Enhancing the Alloy Analyzer with Patterns of Analysis

William Heaven

in collaboration with
Alessandra Russo

Imperial College London
Motivation

- Formal techniques not yet widely adopted by programmers.
- Commercial pressure to produce higher quality software is increasing.
Motivation

- Formal techniques not yet widely adopted by programmers.
- Commercial pressure to produce higher quality software is increasing.

Software developers favour so-called *lightweight* techniques that provide immediate returns and sit comfortably with activity of implementation.
Motivation

- Formal techniques not yet widely adopted by programmers.
- Commercial pressure to produce higher quality software is increasing.
- Software developers favour so-called *lightweight* techniques that provide immediate returns and sit comfortably with activity of implementation.

- Existing lightweight techniques (such as JML and Alloy) still suffer shortcomings
  - Notation
  - Limited or misleading feedback from tools
class BadInvariant {
    //@ invariant x.equals (y) && ! x.equals (y);
    Integer x = new Integer (1);
    Integer y = new Integer (1);

    //@ requires true;
    //@ ensures x != k;
    void setX (Integer k) { x = k; }
}
class BadInvariant {
    //@ invariant x.equals (y) && ! x.equals (y);
    Integer x = new Integer (1);
    Integer y = new Integer (1);

    //@ requires true;
    //@ ensures x != k;
    void setX (Integer k) { x = k; }
}

INV \land PRE \land CODE \rightarrow POST
JML Example

class BadInvariant {
  //@ invariant x.equals (y) && ! x.equals (y);
  Integer x = new Integer (1);
  Integer y = new Integer (1);

  //@ requires true;
  //@ ensures x != k;
  void setX (Integer k) { x = k; }
}

- **INV** \( \land \) **PRE** \( \land \) **CODE** \( \rightarrow \) **POST**
The implication is vacuously true.
Alloy Example

sig Project { }

sig Employee { project : Project }

sig Pool extends Employee { } { no project }

fact { some Pool }

pred PropertyTest () {
    some e : Employee | e not in Pool
}

run PropertyTest for 4
Alloy Example

sig Project { }

sig Employee { project : Project }

sig Pool extends Employee { } { no project }

fact { some Pool }

pred PropertyTest () {
    some e : Employee | e not in Pool
}

run PropertyTest for 4

Analyzer suggests that PropertyTest is inconsistent with the specification.

But is this really all?
Alloy Example

```alloy
sig Project { }

sig Employee { project : Project }

sig Pool extends Employee { } { no project }

fact { some Pool }

pred PropertyTest () {
    some e : Employee | e not in Pool
}
run PropertyTest for 4
```

\[ \models \Gamma \land P \]
Aims & Approach

- Development of a lightweight specification environment for OO programs that provides richer analysis feedback.
Aims & Approach

- Development of a lightweight specification environment for OO programs that provides richer analysis feedback.

- Loy

- Patterns of analysis

Lightweight specification language for OO programs built upon Alloy.

For richer feedback.
Example Loy Specification

class Project {
    manager : Manager
    invariant some manager
}

class Employee {
    project : Project
    invariant no project.manager

    assign (p : Project)
    requires no project
    ensures project' = p
    modifies project
}

class ManagedEmployee extends Employee {
    manager : Manager
    depends manager <- project

    assign (p : Project)
    requires no project
    ensures project' = p and manager' = p.manager
    modifies project
}
Analysis

- Check consistency of
  - invariants
  - invariants and precondition
  - invariants and postcondition
  - precondition and postcondition
  - postcondition and frame condition
  - ...

- Check behavioural subtype properties
  - invariants of subtype imply invariants of supertype
  - overriding postconditions imply overridden postconditions
  - ...

Pattern Application

Check that invariant and postcondition of assign in ManagedEmployee (type B) together imply postcondition of assign in Employee (type A)

$$\phi: \text{assign-POST}_B \land \text{INV}_B \Rightarrow \text{assign-POST}_A$$
Pattern Application

- Check that invariant and postcondition of assign in ManagedEmployee (type B) together imply postcondition of assign in Employee (type A)

\[ \Phi: \text{assign-POST}_B \land \text{INV}_B \rightarrow \text{assign-POST}_A \]

1) Apply pattern for “\(\rightarrow\)” to \(\Phi\)

- Pattern warns of vacuous satisfiability of \(\Phi\) due to unsatisfiable antecedent.
Pattern Application

Check that invariant and postcondition of assign in ManagedEmployee (type B) together imply postcondition of assign in Employee (type A)

\[ \phi: assign-POST_B \land INV_B \rightarrow assign-POST_A \]

1) Apply pattern for “\(\rightarrow\)” to \(\phi\)
   - Pattern warns of vacuous satisfiability of \(\phi\) due to unsatisfiable antecedent.

