KeY-Style Verification for (Hybrid) ABS

Advances after KeY-ABS

Eduard Kamburjan

University of Oslo

KeYnote Series, 23.04.2021

Introduction

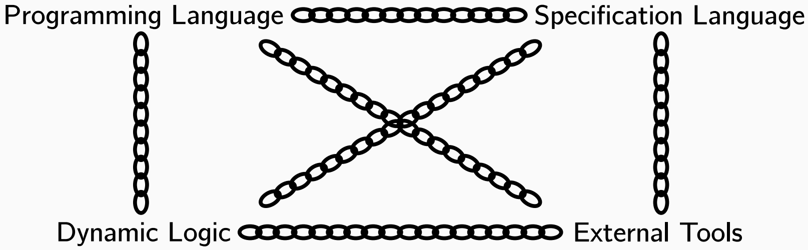
Programming Language

Specification Language

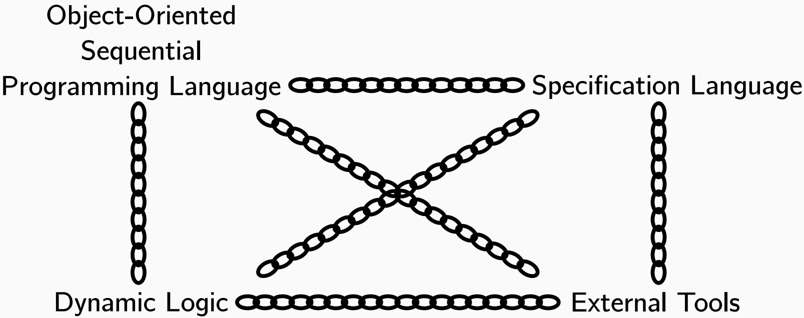
Dynamic Logic

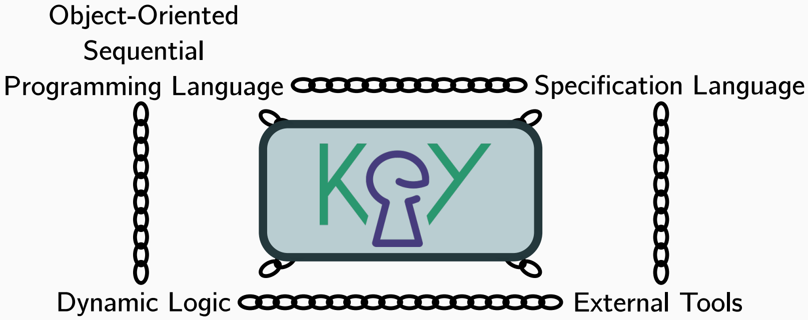
External Tools

Introduction

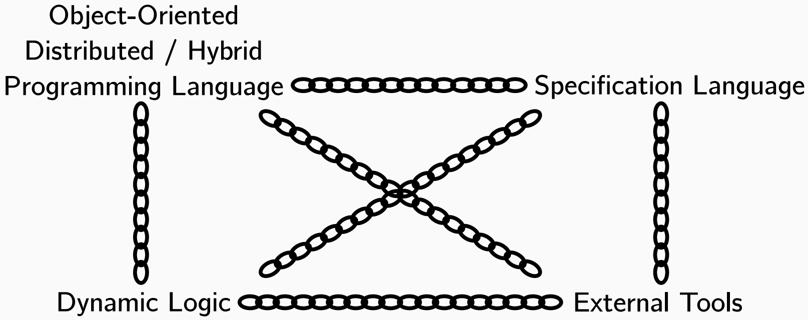


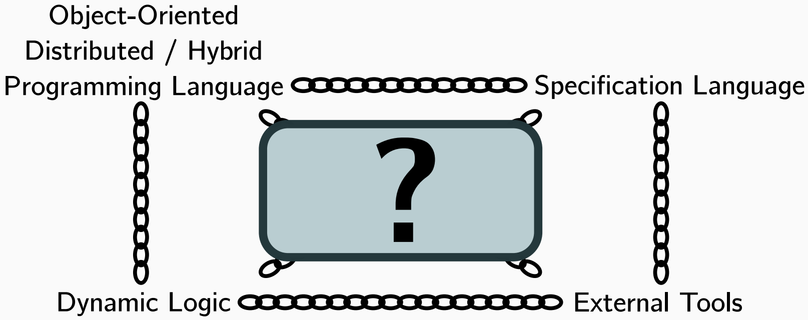
Introduction



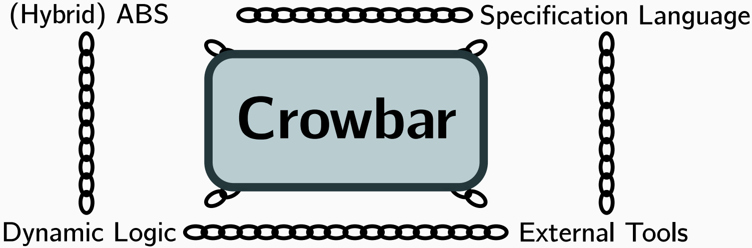

Object-Oriented
Sequential
Programming Language
Specification Language
KeY
Dynamic Logic
External Tools

Introduction




Object-Oriented
Distributed / Hybrid
Programming Language
Specification Language
?
Dynamic Logic
External Tools

Introduction



Publication: *Kamburjan, Scaletta, Rollshausen*, Crowbar: Behavioral Symbolic Execution for Deductive Verification of Active Objects, **abs/2102.10127**

Available at **https://github.com/Edkamb/crowbar-tool**

Preliminaries

Active Objects: Between Actors and Shared Memory

```
class A(Int x, Bool locked, B other) implements A{
  Unit m(){
    Fut<Int> f = other!met(this.x);
    this.locked = True;
    await f?;
    this.locked = False;
    this.x = this.x + f.get;
  }
  Unit setX(Int x){ await !this.locked; this.x = x; }
}
```

- Object-private fields, interleavings only at **await**
- Object-Oriented Actors + Futures + Cooperative Scheduling

ABS

ABS is specifically designed to combine verification, analysis, execution and natural modeling (for programmers).

- Functional Language

```
