1.1 Test Generation and Comparison

Before presenting how test generation works in our approach, let us revisit some of the details of TOTEM. In essence their approach is summarized into the four regular expressions given in Table 1 of their paper, namely:

```
A(uid).C(uid)
I(title).J(title)
I(title).G(title, item).H(item).J(title)
(A(uid)||I(title).G(title,item)).D(uid,item).F(uid,item).
(C(uid)||H(item).J(title))
```

It is this last expression that illustrates best the root cause for the large number of test cases TOTEM generates. In that expression, the creation of a user (A) is interleaved with the sequence title creation (I) followed by item creation (G). Similarly, user deletion (C) is interleaved with item deletion (H) and title deletion (J). The authors explain there are 3 possible sequences for the first interleaving, and 3 others for the second. Then to illustrate their generation algorithm, they apply the four regular expressions above to a very small example: 2 users, 3 titles, 2 items per title, 1 loan per user, no renew or collect fine for loans, and no system monitoring. In order to 'cover' their example (that is, use at least once each possible value of each parameter), they use an 'arbitrary' (their adjective) algorithm to come up with 8 sequences, namely:

Seq1: A(u1).C(u1)

Seq2: I(t1).J(t1)

Seq3: I(t2).G(t2, i21).H(i21).J(t2)

Seq4: I(t3).A(u2).G(t3,i32).D(u2,i32).F(u2,i32).C(u2).H(i32).J(t3)

Seq5: I(t1).G(t1, i11).H(i11).J(t1)

Seq6: I(t1).G(t1, i12).H(i12).J(t1)

Seq7: A(u1).I(t2).G(t2,i22).D(u1,i22).F(u1,i22).H(i22).C(u1).J(t2)

Seq8: I(t3).G(t3, i31).H(i31).J(t3)

Noticing the first two are redundant respectively with Seq7 and Seq 5, they then start discussing combination of these sequences. First Seq3 and Seq4 are combined to produce 495 interleavings since the number of interleavings between two sequences S1 and S2 of length m and n respectively is given by $C(m+n, n) = (m+n)!/(n!m!)$. The crucial detail is that they then arbitrarily pick one of these 495 interleavings, namely S:

S: I(t3).A(u2).I(t2).G(t3,i32).G(t2,i21).D(u2,i32).H(i21).

F(u2,i32).C(u2).H(i32).J(t2).J(t3)

They then compute that there are 1820 interleavings between S and Seq5. Again, they pick *one* of these 1820 interleavings and combine it to Seq6, then one of the resulting interleavings, and combine it with Seq7 ultimately producing 18018 interleavings that include:

S^3: I(t3).I(t1).A(u1).G(t1,i11).G(t1,i12).A(u2).I(t2).G(t2,i22).

G(t3,i32).H(i11).D(u1,i22).G(t2,i21).F(u1,i22).D(u2,i32).

H(i12).H(i21).J(t1).H(i22).C(u1).F(u2,i32).C(u2).H(i32).J(t2)

.J(t3)

Clearly, these 18018 tests are *not* a complete set as they proceed from repeatedly taking only 1 interleaving of the previous combination and interleaving it with the next sequence.

Our claim is that considering use case independence can drastically reduce the number of test cases to generate. In the library system, use case independence allows us to consider that the 3 possible orderings for (A || I.G) are equivalent, as are the three for (C || H.J). In turn, the independence of A, I and G, and of C, H and J, which can be verified using our model, allows us to limit the use of interleavings to only those use cases that cannot be shown to be independent. In TOTEM's small scoped example, this means considering interleavings only between instances of use cases D (borrowing) and F (returning).

In the context of this example, our generation algorithm will:

1) Carry out all relevant user, title and item creation, namely: A(u1), A(u2), I(t1), I(t2), I(t3), G(t1, i11), G(t1, i12), G(t2, i21), G(t2, i22), G(t3,i31) and G(t3,i32).

