Generating Counterexamples for Java Dynamic Logic

Philipp Rümmer
philipp@cs.chalmers.se

9th June 2005

Overview of the Talk

- Notion of counterexamples
- Derivation of counterexamples by disproving
- Example

The talk describes work in progress

DL Correctness Statements, Counterexamples

- Typically in DL: Correctness characterised by validity
  Program is *correct* $\iff$ Formula is *valid*
- Specification through pre-/postconditions, invariants, mostly formulas like

$$\varphi \rightarrow \langle\, p \,\rangle\, \psi, \qquad \varphi \rightarrow [\, p \,]\, \psi$$

  Intuitively: If *p* is started in a state allowed by φ, then after execution ψ holds

- Unfortunately: Most programs are *not* correct
  ▶ Counterexamples (CE), formula is invalid

Counterexamples

Program states are in JavaDL modelled as first-order structures: Values of

- program variables, class attributes
- instance attributes (unary functions)
- arrays (binary functions)

▶ Further symbols not considered for the time being (in specification)

Counterexamples are first-order structures violating a correctness statement (formula)

Counterexamples

- For instance: CEs for

$$\langle\, \text{a.o = 5;} \,\rangle\ \text{a.o} \mathrel{\dot{\neq}} 0$$

  are structures that interpret a with null
- Knowledge about CE could be used to locate bugs (in program *p* or specification φ, ψ)

▶ Tasks: Prove formulas invalid, derive counterexamples

Generating Counterexamples by Disproving

$$\varphi \rightarrow \langle\, p \,\rangle\, \psi \text{ invalid} \quad \text{iff} \quad \neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi) \text{ satisfiable}$$

Use appropriate calculus for satisfiability, extract counterexample from proof

Proving satisfiability $\leftrightarrow$ Building models
For FOL:

- (Finite) Model Finders
- Building Herbrand models
- Saturating clause sets

Situation in JavaDL somewhat different:

- Domains essentially fixed and infinite
- Lots of theories involved

Generating Counterexamples by Disproving

Approaches alternative to disproving:

- Testing
- Extract information from failing verification attempts
- Software model checking, abstraction

Disproving?

- Systematic approach, completeness results possible
- Can derive closed representations of *classes* of CEs
- Efficiency?

▶ How to show satisfiability in KeY?

Used Here: Reduction of Satisfiability to Validity

$$\neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi) \text{ satisfiable}$$

In other (informal) words:

$$\exists\ \textit{initial\_state}.\ \neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi)$$

For this “formula” validity and satisfiability coincide

- How to express *initial_state*? (Higher-order quant.?)

Used Here: Reduction of Satisfiability to Validity

$$\neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi) \text{ satisfiable}$$

In other (informal) words:

$$\exists\ \textit{initial\_state}.\ \neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi)$$

For this “formula” validity and satisfiability coincide

- How to express *initial_state*? (Higher-order quant.?)
  - Program variables: First-order quantification

$$\exists xx.\, \{\, \mathrm{x} := xx \,\} \ldots$$

Used Here: Reduction of Satisfiability to Validity

$$\neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi) \text{ satisfiable}$$

In other (informal) words:

$$\exists\ \mathit{initial\_state}.\ \neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi)$$

For this "formula" validity and satisfiability coincide

- How to express *initial_state*? (Higher-order quant.?)
  - Program variables: First-order quantification

    $$\exists xx.\ \{\ \mathrm{x} := xx\ \}\ \ldots$$

  - Instance attributes: Quantification over lists

    $$\exists l : \mathit{ListOfT}.\ \{\ \mathtt{for}\ i : \mathit{nat}.\ \mathit{obj}_C(i).\mathrm{attr} := l_i\ \}\ \ldots$$

Used Here: Reduction of Satisfiability to Validity

$$\neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi) \text{ satisfiable}$$

In other (informal) words:

$$\exists\ \mathit{initial\_state}.\ \neg(\varphi \rightarrow \langle\, p \,\rangle\, \psi)$$

