Homework 4 (due June 20)

In this homework, you will start handling the memory of a smart contract, both transient (Memory) and persistent (Storage). The handling of shared memory (and not local variables) is one of the main challenges of whole-program static analysis.

1. In your Flows relation from the previous homeworks, add flows via memory and via storage. That is, you should handle `MSTORE/.../MLOAD` and `SSTORE/SLOAD` instructions. The main focus is storage. Memory can be handled similarly, by modeling `MSTORE/MLOAD` instructions at known addresses, but most uses of memory are more complex and are already modeled in the framework via the memory modeling API present under clientlib/memory_modeling. Relation `MemoryStatement_ActualArg` can be used to obtain the variables written in memory (using `MSTORE` statements) as arguments to `MLOAD, SHA3, CALL` statements (and more) without having to model the underlying memory addresses.
2. Add symbolic constants for all addresses that are derived from external values, specifically from `CALLDATALOAD` statements, as in homework 3. For instance, for a `CALLDATALOAD` with instruction id "`0x95a`", you can consider that it produces a new value (symbolic constant) "`input0x95a`". You should perform symbolic evaluation of at least the `SHA3` instruction over such symbolic constants. (You may also need to combine with `Flows` to get meaningful results, unless you also support symbolic arithmetic.) That is, produce constants of the form "`SHA3(SHA3(input0x951))`", which will be added to `Variable_Value` as "values". To avoid infinite recursion, limit the depth of application of the `SHA3` constructor.
3. Even with the above effort, the definition is not even “soundy”! It yields no results for store instructions over unknown addresses. A big step towards completeness (with a likely cost in precision) is to consider every store to an unknown location as a store to all the already-known (constant) locations. This requires negation, so the predicate you'll produce should be evaluated at a later stage than (i.e., after the full evaluation of) the earlier `Flows` relation. You can name this more complete relation `GeneralFlows`. How much less precise is it?
4. Define an intermediate relation between the `Flows` relation of step 1 and `GeneralFlows` of step 3. For instance, you can handle as stores-to-any-address only the stores to a tainted address (according to the 2nd question of Homework 3) that is not constructed via a `SHA3`. We will discuss more options in class.

Apply your analyses over the gigahorse-benchmarks. Examine the impact of the different definitions of `Flows` over the client analyses from Homeworks 2 and 3.