

Symbolic Fault Injection

Evaluation of Symbolic Fault Injection

Luis Torres Klauwer

Contents

- Case Study
 - Evaluate practical use of SFI
 - Application of SFI to a simple system
- Calculating a system's coverage factor ***c***
 - Using SFI and formal methods

SFI Case Study

GOALS

- Directly compare “effectiveness” of conventional fault injection vs. symbolic fault injection
- Determine how “applicable” symbolic fault injection is in practice.

SFI Case Study

~~Conventional Fault Injection vs. Symbolic Fault Injection~~

Main Problems:

- **CFI usually targets low level mechanisms (i.e. registers) while SFI operates at code level.**
- **No standard methods to evaluate and compare the effectiveness of CFI systems exist in the fault tolerance community (everyone uses it's own criteria)**

SFI Case Study

Practical applicability of SFI

- Test case: Proof the fault tolerance property of the parity code (namely that it detects any single bitflip)
- 4 different implementations of the parity code where tested

SFI Case Study

```
\problem {  
  
\forall int i; ( { anInt:=i }  
  
\<{  
sentIsPair = CheckCodes.hasPairParity(anInt);  
  
//Represents all possible single bitflips in an int -  
//Simulate bitflip during transfer  
inject(anInt);  
  
receivedIsPair = CheckCodes.hasPairParity(anInt);  
  
boolean haveSameParity = (sentIsPair == receivedIsPair);  
  
}\> (haveSameParity=FALSE))  
  
}
```

SFI Case Study

Results:

- It could be proven rigorously but not formally that the implementations comply with the specification.
- KeY: Complex proof obligations resulted. (Containing too many branches or too complex arithmetic formulas, depending on the implementation)

Case Study SFI

Causes for the complexity of the proof obligations:

- Inject statements usually *envelope* the proof with a universal quantifier, which dramatically increases the space of reachable states.
- Most fault tolerance algorithms (especially checksum algorithms) contain loops which contain non trivial instructions (or at least non trivial for the average user).

Case Study SFI

- **Conclusion:**
- SFI is not a practical alternative to CFI at the moment.
- SFI is nevertheless a promising technique (exhaustive verification of one of the parity code implementations would take approximately 4 years and CFI does not provide full coverage)
- SFI will become more and more effective as formal methods evolve (in the case of KeY especially with better and more user friendly ways to handle quantifiers and loops).

Coverage Factor

$$c = \frac{\textit{covered faults}}{\textit{total faults}}$$

MOTIVATION:

- Conventional Fault Injection:
 - Injection of large number of random faults
 - Statistical calculations
- Symbolic Fault Injection:
 - Requires one run for a specific fault model
 - Returns precise value

Coverage Factor

Basic Idea

- Inject a symbolic fault and “count” the possible cases caused by the injection.
- “Count” the cases represented by the proof branches that comply with the specification.
- Calculate c for the given fault model.

Coverage Factor

Problematic:

- Define rules which allow to count the cases represented by the branches of a proof
- Is this compatible/integrateable in KeY?

**QUESTIONS, REMARKS
AND
SUGGESTIONS**