data IntList = Cons(Int, IntList) | Nil;
def Int length(IntList l) = case l { ... };
```

- Symbolic time

```
println(now()); //0
duration(1,1);
println(now()); //1
```

- ...

KeY-ABS

KeY-ABS

Developed 2015, keeps track of communication events during symbolic execution in a *history*. Trace properties are verified as object invariants over the history.

- FO logic over histories is not a good specification language
- Requires full symbolic execution to detect errors in the beginning of the method
- Implementation still retains Java-bindings:
  - Hard to connect with external tools
  - Hard to prototype new specifications
  - Hard to include functional sublanguage

Behavioral Program Logic

Trace Properties

In a concurrent setting, (a) most properties of interest are trace-based and (b) no general scheme is established.

The Many Faces of the Box Modality for Traces

- $[\mathtt{s}]\forall i \in \mathbb{N}.\, \mathtt{history}[i] \doteq\!\!\!\!/\;\, \mathsf{invEv}$ ABSDL [Din et al., SEFM'12]

Behavioral Program Logic

Trace Properties

In a concurrent setting, (a) most properties of interest are trace-based and (b) no general scheme is established.

The Many Faces of the Box Modality for Traces

- $[\mathtt{s}]\forall i \in \mathbb{N}.\, \mathtt{history}[i] \dot{\neq} \mathsf{invEv}$ ABSDL [Din et al., SEFM'12]
- $[\mathtt{s}]\Box\mathbf{this}.\mathtt{f>0}$ DTL [Beckert and Bruns, CADE'13]

Behavioral Program Logic

Trace Properties

In a concurrent setting, (a) most properties of interest are trace-based and (b) no general scheme is established.

