Snugglebug

Work-In-Progress

Stephen Fink
Satish Chandra
Manu Sridharan

IBM T. J. Watson Research Center
March 28, 2008

Technology Quarterly

SOFTWARE

Building a better bug-trap

Jun 19th 2003

From *The Economist* print edition

People who write it are human first and programmers only second—in short, they make mistakes, lots of them. Can software help them write better software?

Software bugtraps

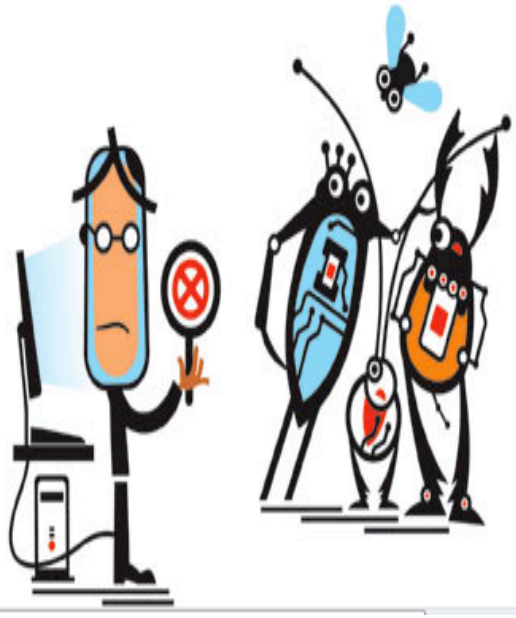
Software that makes software better

Mar 6th 2008

From *The Economist* print edition

Computing: Programmers are using a variety of software tools to help them produce better code and keep bugs at bay

Illustration by Petra Stefankova



What's wrong with current bug finding tools?

1. False positives. Lots of them.

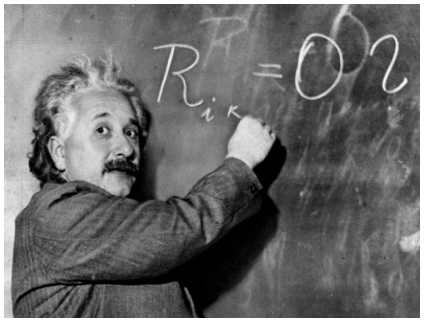
Mostly local pattern matching

Hundreds of "rules"

Fragile ad hoc ranking heuristics

So all we need is better analysis technology?

- precise, scalable interprocedural analysis to move beyond local scope and eliminate false positives??



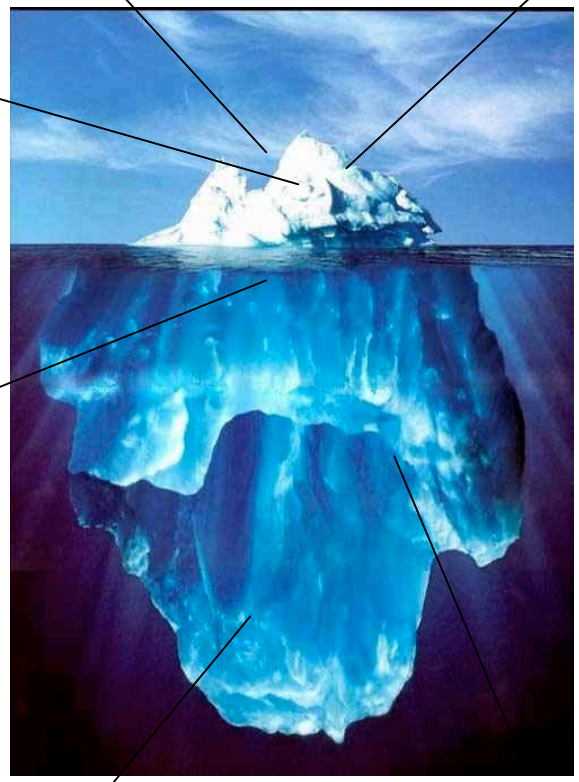
? What if God provided infinitely precise analysis ?

What specifications do tools check?