For this "formula" validity and satisfiability coincide

- How to express *initial_state*? (Higher-order quant.?)
  - Program variables: First-order quantification

    $$\exists xx.\ \{\ \mathrm{x} := xx\ \}\ \ldots$$

  - Instance attributes: Quantification over lists

    $$\exists l : \mathit{ListOfT}.\ \{\ \mathtt{for}\ i : \mathit{nat}.\ \mathit{obj}_C(i).\mathrm{attr} := l_i\ \}\ \ldots$$

  - Arrays: (FO) Quantification over lists of lists

Reduction of Satisfiability to Validity

- Justification: A Java program only has countably many states
  ▶ Only finitely many objects used
- Searching for lists efficiently possible using metavariables
  ▶ Use unification to construct system snapshots (substitution that closes proof)

Example: Swapping of Array Cells

Program swapping array cells a[i], a[j] of type *int*:

```
a[j] += a[i];
a[i]  = a[j]-a[i];
a[j] -= a[i];
```

Example: Swapping of Array Cells

Program swapping array cells a [i], a [j] of type *int*:

$$\exists x.\ \exists y.\ \ a[i] \doteq x \ \wedge\ a[j] \doteq y \wedge$$

```
⟨a[j] += a[i];
 a[i]  = a[j]−a[i];
 a[j] −= a[i];⟩
```

$$a[i] \doteq y \wedge a[j] \doteq x$$

Specification telling that after execution cells are swapped

Example: Swapping of Array Cells

Program swapping array cells a[i], a[j] of type *int*:

$$
\begin{array}{l}
\mathtt{a} \doteq\!\!\!\!/\; \mathsf{null} \\
\to\ \exists x.\ \exists y.\ \ \mathtt{a[i]} \doteq x \ \wedge\ \mathtt{a[j]} \doteq y \wedge \\
\qquad\qquad \langle\, \mathtt{a[j]\ +=\ a[i];} \\
\qquad\qquad\ \ \mathtt{a[i]\ \ \ =\ a[j]-a[i];} \\
\qquad\qquad\ \ \mathtt{a[j]\ -=\ a[i];}\, \rangle \\
\qquad\qquad\qquad\qquad \mathtt{a[i]} \doteq y \wedge \mathtt{a[j]} \doteq x
\end{array}
$$

Precondition: a must not be null

Example: Swapping of Array Cells

Program swapping array cells a [i], a [j] of type *int*:

$$a \not\doteq \mathrm{null} \;\wedge\; i \in [0, a.\mathit{length}) \;\wedge\; j \in [0, a.\mathit{length})$$
$$\rightarrow\; \exists x.\,\exists y.\;\; a[i] \doteq x \;\wedge\; a[j] \doteq y \;\wedge$$
$$\langle\, \texttt{a[j] += a[i];}\;\; \texttt{a[i] = a[j]-a[i];}\;\; \texttt{a[j] -= a[i];} \,\rangle$$
$$a[i] \doteq y \wedge a[j] \doteq x$$

Precondition: Indexes must be within bounds

Example: Swapping of Array Cells

Program swapping array cells a[i], a[j] of type *int*:

$$
\begin{array}{l}
a \not\doteq \mathrm{null} \ \wedge\ i \in [0, a.\mathit{length}) \ \wedge\ j \in [0, a.\mathit{length}) \\
\to\ \exists x.\, \exists y.\ a[i] \doteq x \ \wedge\ a[j] \doteq y \ \wedge \\
\qquad \langle\, \mathtt{a[j]\ +=\ a[i];} \\
\qquad\ \ \mathtt{a[i]\ \ =\ a[j]-a[i];} \\
\qquad\ \ \mathtt{a[j]\ -=\ a[i];}\,\rangle \\
\qquad\qquad a[i] \doteq y \wedge a[j] \doteq x
\end{array}
$$

Is this formula valid?

