Hybrid Systems Verification with Isabelle/HOL: Simpler Syntax, Better Models, Faster Proofs

Simon Foster[1] Mario Gleirscher[2]
Jonathan Julián Huerta y Munive[3] Georg Struth[4]

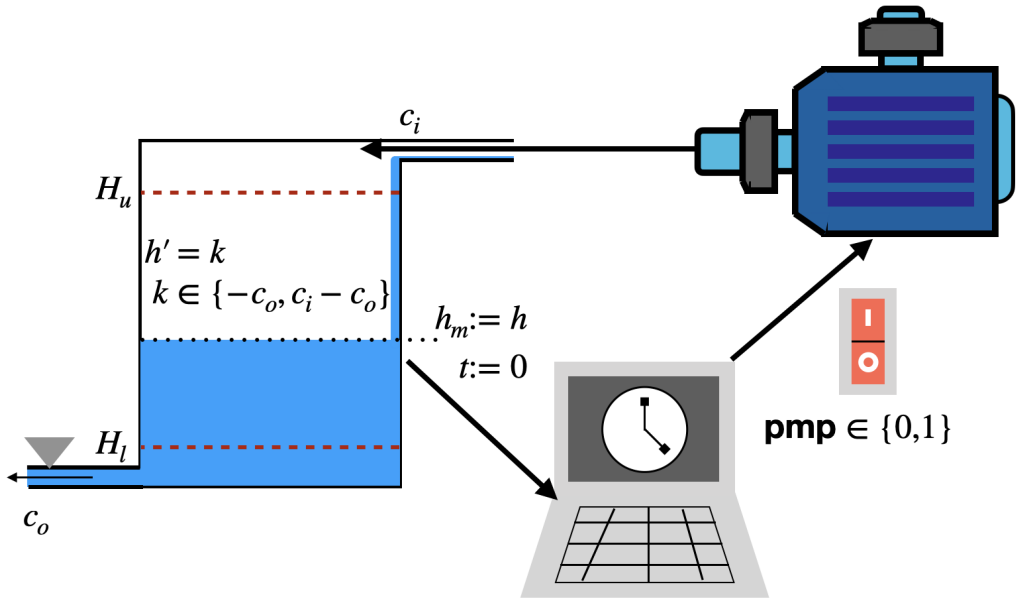
[1]University of York, UK

[2]University of Bremen, Germany

[3]University of Copenhagen, Denmark

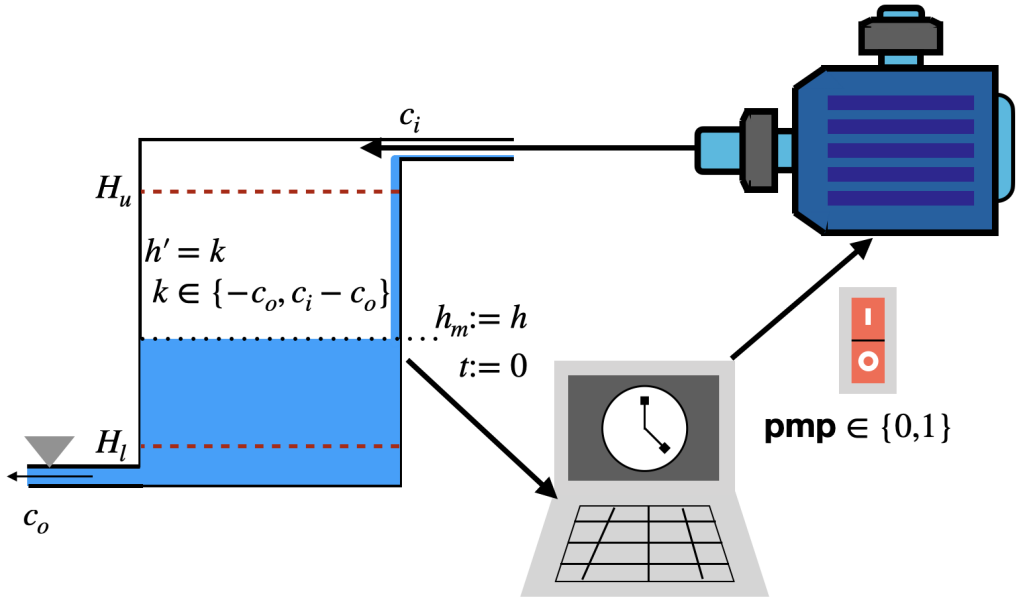
[4]University of Sheffield, UK

Verification of hybrid systems



$$\begin{aligned}
\text{dynamics} &= h' = k \\
\text{pre} &= H_l \leq h_m \leq H_u \\
\text{pos} &= H_l \leq h_m \leq H_u && \text{invariant} \\
\text{control} &= t := 0 \,;\, h_m := h \,;\, \texttt{if pmp} \ldots \\
\text{tank} &= (\text{control} \,;\, \text{dynamics})^* && \text{hybrid program} \\
\{\text{pre}\}\text{tank}\{\text{pos}\} & && \text{correctness spec}
\end{aligned}$$

Verification of hybrid systems in Isabelle/HOL




```
lemma tank_correct:
  "{t = 0 ∧ h = hₘ ∧ Hₗ ≤ h ∧ h ≤ Hᵤ}
    LOOP (
      (t, hₘ) ::= (0, h);
      IF ¬ pmp ∧ hₘ ≤ Hₗ + 1 THEN pmp ::= True ELSE
      IF   pmp ∧ hₘ ≥ Hᵤ - 1 THEN pmp ::= False ELSE skip;
      IF pmp THEN
        {h` = cᵢ - cₒ, t` = 1 | t ≤ (Hᵤ - hₘ)/(cᵢ - cₒ)}
      ELSE
        {h` = - cₒ, t` = 1 | t ≤ (Hₗ - hₘ)/(- cₒ)}