The Many Faces of the Box Modality for Traces

- $[\mathtt{s}]\forall i \in \mathbb{N}.\,\mathtt{history}[i] \mathrel{\dot{\neq}} \mathsf{invEv}$ — ABSDL [Din et al., SEFM'12]
- $[\mathtt{s}]\Box\mathbf{this}.\mathtt{f}{>}0$ — DTL [Beckert and Bruns, CADE'13]
- $[\mathtt{s}]\mathbf{finite} * * \lceil \mathbf{this}.\mathtt{f}{>}0 \rceil * * \mathbf{finite}$ — DLCT [Din et al., TABLEAUX'15&'17]

Behavioral Program Logic

Trace Properties

In a concurrent setting, (a) most properties of interest are trace-based and (b) no general scheme is established.

The Many Faces of the Box Modality for Traces

- $[\mathtt{s}]\forall i \in \mathbb{N}.\,\mathtt{history}[i] \doteq\!\!\!\!/\ \mathsf{invEv}$ — ABSDL [Din et al., SEFM'12]
- $[\mathtt{s}]\square\,\mathbf{this}.\mathtt{f>0}$ — DTL [Beckert and Bruns, CADE'13]
- $[\mathtt{s}]\mathbf{finite} * * \lceil \mathbf{this}.\mathtt{f>0} \rceil * * \mathbf{finite}$ — DLCT [Din et al., TABLEAUX'15&'17]
- $\mathtt{s} \vdash \mathtt{X!m}\langle \mathbf{this}.\mathtt{f>0} \rangle.\mathtt{Y!n}.\mathbf{end}$ — Session Types for AO [Kamburjan and Chen, iFM'18]

Behavioral Program Logic

Trace Properties

In a concurrent setting, (a) most properties of interest are trace-based and (b) no general scheme is established.

The Many Faces of the Box Modality for Traces

- $[\mathtt{s}]\forall i \in \mathbb{N}.\,\mathtt{history}[i] \not\doteq \mathsf{invEv}$ — ABSDL [Din et al., SEFM'12]
- $[\mathtt{s}]\Box\mathbf{this}.\mathtt{f>0}$ — DTL [Beckert and Bruns, CADE'13]
- $[\mathtt{s}]\mathbf{finite} **\lceil \mathbf{this}.\mathtt{f>0}\rceil ** \mathbf{finite}$ — DLCT [Din et al., TABLEAUX'15&'17]
- $\mathtt{s} \vdash \mathtt{X!m}\langle \mathbf{this}.\mathtt{f>0}\rangle.\mathtt{Y!n}.\mathbf{end}$ — Session Types for AO [Kamburjan and Chen, iFM'18]
- And more....

Behavioral Modalities

[ ]

$$[\underbrace{\texttt{s}}_{\textit{statement}} \qquad\qquad\qquad]$$

$$\left[\ \underbrace{\mathtt{s}}_{\textit{statement}} \quad \underbrace{\tau}_{\textit{type}}\ \right]$$

$$\left[\ \underbrace{\mathtt{s}}_{\textit{statement}}\ \overset{\overbrace{\ \alpha\ }^{\textit{semantics}}}{\Vdash}\ \underbrace{\tau}_{\textit{type}}\ \right]$$

BPL

Example

Trace-specifications are too complex for simple post-conditions.

- ABSDL has object-invariant *implicit*
- BPL makes structure explicit

BPL

Example

Trace-specifications are too complex for simple post-conditions.

- ABSDL has object-invariant *implicit*
- BPL makes structure explicit

$$\textbf{(BPL)}\ \frac{\Gamma \;\Rightarrow\; \{U\}\{\mathtt{x\ :=\ v}\}[\mathtt{s} \Vdash (\phi, \mathit{inv})], \Delta}{\Gamma \;\Rightarrow\; \{U\}[\mathtt{x\ =\ v;\ s} \Vdash (\phi, \mathit{inv})], \Delta}$$

BPL

Example

Trace-specifications are too complex for simple post-conditions.

