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.

Motivation (cont.)

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.

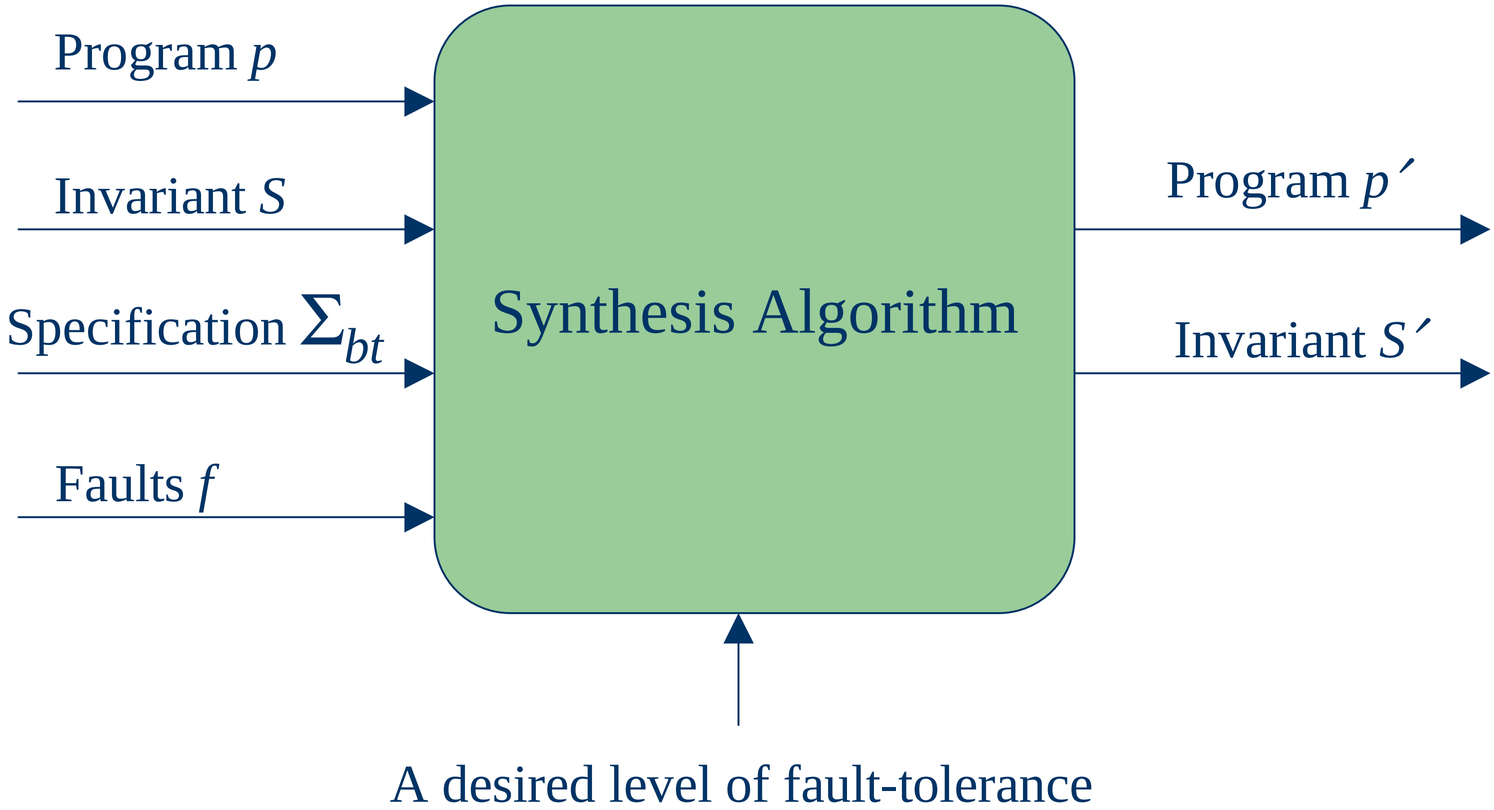
A Fault-Tolerance Framework in PVS

FT [state : **TYPE**]: **THEORY**
BEGIN
ASSUMING
ST_is_finite : **ASSUMPTION** is_finite_type[state] — *State* is a finite type
TR_is_finite : **ASSUMPTION** is_finite_type[[state, state]] — *Transition* is a finite type
ENDASSUMING

