM Frontend-Backend Compiler Architecture (from medium.com)

Clang is used as follows from the command line in Linux for C code in file `add.c`:

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

For this to work, `clang-12` must be installed. Switch `-O0` means "no optimization": this level compiles the fastest and generates the most debuggable code. Switch `-O1` instead means generated bitcode more closely matches the C/C++ source, making the results more comprehensible. Switch `-g` turns on debugging symbols so SAW can find source locations of functions, names of variables, etc.. In this example, compiled output is placed in `add.bc`. This is not human readable but can be converted to human readable by doing this: `llvm-dis`

`add.bc` the result of which is `add.ll`. The extension `.bc` stands for bitcode. Consult the manual, Page 28, for helpul notes on compiling for SAW.

SAW can load a bitcode module with this: `m <- llvm_load_module "add.bc";` for the `clang-12` created `add.bc`.

Exercise 1:
The DES specification in Cryptol is provided in files `DES.cry` and `Cipher.cry`. Add a property that will allow you to find all weak keys for DES.

Definition: *weak keys of DES* are keys that result in all per-round keys being identical.

Help: in `DES.cry` there is a function called `expandKey` that takes a key as argument and produces a sequence of 16 per-round keys. All you have to do is create a function that checks whether all 16 numbers in the sequence are the same. Then create a property that compares the output of that function to something such that the comparison is `True` if and only if all 16 per-round subkeys are the same.

Exercise 2:
A key is not desirable if encrypting twice produces the original plaintext. Determine whether such keys exist for DES and, if so, provide one such.