Translating Timed I/O Automata to PVS

Hongping Lim, Dilsun Kaynar, Nancy Lynch and Sayan Mitra

Theory of Distributed Systems Group (TDS)
Computer Science and AI Lab, MIT

FORMATS 2005, Uppsala, Sweden
Project Goals

- Develop formal framework for modeling and reasoning about complex, interacting systems
  - Timing-dependent behavior: schedules, deadlines
  - Hybrid behavior: continuous interactions
  - Probabilistic behavior
- Build language for specifying formal models
  - Extend of IOA language
- Build Tool support based on the specification language
  - Interface to Theorem Provers
  - Simulator
  - Model checking
Flavors of I/O automaton models

- Infinite state automata with external interface, abstraction, composition
- Basic IOA (synchronous distributed algorithms)
  - Sequential order of actions, no timing information
  - Interaction through shared actions
- TIOA (timing based systems, hybrid systems || environment)
  - Actions and trajectories
  - Trajectories may describe complex continuous dynamics
  - No continuous interaction between components
- HIOA (embedded systems, software + physical processes)
  - Continuous interactions through shared variables
- PIOA, PTIOA, PHIOA (security protocols, stochastic hybrid systems)
  - Probabilistic transitions, trajectories, …
Outline

- Introduction
- TIOA model and language
- Translation to PVS
- Examples
TIOA model

**Timed I/O Automaton** [Kaynar,Lynch,Segala,Vaandrager]
- State variables $X$ ( + input/output variables = HIOA)
- Start states $\Theta$
- Actions $A$, partitioned into input, output, and internal subsets
- Discrete transitions $D$, $(x,a,x')$
- Trajectories $T$, $\tau$ maps interval of time to variable values

- Executions $\alpha = \tau_0 a_1 \tau_1 a_2 \tau_2 \ldots a_n \tau_n$
- Invariant properties, proof by induction

- Traces
- Simulation relations: sufficient conditions for $\text{traces}(A) \subseteq \text{traces}(B)$
- Composition, $A || B$

$\text{traces}(A) \subseteq \text{traces}(B) \Rightarrow \text{traces}(A || C) \subseteq \text{traces}(B || C)$
Two task example

- **Model & Language**
  - PVS Translation

- **Examples**

International Conference on Formal Modelling and Analysis of Timed Systems 2005
Variables

- \(\textit{count, flag, t}\)
- \(\textit{u_reset, l_reset}\)
- \(\textit{u_count, l_count}\)

+ action

- Precondition:
  - not \(\textit{flag}\) and \(t \geq l\_count\)
- Effect:
  - \(\textit{count}++\)
  - \(\textit{l_count} = t + a_1\)
  - \(\textit{u_count} = t + a_2\)

reset action

- Precondition:
  - not \(\textit{flag}\) and \(t \geq l\_reset\)
- Effect:
  - \(\textit{flag} = \text{true}\)

trajdef

- Evolve: \(d(t) = 1\)
- Stop when: \(t = u\_count\ or\ t = u\_reset\)
Upper bound for stop

How late can it **stop**?

\[ b_2 + a_2 + \frac{b_2 a_2}{a_1} \]

How early?

If \( a_2 \geq b_1 \) then \( a_1 \) else \( \min(b_1, a_1) + \frac{(b_1 - a_1)a_1}{a_2} \)
Case studies: Two task example

- Abstract automaton $B$ with one action $\text{stop}$
  - $u_{\text{stop}} = b_2 + a_2 + \frac{b_2a_2}{a_1}$
  - $l_{\text{stop}} = \text{if } a_2 \geq b_1 \text{ then } a_1 \text{ else}$
    \[ \min(b_1, a_1) + \frac{(b_1 - a_1)a_1}{a_2} \]

- Prove trace inclusion (time bounds for $\text{stop}$) with simulation relations

- Prove forward simulation $R \subseteq Q_A \times Q_B$
Simulation Relation

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Model &amp; Language</th>
<th>PVS Translation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>🗳 Requires creativity/insight to come up with the right $R$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🗳 Proof by induction</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. There are related start states</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Every action/trajectory of $A$ can be emulated by an execution fragment of $B$ with the same trace.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🗳 Use PVS prover [SRI] for proving interactively</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🗳 Strategies set up the induction and case analysis automatically</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>🗳 Nonlinear real inequalities handled by Field and Manip strategy packages [Muñoz, deVito]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$\sim \text{flag} \lor l\_\text{count} \leq u\_\text{reset} \rightarrow u\_\text{stop} \geq u\_\text{reset} + a_2 (\text{count} + 2 + (u\_\text{reset} - l\_\text{count})/a_1)$$

$$\text{flag} \lor l\_\text{count} > u\_\text{reset} \rightarrow u\_\text{stop} \geq u\_\text{count} + a_2 \text{count}$$

$$\sim \text{flag} \land u\_\text{count} < l\_\text{reset} \rightarrow l\_\text{stop} \leq \min(l\_\text{reset}, l\_\text{count}) + a_1 (\text{count} + (l\_\text{reset} - u\_\text{count})/a_2)$$

$$\text{flag} \lor u\_\text{count} \geq l\_\text{reset} \rightarrow l\_\text{stop} \leq l\_\text{count} + a_1 \text{count}$$
Translation to PVS