Transition: **TYPE** = [state, state]
StatePred: **TYPE** = finite_set [state]
Action: **TYPE** = finite_set [Transition] — *set of transitions*
Computation (Z: Action): **TYPE** = $\{A\text{: sequence[state]} \mid \forall n\text{: } (A_n, A_{n+1}) \in Z\}$

StateSpace: StatePred = fullset [state] — *The state space*
S: StatePred — *invariant of fault-intolerant program*
p: Action — *program*
f: Action — *set of faults*
Σ_{bt}: Action — *set of bad transitions*

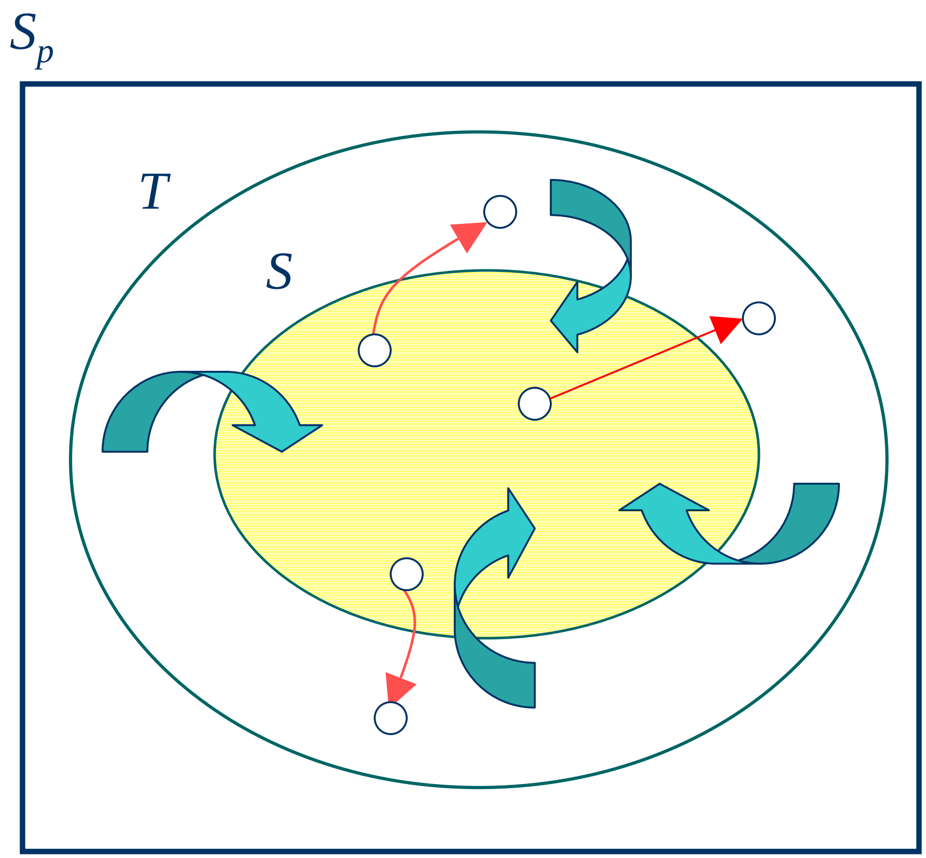
The Synthesis Problem



Automatic Synthesis of Nonmasking Tolerance

- $S' = S$
- $p' = p \cup$

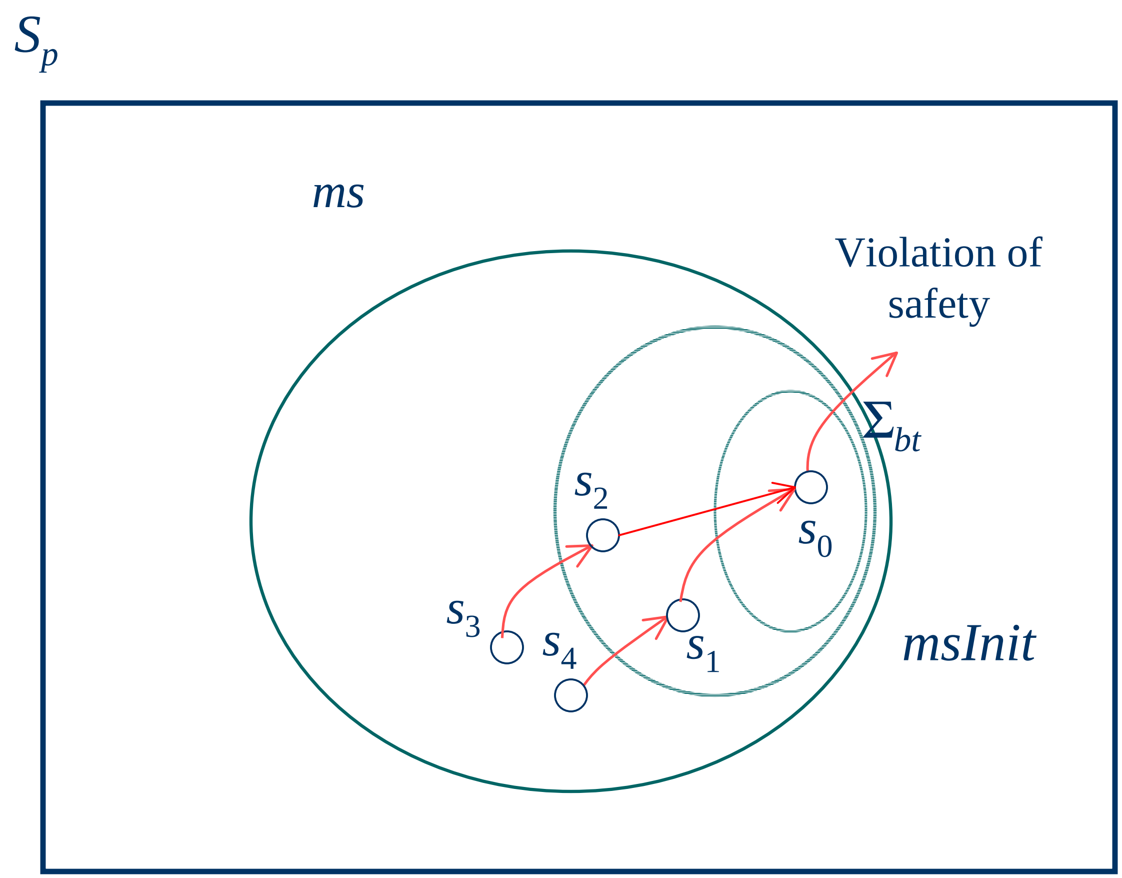
  $\{(s_0, s_1) \mid s_0 \in T - S \wedge s_1 \in S\}$



