Towards Reusing Formal Proofs in Verification of Fault-Tolerance

Borzoo Bonakdarpour
Sandeep S. Kulkarni

Automated Formal Methods (AFM’06)
Motivation

- We need to gain confidence on the correctness of fault-tolerance properties.

- In the literature, the main focus has been on verification of concrete fault-tolerant systems.

- We need more general verifications, so that we are not required to verify individual programs.
We verify the correctness of algorithms that synthesize fault-tolerant programs; all synthesized programs will be correct-by-construction.

We use the theorem prover PVS as our verification tool.
Outline

- Review of previous results [LOPSTR’04, TPHoLs’04]
  - A formal framework for fault-Tolerance
  - A fixpoint calculation library on finite sets
  - Mechanical verification of automatic addition of fault-tolerance

- Mechanical verification of automatic synthesis of multitolerance by reusing formal proofs [AFM’06]

- Conclusions and future work
Levels of Fault-Tolerance

- **Nonmasking**: A program is nonmasking fault-tolerant, if after occurrence of faults it eventually recovers to its normal behavior.

- **Masking**: A program is masking fault-tolerant, if after occurrence of faults it eventually recovers to its normal behavior without violating safety.
A Fault-Tolerance Framework in PVS

FT [state : TYPE]: THEORY
BEGIN
ASSUMING
   ST_is_finite : ASSUMPTION is_finite_type[state]  
   TR_is_finite : ASSUMPTION is_finite_type[[state, state]]
ENDASSUMING

Transition: TYPE = [state, state]
StatePred: TYPE = finite_set [state]
Action: TYPE = finite_set [Transition]
Computation (Z: Action): TYPE = \{ A: sequence[state] | \forall n: (A_n, A_{n+1}) \in Z \} 

StateSpace: StatePred = fullset [state]  
S: StatePred  
p: Action  
f: Action  
Σ_{bt}: Action  

State is a finite type
Transition is a finite type
set of transitions

The state space
invariant of fault-intolerant program
program
set of faults
set of bad transitions
The Synthesis Problem

- Program $p$
- Invariant $S$
- Specification $\Sigma_{bt}$
- Faults $f$

Synthesis Algorithm

Program $p'$
Invariant $S'$

A desired level of fault-tolerance
Automatic Synthesis of Nonmasking Tolerance

- \( S' = S \)
- \( p' = p \cup \{(s_0, s_1) \mid s_0 \in T - S \land s_1 \in S\} \)
**Step (1):** Identifying the set of states and transitions from where safety may be violated by a sequence of fault transitions.

\[ mt = \{ (s_0, s_1) \mid s_1 \in ms \lor (s_0, s_1) \in \Sigma_{bt} \} \]
Step (2): Identifying and removing deadlock states
A Fixpoint Theory on Finite Sets

Suppose $X$ is a state predicate and $g(X)$ denotes the set of deadlock states of $X$:

\[
X_1 = X - g(X) \\
X_2 = X_1 - g(X_1) \\
\vdots \\
X_n = X_{n-1} - g(X_{n-1}) \quad \text{where } g(X_{n-1}) = \emptyset \\
X_n = X_{n-1} \\
X_{n+1} = X_n
\]
Largest Fixpoint

DecFunc: \( \text{TYPE} = [A : \text{StatePred} \rightarrow \{B : \text{StatePred} \mid B \subseteq A\}] \)

Dec \((i : \text{nat}, X : \text{StatePred})(g : \text{DecFunc})\): RECURSIVE \text{StatePred} =

\[
\begin{align*}
\text{IF } i = 0 \text{ THEN} \\
X \\
\text{ELSE} \\
Dec(i - 1, X)(g) - g(Dec(i - 1, X)(g)) \\
\text{ENDIF}
\end{align*}
\]

MEASURE \((\lambda (x : \text{nat}, y : \text{StatePred}) : x)\)

LgFix \((X : \text{StatePred})(g : \text{DecFunc})\): \text{StatePred} = \{s \mid \forall (k : \text{nat}): s \in \text{Dec}(k, X)(g))\}
Theorem [LOPSTR’04]: Further recalculation of fixpoint returns the empty set:

\[ g \left( \text{LgFix} \ (X)(g) \right) = \emptyset \]

\text{DeadlockStates} \ (p:\ \text{Action})(ds:\ \text{StatePred}): \text{StatePred} = \{ s_0 \mid (s_0 \in ds) \land (\forall s_1: (s_1 \in ds) \Rightarrow (s_0, s_1) \notin p) \}\]

\[ S_1: \text{StatePred} = \text{LgFix} \ (S\ -\ ms)(\text{DeadlockStates}(p\ -\ mt)) \]
Let $T_1 = true - ms$

Repeat
- Recalculate $S_1$ and $T_1$ such that:
  - $S_1$ is reachable from all states in $T_1 - S_1$.
  - $T_1$ is closed in $p_1 \cup f$.

Until $S_1$ and $T_1$ remain unchanged

Remove cycles from $T_1 - S_1$
Multitolerant programs tolerate different classes of faults and provide different level of fault-tolerance to each class.

