Updates on Hybrid ABS

Eduard Kamburjan
University of Oslo

ABS Workshop’21, 26.08.21
Hybrid Active Objects

Hybrid ABS (HABS) is a conservative extension of Timed ABS with continuous dynamics for state changes during time advance.

This talk: recent results, on-going work and outlook, mainly verification.

Post-Regions

Generalizing method post-conditions to hybrid objects.

- Analyze local structure of object to derive how long continuous dynamics have to stay safe upon method termination (HSCC'21\(^1\))
- Analyze global structure for more loosely coupled systems (on-going)

\(^1\)https://www.youtube.com/watch?v=KTPs9B9jobo
Example: Water Tank

```java
class CSingleTank(Real inVal) {
    physical {
        Real lvl = inVal : lvl' = flow;
        Real flow = -0.5 : flow' = 0;
    }
    { this!up(); this!low(); }
    Unit low() {
        await diff lvl <= 3 & flow <= 0;
        flow = 0.5; this!low();
    }
    Unit up() {
        await diff lvl >= 10 & flow >= 0;
        flow = -0.5; this!up();
    }
}
```

Is $3 \leq lvl \leq 10$ an invariant (if $3 \leq \text{inVal} \leq 10$)?
Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

\[ \phi ::= \forall x. \phi \mid \phi \lor \phi \mid \neg \phi \mid \ldots \mid [\alpha] \phi \]

\[ \alpha ::= ?\phi \mid v := t \mid v := * \mid \{v' = f(v) \& \phi\} \mid \ldots \]

Example

Set a variable to 0, let it raise with slope 1 while it is below 5 and discard all runs where it is above 5.

\[ [x := 0; \{x' = 1 \& x \leq 5\}; ?x \geq 5] \]

This formula is valid.
Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

\[ \phi ::= \forall x. \phi \mid \phi \lor \phi \mid \neg \phi \mid \ldots \mid [\alpha] \phi \]
\[ \alpha ::= ?\phi \mid v := t \mid v := \ast \mid \{v' = f(v) \& \phi\} \mid \ldots \]

Example

Set a variable to 0, let it raise with slope 1 while it is below 5 and discard all runs where it is above 5.

\[ \lfloor x := 0; \{x' = 1 \& x \leq 5\}; ?x \geq 5 \rfloor x \models 5 \]

This formula is valid.
Internal Post-Regions
Preliminaries

- We assume that every method starts with an `await diff` statement. If it does not, add `await diff true`.
- The leading guard of a method $m$ is denoted $trig_m$.
- Only `Real` variables are manipulated.
- Weak negation is denoted $\neg e_1 \geq e_2 \iff e_1 \leq e_2$

Safety

An object is safe w.r.t. some formula $\phi$, if its state is a model for $\phi$ (a) whenever a method starts and (b) whenever time advances.

For this talk, all `await` are leading and no `get` or `duration` occur.
### Proof Obligations with Dynamic Logic

In discrete systems, an object invariant $I$ can be checked *modularly* with dynamic logic by showing that every method preserves $I$.

\[ I \rightarrow [s]I \]

This uses that the state does not change in inactive objects.
In discrete systems, an object invariant $I$ can be checked *modularly* with dynamic logic by showing that every method preserves $I$.

\[ I \rightarrow [s]I \]

Proof Obligation for Java

This uses that the state does not change in inactive objects.
Proof Obligations with Dynamic Logic

In discrete systems, an object invariant $I$ can be checked *modularly* with dynamic logic by showing that every method preserves $I$.

$$I \rightarrow [s]/I$$  

**Proof Obligation for Java**

This uses that the state does not change in inactive objects.
Let $C$ be a class with dynamics $\text{ode}$. Each object of $C$ is safe w.r.t. $\text{inv}$ and precondition $\text{pre}$ if for every method the following holds:

$$\text{inv} \rightarrow [?\text{trig}_m; \text{trans}(s_m)] (\text{inv} \land [\text{ode}&\text{true}]\text{inv})$$

And additionally for the constructor:

$$\text{pre} \rightarrow [\text{trans}(s_{\text{init}})] (\text{inv} \land [\text{ode}&\text{true}]\text{inv})$$
Theorem

Let $c$ be a class with dynamics $\text{ode}$. For each method $m$ let $CM_m$ be the set of methods which are guaranteed to called in every execution. Each object of $c$ is safe w.r.t. $\text{inv}$ if for every method $m$ the following holds:

$$\text{inv} \rightarrow ??trig_m; \text{trans}(s_m) \left( \text{inv} \land \left[ \text{ode} \land \bigwedge_{m' \in CM_m} \sim trig_{m'} \right] \text{inv} \right)$$

And analogously for the constructor.
### Definition

A controller is a method of the form

```
1 Unit m() { await diff g; s; this!m(); }
```

which (a) is called from the constructor and (b) contains no communication statements within `s`. 
Structurally Controlled Regions

**Definition**

A controller is a method of the form

```plaintext
1 Unit m(){ await diff g; s; this!m(); }
```

which (a) is called from the constructor and (b) contains no communication statements within `s`.