Automatic Synthesis of Masking Tolerance

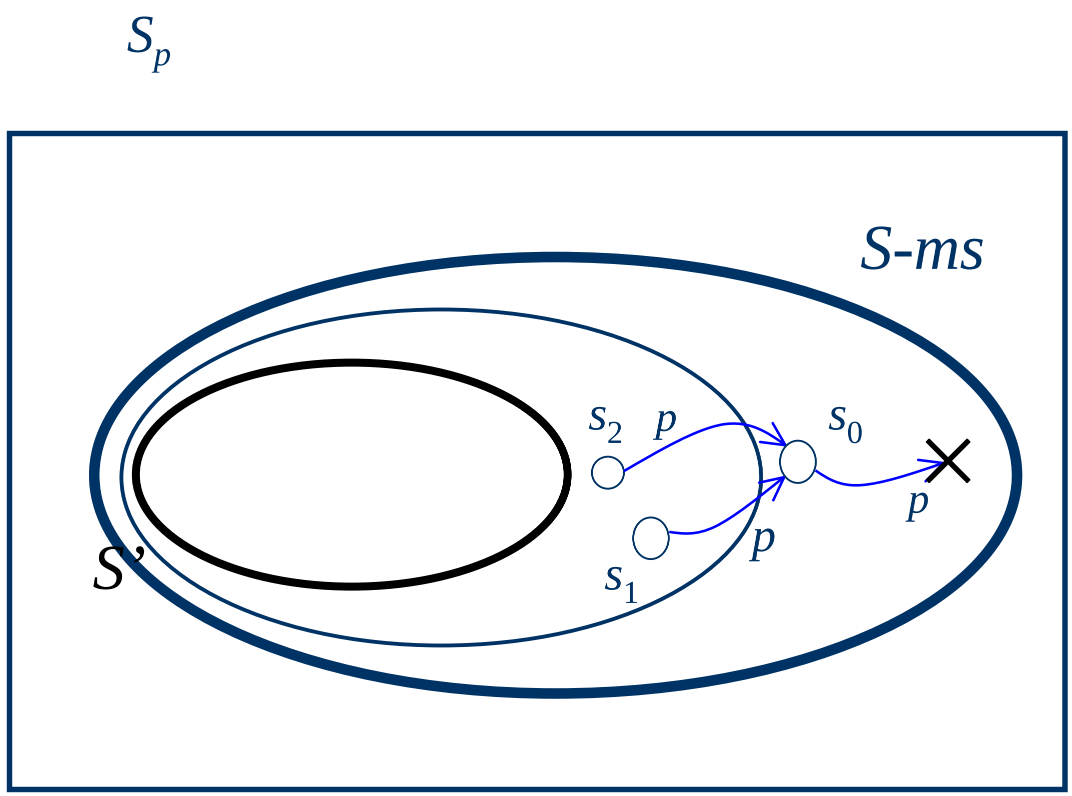
Step (1): Identifying the set of states and transitions from where safety may be violated by a sequence of fault transitions.

$$mt = \{(s_0, s_1) \mid s_1 \in ms \vee (s_0, s_1) \in \Sigma_{bt}\}$$



Automatic Synthesis of Masking Tolerance (cont.)

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***:

$$X_1 = X - g\,(X)$$

$$X_2 = X_1 - g\,(X_1)$$

$$\vdots$$

$$X_n = X_{n\text{-}1} - g\,(X_{n\text{-}1}) \ \text{ where } g\,(X_{n\text{-}1}) \ = \ \varnothing$$

$$X_n = X_{n\text{-}1}$$

$$X_{n+1} = X_n$$

Largest Fixpoint

DecFunc: **TYPE** = [A : **StatePred** $\rightarrow$ {B: **StatePred** | $B \subseteq A$)}]

Dec (i : **nat**, X : **StatePred**)(g : **DecFunc**): **RECURSIVE** **StatePred** =

IF $i = 0$ **THEN**

X

ELSE

$$Dec(i-1, X)(g) - g(Dec(i-1, X)(g))$$

ENDIF

MEASURE (λ (x : **nat**, y : **StatePred**): x)

LgFix (X : **StatePred**)(g : **DecFunc**): **StatePred** =

$\{s \mid \forall\ (k\text{: }\mathbf{nat})\text{: } s \in \text{Dec}\ (k, X)(g))\}$

