Updates on Hybrid ABS

Eduard Kamburjan
University of Oslo

ABS Workshop'21, 26.08.21

Hybrid ABS

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.

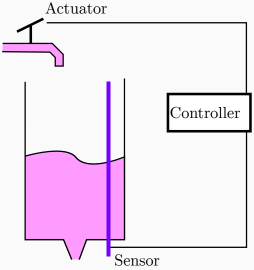
Post-Regions

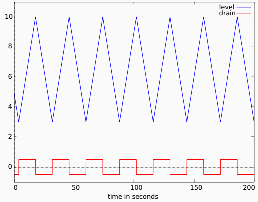
Generalizing method post-conditions to hybrid objects.

- Analyze local structure of object to derive how long continous 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






```
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` ≤ 10 an invariant (if $3 \leq$ `inVal` ≤ 10)?

Differential Dynamic Logic

Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

$$\phi ::= \forall x.\ \phi \mid \phi \vee \phi \mid \neg\phi \mid \ \dots \ \mid [\alpha]\phi$$
$$\alpha ::= ?\phi \mid \mathtt{v} := t \mid \mathtt{v} := * \mid \{\mathtt{v}' = f(\mathtt{v}) \& \phi\} \mid \ \dots$$

Differential Dynamic Logic

Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

$$\phi ::= \forall x.\ \phi \mid \phi \vee \phi \mid \neg\phi \mid \ \dots \ \mid [\alpha]\phi$$
$$\alpha ::= ?\phi \mid \mathtt{v} := t \mid \mathtt{v} := * \mid \{\mathtt{v}' = f(\mathtt{v})\&\phi\} \mid \ \dots$$

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.

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

This formula is valid.

Internal Post-Regions

Setup

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 $trig_{\mathtt{m}}$.
- Only `Real` variables are manipulated.
- Weak negation is denoted $\tilde{\neg} e_1 \geq e_2 \iff e_1 \leq e_2$

Safety

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

For this talk, all **await** are leading and no **get** or **duration** occur.

Object Invariants

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.

$$I \rightarrow [\mathrm{s}] I \qquad \text{Proof Obligation for Java}$$

This uses that the state does not change in inactive objects.

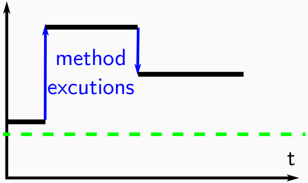
Object Invariants

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.

$$I \rightarrow [\mathrm{s}] I \qquad \text{Proof Obligation for Java}$$

This uses that the state does not change in inactive objects.



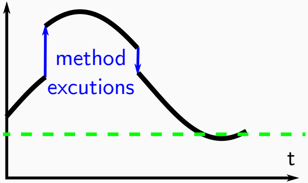
Object Invariants

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.

$$I \rightarrow [\mathrm{s}] I \qquad \text{Proof Obligation for Java}$$

This uses that the state does not change in inactive objects.



Basic Regions

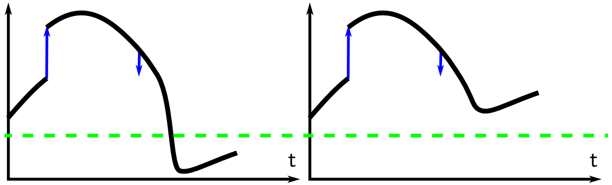
Theorem

Let $\texttt{C}$ *be a class with dynamics* $\texttt{ode}$*. Each object of* $\texttt{C}$ *is safe w.r.t.* inv *and precondition* pre *if for every method the following holds:*

$$\mathsf{inv} \rightarrow \left[?\mathit{trig}_{\mathtt{m}};\mathsf{trans}(\mathtt{s}_{\mathtt{m}})\right]\left(\mathsf{inv} \wedge [\mathtt{ode}\&\mathsf{true}]\mathsf{inv}\right)$$

And additionally for the constructor:

$$\mathsf{pre} \rightarrow \left[\mathsf{trans}(\mathtt{s}_{\mathtt{init}})\right]\left(\mathsf{inv} \wedge [\mathtt{ode}\&\mathsf{true}]\mathsf{inv}\right)$$



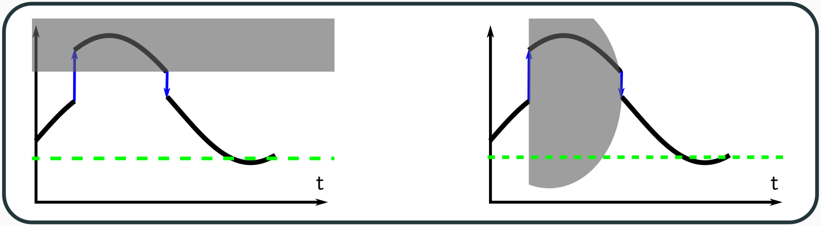
Locally Controlled Regions

Theorem

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

$$\mathsf{inv} \rightarrow [?\mathit{trig}_{\mathtt{m}}; \mathsf{trans}(\mathsf{s}_{\mathtt{m}})] \left(\mathsf{inv} \wedge \left[\mathtt{ode}\& \bigwedge_{\mathtt{m}' \in \mathsf{CM}_{\mathtt{m}}} \tilde{\neg}\mathit{trig}_{\mathtt{m}'}\right] \mathsf{inv}\right)$$

And analogously for the constructor.



Structurally Controlled Regions

Definition

A controller is a method of the form

