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

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; }
}
```

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 ∧ PRE ∧ CODE -> 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* ∧ *PRE* ∧ *CODE -> POST*
- ESC/Java2 passes *setX.*

.. implication vacuously true.

Alloy Example

```
sig Project { }
sig Employee { project : Project }
sig Pool extends Employee { } { no project }
fact { some Pool }
```

Γ

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

P

Alloy Example

```
sig Project { }
sig Employee { project : Project }
sig Pool extends Employee { } { no project }
fact { some Pool }
```

Γ

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

P

- Analyzer suggests that *PropertyTest* is inconsistent with the specification.
- But is this really all?

Alloy Example

```
sig Project { }
sig Employee { project : Project }
sig Pool extends Employee { } { no project }
fact { some Pool }
```

Γ

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

P

- $\Vdash \Gamma \wedge P$

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

Lightweight specification language for OO programs built upon Alloy.

- Patterns of analysis

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*)

$$\Phi: \quad \textit{assign-POST}_B \wedge \textit{INV}_B \dashrightarrow \textit{assign-POST}_A$$

Pattern Application

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

$$\Phi: \boxed{assign\text{-}POST_B \wedge INV_B \ \texttt{-->}\ assign\text{-}POST_A}$$

1) Apply pattern for “ -->” to Φ
   - Pattern warns of vacuous satisfiability of Φ 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*)

$$\Phi: \boxed{\textit{assign-POST}_B \wedge \textit{INV}_B \;\texttt{-->}\; \textit{assign-POST}_A}$$

1) Apply pattern for " --> " to Φ
   - Pattern warns of vacuous satisfiability of Φ due to unsatisfiable antecedent.
2) Apply pattern for "∧" to antecedent
   - Pattern checks satisfiability of each combination of conjunct and identifies unsatisfiability of $\textit{assign-POST}_B \wedge \textit{INV}_B$.

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
}
```