- ABSDL has object-invariant *implicit*
- BPL makes structure explicit

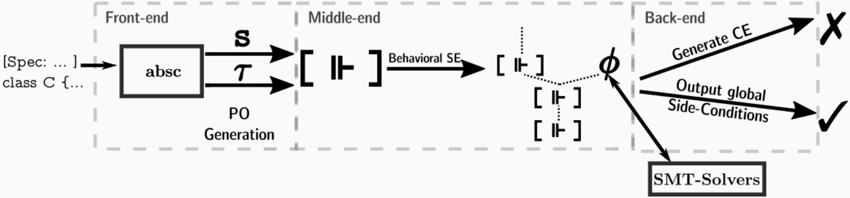
$$\textbf{(BPL)}\ \frac{\Gamma \ \Rightarrow\ \{U\}\{\mathtt{x\ :=\ v}\}[\mathtt{s} \Vdash (\phi, \mathit{inv})], \Delta}{\Gamma \ \Rightarrow\ \{U\}[\mathtt{x\ =\ v;\ s} \Vdash (\phi, \mathit{inv})], \Delta}$$

$$\textbf{(BPL)}\ \frac{\Gamma \ \Rightarrow\ \{U\}\mathit{inv}, \Delta \qquad \Gamma, \{U_{\mathcal{A}}\}\mathit{inv} \ \Rightarrow\ \{U_{\mathcal{A}}\}[\mathtt{s} \Vdash (\phi, \mathit{inv})], \Delta}{\Gamma \ \Rightarrow\ \{U\}[\textbf{await}\ \mathtt{e?;\ s} \Vdash (\phi, \mathit{inv})], \Delta}$$

$$\textbf{(BPL)}\ \frac{\ldots \qquad \Gamma\{U_{\mathcal{A}}\}I \ \Rightarrow\ \{U_{\mathcal{A}}\}[\mathtt{s} \Vdash (I, \mathit{inv})], \Delta}{\Gamma \ \Rightarrow\ \{U\}[\textbf{while}\mathtt{(e)\{s\}s'} \Vdash (\phi, \mathit{inv})], \Delta}$$

Crowbar

Structure



Behavioral Symbolic Execution

`Crowbar` is a symbolic execution engine to prototype behavioral symbolic execution: SE influenced by its context.

Aims

- Investigate how SE can cooperate with rest of static toolchain
- Quicker development cycles than KeY/Java

Frontend: Specification

Supported Specification Appraoches

- Cooperative method contracts (with \old and \last)
- Object invariants
- Session Types

Frontend: Specification

Supported Specification Appraoches

- Cooperative method contracts (with \old and \last)
- Object invariants
- Session Types

- Only user-input is a complete ABS program to integrate with the parser and type system.
- Specifications are annotated directly in the program.

```
...
[Spec: LoopInv(i>=0)]
while(i > 0) i = i-1;
...
```

Nullability Guides

Nullability Types

Most null-pointer exceptions can be handled by the type system.

ABS has a lightweight analysis to mark expression as non-null.

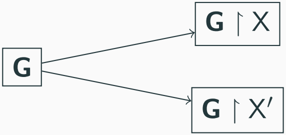
```
Unit m([NonNull] C o, C o2){
  Int i = o.m(); //safe
  Int j = o2.m();
  Int k = o2.m(); //safe
  return i + j + k;
}
```

- Crowbar keeps this information in the AST
- Safe accesses do not cause branching