Buffer overflows

Misc. "rules"

Null Derefs

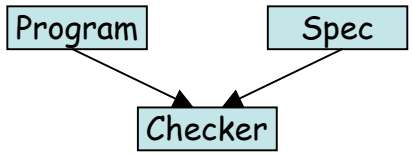


Claim: If you read bug trackers, vast majority of critical defects discovered in the field are **below the waterline.**

```
foo(x) {
   if (x == null) BOOM;
}
```

"The ATM was not supposed to e-mail my PIN to my ex-wife".

"The form did not resize correctly when using a Korean font"



When a tool reports a finding, it means either:

BUGGY CODE: The code is buggy. ☺

BUGGY SPEC: The specification is buggy. ☹

FALSE ALARM: The analysis is inexact. ☹

Our goals:

1. Eliminate FALSE ALARM

   Always generate concrete witnesses (JUnit tests)

   - Developer can step through with debugger, understand cause completely
   - Can add test to regression suite
   - Give up on verification

2. Attack BUGGY SPECS with analysis-driven feedback loop to acquire specifications

   Reduce Costs

   - Specifications manifest as assertions in source language

     JML?
   - Machine infers specifications. Simple UI for developer to Accept/reject specifications

   Increase Benefits

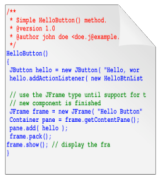
   - Generate counterexamples that violate accepted specifications

This sounds like …

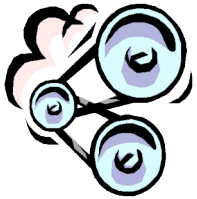
Agitator, Alloy, Boogie, CUTE, DART, Daikon, DIDUCE, DSD-Crasher, Dynamine, DySy, ESC, Korat, Java Pathfinder, JCrasher, jCUTE, Jex, JML, Houdini, MAPO, Metal, Miniatur, Perracotta, Pex, PreFIX, PR-Miner, Randoop, Saturn, SMART, TestEra, SPEC#, Symestra, Synergy,

Your Project (egregiously omitted) …

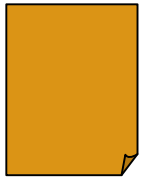
Today's workflow:



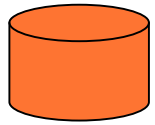
Code

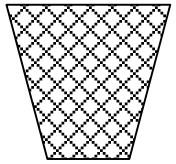


Analysis



Report

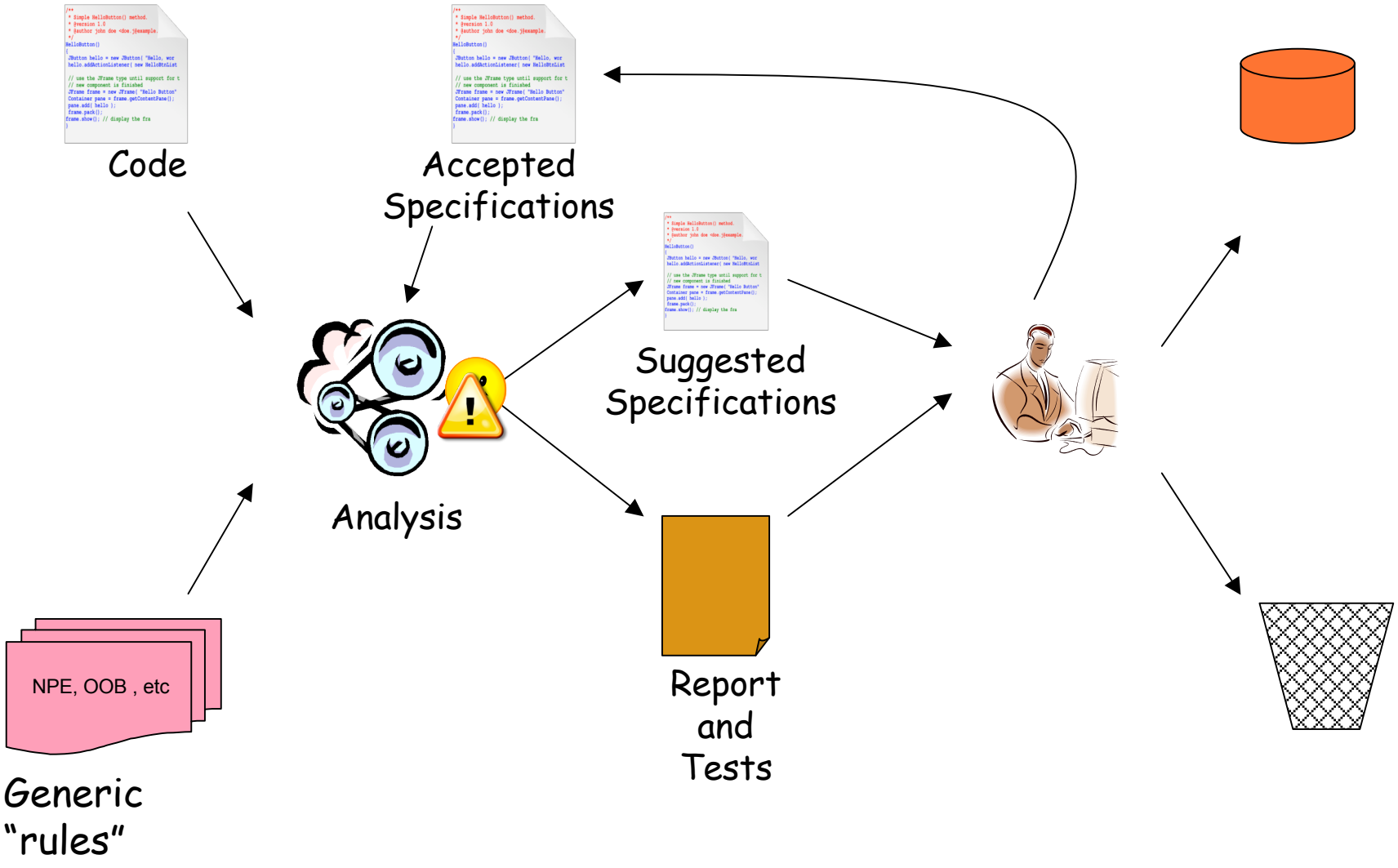




NPE, OOB , etc

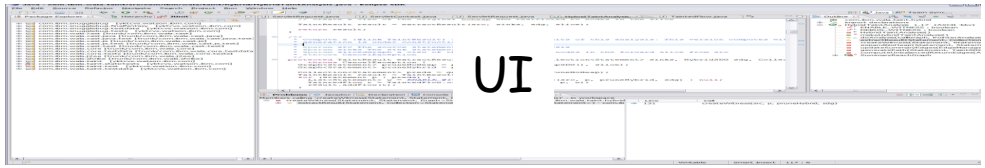
Generic
"rules"

Snugglebug workflow:



DEMO?

Technology Overview



Candidate identification

Identify program states (goals) we would like to reach

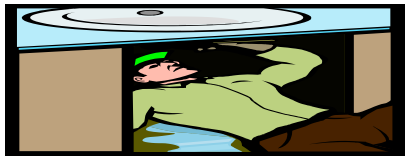
Witness Generation

Generate a unit test that reaches a goal state

Specification Acquisition

Acquire some formal specification of kosher and trief (non-kosher) program states