      )
      INV (0 ≤ t ∧ h = ((pmp * cᵢ) - cₒ)*t + hₘ ∧ Hₗ ≤ h ∧ h ≤ Hᵤ)
    {Hₗ ≤ h ∧ h ≤ Hᵤ}"
  using ci co by dProve
```

Previous work: Isabelle verification framework

1. Algebra of hybrid programs
   - Predicate transformers $|-] : S^{\mathbb{B}} \to S^{\mathbb{B}}$
   - $X ::= x := e \mid x' = f \,\&\, G \mid ?P \mid X \,;\, X \mid X + X \mid X^*$
   - **if** p **then** x **else** $y = p \,;\, x + (\bar{p}) \,;\, y$
2. Algebra semantics
   - State transformer $f : S \to \mathcal{P}\, S$
   - $(x := e)\, s = \{s[x \mapsto e\, s]\}$
   - $(x' = f \;\&\; G)\, s = \{X\, t \mid \dot{X}\, t = f(t, X\, t) \text{ and } X\, 0 = s \text{ and } G_{\downarrow}\}$
3. Program store model
4. Verification workflows
   - Certify unique solutions (flows) φ
   - Invariant reasoning I' inspired by differential dynamic logic ($\mathrm{d}\mathcal{L}$)
   - Direct reasoning about dynamics φ

How to turn this framework towards a formal method?

Contributions

1. More refined hybrid store model based on lenses
2. More user-friendly specification language (cleaner syntax)
3. More inference and calculational rules (derivatives, framing, $\mathrm{d}\mathcal{L}$-inspired ghost rule)
4. More effective tactics