Here the only ordering that matters proceeds from enforcing valid sequential dependencies, which requires the instances of I to precede those of G.

2) Consider the possible interleavings of the only two loans this scoped example deals with, namely the interleavings of (D(u1,i22).F(u1,i22)) with (D(u2,i32).F(u2,i32)).
3) Finally, carry out all relevant user, title and item deletion. Here, since deleting a title automatically deletes its copies, in principle order does not matter. However, to avoid unsuccessful instances of use cases (that would result from attempting to delete a copy of a title that has already been deleted), our generation algorithm obeys the sequential dependency between J and H and correctly deletes items and then titles.

For the two loans of this scoped example, there are 6 interleavings:

D(u1,i22).F(u1,i22). D(u2,i32).F(u2,i32)

D(u1,i22). D(u2,i32). F(u1,i22).F(u2,i32)

D(u1,i22). D(u2,i32). F(u2,i32). F(u1,i22)

D(u2,i32).F(u2,i32). D(u1,i22).F(u1,i22)

D(u2,i32). D(u1,i22). F(u2,i32). F(u1,i22)

D(u2,i32). D(u1,i22). F(u1,i22). F(u2,i32).

The generation of these interleavings (which respect the sequential dependency that any D(i,j) must precede the corresponding F(I,j)) is fully automated and determines the number of tests to run for this very limited example. In other words, there are 6 test cases to run that are specific to the valid paths of the scoped example. The key difference between TOTEM's approach and ours is that, in ours, all user/title/item creations are carried out as the

set-up of each of these 6 testcases, and all user/title/item deletions are carried out as the teardown of each of these 6 testcases. The generation algorithm also generates a test case for each unsuccessful path of each scenario. That is, for unsuccessful paths, there is no need to consider any interleaving or combination of sequences. However, because of how scenarios can be monitored only if they are correctly triggered, our verification tool must allow bypassing a trigger statement for the purpose of testing unsuccessful paths. For example, given that the borrowing use case can only be triggered if the borrowing user has been correctly created, in order to test an attempt to borrow with an invalid user, we must bypass the trigger statement of the borrowing scenario.

Recapitulating, given a scoped example to test (as the very small example considered in TOTEM), we restrict interleaving to only use cases that cannot be shown to be independent of others. For those use cases only, we directly reuse TOTEM's approach to test generation. All object creation and deletion relevant to independent use cases is handled in the setup and teardown shared by all example-specific test cases. A test case is also generated for each unsuccessful paths of each (independent or not) use case. (TOTEM does not address generation for unsuccessful paths of use cases.). In the worst case, there are no independent use cases and our approach boils down to reusing TOTEM's generation algorithm. But, as pointed out by Jacobson (1992), the absence of independent use cases typically indicates an undesirable high level of coupling in the system and should be corrected. Assuming a system will have some independent use cases, eliminating these when interleaving will necessarily generate fewer cases than in TOTEM as it is the length of sequences that correlates to the number of possible interleavings.

At this point of the discussion, given the simplicity of the scoped example, one must consider how cycles (e.g., renewals) may complicate the generation algorithm. On this issue, TOTEM follows Binder's (2000) guideline: each loop or cycle is to be replaced by a fixed number of paths corresponding to at least the minimum and the maximum number of iterations through it. TOTEM does not further discuss this issue and, by default, we adopt their approach. That is, currently, we assume that any use case involved in a cycle is not an independent one and reuse TOTEM's approach to generation of tests to deal with it. We do remark however that future work should investigate this issue further as there is room for improvement. For example, our model does have the ability to verify that if one loan is renewed, all other loans are not affected (i.e., do not exhibit a state change). The open problem to eventually address is whether such ability could somehow reduce the number of interleavings to generate.