Program Analysis

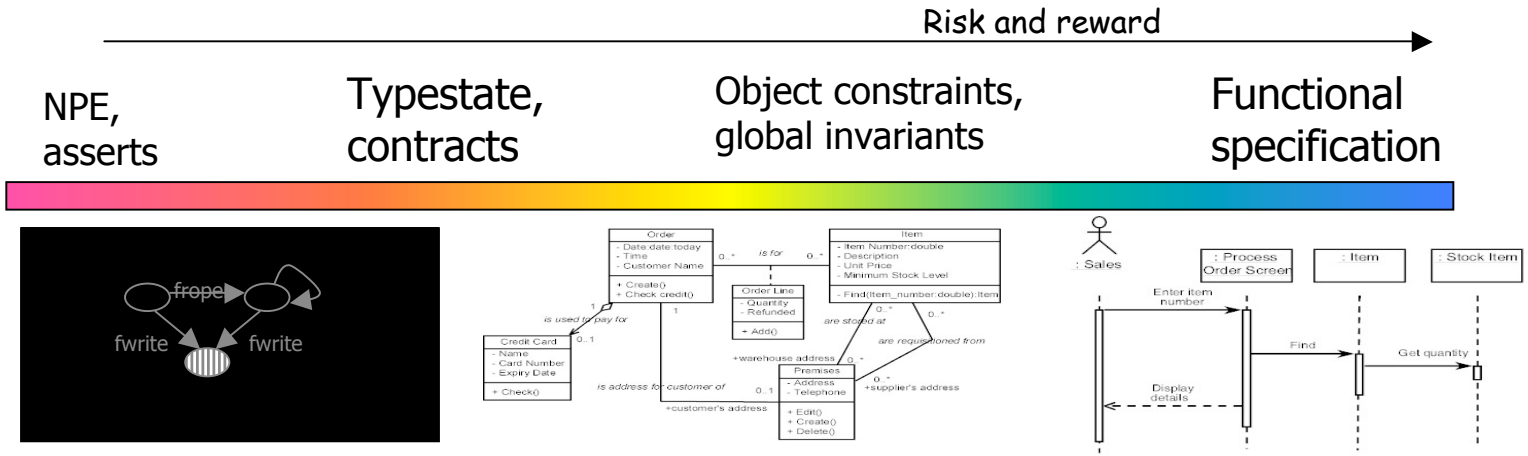


What are the risks?

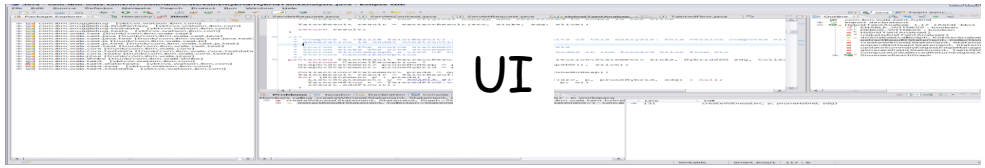
Analysis Technology Inadequate

Concrete test case generation, respecting public APIs, over huge code bases, testing non-trivial properties

Can we really learn powerful specs? Can we express them in ways that a human will relate to?



Analysis Technology

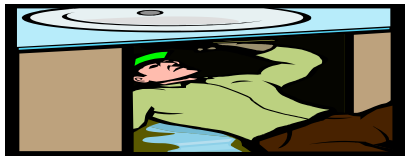


Candidate identification

Witness Generation

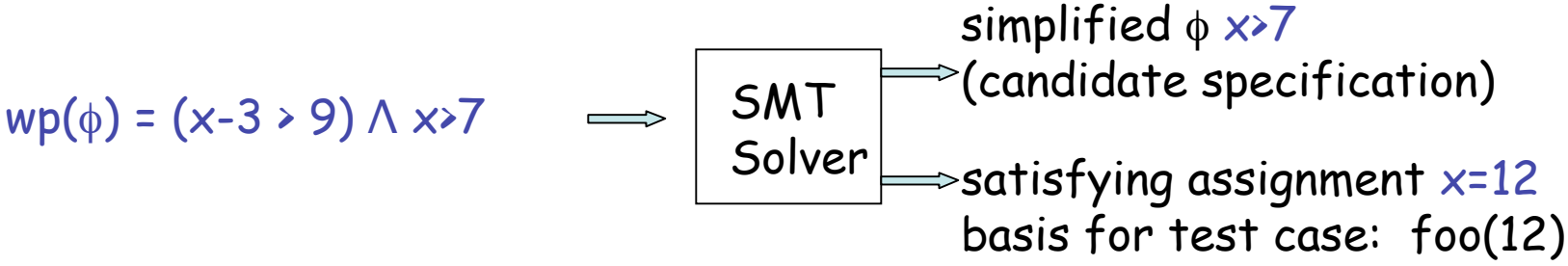
Specification Acquisition

Program Analysis



Symbolic Search Via Weakest Precondition

Symbolic Search via Weakest Precondition (Intro)




```
void foo (int x) {
                    ϕ= wp(ϕ) = (x-3 > 9) ∧ x>7
  if (x > 7) {
                    ϕ:= wp(ϕ)= ϕ[x-3|y] = (x-3 > 9)
    int y = x -3;
                    ϕ:= wp(ϕ) = (y > 9)
    if (y > 9) {
                    ϕ:= true
      BOOM;
    }
}
```