Example: Proving Formula Invalid

- In the sequel handling of objects is simplified (a $\not\doteq$ null is left out)
- Make quantification of occurring symbols explicit:

$$\forall l : \mathit{ListOfInt}.\ \forall len.\ \forall ii.\ \forall jj.$$

$$\{\, \mathrm{i} := ii,\ \mathrm{j} := jj,\ \mathrm{a}.length := len,\ \texttt{for}\ k \in [0, l.len).\ \mathrm{a}[\,k\,] := l_k \,\}$$

$$\mathrm{i} \in [0, \mathrm{a}.length) \ \land\ \mathrm{j} \in [0, \mathrm{a}.length)$$

$$\rightarrow\ \exists x.\ \exists y.\ \ \mathrm{a}[\mathrm{i}] \doteq x \ \land\ \mathrm{a}[\mathrm{j}] \doteq y \ \land$$

$$\langle\, \texttt{a[j] += a[i];} \quad \texttt{a[i] = a[j]-a[i];} \quad \texttt{a[j] -= a[i];} \,\rangle$$

$$\mathrm{a}[\mathrm{i}] \doteq y \land \mathrm{a}[\mathrm{j}] \doteq x$$

Example: Proving Formula Invalid

- Negate formula; quantified variables can be replaced with metavariables:

$$\{\, \mathrm{i} := \mathit{II},\ \mathrm{j} := \mathit{JJ},\ \mathrm{a}.\mathit{length} := \mathit{LEN},\ \mathtt{for}\ k \in [0, L.\mathit{len}).\ \mathrm{a}[k] := L_k \,\}$$

$$\neg(\ \mathrm{i} \in [0, \mathrm{a}.\mathit{length}) \wedge \mathrm{j} \in [0, \mathrm{a}.\mathit{length}) \rightarrow \exists x.\, \exists y.\ \mathrm{a}[\mathrm{i}] \doteq x \wedge \mathrm{a}[\mathrm{j}] \doteq y \wedge$$

$$\langle\, \mathtt{a[j]\ +=\ a[i];}\ \ \mathtt{a[i]\ \ =\ a[j]-a[i];}\ \ \mathtt{a[j]\ -=\ a[i];} \,\rangle\ \mathrm{a}[\mathrm{i}] \doteq y \wedge \mathrm{a}[\mathrm{j}] \doteq x\ )$$

- Calculus of KeY can then eliminate program by symbolic execution ...

Example: Proving Formula Invalid

- Afterwards in the proof tree three goals remain:

$$\vdash \ \mathrm{II} \in [0, LEN) \ \wedge \ \mathrm{JJ} \in [0, LEN)$$

$$\ldots \ \vdash \ II \doteq JJ$$

$$\ldots \ L_{II} \doteq 0, \ II \doteq JJ \ \vdash$$

  - ▶ case distinction to treat equal array indexes *II*, *JJ*
- Proof is closed e.g. by substitution

$$[II/0, \ JJ/0, \ L/[1], \ LEN/1]$$

- Class of counterexamples described by

$$II \doteq JJ \ \wedge \ L_{II} \not\doteq 0 \ \wedge \ \mathrm{II} \in [0, LEN)$$

For JavaDL: Disproving Easier than Proving?

- For partial incorrectness loops can be treated without induction
  - ▶ Non-interactive proof procedure seems feasible
- The construction shows that relatively complete calculi for disproving in a fragment of JavaDL exist
  - Restricted vocabulary
  - No evil formulas talking about infinitely many objects
- Method very similar to testing: Simultaneous testing of all possible initial states (symbolically)

Next Steps . . . What could be investigated . . .

- Consider disproving (obligations) in practice/for real-world programs ▶ Master thesis of Muhammad Ali Shah
- Automatic extraction of counterexamples
- Treat shortcomings of KeY: Arithmetic, Equations
- Disproving when leaving the JavaDL fragment?

- Disproving for showing program correctness?
  Program p is *correct* $\iff$ Formula is *satisfiable*

- Different direction: Try to locate bugs more precisely