So far, we have claimed that assuming that there are some independent use cases in a system allows us to reduce the number of interleavings to generate (using TOTEM's algorithm) by eliminating these independent use cases from the possible sequences of use cases to consider. But it is crucial to acknowledge that, in our approach, there are more tests to be generated (beyond those associated with unsuccessful paths and interleaving between non-independent use cases). Indeed, additionally, both the setup and the teardown require tests to verify the mutual independence between some use cases, on which our approach rests. Most importantly, the tests for the setup are independent of all others, as are those for the teardown. For the setup, we need tests to verify that a user/title/item creation does not affect other users, nor existing titles, nor existing items, nor existing loans (as explained in 3.2). Similarly, for the teardown, we need tests to verify that a user/title/item deletion does not

affect other users, nor existing titles, nor existing items, nor existing loans (again as explained in 3.2). To do so, it is crucial to remark that such tests should not be limited to a particular scoped example. Instead, there should be one testplan for setup and one for the teardown, regardless of the scope used for the testing of non-independent use cases. For our library system, according to our model, we have users, titles, items and loans to consider. It is left to the tester to define how many instances whose states are monitored need to be created. We have arbitrarily chosen 5 as a representative number of instances to consider. The task is *not* to consider all possible interleavings of creating 5 users, 5 titles, and 5 items (which leads to 15! tests). Instead it consists of two steps:

- Step1: minimal set of existing instances
    - Test each of the 3 valid orderings of (A(u1) || I(t1).G(t1,i11)), which addresses creating the first user, the first title, the first item
    - For each of the 3 valid orderings of (A(u1) || I(t1).G(t1,i11))
        - Test A(u2), test I(t2), test G(t1, i12), test G(t2, i21) separately
        - Test the 4! interleavings of A(u2) || I(t2) || G(t1, i12) || G(t2, i21)
- Step 2: representative set of existing instances
    - Once and only once all tests of step 1 have passed, create 5 users, 5 titles, 5 items per title. Add 1 random loan to one user, 2 random loans to another, and 5 random loans to a third user.
    - Test A(u6), test I(t6), test G(t6, i61), test G(t6, i62), test G(t6, i63), test G(t5, i56) and test G(t5, i57) separately

The idea of step 1 is to ensure that, regardless of the order of creation of a set of instances whose states are to remain unchanged, the addition of a new user or a new title, or a

new item (to an existing or a new title) indeed does not trigger any incorrect state change. This step proceeds from varying the values used for the different parameters of the independent use cases. Step 2 follows the same idea but once a representative number of instances have been created. Given step 1, we do not consider the ordering of these creations.

For step 1, *for each possible interleaving*, there is small set of tests to perform, which is obtained by testing a new value of each parameter in all possible contexts of use. This explains why we test both adding a new item to an existing title and adding a new item to a new title. For step 2, there is a fixed set of tests, which also proceeds from testing a new value of each parameter in all possible contexts of use.

Quantitatively, for this library system, we end up with 3 + (3*4) + (3*4!) tests for step1, and 7 tests for step 2, for a total of 94 tests for the setup testplan. It should be emphasized that, contrary to TOTEM that relies on combining the current sequence with only one instance of the previous interleavings, here all possible interleavings are considered. That is, our approach appears to be more complete with respect to setup use cases.

For the teardown testplan, we adopt a similar strategy but must consider two additional difficulties: deleting the last instance of some type and deleting a title that has items that have not been deleted.

- Perform A(u1), I(t1), G(t1, i11) (in any order since it is assumed the setup has demonstrated their independence). Test each of the 3 valid orderings of a (C(u1) || (J(t1,i11). H(t1))), which addresses deleting the last instance of a user, a title, an item.