IPA WP Via (Partial) Tabulation

Reps-Horwitz-Sagiv POPL 95 Tabulation Solver (WALA)

- explore all paths at once, IPA with underapproximate abstraction

```
                    φ:= 1 > 3 ∧ 1 ≤ 2
x = min(1,2);
                    φ:= x > 3
Y = min(x,3);
                    φ:= x > 3
z = min(x,4);
                    φ:= wp(φ) = (z > 3)
if (z > 3)
   BOOM;
                    φ:= true
```

```
                                                    φ:= T
int min(a, b) {
φ1:= a > 3 ∧ a ≤ b                  φ2:= b > 3 ∧ a > b
   if (a <= b)
      r = a;                        φ:= a > 3
   else
      r = b;                        φ:= b > 3
   return r;
}                                   φ:= r > 3
                                                    φ:= T
```

φ	Wp(min,φ)
r > 3	a > 3 ∧ a ≤ b
r > 3	b > 3 ∧ a > b
T	T

Effective Modular Analysis?

Tabulation is fully automatic

Maintain (large?) database of partial transfer functions

Precompute partial predicate transformers for standard libraries

- WP(true), WP(throws an exception)
- WP(other common conditions?)

Key issue: Separation. What is the frame condition?

"logical mod/ref"

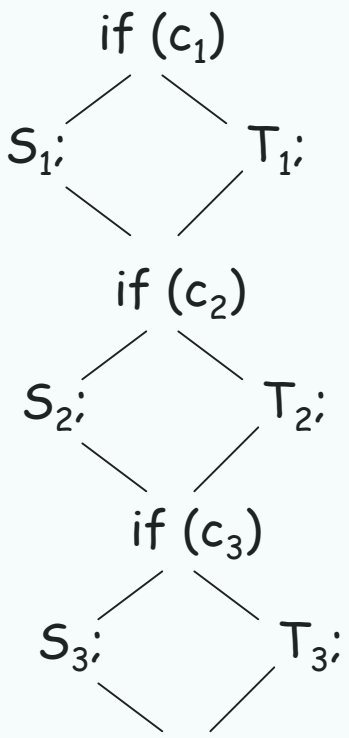
abstract interpretation

Open question: degree of reuse?

Dealing with exponential explosion

(Without even worrying about loops …)

Paths



Substitution run amuck (FS POPL02)

```
x = …
y = x + x
z = y + y
w = z + z
v = w + w
```

