Feature Interaction Mutations - Supplement

Brady J. Garvin and Myra B. Cohen
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
{bgarvin,myra}@cse.unl.edu

I. Mutations

GCC Bug #41643. In the simplest GCC interaction fault, an `if` that should have tested a condition in the form `(!foo||!bar)` was wrongly implemented so that it only checked `!foo`.

```
if(foo){                  if(foo){
  if(bar){             Δ    if(true){
    goto L1;                  goto L1;
  }                         }
}                         }
//...                     //...
L1:                       L1:
```

Although the change is a simple first-order mutation, it occurred in optional code, and the predicate could only be falsified by statements in code for different features. The mutated statement, a use, and one of its defs were in the reachable blocks of every critical pair.

The general mutation suggested is to identify a viable def/use pair that is not contained by any non-interaction pair's reachable set, and then to apply a normal mutation to either the def or the use.

GCC Bug #39794. As with bug #41643, the mutation corresponding to GCC Bug #39794 is semantically a guard condition changed to `true` or `false`.

```
if(xyzzy==null){     Δ if(true){
  xyzzy=...;              xyzzy=...;
}                       }
```

The fix is actually somewhat more complicated because the GCC developers had to move one definition of `xyzzy` (which represents the canonicalized expression for a memory address) earlier and then change a function signature in order to introduce the guard.

The effects of this mutation are similar to the effects of #41643, but the reasons for it being an interaction fault are different. Although the `if`'s use is in code for an optional feature (a dead store elimination pass), it can draw on a wide variety of defs, not all of which are in optional code. The fault is only an interaction fault because the subsequent assignment almost always leads to equivalent gen and kill sets and therefore the same ultimate outcome. We identified at least twelve features that would disturb the inputs to the dead store elimination pass enough to mask the fault.

GCC Bug #40087. The most complicated of the three bugs was #40087, which arguably was several faults caused by a single misunderstanding. In five places, and in four different ways, the mutation followed the pattern where a guard was altered, eventually affecting the value escaping a function in optional code; at the same time, all of the code that could use the def was governed by a different set of features.

The unique aspect of #40087 is in how the guards changed, because at two points the correct condition to test is `false`. Thus, it would perhaps be better to classify those mutations separately, as introducing a definition.

```
//...                //...
                   Δ plugh=...
//...                //...
```

For this kind of mutation to yield an interaction fault, the location of the injected definition and the locations where it could be read must not be within the reachable set of any non-interaction pair.