**Theorem**

Let \( C \) be a class with dynamics \( \text{ode} \). Let \( \text{Ctrl} \) be the set of controllers and \( \text{CM}_n \) be as before. Each object of \( C \) is safe w.r.t. \( \text{inv} \) if for every method \( m \) the following holds:

\[
\text{inv} \rightarrow [?\text{trig}_m; \text{trans}(s_m)] \left( \text{inv} \land \left[ \text{ode} \& \bigwedge_{m' \in \text{CM}_m \cup \text{Ctrl}} \neg \text{trig}_{m'} \right] \text{inv} \right)
\]

And analogously for the constructor.
Structurally Controlled Regions

class StructureTank(){
physical{Real lvl = 5 : lvl’ = flow; ...}
{ this!up(); this!low(); }
Unit low(){await diff lvl <= 3 & φ₁; flow = 0.5; this!low();}
Unit up(){await diff lvl >= 10 & φ₂; flow = -0.5; this!up();}
}

inv → [lvl <= 3 ∧ φ₁; flow := 0.5]

\[
\left( inv ∧ [lvl’ = flow ∧ (lvl >= 3 ∨ \sim φ₁) ∧ (lvl <= 10 ∨ \sim φ₂)] inv \right)
\]

Modularity

- Changing a controller method requires to re-verify all methods.
- Changing a method requires reverification of its (guaranteed) callers.
- Otherwise, only the changed method must be reverified.
External Post-Regions
So far, locally and structurally controlled regions are computed _internally_. Controller and controllee are tightly coupled within one object.

```java
1 class Tank(Real inVal) implements Tank {
2     physical { ... }
3     /* timed_reQUIRES 1 */
4     Unit check(){
5         if(level <= 3.5) drain = 0.5;
6         if(level >= 9.5) drain = -0.5;
7     }
8 }
9 class FlowCtrl(){
10     Unit ctrl(Tank t) {
11         await duration(1,1);
12         t!check();
13         this.ctrl(t);
14     }
15 }
```
External Control

**Typing Control**

Use behavioral types to keep track of

1. Which object is controlling an exposed method (≈ ownership)
2. Who often does this object call the method (≈ deadline)

**Proof obligations do not change, but are justified differently.**

**Loose Coupling**

This way, we can type check loose coupling:

1. Controller may change after some time
2. Multiple controllers can control one HAO
IoT systems in HABS

- The behavioral type system is for *Timed ABS*
- We can reuse all analyses for ABS for cloud based CPS
- This is exactly the structure of the IoT
Conclusion
Modelica

*Modelica* is an OO language with differential equations as its semantics. Describe equations for physical behavior by using `physical` as an interface.

```modelica
model Growth "This is a modelica style comment"
  output Real value; input Real lm;
  equation
  der(value) = 1/2*(lm-value);
end Growth;

class C {
  physical Real v = 5; ....
  physical{
    Growth g(lm=lm, value=v); is(g.value, this.v); is(g.lm, this.l);
    //  der(v) = 1/2*(l-v)  //alternative
  }
}
```
## Conclusion

### Summary

- Generalizing pre-/post-condition reasoning to hybrid systems
- Implemented for Hybrid ABS with KeYmaera X as backend
- On-going: verifying loosely coupled systems

### Future Work

- Simulation and modeling with Modelica/FMUs
- Verification of global properties of HABS programs
- Resource-aware hybrid systems
- Verification of hybrid objects with rich data types
## Summary

- Generalizing pre-/post-condition reasoning to hybrid systems
- Implemented for Hybrid ABS with KeYmaera X as backend
- On-going: verifying loosely coupled systems

## Future Work

- Simulation and modeling with Modelica/FMUs
- Verification of global properties of HABS programs
- Resource-aware hybrid systems
- Verification of hybrid objects with rich data types

Thank you for your attention