Using CPAL to model and validate the timing behaviour of embedded systems

Sebastian Altmeyer, Nicolas Navet, Loïc Fejoz
Cyber Physical Action Language (CPAL)

- C-like intuitive language (with automata and real-time abstractions)
- model functional and temporal behaviour of CPS
- simulate CPS (both types of behaviour)

(still under development)
The **challenging part** of the challenge

- **not** a **standard** scheduling problem
- hidden **ambiguity** in the model
- pen & paper solutions seemed trivial

How to **solve** the challenge with **CPAL**

- **low effort** to model the challenge
- **quick simulation** results
- explicit dis-ambiguity

(yet, simulation ≠ formal verification)
struct Frame {
    uint32: id;
    uint32: emission_time;
};

processdef T1_PreProcessor(
    in channel<Frame>: input,
    out channel<Frame>: output)
{
    state Main {
        /* removes reflections
         * normalizes intensity, etc. */
        assert(input.notEmpty());
        output.push(input.pop());
    }
}

var queue<Frame>: cam_to_t1[1];
var queue<Frame>: t1_to_t2[1];
var Frame: t2_to_t3;
var queue<Frame>: t3_to_t4[n];
var queue<Frame>: t4_to_monitor[1];

process T1_PreProcessor:
    t1[cam_to_t1.notEmpty()](cam_to_t1, t1_to_t2);
    @cpal:time {
        t1.execution_time = 28ms;
    }
    ...

Explicit Disambiguation

- task release times
- mutable or immutable clock drifts
- clock drift distribution
- execution time distribution

always the least-favorable configuration chosen
Simulation of Challenge 1A

- $10^8$ frames in total simulated (in less than 8 hours)
- $10^3$ release patterns, $10^5$ frames per pattern
- mutable drifts
- normal distributions
Simulation vs. Pen & Paper

<table>
<thead>
<tr>
<th>buffer (n)</th>
<th>frame</th>
<th>simulation</th>
<th>pen &amp; paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>63 ms</td>
<td>63 ms</td>
</tr>
<tr>
<td>1</td>
<td>&gt; 1</td>
<td>89.7694 ms</td>
<td>89.6656 ms</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>63 ms</td>
<td>63 ms</td>
</tr>
<tr>
<td>3</td>
<td>&gt; 1</td>
<td>90.0226 ms</td>
<td>89.6656 ms</td>
</tr>
<tr>
<td>max</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>144.9224 ms</td>
<td>&lt; 146 ms</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>222.9026 ms</td>
<td>&lt; 226 ms</td>
</tr>
</tbody>
</table>

Error in first pen & paper solution identified using simulation
Simulation of Challenge 1B

- $10^8$ frames in total simulated (in less than 8 hours)
- $10^3$ release patterns, $10^5$ frames per pattern
- Immutable drifts, worst-case clock drifts
- Normal distribution of exec time
Simulation of Challenge 1B: Observations

$n = 1$

- minimal distance: 2
- overload situations
- lost frames very frequent

$n = 3$

- minimal distance > 3800
- no bursts
- two spikes

No pen & paper solution to 1B.
CPAL Model of Challenge 2

Diagram:
- Sensors [100ms] → T6 [sensorsSource.notEmpty()] → T7 [tc_to_cc.notEmpty()] → Camera [cameraCmds.notEmpty()]
- T5 [tc_to_tpc.notEmpty()] → T2 [40ms] → Tracking & Camera Control
Simulation of Challenge 2

- CPAL simulation does not yet support pre-emption

- taskset \( T5, T6, T7 \) mutually non-pre-emptive (simulation possible)
- taskset \( T5, T6, T7 \) treated as artificial task \( Tx \):
- \( \Rightarrow \) reduction to standard response-time analysis!
Conclusions

CPAL doesn’t offer **automated** formal verification, but:

- **intuitive modelling** (< 4 hours for the both challenges)
- **quick simulation** (< 8 hours for all simulations)
- **unambiguous description**

Integration with formal verification tool future work.