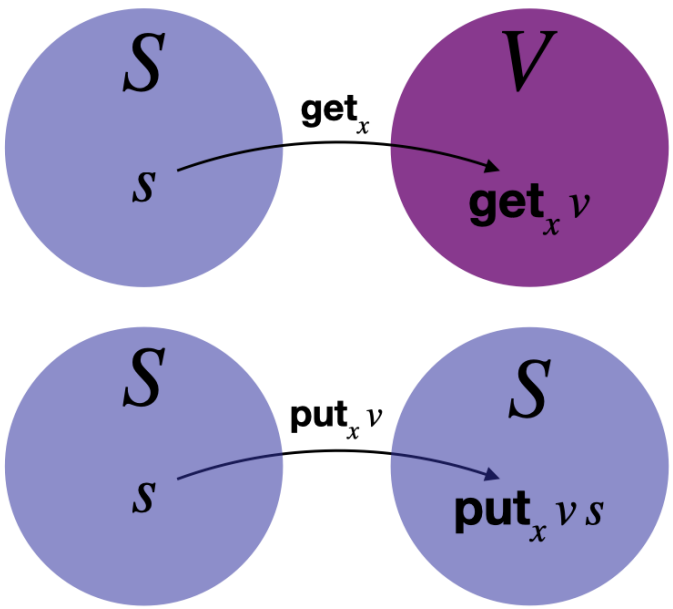


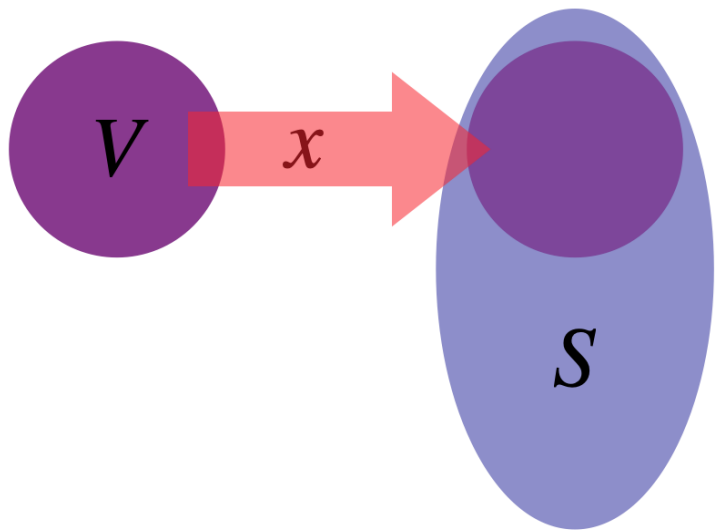
https://github.com/isabelle-utp/Hybrid-Verification

Hybrid store model

Lenses

- Lenses $x = (\mathcal{V}, \mathcal{S}, get_x, put_x)$ are variables denoted $x : \mathcal{V} \Longrightarrow \mathcal{S}$ where $get_x : \mathcal{S} \to \mathcal{V}$ access and $put_x : \mathcal{V} \to \mathcal{S} \to \mathcal{S}$ updates.
- Combinators: indep. $(x \bowtie y)$, unrestriction $(x \sharp e)$, preorder $(x \preceq y)$





Hybrid store model

Lenses

- Lenses $x = (\mathcal{V}, \mathcal{S}, get_x, put_x)$ are variables denoted $x : \mathcal{V} \Longrightarrow \mathcal{S}$ where $get_x : \mathcal{S} \to \mathcal{V}$ access and $put_x : \mathcal{V} \to \mathcal{S} \to \mathcal{S}$ updates.
- Combinators: indep. $(x \bowtie y)$, unrestriction $(x \sharp e)$, preorder $(x \preceq y)$

Syntax translations

$$(t)_e \rightleftharpoons [(t)^e]_e, \quad (x)^e \rightleftharpoons \begin{cases} \lambda s.\, get_x\ s & \text{if } x \text{ is a lens,} \\ \lambda s.\, x & \text{otherwise,} \end{cases} \quad (f\ t)^e \rightleftharpoons \lambda s.\, f\ ((t)^e\ s),$$

