Temporal Refinement Using SMT and Model Checking with an Application to Physical-Layer Protocols

Lee Pike (Presenting), Galois, Inc.
leepike@galois.com

Geoffrey M. Brown, Indiana University
geobrown@cs.indiana.edu

May 31, 2007
Problem: Verify a Parameterized UART Design

Universal asynchronous receiver-transmitters (UARTS) are hardware devices that allow two independently-clocked units to reliably communicate serial data. They implement a real-time protocol (e.g., 8N1). UARTS can be found in both systems like

- Microcontrollers in, e.g., microwaves ovens.
- ROBUS nodes in NASA’s SPIDER fly-by-wire bus.
“Reliably transmitting” serial data requires real-time constraints to be met that relate the

- **relative drift rates** of the sender’s and receiver’s clocks,
- **clock jitter** and other adverse affects on the relative clock rates,
- **signal stabilization** (reliable sampling) and **signal settling** (unreliable sampling) time on the wire.
Unreliable Sampling

Data Sent

Received Signal

Decoder Clock
Problem: Verify a Parameterized UART Design

Desiderata:

- A **parameterized proof** over a range of clock frequencies, error rates, etc.
- An **automated proof** (comparisons with PVS and ACL2 follow).
- A **compositional proof** (composing a real-time protocol specification with a synchronous hardware specification).
Some Approaches for Real-Time Verification

Finite-state model checking (e.g., using BDDs)

- Is great for verifying abstract asynchronous-interleaving models of real-time protocols and synchronous hardware.
- An error in the spec means an error in the real-time implementation.

But correctness of the spec doesn’t guarantee correctness of the implementation...
Some Approaches for Real-Time Verification

Goal: show that real-time constraints ensure the protocol behaves correctly. Some approaches:

- You can try real-time model-checking (e.g., Uppaal).
- You can try mechanical theorem-proving (e.g., PVS, ACL2).
- You can try infinite-state bounded model-checking – inf-bmc MC (i.e., SMT + some algorithm for checking LTL safety properties, like $k$-induction).

Let’s look at these three choices in turn...

By the way, the 8N1 protocol is just representative: another physical layer protocol is the Biphasé Mark protocol (BMP) used in CD player decoders and ethernet, for example.
You can try real-time model-checker

- **Automation**: Fully-automatic (inf-bmc MC requires manually-stated invariants).
- **Compositionality**: Real-time model checkers aren’t particularly good for specifying and verifying synchronous hardware.
- **Parameterization**: Although partially-parameterized BMP verifications exist using real-time model-checkers (Uppaal and HyTech), no fully-parameterized verification exists.
- Ultimately, SMT + MC is ultimately more powerful – you can verify a broad range of theories (e.g., lists + linear real arithmetic + fixed-width bitvectors + ...).
You can try mechanical theorem-proving

**Automation:** Compared to our verification of BMP in SAL (TACAS, 2006):

- One PVS effort required 37 invariants and 4000 individual proof directives (before “optimizing” the proofs).
- Ours required five invariants, each of which is proved automatically by SAL.
- In the other PVS effort, it takes 5 hours for PVS to check the manually-generated proof scripts.
- Ours requires just a few minutes to generate the proofs.
- J. Moore reports the BMP verification as one of his “best ideas” in his career.\(^1\)
- Our initial effort in SAL took a couple days.

...And we found a significant bug in a UART application note.

\(^1\)http://www.cs.utexas.edu/users/moore/best-ideas/
Or, you can use SMT + MC

Inf-bmc MC may satisfy our desiderata of parameterized and automated proofs, but so far, not compositionality: An invariant constructed for inf-bmc MC must apply to both the synchronous hardware and real-time constraints.

So how do we get the best of both worlds: automated, compositional proofs that apply to a high-fidelity real-time model?