Top-Down Specification with Session Types

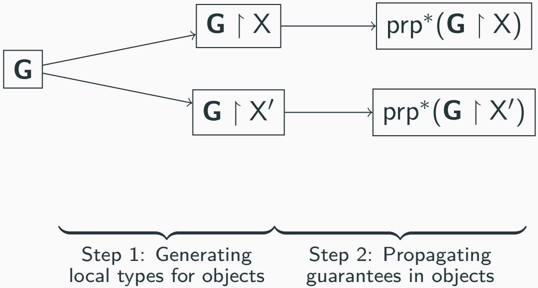
G

Top-Down Specification with Session Types

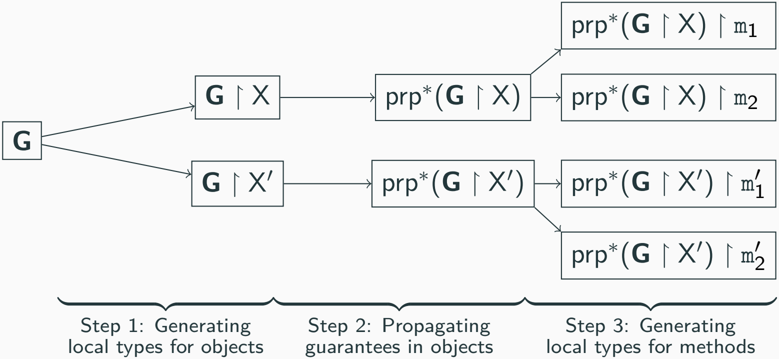


Step 1: Generating
local types for objects

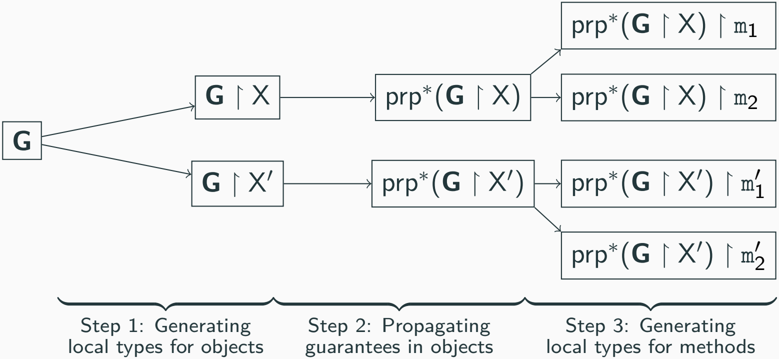
Top-Down Specification with Session Types



Top-Down Specification with Session Types



Top-Down Specification with Session Types



- Propgation is outside `Crowbar`
- Each class generates a static node for projection

Session Types

```
[Spec:Role("server", this.s)][Spec:Role("db", this.d)]
[Spec:ObjInv(...)]
class C(Server s, Client c, Database d) {
[Spec:Local("db!reset().(server!m(a > 3))*.Put()")]
Unit sideconditionInLoop() {
  Fut<Int> sth = this.d!reset();
  Int a = 10;
  [Spec: WhileInv(this.s != null)]
  while(a > 5)  sth = this.s!m(a--);
  }
}
```

$$\textbf{(met-V)}\ \frac{\Gamma \Rightarrow \{U\}(\mathtt{X} \doteq \textbf{this}.\mathtt{f} \wedge \phi), \Delta \qquad \Gamma \Rightarrow \{U\}\{\mathtt{v} := f\}[\mathtt{s} \overset{\text{met}}{\Vdash} \mathtt{L}], \Delta}{\Gamma \Rightarrow \{U\}[\mathtt{v} = \textbf{this}.\mathtt{f!m();}\ \mathtt{s} \overset{\text{met}}{\Vdash} \mathtt{X!m}\langle\phi\rangle.\mathtt{L}], \Delta}$$

Functional Layer

Functions and Data in ABS

ABS has a functional sublanguage for ADTs.

Each definition is translated into an assignment with contracts.

```
[Spec: Requires(n >= 0)][Spec: Ensures(result >= 0)]
def Int fac(Int n) = if(n<=1) then 1 else n*fac(n-1);
```

```
[Spec: Requires(n >= 0)][Spec: Ensures(result >= 0)]
Int fac(Int n){
    return if(n<=1) then 1 else n*this.fac(n-1);
}
```

Functional Layer

Functions and Data in ABS

ABS has a functional sublanguage for ADTs.