For program variables x and y and logical variable z:

$$((x+y)^2/z)_e \rightleftharpoons [\lambda s.\, (get_x\ s + get_y\ s)^2/z]_e$$

Hybrid store model

Lenses

- Lenses $x = (\mathcal{V}, \mathcal{S}, get_x, put_x)$ are variables denoted $x : \mathcal{V} \Longrightarrow \mathcal{S}$ where $get_x : \mathcal{S} \to \mathcal{V}$ access and $put_x : \mathcal{V} \to \mathcal{S} \to \mathcal{S}$ updates.
- Combinators: indep. $(x \bowtie y)$, unrestriction $(x \sharp e)$, preorder $(x \preceq y)$

User friendly specification language

- User input:

```
"(0 ≤ y ∧ v² ≤ 2 * g * (H - y))ₑ"
```

- Isabelle output:

```
"(0 ≤ $y ∧ ($v)² ≤ 2 * g * (H - $y))ₑ"
```

- Isabelle internal processing:

```
"[λs. 0 ≤ get↘y↙ s ∧ (get↘v↙ s)² ≤ 2 * g * (H - get↘y↙ s)]ₑ"
```

More inference rules via framing

Framing: project to a part of the state space

- Take any substitution on $\mathcal{S}$, e.g. $f : \mathcal{S} \to \mathcal{S}$.
- *Frame* it with λ where $\lambda : \mathcal{C} \Longrightarrow \mathcal{S}$ to $f_\lambda : \mathcal{S} \to \mathcal{C} \to \mathcal{C}$ s.t. $f_\lambda\ s = get_\lambda \circ f \circ (\lambda c.\ put_\lambda\ c\ s)$.
- If $\mathcal{C}$ represents the "continuous" part of $\mathcal{S}$, then
  - $f_\lambda\ s : \mathcal{C} \to \mathcal{C}$ is the restriction of f to $\mathcal{C}$
  - $f_\lambda\ s$ is a vector field
  - It is supplied with the full store s before restriction.

Framing rule as in separation logic

$$\frac{\alpha\ nmods\ \{x_1, \ldots, x_n\} \qquad - \{x_1, \ldots, x_n\} \sharp I \qquad \{P\}\ \alpha\ \{Q\}}{\{P \wedge I\}\ \alpha\ \{Q \wedge I\}}\ Frame$$

More inference rules via framing

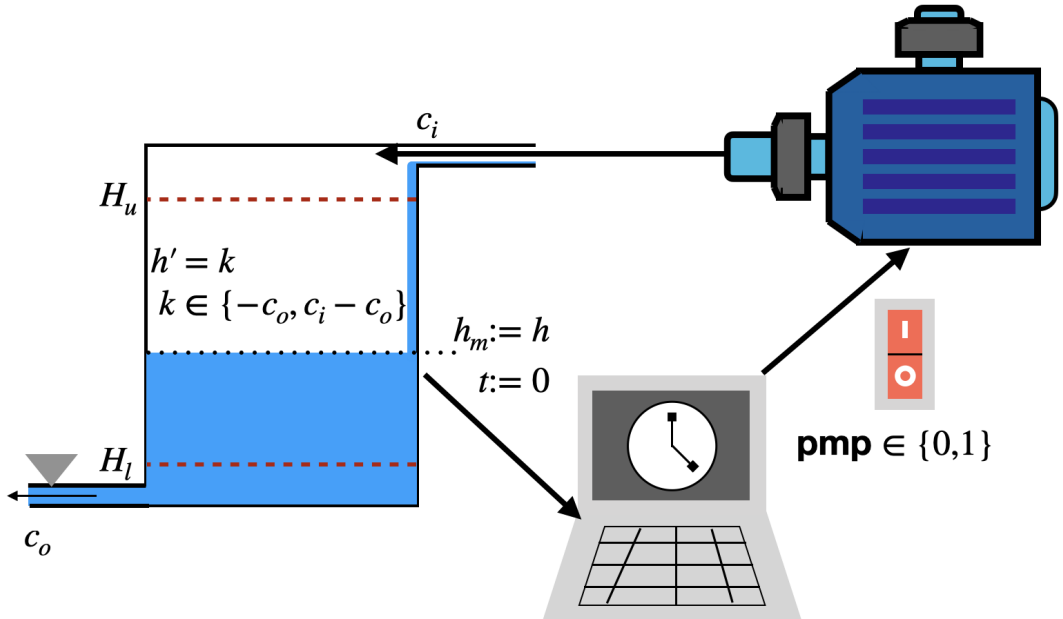
Equational certification of framed derivatives

