A Formal Verification of the SPIDER Reintegration Protocol

Lee Pike

Formal Methods Group
NASA Langley Research Center
lee.s.pike@nasa.gov

May 12, 2005
Acknowledgments

- The reintegration protocol was developed by Wilfredo Torres-Pomales, Mahyar Malekpour, and Paul Miner (NASA LaRC).
- Bruno Dutertre and Leonardo de Moura (SRI) provided many helpful suggestions concerning Timeout Automata and SAL.
SPIDER Overview

The Reintegration Protocol

The Verification
   SAL
   $k$-Induction
   An Optimized Model of Timed Computation

Conclusion
Motivation

- Safety-critical distributed x-by-wire applications are deployed in inhospitable environments.
- Failure rates must be on the order of $10^{-9}$ per hour of operation.
Bus Architecture Desiderata

- Integration
  - Off-the-shelf application integration
  - Off-the-shelf fault-tolerance
  - Eliminate redundancy
- Partitioning
  - Fault-partitioning
  - Modular certification
- Predictability
  - Hard real-time guarantees
  - A “virtual” TDMA bus

---

1John Rushby’s *A Comparison of Bus Architectures for Safety-Critical Embedded Systems*
SPIDER Architecture

Processor Elements

ROBUS

Software

Hardware
BIU/RMU Modes of Operation

- Self-Test Mode
- Initialization Mode
- Preservation Mode
- Reintegration Mode

Continuous on-line diagnosis...
The Frame Property

- $l$: number of faulty nodes not accused by the reintegrator
- $\pi$: maximum skew of nonfaulty nodes
- $P$: frame duration

\[ P > l\pi + 2\pi \]
Reintegration Overview

- Preliminary Diagnosis Mode
- Frame Synchronization Mode
- Synchronization Capture Mode
Safety Properties

Theorem (No Operational Accusations)
For all operational nodes $i$, $\text{accs}[i]$ does not hold during the reintegration protocol.

Theorem (Synchronization Acquisition)
For all operational nodes $i$, $|\text{clock} - \text{echo}(i)| < \pi$ upon termination of the reintegration protocol.
Motivation

- Next formal verification challenge in SPIDER.
- First formal verification of a reintegration protocol (called for by Rushby\textsuperscript{2}).
- Clique-avoidance.
- Uses recently-developed and relatively unstudied techniques combining bounded model-checking and decision procedures.

\textsuperscript{2}Overview of the Time-Triggered Architecture, 1999.
SRI’s SAL Toolset

- Symbolic model-checker (BDDs)
- Witness symbolic model-checker
- Bounded model-checker
- Simulator
- Parser
- Infinite-state bounded model-checker
- Future releases include:
  - Explicit-state model-checker
  - MDDs in the future for symbolic model-checking

All of which are “state-of-the-art”
The Language: Bakery Example

PC: TYPE = {sleeping, trying, critical};

job: MODULE =
BEGIN
  INPUT  y2 : NATURAL
  OUTPUT y1 : NATURAL
  LOCAL  pc : PC

  INITIALIZATION
  pc = sleeping;
  y1 = 0

  TRANSITION
  [
    pc = sleeping --> y1' = y2 + 1;
    pc' = trying
  ]
  []
  pc = trying AND (y2 = 0 OR y1 < y2) --> pc' = critical
  []
  pc = critical --> y1' = 0;
  pc' = sleeping
]
END;
Induction (over Transition Systems)

Let \( \langle S, S^0, \rightarrow \rangle \) be a transition system.

For state predicate \( I \), show

- **Base**: If \( s \in S^0 \), then \( I(s) \);
- **IS**: If \( I(s) \) and \( s \rightarrow s' \), then \( I(s') \).

Conclude that for all reachable \( s \), \( I(s) \).
Strengthening Induction

Induction can be generalized in two ways.

- Strengthen the invariant (hard!)
- Strengthen the induction principle...
For state predicate $I$, show

- **Base**: If $s_0 \in S^0$, then for all trajectories $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$, $I(s_i)$ for $0 \leq i \leq k$;
- **IS**: For all trajectories $s_0 \rightarrow s_1 \rightarrow \ldots \rightarrow s_k$, if $I(s_i)$ for $0 \leq i \leq k - 1$, then $I(s_k)$.

Conclude that for all reachable $s$, $I(s)$.

Induction is the special case when $k = 1$. 
Induction

States

Reachable states

I(s)

A Formal Verification of the SPIDER Reintegration Protocol

Lee Pike
$k$-Induction

- States
- Reachable states
- $I(s)$

[Diagram showing states and reachable states with arrows indicating transitions]
Timeout Automata\(^3\) (Semantics)

An *explicit* real-time model.

▶ **Vocabulary:**
  ▶ A set of state variables.
  ▶ A *global clock*, \( c \in \mathbb{R}^{0\leq} \).
  ▶ A set of *timeout* variables \( T \) such that for \( t \in T, t \in \mathbb{R}^{0\leq} \).

▶ **Construct a transition system** \( \langle S, S^0, \rightarrow \rangle \):
  ▶ States are mappings of all variables to values.
  ▶ Transitions are either *time transitions* or *discrete transitions*.
    ▶ Time transitions are enabled if the clock is less than all timeouts. Updates clock to least timeout.
    ▶ Discrete transitions are enabled if the clock equals some timeout. Updates state variables and timeouts.