2) Apply pattern for “\(\land\)” to antecedent
   - Pattern checks satisfiability of each combination of conjunct and identifies unsatisfiability of \(assign-POST_B \land INV_B \rightarrow assign-POST_A\).
Example Loy Specification

```loyalty
class Project {
    manager : Manager
    invariant some manager
}

class Employee {
    project : Project
    invariant
        no project.manager

    assign (p : Project)
        requires no project
        ensures project' = p
        modifies project
}

class ManagedEmployee extends Employee {
    manager : Manager
    depends manager <- project

    assign (p : Project)
        requires no project
        ensures project' = p and
            manager' = p.manager
        modifies project
}
```
Negation and Conjunction

**Negation**

\[ \text{SAT } [\neg A]_T \]

**YES**

**apply** \([A]_T \)

**NO**

**Q:** Why is \(A\) valid?

**apply** \([A]_T \)

**Q:** Is \(A_i\) vacuously satisfied?

**apply** \([A_i]_T\), \(1 \leq i \leq n\)

**Conjunction**

\[ \text{SAT } [A_1 \land \ldots \land A_n]_T \]

**YES**

**Q:** Why is \(A_1 \land \ldots \land A_n\) unsatisfiable?

**apply** \([A_i \land \ldots \land A_j]_T\), \(1 \leq i < j \leq n, 1 \leq k < n\)

**NO**

\[ \text{SAT } [A]_T, \text{ SAT } [\neg A]_T \]
Implication

SAT [A -> B] \( T \)

YES

SAT [A] \( T \)

NO

WARNING

SAT [¬B] \( T \)

YES

apply [A] \( T \),

apply [B] \( T \)

NO

WARNING

SAT [(A \( \land \) ¬A) -> B] \( T \)

YES

SAT [A \( \land \) ¬A] \( T \)

NO

WARNING [vacuously SAT.]

Q: Why is A valid? apply[A] \( T \),

Q: Why is B unsatisfiable? apply[B] \( T \)
Universal Quantification

Q: Is $A(x)$ vacuously satisfiable? apply $[A(x)]_{(x,X)} + T$

**YES**

We know formula is unsatisfiable for at least one value of $x$. This SAT query will provide a value.

**NO**

**WARNING**

We know formula is unsatisfiable for at least one value of $x$. This SAT query will provide a value.

SAT $[\forall x \in X. A(x)]_{T}$

SAT $[\neg \exists y \in Y. P(x,y)]_{(x,X)} + T$

Q: Is $A(x)$ satisfiable? apply $[A(x)]_{(x,X)} + T$

**YES**

**NO**

**YES**
Existential Quantification

SAT \[\text{some } y : Y . A (y)\] \(\tau\)

**YES**

Q: Is \(A(y)\) vacuously satisfiable?
apply \(\{A(y)\}_{(y,Y)} + \tau\)

**NO**

Q: Why is \(A(y)\) unsatisfiable?
apply \(\{A(y)\}_{(y,Y)} + \tau\)

Y \(\neq\) \{\}

**YES**

**NO**

WARNING [empty domain: vacuously unsatisfiable]
Future Work

- Finish work on implementing prototype tool on top of Alloy Analyzer.
Future Work

- Finish work on implementing prototype tool on top of Alloy Analyzer.
- Address main limitation that satisfiability checking is labour intensive – one approach to be investigated is the implementation of a change-management system to avoid unnecessary re-analysis of satisfiability.
Future Work

- Finish work on implementing prototype tool on top of Alloy Analyzer.

- Address main limitation that satisfiability checking is labour intensive – one approach to be investigated is the implementation of a change-management system to avoid unnecessary re-analysis of satisfiability.

- Investigate complexity and completeness issues of the approach.