$$
\begin{aligned}
\mathcal{D}^f_A(k) &= 0 && \text{if } A \sharp k,\\
\mathcal{D}^f_A(x) &= 0 && \text{if } x \bowtie A\\
\mathcal{D}^f_A(x) &= get_x \circ f && \text{if } x \preceq A \text{ and } get_x \text{ is bounded linear}\\
\mathcal{D}^f_A(e_1 + e_2) &= (\mathcal{D}^f_A(e_1)) + (\mathcal{D}^f_A(e_2)),\\
\mathcal{D}^f_A(e_1 \cdot e_2) &= (e_1 \cdot \mathcal{D}^f_A(e_2)) + (\mathcal{D}^f_A(e_1) \cdot e_2),\\
\mathcal{D}^f_A(e^n) &= n \cdot (\mathcal{D}^f_A(e)) \cdot e^{(n-1)},\\
\mathcal{D}^f_A(\ln(e)) &= (\mathcal{D}^f_A(e))/e && \text{if } e > 0.
\end{aligned}
$$

Ghost rule from $\mathsf{d}\mathcal{L}$

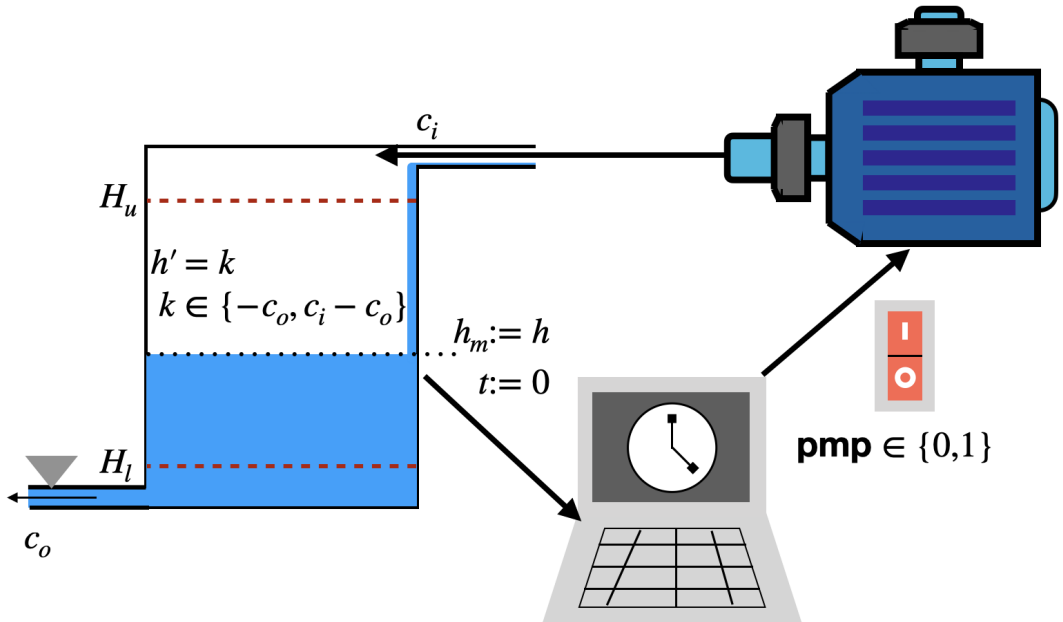
$$
\frac{y \bowtie \mathbf{x} \quad y \sharp (G, e) \quad \{I\}\ \{(\mathbf{x}, y)' = (e, k \cdot y) \mid G\}\ \{I\}}{\{\exists v.\, I[v/y]\}\ \{\mathbf{x}' = e \mid G\}\ \{\exists v.\, I[v/y]\}}\, dGhost
$$