- Stylized proofs lead to partial automation [Archer, Mitra 05]
- Proof management (E.g., ABD implementation [Chockler, Lynch, Mitra, Tauber, DISC’05])
- Rechecking proofs after making changes to spec
- Generation of human readable proofs

- Rewrite TIOA specs for the theorem prover! Different language and style.

- Why not specify automata directly in PVS?
  - TIOA provides structures for natural description of automata
    - Programs for effects as opposed to functions or relations
    - Differential equations and stopping conditions for trajectories
  - Other TIOA tools
States and Actions

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Model &amp; Language</th>
<th>PVS Translation</th>
<th>Examples</th>
</tr>
</thead>
</table>

- **TIOA**
  - States variables →
  - Initial state →
  - Actions →
  - Preconditions →
  - Effects →
  - Trajdefs →

- **PVS**
  - tuple of variables
  - predicate on state variables
  - new datatype called *actions*
  - predicates on state variables, action parameters, automaton parameters
  - …
  - …
Translating action effects

- TIOA effects are nondeterministic, e.g.,
  \[ \text{plus\_something}(): \text{effect } x := x + \text{choose } [1,5]; \]

- TIOA effects are programs with operational semantics:
  \[
  x := x + 5; \ y := 2.x; \ldots
  \]

- We want the PVS effects to be functions:
  - \( \text{plus}(k \in [1,5]): x' := x + k \)
  - Substitution:
    \[
    x' = x + 5; \ y' = 2 (x + 5)
    \]
  - PVS assignments:
    \[
    s' = \text{LET } s := s \text{ WITH } [x := x(s) + 5] \text{ in}
    \]
    \[
    \text{LET } s := s \text{ WITH } [y := 2 \times x(s)] \text{ in } s
    \]
Trajdef $\rightarrow$ time_elapse action

- Trajdef
- Evolve: $d(x) = c$
- Stop when: $x \in D$

- PVS action
- time_elapse($t: \mathbb{R}^0$, $\tau: [0,t] \rightarrow \mathbb{R}^2$)
- Enabled if
  - for all $t_1 \in [0,t]$,
    - If $\tau(t_1) \in D$ then $t_1 = t$
    - $\tau(t_1) = x + c \cdot t_1$
- Effect
  - $s' = \tau(t)$

International Conference on Formal Modelling and Analysis of Timed Systems 2005
Works for general *trajdefs*

- **Trajdef**
- **Evolve:** \( d(x) = cx \)
- **Stop when:** \( x \in D \)

- **PVS action**
- **time_elapse** \((dt:R^{\geq 0}, \tau: [0,t] \rightarrow R^2)\)
- Enabled at s If
  - for all \( t_1 \in [0,t], \)
    - If \( \tau(t_1) \in D \) then \( t_1 = t \)
    - \( \tau(t_1) = x \cdot e^{ct_1} \)
- **Effect**
  - \( s' = \tau(t) \)
Case Studies
Fischer’s Mutual Exclusion Algorithm

- $N$ processes, each go through *try, test, etc.*, to get to *critical*
- Transitions determined by deadlines
- Single TIOA written as the composition of $N$ automata
- Safety property: no two processes are *critical* simultaneously

- Automata and properties translated to PVS
- Invariant proved using induction
### Small Aircraft Transportation System (SATS) [Muñoz, NASA]

- Discrete model: airport space partitioned into several logical zones
- Each zones represented by a queue
- Transitions represent aircrafts moving from one zone to another
- Various constraints on transitions

### Safety property: upper bound on the number of aircrafts in each zone

### TIOA description of system

- Special operators declared in TIOA and defined in PVS, e.g. recursive functions

### Translated to PVS

### Properties written directly in PVS and proved
Ongoing: ABD

- ABD atomic register implementation
  - Read and write quorums for 2 phase reads and writes
  - Partial functions & graphs

- Proof of correctness using an abstract Partial Order automaton
- Simulation proof, ABD implements PO-automaton

- Directly coded in PVS
Conclusion

- Translator is part of TIOA toolkit, implemented in Java

- Future directions:
  - Extend translator
    - systems with linear dynamics
    - composed TIOAs
  - Develop new PVS strategies
  - New case studies
    - Linear hybrid systems
    - Atomic registers implementations
    - Continuous SATS