Dependent Types for Verifying Real-Time Constraints in ATS

William Blair, Hongwei Xi

Motivation and Implementation

Many hard real-time applications are multi-periodic systems comprised of communicating tasks. Composing these systems is complicated by strict timeliness constraints and a preference for lockless communication in order to avoid deadlocks.

Prelude\textsuperscript{1} was developed to augment declarative synchronous languages like Lustre with real-time primitives to simplify this process.

Synchronous Language Domains

- Aerospace
- Transportation
- Industrial Control

ATS is primarily used as a safe front end to C. Its type system, however, is flexible enough to accommodate the semantics of synchronous languages like Prelude. Rather than replace Prelude with its simple syntax, we want to explore advantages of using ATS as a verification back end.

Clock Calculus

All values in Prelude are called Flows and have the following properties:

- Represent an infinite stream of tuples (v,t).
- The value v occurs at time t.
- Model communication between tasks.
- Each flow has its own clock (n:int, p: rat) where n is the period of the flow, and p its phase offset.
- Two flows are synchronous if they have the same clock.
- Example of 2 out-of-sync clocks:
  - E has clock (2, 0), O has clock (2, \( \frac{3}{2} \))

<table>
<thead>
<tr>
<th>Time</th>
<th>E ( v_0 )</th>
<th>( v_1 )</th>
<th>( v_2 )</th>
<th>( v_3 )</th>
<th>( v_4 )</th>
<th>( v_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td>( v_4 )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td>( v_4 )</td>
<td>( v_5 )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( v_3 )</td>
<td>( v_4 )</td>
<td>( v_5 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>( v_5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>( v_0 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Communication Between a Fast and Slow Task

A Prelude program is organized as a set of defined and imported nodes that take in and produce flows. Imported nodes are external C functions with annotations to denote their worst case execution time. If a node’s clocks are not given, they may be inferred from the program. Sensor flows are considered system input and actuator flows are system output.

Example of 2 out-of-sync clocks:

\( \text{Flow (a, n, p),} \) where \( a, n, p \) are positive integers.

\( \text{Flow (a, n, p),} \) where \( a, n, p \) are rational.

Clock Transformations

If we want communication between tasks, the programmer is required to make all flows in and out of imported nodes synchronous. He or she has several tools to accomplish this. In the table below, we consider a flow F of clock (50, 0) and show example flows from applying some basic operators.

The effects these transformations have on the flow’s clock is captured by dependent types in ATS. This allows us to use the type checker to check synchronization constraints outlined in Prelude’s Clock Calculus.

<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td>( v_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( F \times 2 )</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td>( v_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( F \div 2 )</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( 0 \times F )</td>
<td>( v_0 )</td>
<td>( v_1 )</td>
<td>( v_2 )</td>
<td>( v_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clocks as Dependent Types

We recently modified the ATS type checker to use a general purpose SMT solver for solving program constraints. This allows us to incorporate real numbers into the ATS statics and capture clock operator invariants with dependent types.

Encoding Transformations

For any flow, the date given by its phase offset must be valid (a natural number). In the following definitions, we enforce this so that if invalid flows are used, a type error will occur.

\text{fun}
\text{multiply (a:t@ype) \{ n,k:pos | divides(k,n) \} \{ pirat | is_nat (n * p) \} (}
\text{Flow (a, n, p), int k)}
\text{Flow (a, n / k, p * k) // over-sample}

\text{fun}
\text{divide (a:t@ype) \{ n,k:pos \} \{ pirat | is_nat (n * p) \} (}
\text{Flow (a, n, p), int k)}
\text{Flow (a, n * k, p / k) // under-sample}

\text{fun}
\text{shift (a:t@ype) \{ n:pos \} \{ pirat | is_nat(n*(p+k)) \} (}
\text{Flow (a, n, p), rational (k)}
\text{Flow (a, n, p + k) // phase shift}

Using the types we give to built-in operators like those above, we can type check ATS code compiled from Prelude programs to enforce timing constraints.

Future Work

- We would like to see if there are synchronous invariants ATS can capture that are out of reach for the default Prelude type checker.
- Some translation is required from ATS type errors to corresponding Prelude errors.
- A tool where Prelude programs are derived from schematics like the given diagram could be useful.

Reference