Formalisation of the water tank



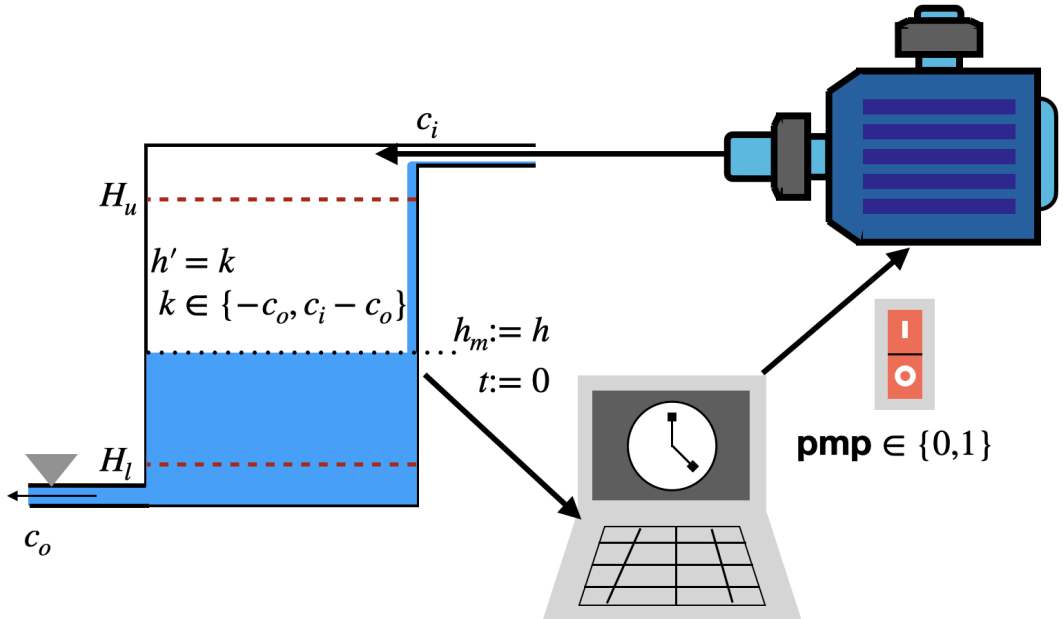

```
dataspace water_tank =
  constants Hl::real Hu::real co::real ci::real
  assumes co: "0 < co" and ci: "co < ci"
  variables pmp::"bool" h::"real" hm::"real" t::"real"
```