Each definition is translated into an assignment with contracts.

```
[Spec: Requires(n >= 0)][Spec: Ensures(result >= 0)]
def Int fac(Int n) = if(n<=1) then 1 else n*fac(n-1);
```

$$\big(\forall\ \mathtt{Int}\ x.\ x \geq 0 \rightarrow \mathtt{fac}(x) \geq 0\big) \land \mathtt{n} \geq 0$$

$$\rightarrow \big[\mathtt{result\ =\ \textbf{if}(n\ <=\ 1)\ \textbf{then}\ 1\ \textbf{else}\ n*fac(n-1);} \Vdash^{\alpha_{\text{pst}}} \mathtt{result} \geq 0\big]$$

- ABS does not support first-order function passing
- ADTs are translated into SMT-LIB datatypes

Middleend: BPL

Behavioral Symbolic Execution

Crowbar implements symbolic execution with *guides*: additional inputs that guide execution and shape the symbolic execution tree.

Rules

- Rules Kotlin classes implementing

```
abstract class Rule( val conclusion : Modality ) {
abstract fun transform(cond :MatchCondition,
                       input:SymbolicState): List<SymbolicTree>
}
```

- Matching is implemented directly on the AST using reflection: Schemavariables are any instances implements `AbstractVar`

Counterexample Generation

User Feedback

While non-interactive, `Crowbar` must still give comprehensive feedback to user and developer. We generate a *program* from failing proof branch and annotate relation to specification.

DEMO

Experiences with `Crowbar`

Experiences with Crowbar

C2ABS [Wasser et al., SCP'21]

Translates ACSL-specified C-Code into ABS.

Underspecified semantics becomes non-deterministic concurrency.

Example

Following code returns 1 (clang) or 2 (gcc)

```
int x;
int id_set_x(int val){
    x=1; return val;
}
int main(void){
    x=0; return x + id_set_x(1);
}
```

Experiences with `Crowbar`

Case Study

Highly underspecified variant of `fib`(n) which returns a number between 1 and the nth fibonnaci number based on evaluation order.

- 4 C functions, each with post-conditions, 1 Strong invariant

Translation generates 260 lines of ABS code

- 5 classes (with invariants and creation conditions)
- 5 interfaces with 19 method contracts
- 1 function with contract

Old KeY-ABS case study: 140 LoC, 1 class, 1 invariant, interactive

Extensions

Advances in Language Coverage over KeY-ABS

- Covers complete imperative layer of CoreABS without exception handlers
- Covers functional layer without `let`
- Specification integrated into ABS

Missing Pieces

- Explicit history using the functional layer and ghost statements
- First-Order Specification and full ABS Session Types
- Additional backends (Why3, KeY-Java, ...)
- Restarting SE for further modalities

Hybrid ABS

Hybrid ABS

Distributed Cyber-Physical Systems

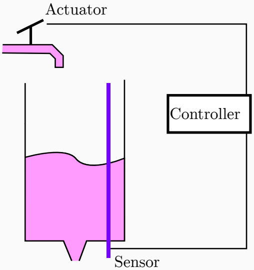
Many modern system are distributed CPS with isolated dynamics: IoT, Industrie 4.0, Digital Twins, ...

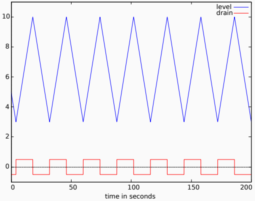
How to (a) model (b) simulate and (c) verify such systems?

Hybrid ABS

- Modeling: Hybrid ABS = ABS + ODEs.
- Verifying: Crowbar + KeYmaera X

Water Tank






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

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.

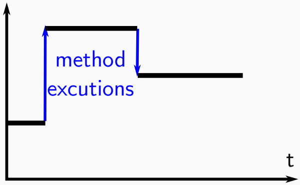
Object Invariants

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.

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

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



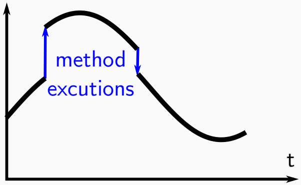
Object Invariants

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.

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

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



Object Invariants

Challenge: Express that I holds until the next method runs.

Object Invariants

Challenge: Express that I holds until the next method runs.

Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

$\phi ::= \forall x.\, \phi \mid \ldots \mid [\alpha]\phi \qquad \alpha ::= ?\phi \mid \mathtt{v} := t \mid \{\mathtt{v}' = f(\mathtt{v}) \& \phi\} \mid \ldots$

Object Invariants

Challenge: Express that I holds until the next method runs.

