SIL Bug Reducer

This directory contains a script called bug_reducer. It is a reimplementation of llvm's bugpoint using a python script + high level IR utilities. This will enable bugpoint to easily be ported to newer IRs.

An example invocation would be:

```
./swift/utils/bug-reducer/bug_reducer.py \
    opt \
    --sdk=$(xcrun --sdk macosx --toolchain default --show-sdk-path) \
    --module-name=${MODULE_NAME} \
    --work-dir=${PWD}/bug_reducer \
    --module-cache=${PWD}/bug_reducer/module-cache \
    ${SWIFT_BUILD_DIR} \
    ${SIB_FILE}
```

This is still very alpha and needs a lot of work including tests, so please do not expect it to work at all until tests are available. Once testing is available, please only use this if you are willing to tolerate bugs (and hopefully help fix them/add tests).

Tasks

1. A lot of this code was inspired by llvm's bisect tool. This includes a bit of the code style, we should clean this up and pythonify these parts of the tool.
2. Reduction at a function level is complete, we should look into block/instruction reduction techniques to reduce SIL further.
3. Then we need to implement miscompile detection support. This implies implementing support for codegening code from this tool using swiftc. This implies splitting modules into optimized and unoptimized parts. Luckily, sil-func-extractor can perform this task modulo one task*.

- Specifically, we need to be able to create internal shims that call shared functions so that if a shared function is partitioned on the opposite side of any of its call sites, we can avoid having to create a shared function declaration in the other partition. We avoid this since creating shared function declarations is illegal in SIL.

High Level Example

Imagine that I have a test case foo.swift that causes the optimizer to assert in the normal performance pipeline. I decide I want to use the bug_reducer to reduce the passes and or size of the test case. First I emit a SIB file with the performance optimizations disabled:

```
${SWIFTC_CMDLINE} -emit-sib -Xllvm -disable-sil-optzns -O -o ${OUTPUT}.sib
```

Then I invoke the bug reducer as follows:

```
./swift/utils/bug-reducer/bug_reducer.py \
    opt \
    --sdk=$(xcrun --sdk macosx --toolchain default --show-sdk-path) \
    --module-name=${MODULE_NAME} \
    --work-dir=${PWD}/bug_reducer \
    --module-cache=${PWD}/bug_reducer/module-cache \
    --reduce-sil \
    ${SWIFT_BUILD_DIR} \
    ${OUTPUT_SIB}
```

Then the bug_reducer will first attempt to reduce the passes. Then, it will attempt to reduce the test case by splitting the module and only optimizing part of it. The output will look something like:

```
*** Successfully Reduced file!
*** Final File: ${FINAL_SIB_FILE}
*** Final Functions: ${FINAL_SET_OF_FUNCTIONS}
*** Repro command line: ${FULL_FINAL_REPRO_CMDLINE}
*** Final Passes: ${FINAL_PASSES}
```

Some notes:

1. The final sib file produced *may be different* than the sib file that you inputted. This is because the bug_reducer may have:
    a. Created an intermediate optimized sib file with some of the passes applied (this ensures that a module with only one pass can be provided)
    b. (In a future version) Split the module into optimized and unoptimized parts to reduce the code even further.
2. Repro command line will be the actual final full command line used to reproduce the crasher. You should be able to copy/paste the command line directly into LLDB to reproduce. No further work is required.