KeY-Style Verification of ABS with Crowbar

Eduard Kamburjan

University of Oslo

ABS Workshop'21, 26.08.21

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

- $[\mathtt{s}]\forall i \in \mathbb{N}.\,\mathtt{history}[i] \doteq\!\!\!\!/\ \mathsf{invEv}$ 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

- $[\mathtt{s}]\forall i \in \mathbb{N}.\, \mathtt{history}[i] \dot{\neq} \mathsf{invEv}$ — 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]
- $[\mathtt{s}]\Box\mathbf{this}.\mathtt{f>0}$ — 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]

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

$$[\qquad\qquad\qquad\qquad\qquad\qquad]$$

Behavioral Modalities

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

Behavioral Modalities

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

Behavioral Modalities

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

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

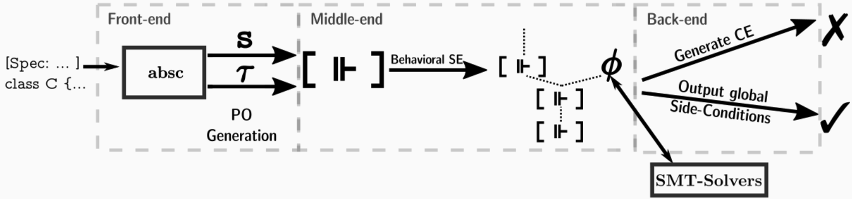
$$\textbf{(BPL)}\ \frac{\Gamma \Rightarrow \{U\}\{\mathtt{x\ :=\ v}\}[\mathtt{s} \Vdash (\phi, \mathit{inv})], \Delta}{\Gamma \Rightarrow \{U\}[\mathtt{x\ =\ v;\ s} \Vdash (\phi, \mathit{inv})], \Delta}$$

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

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

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

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.

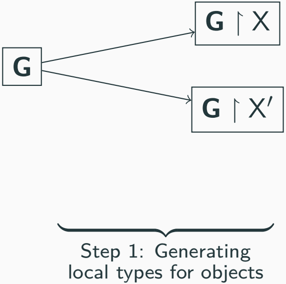
ABS has a lightweight analysis to mark expression as non-null.

```
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;
}
```

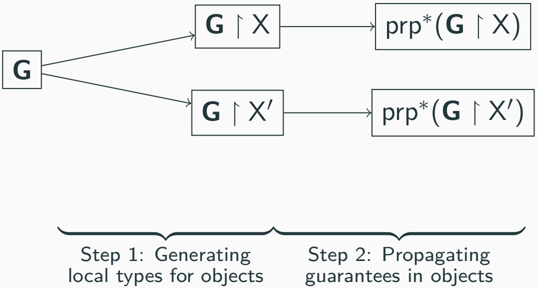
- Crowbar keeps this information in the AST
- Safe accesses do not cause branching

Top-Down Specification with Session Types

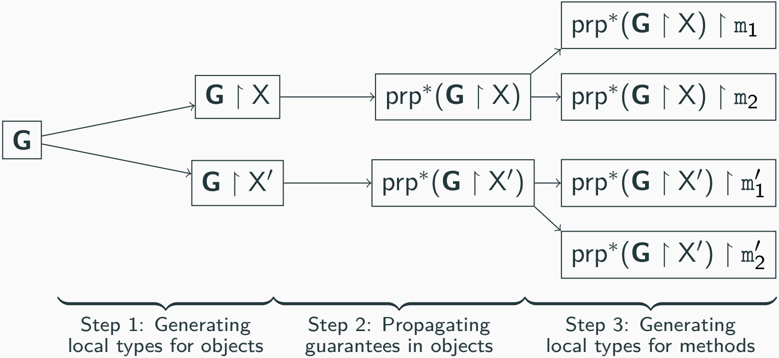
Top-Down Specification with Session Types



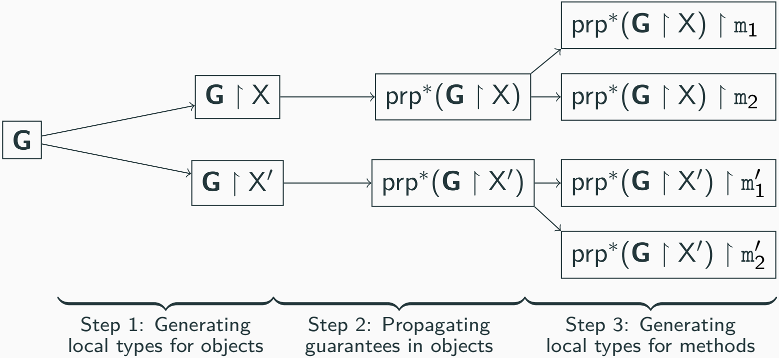
Top-Down Specification with Session Types



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--);
  }
}
```

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

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);
```

$$(\forall\ \texttt{Int}\ x.\ x \geq 0 \rightarrow \texttt{fac}(x) \geq 0) \land \texttt{n} \geq 0$$
$$\rightarrow [\texttt{result = if(n <= 1) then 1 else n*fac(n-1);} \Vdash^{\alpha_{\mathsf{pst}}} \texttt{result} \geq 0]$$

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

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 almost complete imperative layer of CoreABS (e.g., no 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

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

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

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

Long-term goal

Reintegration with KeY as a KeY-ABS successor

Thank you for your attention