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

- \( [s] \forall i \in \mathbb{N}. \text{history}[i] \neq \text{invEv} \)

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
- \([s] \forall i \in \mathbb{N}. \text{history}[i] \neq \text{invEv}\)  
  ABDL [Din et al., SEFM’12]
- \([s] \Box \text{this}. f > 0\)  
  DTL [Beckert and Bruns, CADE’13]

Kamburjan — KeY-Style Verification of ABS with Crowbar
## 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

<table>
<thead>
<tr>
<th>Formula</th>
<th>Description</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[s] \forall i \in \mathbb{N}. \text{history}[i] \neq \text{invEv}$</td>
<td>ABDL [Din et al., SEFM'12]</td>
<td></td>
</tr>
<tr>
<td>$[s] \Box \text{this}.f&gt;0$</td>
<td>DTL [Beckert and Bruns, CADE'13]</td>
<td></td>
</tr>
<tr>
<td>$[s] \text{finite} \ast \ast [\text{this}.f&gt;0] \ast \ast \text{finite}$</td>
<td>DLCT [Din et al., TABLEAUX’15&amp;’17]</td>
<td></td>
</tr>
</tbody>
</table>
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

- \([s] \forall i \in \mathbb{N}. \text{history}[i] \neq \text{invEv}\)  
  \(\text{ABSDL [Din et al., SEFM'12]}\)

- \([s] \Box \text{this} . f>0\)  
  \(\text{DTL [Beckert and Bruns, CADE'13]}\)

- \([s] \text{finite} \ast \ast [\text{this} . f>0] \ast \ast \text{finite}\)  
  \(\text{DLCT [Din et al., TABLEAUX'15&'17]}\)

- \(s \vdash X!m(\text{this} . f>0) . Y!n . \text{end}\)  
  \(\text{Session Types for AO [Kamburjan and Chen, iFM'18]}\)
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

- \([s] \forall i \in \mathbb{N}. \text{history}[i] \neq \text{invEv}\)
- \([s] \Box \text{this}. f > 0\)
- \([s] \text{finite} \ast \ast \text{this}. f > 0 \ast \ast \text{finite}\)
- \(s \vdash X!m\langle \text{this}. f > 0\rangle . Y!n. \text{end}\)
- And more....
Behavioral Modalities

\[ \alpha \models \tau \]

Kamburjan — KeY-Style Verification of ABS with Crowbar
Behavioral Modalities

statement
Behavioral Modalities

\[ S \quad \text{statement} \quad \tau \quad \text{type} \]
Behavioral Modalities

\[ \alpha \models \tau \]

semantics

statement type

\( [s \models \alpha] \)
### Example

Trace-specifications are too complex for simple post-conditions.
- ABDL has object-invariant *implicit*
- BPL makes structure explicit
Example

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

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

\[
\begin{align*}
\text{(BPL)} & \quad \Gamma \Rightarrow \{ U \} \{ x := v \} [ s \models (\phi, \text{inv}) ], \Delta \\
& \quad \Gamma \Rightarrow \{ U \} [ x = v ; \ s \models (\phi, \text{inv}) ], \Delta
\end{align*}
\]
### Example

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

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

\[
\begin{align*}
\Gamma & \Rightarrow \{U\} \{x := v\} [s \models (\phi, inv)], \Delta \\
\Gamma & \Rightarrow \{U\} [x = v; s \models (\phi, inv)], \Delta \\
\Gamma, \{U_A\} inv & \Rightarrow \{U_A\} [s \models (\phi, inv)], \Delta \\
\Gamma & \Rightarrow \{U\} [\text{await} \ e?; s \models (\phi, inv)], \Delta \\
\Gamma & \Rightarrow \{U\} [\text{while} (e) \{s\} s' \models (\phi, inv)], \Delta
\end{align*}
\]
Crowbar
**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
## Supported Specification Approaches

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

- 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.

```
1 ... 
2 [Spec: LoopInv(i>=0)]
3 while(i > 0) i = i-1;
4 ... 
```
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.

```java
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
Step 1: Generating local types for objects

Prp\star(G|X)\star(G|X)

Step 2: Propagating guarantees in objects

Prp\star(G|X\prime|m\prime2)\star(G|X\prime|m\prime1)\star(G|X|m2)\star(G|X|m1)

Step 3: Generating local types for methods

• Propagation is outside Crowbar
• Each class generates a static node for projection
Step 1: Generating local types for objects
Top-Down Specification with Session Types

Step 1: Generating local types for objects

Step 2: Propagating guarantees in objects
Top-Down Specification with Session Types

Step 1: Generating local types for objects

Step 2: Propagating guarantees in objects

Step 3: Generating local types for methods

G ↠ G ↠ X ↠ prp∗(G ↠ X) ↠ prp∗(G ↠ X) ↠ m2
G ↠ X′ ↠ prp∗(G ↠ X′) ↠ prp∗(G ↠ X′) ↠ m′1
G ↠ prp∗(G ↠ X′) ↠ prp∗(G ↠ X′) ↠ m′2
G ↠ prp∗(G ↠ X) ↠ prp∗(G ↠ X) ↠ m1
Step 1: Generating local types for objects

Step 2: Propagating guarantees in objects

Step 3: Generating local types for methods

- Propagation is outside Crowbar
- Each class generates a static node for projection
Session Types

```java
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--);
  }
}
```

\[
\Gamma \Rightarrow \{ U \}[x = this.f \wedge \phi], \Delta
\]
\[
\Gamma \Rightarrow \{ U \}[v := f][s \models L], \Delta
\]
\[
\Gamma \Rightarrow \{ U \}[v = this.f!m(); s \models x!m(\phi).L], \Delta
\]
ABS has a functional sublanguage for ADTs. Each definition is translated into an assignment with contracts.

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

```plaintext
1 [Spec: Requires(n >= 0)] [Spec: Ensures(result >= 0)]
2 Int fac(Int n){
3    return if(n<=1) then 1 else n*this.fact(n-1);
4 }
```
ABS has a functional sublanguage for ADTs. Each definition is translated into an assignment with contracts.

\[
1 \text{ [Spec: Requires}(n \geq 0)\text{]} [\text{Spec: Ensures(result} \geq 0)]
\]

\[
2 \text{ def Int fac(Int n) = if}(n \leq 1) \text{ then 1 else n*fac(n-1);}
\]

\[
(\forall \text{ Int } x. \ x \geq 0 \rightarrow \text{fac}(x) \geq 0) \land n \geq 0
\]

\[
\rightarrow [\text{result} = \text{if}(n \leq 1) \text{ then 1 else n*fac(n-1); } |\alpha_{pst} \text{ result} \geq 0]
\]

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

```c
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 \( \text{fib}(n) \) which returns a number between 1 and the \( n \)th 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
### 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

Thank you for your attention.
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