Answer: Automated temporal refinement proofs.
## Approach Overview

<table>
<thead>
<tr>
<th>Specification</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Finite-state model</td>
<td>• Infinite-state model (time is modeled by $\mathbb{R}$)</td>
</tr>
<tr>
<td>• Nondeterministic interleaving asynchronous semantics</td>
<td>• Linear real-time constraints</td>
</tr>
<tr>
<td>• Safety properties proved by BDD model checking</td>
<td>• Safety property inherited from the specification</td>
</tr>
<tr>
<td>• Compositional with other finite-state specifications</td>
<td>• Refinement proof, for real-time portion by <em>infinite-state bounded model checking</em></td>
</tr>
</tbody>
</table>
Outline for the rest of the talk

1. Basic physical layer protocol model
2. Refinement

Some things I will not discuss (but are in the paper, specs online, and slides appendix):

- Composition of synchronous HW specs with the protocol model.
- How to easily generate invariants using inf-bmc model-checking:
  1. $k$-induction
  2. Disjunctive invariants
Generic Architecture for Physical-Layer Protocol Models

General model (for both the finite-state specification and infinite-state refinement):

```
tx
  tenv -- tbit --> tenc
  tready

  tclock

rx
  rdec
  rbit

  rclock
  tdata
```
SAL Composition

**Finite-state model:**

\[
\begin{align*}
\text{tx} & : \text{MODULE} = \text{tclock} \parallel \text{tenv} \parallel \text{tenc}; \\
\text{rx} & : \text{MODULE} = \text{rclock} \parallel \text{rdec}; \\
\text{system} & : \text{MODULE} = \text{tx} [] \text{rx};
\end{align*}
\]

**Infinite-state model:**

\[
\begin{align*}
\text{tx}_\text{rt} & : \text{MODULE} = \text{tclock}_\text{rt} \parallel \text{tenv} \parallel \text{tenc}; \\
\text{rx}_\text{rt} & : \text{MODULE} = \text{rclock}_\text{rt} \parallel \text{rdec}_\text{rt}; \\
\text{system}_\text{rt} & : \text{MODULE} = (\text{tx}_\text{rt} [] \text{rx}_\text{rt}) \parallel \text{constraint};
\end{align*}
\]

In the talk, we’ll focus on the clocks, where the principle refinement is, and skip the other modules (check out the paper, though!).


Temporal Refinement Using SMT and Model Checking with an Application to Physical-Layer Protocols

Refinement Approach

We demonstrate an Abadi-Lamport refinement mapping:\(^2\)

I implements S if every externally visible behavior of I is also allowed by S. We prove that if I allows the behavior

$$\langle\langle (e_0, z_0), (e_1, z_1), (e_2, z_2), \ldots \rangle\rangle$$

where each e_i is an externally-visible state, and where each z_i is an internal state, then there exist internal states y_i such that S allows

$$\langle\langle (e_0, y_0), (e_1, y_1), (e_2, y_2), \ldots \rangle\rangle$$

Refinement in SAL: Guard Weakening

Refinement mappings can be difficult to discover. For our models, they usually reduce to guard weakening in SAL:

Let $I = G_0 \rightarrow S_0[] \ldots [] G_N \rightarrow S_N$

Let $S = G'_0 \rightarrow S_0[] \ldots [] G'_N \rightarrow S_N$

Theorems of the form $G_i \Rightarrow G'_i$ are the refinement conditions.
Refining the Clocks

- The main refinement is from finite-state to infinite-state clocks.
- Prove infinite-state guards imply finite-state guards.

For the transmitter’s clock:

- **Finite-state guard**: \( t\text{state} = r\text{state} \) (i.e., the transmitter is ready to send the next bit).
- **Infinite-state guard**: \( t\text{clk} = \text{time}(t\text{clk}, r\text{clk}) \) (i.e., it’s the transmitter’s turn to execute, based on linear constraints).

\[
\text{tclock\_thm} : \text{THEOREM system\_rt \mid- } G(t\text{clk} = \text{time}(t\text{clk}, r\text{clk}) \\
\qquad \Rightarrow t\text{state} = r\text{state});
\]

And similarly for the receiver’s clock.

- Proof by \( k \)-induction over the infinite-state model.
Lingering Thoughts on Real-Time Verification Using SMT

We use what Leslie Lamport calls an *explicit-time* model\(^3\) for real-time verification without a real-time model-checker. Some benefits:

- No new languages and simple semantics (*timeout automata*\(^4\)).
- SMT is extensible (the theory of arrays, lists, uninterpreted functions, etc.)
- Compositional with non real-time specifications.

Possible future work we’d like to see:

- Algebraic framework for generating refinement conditions.
- Data refinement.
- Dealing with non-linear temporal constraints.

