Homework 3 (due June 7)

In this third 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. (There are more instructions that play the role of an MSTORE--e.g., see the predicate StatementStoresMemory--but for the homework it is enough to limit your attention to MSTORE.) At this stage, you should support only stores and loads to constant memory locations (i.e., use predicate Variable_Value). You can experiment with also handling memory locations that have constants flow to them (i.e., combine Variable_Value and Flows) to see how much you lose in precision and gain in completeness.
2. Add symbolic constants for all addresses that are derived from external values, specifically from CALLDATALOAD statements, as in homework 2. 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 LocalFlows 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 2) that is not constructed via a SHA3. We will discuss more options in class.

Apply your analyses over all 800+ contracts given, using analyze.py. Examine the impact of the different definitions of Flows over the client analyses from Homeworks 1 and 2.