- Perform A(u1), I(t1) and test the 2 orderings of (C(u1) || H(t1)). This addresses the independence of user deletion from title deletion when there are no copies associated with this title.
- Perform A(u1), I(t1), G(t1, i11) and test the 2 orderings of (C(u1) || H(t1)). This addresses the independence of user deletion from title deletion when the latter involves deleting a single copy associated with this title.
- Perform A(u1), I(t1), G(t1, i11), G(t1, i12), G(t1, i13) and test the 2 orderings of (C(u1) || H(t1)). This addresses the independence of user deletion from title deletion when the latter involves deleting several copies associated with this title.
- Perform A(u1), I(t1), G(t1, i11), G(t1, i12), G(t1, i13). Then perform J(t1, i12) and then test the 2 orderings of (C(u1) || H(t1)). This addresses the independence of user deletion from title deletion when the latter involves deleting copies associated with this title before and after user deletion.
- Perform A(u1), I(t1), G(t1, i11), G(t1, i12), G(t1, i13). Test each of the 30 possible orderings of **R**: {C(u1) || [(J(t1,i11) || J(t1,i12) || J(t1,i13)).H(t1)]} (different styles of brackets are used here merely to ease readability). This establishes that the deletion of ANY user is independent of the deletion of multiple copies of a same title.
- Perform A(u2), I(t2), G(t2, i21), G(t2, i22), G(t2, i23) in any order. Previous tests make it irrelevant to consider the interleavings of C(u2) with H(t2) and J(t2, i21), J(t2,i22) and J(t2,i12) given u1 is equivalent to u2, t1 to t2, etc.

Perform A(u1), I(t1), G(t1, i11), G(t1, i12), G(t1, i13). Consider the regular expression **R** given in the previous bullet and choose at random one of its 30 possible orderings (since they have been shown to be equivalent in the previous bullet). Call this particular sequence S.

- Test the 6 orderings given by (C(u2) || S).
- Test the 6 orderings given by (H(t2) || S).
- Since they have been shown to be equivalent, choose one of the six possible orderings of J(t2,i21), J(t2,i22), J(t2,i23) followed by H(t2) and test its 126 interleavings with S.

- create 5 users, 5 titles, 5 items per title. Add 1 random loan to one user, 2 random loans to another, and 5 random loans to a third user. Test separately the deletion of each user, item and title.
- create 5 users, 5 titles, 5 items per title. Add 1 random loan to one user, 2 random loans to another, and 5 random loans to a third user. Given interleavings between user, title and item deletions have been tested and deemed equivalent, randomly select and verify a single sequence that tests the deletion of each user, each title and each item.

Despite attempting to be systematic, the test plan for the teardown, much like the one for the setup, is not complete, relying instead on our assumption that it offers sufficient coverage to conclude to the independence of the relevant use cases. The same assumption is made in TOTEM while trying to justify why only 1 sequence of a current set of interleavings is used to further combine with other sequences they have chosen arbitrarily to cover their scoped

example… Moreover, TOTEM does not address all deletion sequences we consider. Specifically, it does not consider deleting a title that has still items associated with it.

Should we not tackle completeness and test all possible orderings of the following expression?

(C(u1) || C(u2) || ((J(t1,i11) || J(t1,i12)) || J(t1,i13)
|| J(t2,i21) || J(t2,i22) || J(t2,i23))
. (H(t1) || H(t2))))

Given there are 129,600 such orderings, this is clearly not feasible and demonstrates Binder's (2000) point that the goal of 'complete coverage' is generally impossible to achieve, testers having to rely more than often on their intuition about what constitutes sufficient coverage. For example, a brute-force approach like tackling the 129,600 interleavings above, ignores completely the relevance of equivalence partitioning. Testing interleavings for a user, a title and copies of this title amounts to testing for any user, any title, and any copy of this title. This is the key observation on which this part of our test generation approach rests.

In total, testing the teardown requires 3+2+2+2+2+30+6+6+126+15+1= 195 tests, and the total scoped example needs 94 (setup) + 6 (specific to the scope example) + 20 (for unsuccessful paths) + 195 (teardown) = 315 tests (in contrast to the 18018 of TOTEM only for valid paths). Furthermore, both the setup and teardown testplans in fact cover more tests than TOTEM has to cover (A || I.J) and (C || J.H) since these testplans are not specific to the scoped example.