If faults from different classes occur, the multitolerant program provides the minimum level of fault-tolerance:

<table>
<thead>
<tr>
<th>Level of FT</th>
<th>Nonmasking</th>
<th>Masking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nonmasking</td>
<td>Nonmasking</td>
<td>Nonmasking</td>
</tr>
<tr>
<td>Masking</td>
<td>Nonmasking</td>
<td>Masking</td>
</tr>
</tbody>
</table>
Revisiting the Synthesis Problem

Program \( p \)

Invariant \( S \)

Specification \( \Sigma_{bt} \)

\( f_{\text{nonmasking}} \)

\( f_{\text{masking}} \)

Synthesis Algorithm

A desired level of multitolerance

Program \( p' \)

Invariant \( S' \)
Generalizing Formal Specification

add_multift [state : TYPE]: THEORY

BEGIN
  IMPORTING add_nonmasking[state]
  IMPORTING add_masking[state]

  f_nonmasking: Action
  f_masking: Action
  f_nonmasking-masking: Action = f_nonmasking ∪ f_masking

  msInit(anyFault : Action) : StatePred = {s₀ | ∃s₁ : ((s₀, s₁) ∈ anyFault ∧ (s₀, s₁) ∈ Σ_bt )} // faults directly violate safety

  RevReachStates(anyFault : Action)(rs : StatePred) : StatePred = {s₀ | ∃s₁ : (s₁ ∈ rs ∧ (s₀, s₁) ∈ anyFault ∧ s₀ ∉ rs)} // backward reachability

  ms(anyFault : Action) : StatePred = SmFix (msInit(anyFault))(RevReachStates(anyFault)) // Fixpoint of RRS
Formal Spec. of Nonmasking-Masking Synthesis

\[ S' : \text{StatePred} = \text{add}_\text{masking} \cdot S_1( f_{\text{masking}} ) \]

// intermediate program transitions
\[ p_1 : \text{Action} = \text{add}_\text{masking} \cdot p_1( f_{\text{masking}} ) \]

// faults-span
\[ T_1 : \text{StatePred} = \text{add}_\text{masking} \cdot T'( f_{\text{masking}} ) \]

// program transitions
\[ p' : \text{Action} = \text{add}_\text{nonmasking} \cdot p'( T_{\text{masking}}( f_{\text{nonmasking}}), p_1( f_{\text{masking}} ) ) \]
Verification of Synthesis of Multitolerance

1- Theorems involving fixpoint calculations.

Theorem (1): All computations of a nonmasking- masking program are infinite:

$$\text{DeadlockStates}(p')(S') = \{\}$$
Formal Proof of Theorem (1)

\[ \begin{align*}
\{1\} \quad & DeadlockStates(p')(S') = \emptyset \\
\text{Rule? (expand "S")] \} & \text{Applying theorem1} \\
\text{this simplifies to:} & \text{theorem1} : \\
\{1\} & DeadlockStates(p') \\
& (LgFix (S - ms)) \\
& (DeadlockStates (p - mt))) \\
\{1\} & \forall (X: \text{StatePred[state]}, g: \text{DecFunc[state]}): \\
& g (LgFix(X)(g)) = \emptyset \\
\{1\} \quad & DeadlockStates (p') \\
& (ConstructInvariant (S - ms, p - mt)) = \emptyset \\
\text{Rule? (inst -1 "S - ms" "DeadlockStates(p')")} & \text{Instantiating quantified variables,} \\
& \text{Q.E.D.}
\end{align*} \]
2- Theorems involving induction and case analysis.

**Lemma (1):** In the presence of faults, no computation prefix of a nonmasking-masking program that starts from a state in $S'$, reaches a state in $ms$:

$$\forall j: (\forall c: \text{prefix } (p' \cup f_{\text{masking}}, j) \mid c_0 \in S' :$$

$$\forall k \mid k < j: c_k \not\in ms)$$
Proof Idea on Satisfying Safety

Use induction on $j$

- Skolemize

[Program transition] [Fault transition]

- Trivial
- Apply the fixpoint theorem
3- Theorems involving only application of another theorem.

**Theorem (2)**: In the presence of faults, no computation prefix of a failsafe fault-tolerant program that starts from a state in $S'$ violates safety:

$$\forall j: (\forall c: \text{prefix} (p' \cup f_{\text{masking}} , j) \mid c_0 \in S' : \forall k \mid k < j: (c_k , c_{k+1}) \notin \Sigma_{bt})$$

**Proof**: By applying Lemma (1).
4- Theorems involving application of a combination of other lemmas, theorems, and possibly other things.

**Theorem (3)**: In the presence of faults, any computation of a nonmasking-masking program that starts from a state in the state space, reaches the invariant $S'$:

$$\forall c \ (p \cup f_{\text{nonmasking-masking}}) \ : \ (\exists j \ | \ j > 0 : c_j \in S_1).$$
Proof Idea

Skolemize

[Program]

[\(f_{\text{masking}}\)]

[\(f_{\text{nonmasking}}\)]

Trivial

Reuse reachability proof of masking f-span

Reuse one-step recovery to f-span of masking
Future Work

- Developing proof strategies

- Verifying the correctness of other synthesis algorithms that:
  - Add fault-tolerance to real-time programs
    [Bonakdarpour and Kulkarni, SSS’06]
  - Enhance the level of fault-tolerance
    [Kulkarni and Ebnenasir, ICDCS’03]
Problem Statement

- **Soundness**: Given, $S, p, f, \Sigma_{bt}$, If $p'$ is the set of transitions of fault-tolerant program with invariant $S'$:
  1. $S' \subseteq S$
  2. $p' \subseteq p$
  3. $p'$ is fault-tolerant (nonmasking / masking) from $S'$