```
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

```
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* `ode`. *Let* Ctrl *be the set of controllers and* $\mathrm{CM}_{\mathtt{n}}$ *be as before. Each object of* `C` *is safe w.r.t.* inv *if for every method* `m` *the following holds:*

$$\mathrm{inv} \rightarrow [?\mathit{trig}_{\mathtt{m}}; \mathrm{trans}(\mathtt{s}_{\mathtt{m}})] \left(\mathrm{inv} \wedge \left[\mathrm{ode} \& \bigwedge_{\mathtt{m}' \in \mathrm{CM}_{\mathtt{m}} \cup \mathrm{Ctrl}} \tilde{\neg} \mathit{trig}_{\mathtt{m}'} \right] \mathrm{inv} \right)$$

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();}
}
```

$$\text{inv} \rightarrow [?\texttt{lvl <= 3} \land \phi_1; \texttt{flow := 0.5}]$$

$$\Big(\text{inv} \land \big[\texttt{lvl' = flow}\&(\texttt{lvl >= 3} \lor \tilde{\neg}\phi_1) \land (\texttt{lvl <= 10} \lor \tilde{\neg}\phi_2)\big]\text{inv}\Big)$$

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

External Post-Regions

So far, locally and structurally controlled regions are computed *internally*. Controller and controllee are tighly coupled within one object.

```
class Tank(Real inVal) implements Tank {
  physical {  ...  }
  /* timed_requires 1 */
  Unit check(){
    if(level <= 3.5) drain =  0.5;
    if(level >= 9.5) drain = -0.5;
  }
}
class FlowCtrl(){
  Unit ctrl(Tank t) {
    await duration(1,1);
    t!check();
    this.ctrl(t);
  }
}
```

External Control

Typing Control

Use behavioral types to keep track of

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

Proof obligations do not change, but are justified differently.

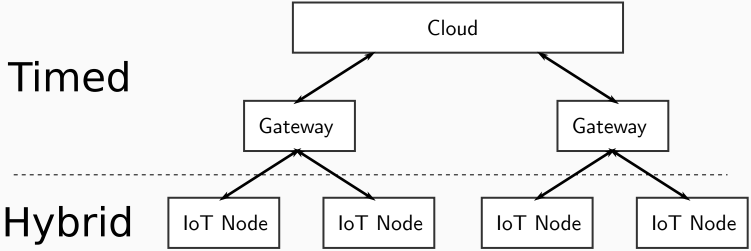
Loose Coupling

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

Modeling with Modelica

Modelica

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

```
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

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

Thank you for your attention