\(^3\) *CHARME*, 2005
Getting our Specifications and SAL

8N1 and BMP Specs & Proofs


Google: pike sal refinement

SRI’s SAL

http://sal.csl.sri.com

Google: SRI SAL

Thanks:

- Learned about real-time verification in SAL from a talk by Bruno Dutertre at the National Institute of Aerospace.
- Initial work motivated by SPIDER and began at NASA Langley.
Appendix.
Finite-State Clock Specifications

STATE : TYPE = [0..9];
tclock : MODULE =
BEGIN
  INPUT rstate : STATE
  INPUT tstate : STATE
  TRANSITION [ tstate = rstate --> ]
END;
rclock : MODULE =
BEGIN
  INPUT tstate : STATE
  INPUT rstate : STATE
  TRANSITION [ rstate /= tstate
              OR tstate = 9 --> ]
END;
The Clocks

- Finite-state: clocks enforce the proper interleaving of the sender’s and receivers’ states.
- Possible finite-state interleavings (“…” means idling and “⋯” means truncation for readability):

\[(t_{\text{state}}, r_{\text{state}}) = (9, 9), (9, 9), \ldots, (0, 9), \ldots, (0, 0), (1, 0), (1, 1), \ldots, (8, 7), (8, 8), (9, 8), (9, 9), \ldots\]

Recall the 8N1 protocol description.
Finite-State Correctness

**Main Correctness Theorem:** When \( rx \) is sampling the bit just sent by \( tx \), and \( rx \) is not in its initial state, then the bit sent is the bit received.

\[
\text{Serial_Thm} : \text{THEOREM system} \vdash G( tstate = rstate \quad \text{AND} \quad rstate /= 9 \Rightarrow rbit = tbit);
\]

(Proved with BDDs.)
In the infinite-state implementation, clocks enforce a nondeterministic real-time interleaving of the asynchronous modules.

The model used is an explicit real-time model *Timeout Automata*.\(^5\)

**Intuition:**
- Timeouts are associated with state-machines (or SAL modules).
- A timeout represents the future time at which the state-machine will make a transition (i.e., update its state variables).
- When a state-machine transitions, its timeout is updated (possibly nondeterministically) to some future time.
- The “current time” is the least-valued timeout.

---

Construct a transition system $\langle S, S^0, \rightarrow \rangle$:

- A partition on the state variables for $S$, and associated with each partition is a timeout $t \in \mathbb{R}$.
- A set of transition relations, such that $\rightarrow_t$ associated with timeout $t$ and is enabled if for all timeouts $t'$, $t \leq t'$ ($\rightarrow$ is the union of $\rightarrow_t$ for all $t$.)
Infinite-State Clock Implementation

- Returns the least-valued timeout between the transmitter and receiver:
  \[
  \text{time}(t1: \text{TIME}, t2: \text{TIME}): \text{TIME} = \\
  \quad \text{IF } t1 \leq t2 \text{ THEN } t1 \text{ ELSE } t2 \text{ ENDIF};
  \]

- *Higher-order(!)* function giving a range over which a timeout may be updated:
  \[
  \text{timeout}(\text{min: TIME}, \text{max: TIME}): [\text{TIME} \rightarrow \text{BOOLEAN}] = \\
  \quad \{x : \text{TIME} \mid \text{min} \leq x \text{ AND } x \leq \text{max}\};
  \]

- Sample transition from the receiver:
  \[
  \text{rclock}_\text{rt} : \text{MODULE} = \\
  \quad \ldots \\
  \quad \text{TRANSITION} \\
  \quad [ \text{rclk} = \text{time(rclk, tclk)} \rightarrow \text{rclk'} \text{ IN} \\
  \quad \ldots \\
  \quad \quad \text{timeout(rclk + RSTARTMIN, rclk + RSTARTMAX)} \\
  \quad \ldots 
  \]
Real-Time Constraints