Differential Dynamic Logic

A logic for (algebraic) hybrid programs:

$\phi ::= \forall x.\, \phi \mid \ldots \mid [\alpha]\phi \qquad \alpha ::= ?\phi \mid \mathtt{v} := t \mid \{\mathtt{v}' = f(\mathtt{v})\&\phi\} \mid \ldots$

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.

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

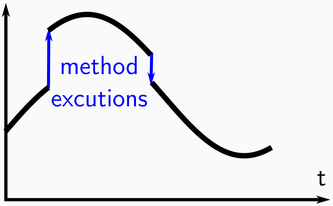
This formula is valid.

Object Invariants

Proof Obligations for Hybrid Active Objects

If we can translate the method body of a method into $d\mathcal{L}$, then we can express that the invariant holds once the method ends and keeps holding when following the dynamics.

$$I \rightarrow [\mathrm{s}]\Big(I \wedge [\text{ode}\&\text{true}]I\Big) \qquad \text{Proof Obligation for } \mathtt{HABS}$$



Using true means that I must hold forever...

Controlled Regions

Controlled Regions

Let g be disjunction of the guards of the *methods that are known to be in the scheduler queue after a method terminates*. To establish I, the following proof obligation suffices:

$$I \rightarrow \left[\mathrm{s}\right]\left(I \wedge \left[\mathrm{ode}\&\neg \mathrm{g}\right] I\right)$$

Example

```
Real m() { ... this!other(); return 0; }
Real other() { await diff x >= 0; ... }
```

$$I \rightarrow \left[\ldots\right]\left(I \wedge \left[\mathrm{ode}\&\mathrm{x} \leq 0\right] I\right)$$

Water Tank

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

$$I \rightarrow [...](I \land [\text{ode}\&\texttt{lvl} \geq 3 \land \texttt{lvl} \leq 10]I)$$

Chisel

Chisel [Kamburjan, HSCC'21]

Post-Regions are implemented as a translation into KeYmaera X.

- Also supports method contracts and local Zeno Behavior.
- Interoperable with `Crowbar` through method contracts.
- Only supports `Real` variables

Chisel

Chisel[Kamburjan, HSCC'21]

Post-Regions are implemented as a translation into KeYmaera X.

- Also supports method contracts and local Zeno Behavior.
- Interoperable with `Crowbar` through method contracts.
- Only supports `Real` variables

Future Work

Is it possible to move all ODEs out of the first program?

duration`(5);` $I \rightarrow \left[\texttt{s}; \{\mathrm{ode}\&t \leq 5\}; \texttt{s'}\right]\left(I \land [\mathrm{ode}\&\mathrm{true}]I\right)$

Conclusion

`Crowbar`: **A flexible framework for prototyping deductive verification of distributed object-oriented programs.**

Conclusion

`Crowbar`: A flexible framework for prototyping deductive verification of distributed object-oriented programs.

On-Going and Future Work

- Redo the KeY-ABS case studies in Crowbar
- Rules as Kotlin DSL
- Comparison of trace specifications/logics in Crowbar
- Verification of coupled objects in Hybrid ABS

Conclusion

`Crowbar`: **A flexible framework for prototyping deductive verification of distributed object-oriented programs.**

On-Going and Future Work

- Redo the KeY-ABS case studies in Crowbar
- Rules as Kotlin DSL
- Comparison of trace specifications/logics in Crowbar
- Verification of coupled objects in Hybrid ABS

Long-term goal

Reintegration with KeY as a KeY-ABS successor

Conclusion

`Crowbar`: **A flexible framework for prototyping deductive verification of distributed object-oriented programs.**

On-Going and Future Work

- Redo the KeY-ABS case studies in Crowbar
- Rules as Kotlin DSL
- Comparison of trace specifications/logics in Crowbar
- Verification of coupled objects in Hybrid ABS

Long-term goal

Reintegration with KeY as a KeY-ABS successor

Thank you for your attention