Largest Fixpoint and Deadlock States

Theorem [LOPSTR'04]**:** Further recalculation of fixpoint returns the empty set :

$$g\ (LgFix\ (X)(g)) = \emptyset$$

$DeadlockStates\ (p$: **Action**$)(ds$: **StatePred**$)$: **StatePred** $=$
$\{s_0 \mid (s_0 \in ds) \wedge (\forall\, s_1 : (s_1 \in ds) \Rightarrow (s_0, s_1) \notin p)\}$

S_1: **StatePred** $= LgFix\ (S - ms)(DeadlockStates(p - mt))$

Automatic Addition of Masking Fault-Tolerance

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

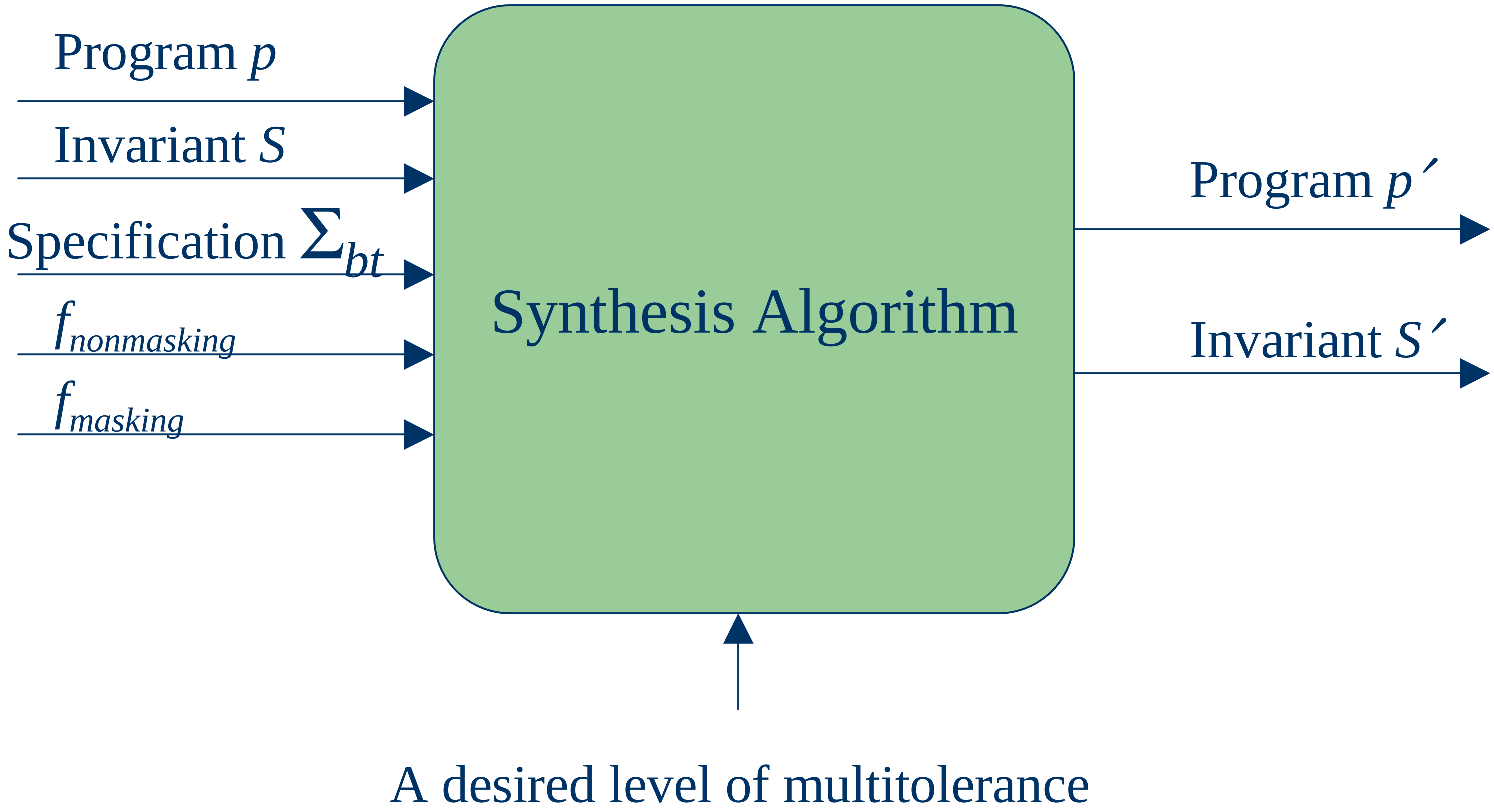
Automatic Synthesis of Multitolerance [DSN'04]