Infinite-state: clocks are constrained by linear inequalities, captured by the types of the following uninterpreted constants:

\[
\begin{align*}
TPERIOD &: \{x : \text{TIME} \mid 0 < x\}; \\
TSETTLE &: \{x : \text{TIME} \mid 0 \leq x \text{ AND } x < TPERIOD\}; \\
TSTABLE &: \text{TIME} = TPERIOD - TSETTLE; \\
RSCANMIN &: \{x : \text{TIME} \mid 0 < x\}; \\
RSCANMAX &: \{x : \text{TIME} \mid RSCANMIN \leq x \text{ AND } x < TSTABLE\}; \\
RSTARTMIN &: \{x : \text{TIME} \mid TPERIOD + TSETTLE < x\}; \\
RSTARTMAX &: \{x : \text{TIME} \mid RSTARTMIN \leq x \text{ AND } \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad x < 2 \times TPERIOD - TSETTLE - RSCANMAX\}; \\
RPERIODMIN &: \{x : \text{TIME} \mid 9 \times TPERIOD + TSETTLE < RSTARTMIN + 8 \times x\}; \\
RPERIODMAX &: \{x : \text{TIME} \mid RPERIODMIN \leq x \text{ AND } \\
&\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad TSETTLE + RSCANMAX + RSTARTMAX + 8 \times x < 10 \times TPERIOD\};
\end{align*}
\]

These capture the error terms represented graphically earlier.
SAL’s Language

- Typed with predicate subtypes.
- Infinite types (e.g., INTEGER and REAL).
- Synchronous (lock-step) and asynchronous (interleaving) composition (|| and [], respectively).
- Quantification (over finite types).
- Recursion (over finite types).
Induction (over Transition Systems)

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

For safety property $P$, show

- **Base**: If $s \in S^0$, then $P(s)$;
- **Induction Step**: If $P(s)$ and $s \rightarrow s'$, then $P(s')$.

Conclude that for all reachable $s$, $P(s)$. 
$k$-Induction Generalization

Generalize from single transitions to trajectories of fixed length.

For safety property $P$, show

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

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

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

States

P(s)

Reachable states
$k$-Induction
k-Induction

counter1: MODULE =
BEGIN
  LOCAL cnt : INTEGER
  LOCAL b : BOOLEAN
  INITIALIZATION
    cnt = 0;
    b = TRUE
  TRANSITION
    [ b --> cnt' = cnt + 2;
      b' = NOT b
    ] ELSE --> cnt' = cnt - 1;
    b' = NOT b
  ] END;

  Thm1 : THEOREM counter1 |- G(cnt >= 0);

  Circuit behavior: b = T F T F T F T F ...  
  cnt = 0 2 1 3 2 4 4 ... 

  Thm1 fails for k = 1, succeeds for k = 2 (why?).
Disjunctive Invariants

*Disjunctive invariants* can be used to weaken safety properties until they become invariant.

- Developed by Pneuli & Rushby, independently.
- A disjunctive invariant can be built iteratively to cover the reachable states from the counterexamples returned by SAL for the hypothesized invariant being verified.
Initial Attempt

States

I0

Reachable states

I0 Not invariant...
Generalization

States

\[ I_0 \lor I_1 \text{ Almost...} \]
Invariant

\( I_0 \lor I_1 \lor I_2 \) There we go!
Disjunctive Invariants

counter1: MODULE =

BEGIN

LOCAL cnt : INTEGER
LOCAL b : BOOLEAN

INITIALIZATION

cnt = 0;
b = TRUE

TRANSITION

[ b --> cnt’ = (-1 * cnt) - 1;
b’ = NOT b

[] ELSE --> cnt’ = (-1 * cnt) + 1;
b’ = NOT b

] END;

Thm2a : THEOREM counter2 |- G(b AND cnt >= 0);

Circuit behavior: 

\begin{align*}
\text{b} &= \ T\ F\ T\ F\ T\ F\ T\ F\ \ldots \\
\text{cnt} &= \ 0\ -1\ 2\ -3\ 4\ -5\ \ldots 
\end{align*}

Thm2a is our initial approximation ...
Disjunctive Invariants

... And fails

SAL's output:

Counterexample:

Step 0:
--- System Variables (assignments) ---
cnt = 0
b = true
------------------------

Step 1:
--- System Variables (assignments) ---
cnt = -1
b = false
------------------------

Thm2b : THEOREM counter2 |- G( (b AND cnt >= 0)
       OR (NOT b AND cnt < 0));

Thm2b succeeds.