More automation to first workflow: certify solutions



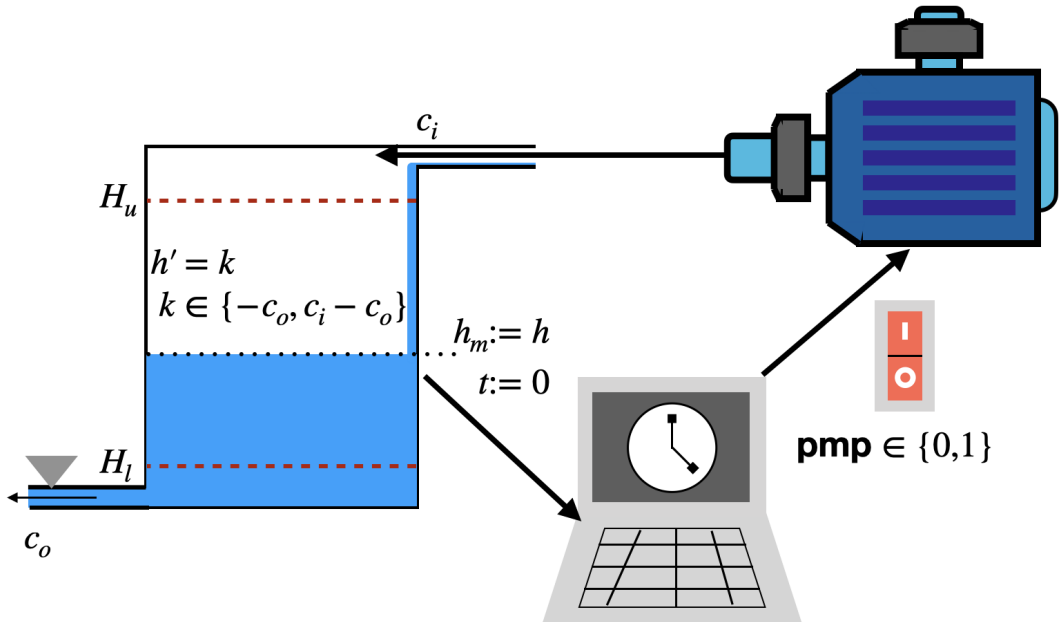

```
abbreviation tank_vec_field :: "real ⇒ 'st ⇒ 'st" ("f")
  where "tank_vec_field k ≡ [h ↝ k, t ↝ 1]"

abbreviation tank_flow :: "real ⇒ real ⇒ 'st ⇒ 'st" ("φ")
  where "tank_flow k τ ≡ [h ↝ k * τ + h, t ↝ τ + t]"

lemma lflow_tank : "local_flow_on (f k) (h+ʟt) UNIV UNIV (φ k)"
  by (local_flow_auto)
```

More automation to second workflow: invariants




```
lemma "{0 ≤ t ∧ h = (cᵢ - cₒ)*t + hₘ ∧ Hₗ ≤ h ∧ h ≤ Hᵤ}
         {h` = cᵢ - cₒ, t` = 1 | t ≤ (Hᵤ - hₘ)/(cᵢ - cₒ)}
       {0 ≤ t ∧ h = (cᵢ - cₒ)*t + hₘ ∧ Hₗ ≤ h ∧ h ≤ Hᵤ}"
  using ci by dInduct_mega
```

Verification of the water tank




```
abbreviation "ctrl ≡ (t, hₘ) ::= (0, h);
     IF ¬ pmp ∧ hₘ ≤ Hₗ + 1 THEN pmp ::= True ELSE
     IF pmp ∧ hₘ ≥ Hᵤ - 1 THEN pmp ::= False ELSE skip"

abbreviation "dyn ≡ IF pmp THEN {h` = cᵢ - cₒ, t` = 1 | t ≤ (Hᵤ - hₘ)/(cᵢ - cₒ)}
     ELSE {h` = - cₒ, t` = 1 | t ≤ (Hₗ - hₘ)/(- cₒ)}"

lemma tank_correct:
  "{Hₗ ≤ h ∧ h ≤ Hᵤ}
    LOOP ctrl ; dyn INV (Hₗ ≤ h ∧ h ≤ Hᵤ)
   {Hₗ ≤ h ∧ h ≤ Hᵤ}"
  using tank_arith[OF _ co ci]
  by (hoare_wp_auto local_flow: lflow_tank)
```

Conclusion

We have transformed our hybrid systems verification framework towards a formal method by adding:

- simpler syntax through our user friendly specification language,
- better models via our new modular hybrid store representation based on lenses,
- faster proofs that emerge from our new rules of inference and tactics

https://github.com/isabelle-utp/Hybrid-Verification

jonathan.munive@di.ku.dk

Conclusion

Future work

- Make this formal method have industrial strength:
  - Replace expression language with Modelica syntax
  - Integrate with a CAS that supplies solutions and invariants, leaving the certification work to Isabelle
  - Code generation of verified controller implementations
  - Explore the absorption of other verification methods into this framework



https://github.com/isabelle-utp/Hybrid-Verification

jonathan.munive@di.ku.dk