- ***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:

Level of FT	***Nonmasking***	***Masking***
Nonmasking	Nonmasking	Nonmasking
Masking	Nonmasking	Masking

Revisiting the Synthesis Problem



Generalizing Formal Specification

add_multift [state : **TYPE**]: **THEORY**

BEGIN

IMPORTING add_nonmasking[state]
IMPORTING add_masking[state]

$f_{nonmasking}$: Action
$f_{masking}$: Action
$f_{nonmasking\text{-}masking}$: Action $= f_{nonmasking} \cup f_{masking}$

msInit(anyFault : Action) : StatePred =
$\{s_0 \mid \exists s_1 : ((s_0, s_1) \in \text{anyFault} \wedge (s_0, s_1) \in \Sigma_{bt})\}$ *// faults directly violate safety*

RevReachStates(anyFault : Action)(*rs* : StatePred) : StatePred = *// backward reachability*
$\{s_0 \mid \exists s_1 : (s_1 \in rs \wedge (s_0, s_1) \in \text{anyFault} \wedge s_0 \notin rs)\}$

ms(anyFault : Action) : StatePred =
SmFix (msInit(anyFault))(RevReachStates(anyFault)) *// Fixpoint of RRS*

Formal Spec. of Nonmasking-Masking Synthesis

// invariant

S' : StatePred = add_masking . $S_1(f_{masking})$

// intermediate program transitions

p_1 : Action = add_masking . $p_1(f_{masking})$

// faults-span

T_1 : StatePred = add_masking . $T'(f_{masking})$

// program transitions

p' : Action = add_nonmasking . $p'(T_{masking}(\mathrm{f}_{nonmasking\_masking}), p_1(f_{masking}))$

Verification of Synthesis of Multitolerance

1- Theorems involving fixpoint calculations.

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

$$DeadlockStates(p')(S') = \{\}$$

Formal Proof of Theorem (1)

|-------
{1} *DeadlockStates*(*p*')(*S*') = ∅

Rule? (**expand "S' "**)
theorem1 :

|-------

{1} DeadlockStates(p')
(*LgFix* (*S* - *ms*))
(*DeadlockStates* (*p* - *mt*))))

Rule? (**lemma "theorem1"**)
Applying theorem1
this simplifies to:
theorem1 :

{-1} ∀ (*X*: StatePred[state], *g*: DecFunc[state]):
g (*LgFix*(*X*)(*g*)) = ∅
|-------
[1] *DeadlockStates* (*p*')
(*ConstructInvariant* (*S* - *ms*, *p* – *mt*)) = ∅

Rule? (**inst -1 " *S - ms*" "*DeadlockStates(p')*"**)
Instantiating quantified variables,
Q.E.D.

More Theorems…

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:

$$\forall j : (\forall c : prefix\,(p' \cup f_{masking},\ j) \mid c_0 \in S' : \forall k \mid k < j : c_k \notin ms)$$

Proof Idea on Satisfying Safety

Use induction on *j*

Skolemize

[Program transition]

Trivial

[Fault transition]

Apply the fixpoint theorem

More Theorems…

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:

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

Proof : By applying Lemma (1).

More Theorems…

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':

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

Proof Idea

Skolemize

[Program]

Trivial

[$f_{masking}$]

Reuse reachability proof of masking f-span

[$f_{nonmasking}$]

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, Σ_{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'