---

No Free Lunch

$k$-induction is exponential with respect to $k$.

- Goal: reduce the size of $k$ for $k$-induction.
- Approach:
  - Optimize the formal model (timeout automata).
  - Optimize the model of the physical world.
Optimization 1: Synchronous Communication

- Communication via shared variables.
- Usual state machine semantics:
  - A transition in which variables are updated by the sender.
  - A transition in which the variables are read.
- Under synchronous semantics, next-state values can be used in guards.

Train-Gate-Controller verification reduced from $k = 14$ to $k = 9$. 
Synchronous Communication SAL Example

Asynchronous Composition

train: MODULE =
    t.state = t0
    AND t.to = time
-->
    t.state' = t1;
    flag1' = TRUE;
    msg1' = approach;

controller: MODULE =
    c.state = c0
    AND flag1 = TRUE
    AND msg1 = approach
-->
    c.state' = c1;
    flag1' = FALSE;
Synchronous Communication SAL Example

Asynchronous Composition

train: MODULE =
  t_state = t0
  AND t_to = time
 -->
    t_state’ = t1;
    flag1’ = TRUE;
    msg1’ = approach;

controller: MODULE =
  c_state = c0
  AND flag1 = TRUE
  AND msg1 = approach
 -->
    c_state’ = c1;
    flag1’ = FALSE;

Synchronous Composition

train: MODULE =
  t_state = t0
  AND t_to = time
 -->
    t_state’ = t1;
    msg1’ = approach;

controller: MODULE =
  c_state = c0
  AND t_to = time
  AND msg1’ = approach
 -->
    c_state’ = c1;
Optimization 2: Clockless Semantics

- Remove *time transitions* from the semantics.
- Transitions guarded by a timeout $t$ are enabled if $t$ is the least of all timeouts.
- Train-Gate-Controller verification reduced from $k = 9$ to $k = 5$. 
Optimization 3

- Typically, a state transition is taken each time the state changes.
- Another approach: “time-triggered simulation.”
- At fixed intervals of time
  - Determine the sequence of events observed by the reintegrator.
  - Update the state of the reintegrator based on these observations simultaneously.

In a timeout-automata model, care must be taken to ensure that the simulation is conservative...
The Peril of Time-Triggered Simulation
The Peril of Time-Triggered Simulation
The Peril of Time-Triggered Simulation
Future Work

- Benchmarks comparing real-time verification technologies (e.g., UPPAAL & SAL).
- Theoretical results for explicit real-time models of computation in formal verification.
- Complete clique-avoidance proof.
Further Information

Some Talks & Papers

http://www.cs.indiana.edu/~lepike/
Google: lee pike

SPIDER Homepage

http://shemesh.larc.nasa.gov/fm/spider/
Google: formal methods spider

NASA Langley Research Center Formal Methods Group

http://shemesh.larc.nasa.gov/fm/
Google: nasa formal methods
State Variables & Initialization

- **accs**: ARRAY of booleans, one for each monitored node
- **seen**: ARRAY of naturals, one for each monitored node
- **mode**: \{prelim\_diag, frame\_synch, synch\_capture\}
- **clock**: $\mathbb{R}^0 \leq$
- **fs\_finish**: $\mathbb{R}^0 \leq$
- **pd\_finish**: $\mathbb{R}^0 \leq$

for each $i$, accs$[i]$ := false;
mode := prelim\_diag;
for each $i$, seen$[i]$ := 0;
Preliminary Diagnosis Mode

\[
pd_{\text{finish}} := \text{clock} + P + \pi;
\]

while \( \text{clock} < pd_{\text{finish}} \) do {
  for each \( i \), when \( \text{echo}(i) \) do {
    if (seen[i] < 2 and not accs[i])
      then seen[i] := seen[i] + 1
    else accs[i] := true;
  }
}

for each \( i \), if \( \text{seen}[i] = 0 \) then \( \text{accs}[i] \);

\( \text{mode} := \text{frame} \_ \text{synch} \);
Frame Synchronization Mode

\[
\text{for each } i, \ seen[i] := 0; \\
fs_{\text{finish}} := clock; \\
\text{while } clock - fs_{\text{finish}} < \pi \text{ do } \{ \\
\quad \text{for each } i, \text{ when } echo(i) \text{ do } \{ \\
\quad \quad \text{if } (seen[i] = 0 \text{ and not } accs[i]) \\
\quad \quad \quad \text{then } \{ \\
\quad \quad \quad \quad fs_{\text{finish}} := clock; \\
\quad \quad \quad \quad seen[i] := seen[i] + 1; \\
\quad \quad \quad \}; \\
\quad \quad \text{else } accs[i] := true; \\
\quad \}; \\
\}; \\
mode := synch_{\text{capture}};
\]
Synchronization Capture Mode

for each \( i \), \( seen[i] := 0; \)
while \( seen\_cnt \leq trusted/2 \) do {
   for each \( i \), when \( echo(i) \) do {
      if (\( seen[i] = 0 \) and not \( accs[i] \))
         then \( seen[i] := seen[i] + 1; \)
   }
};
\( clock := 0; \)