Negation and Conjunction

SAT [¬A] $_T$

YES / *NO*

apply [A] $_T$

Q: Why is A valid?
apply [A] $_T$

Negation

SAT [¬(A ∧ ¬A)] $_T$

YES

SAT [A ∧ ¬A] $_T$

NO

SAT [A] $_T$, SAT [¬A] $_T$

SAT [A1 ∧ .. ∧ An] $_T$

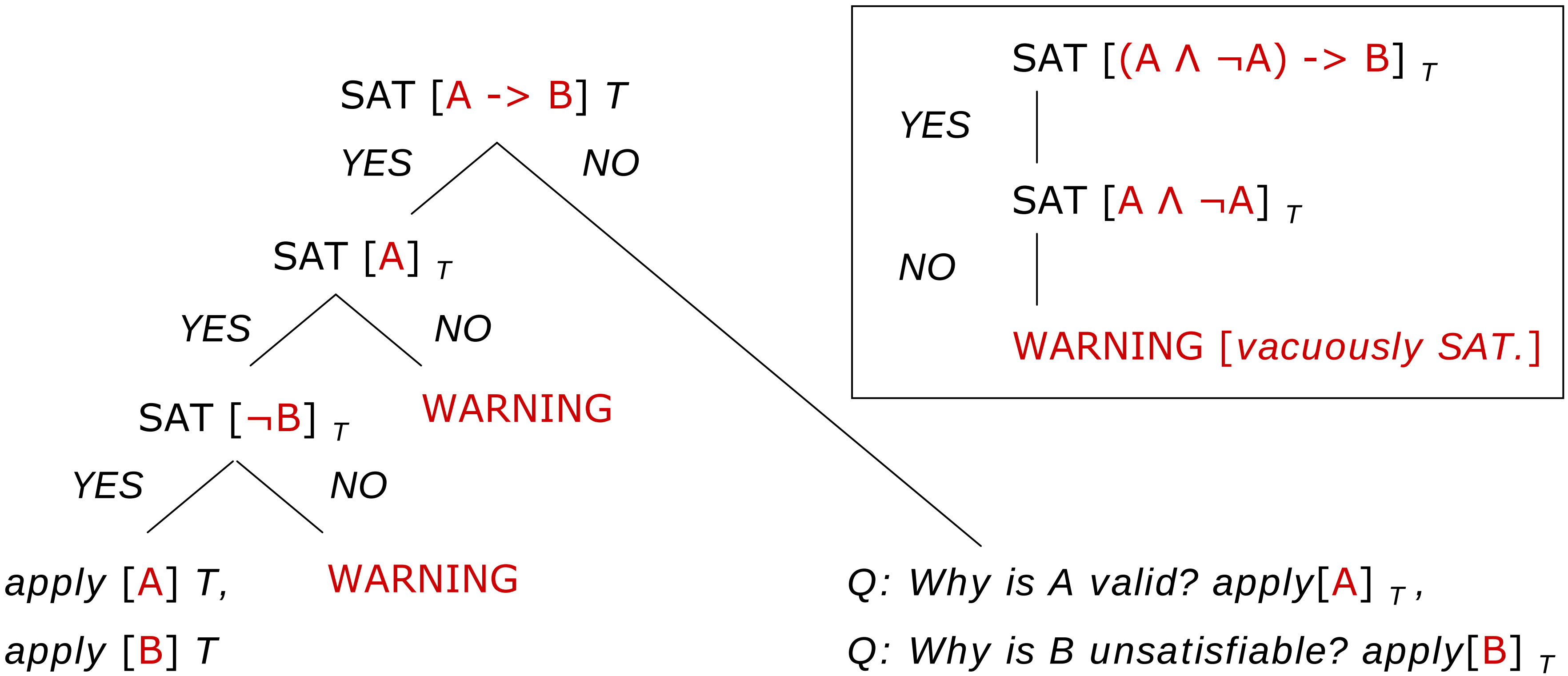
YES / *NO*

Q: Is Ai vacuously satisfied?
apply [Ai] $_T$, $1 \le i \le n$

Q: Why is A1 ∧ .. ∧ An unsatisfiable?
apply [Ai_1 ∧ .. ∧ Aj_k] $_T$,
$1 \le i < j \le n$, $1 \le k < n$

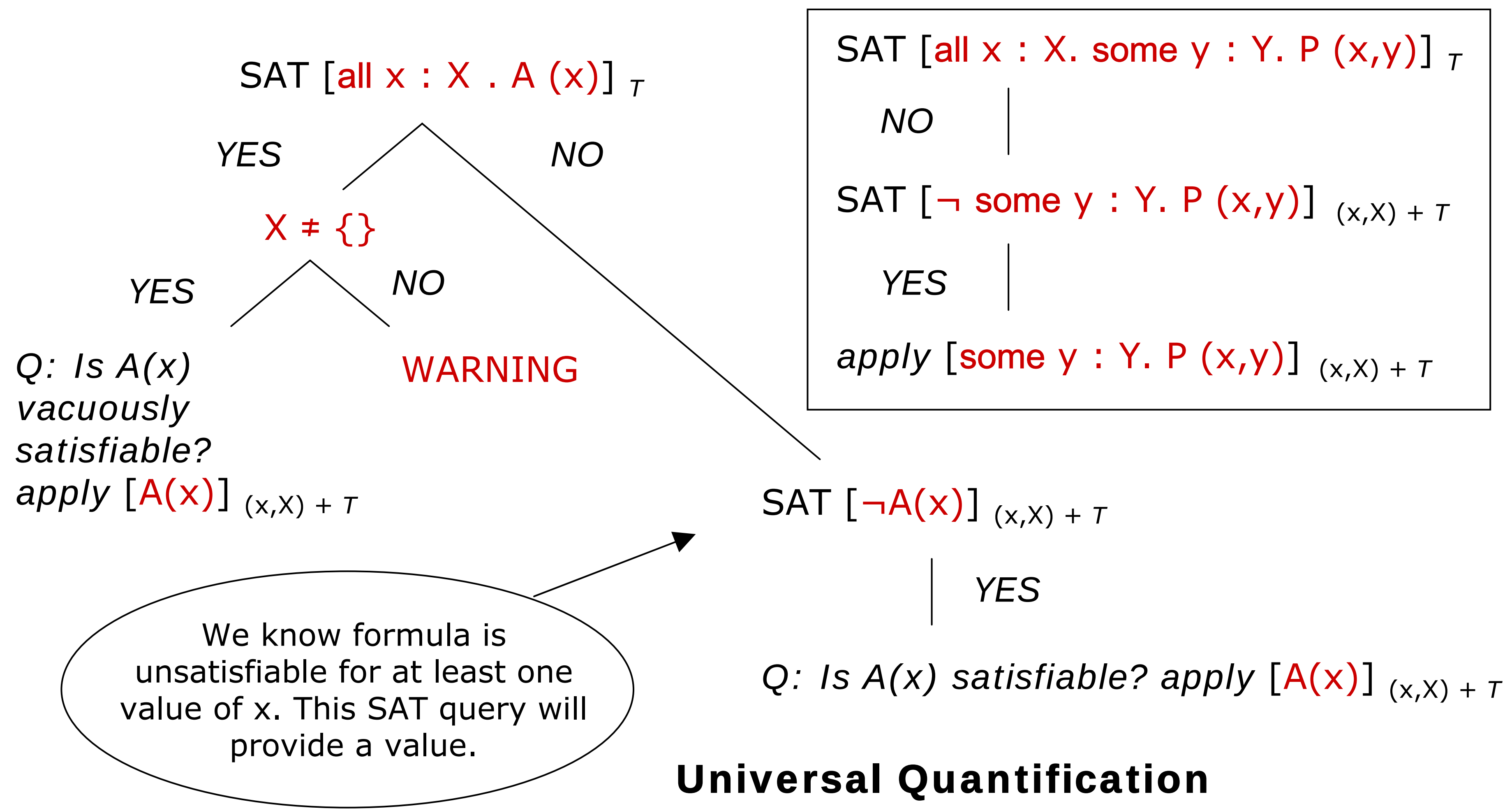
Conjunction

Implication



Implication

Universal Quantification



Existential Quantification

SAT [some y : Y . A (y)] $_T$

YES *NO*

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

SAT [all y : Y. P (x,y)] $_{(x,X) + T}$

NO

Y ≠ {}

NO

WARNING [*empty domain: vacuously unsatisfiable*]

Y ≠ {}

YES *NO*

Q: Why is A(y) unsatisfiable?
apply [A(y)] $_{(y,Y) + T}$

WARNING

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.