Dynamic Dispatch

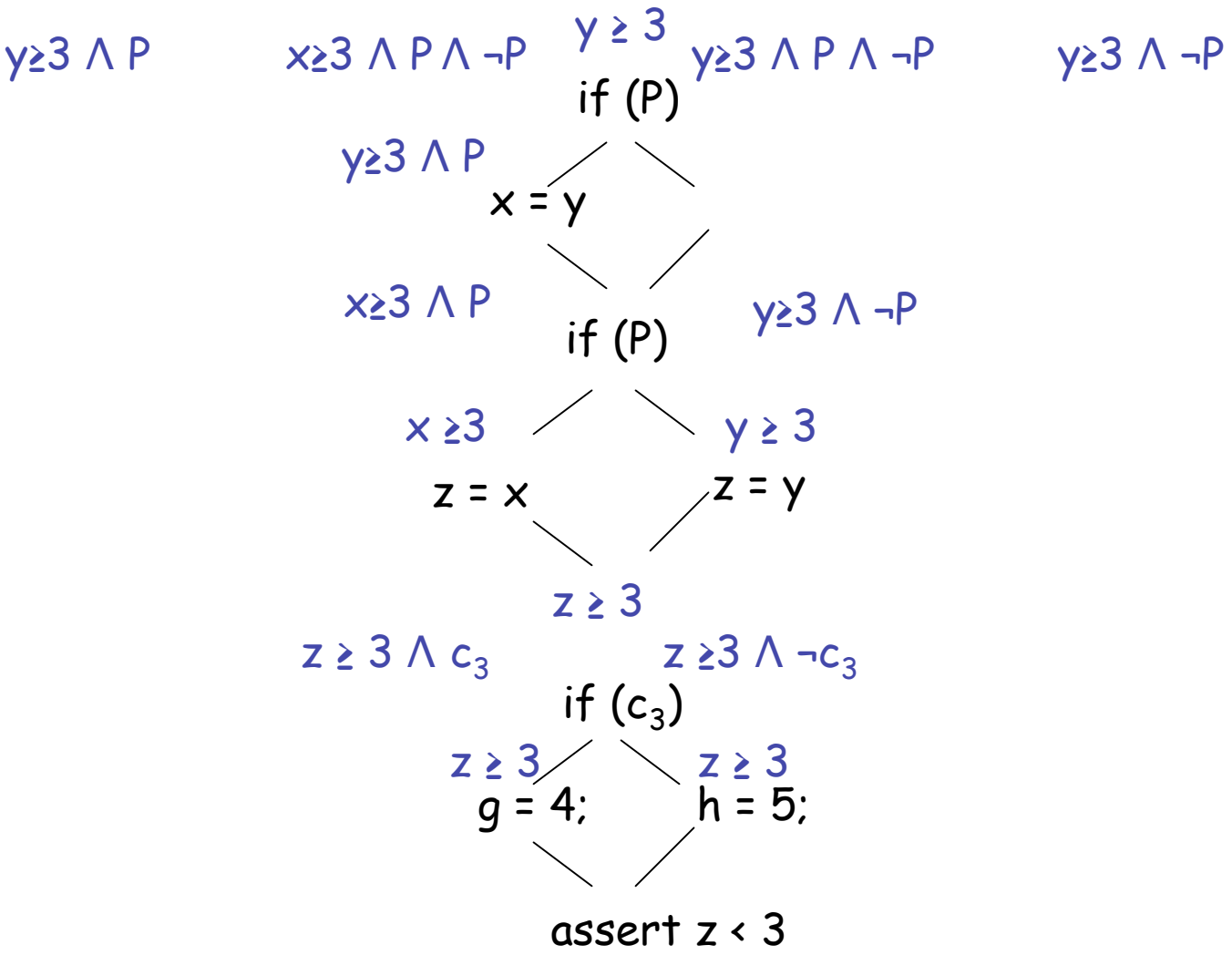
```
s = x.toString();
s += y.toString();
s += z.toString();
s += w.toString();
```

Aliasing and Destructive Updates

```
y.f = x;
z.f = y;
w.g = z;
```

Dealing with exponential explosion

Merge Functions & Search Heuristics



Generating API-conformant test cases.

wp(ϕ) = (b.f == 1)

```
static void foo(Bar b) {
   if (b.getF() == 1) {
      BOOM;
   }
}
class Bar {
   private int f; // f == 0 or 2
   public int getF() { return f; }
   private Bar(int f) {
      this.f = f;
   }
   public static Bar make0() {
      return new Bar(0);
   }
   public static Bar make2() {
      return new Bar(2);
   }
}
```

Solution: Universal Driver

Encodes all reasonable ways of driving the method under test.

Parameterized in a way to facilitate search by an SMT solver.

Partial evaluation of universal driver w.r.t. a satisfying assignment gives a unit test.

Generating API-conformant test cases.

wp(ϕ) = (b.f == 1)

```
static void foo(Bar b) {
    if (b.getF() == 1) {
       BOOM;
    }
}
class Bar {
    private int f; // f == 0 or 2
    public int getF() { return f; }
    private Bar(int f) {
       this.f = f;
    }
    public static Bar make0() {
       return new Bar(0);
    }
    public static Bar make2() {
       return new Bar(2);
    }
}
```

Universal Driver

```
public static void driveFoo(int[] x) {
   int length = x[0];
   int[] y = x[1 : length];
   Bar b = makeBar(y);
   foo(b);
}


public static Bar makeBar(int[] y) {
   switch(y[0]) {
      case 0: return Bar.make0();
      case 1: return Bar.make2();
   }
}
```

SMT: no satisfying assignment for driveFoo().

Generating API-conformant test cases.

wp(ϕ) = (b.f == 1)

```
static void foo(Bar b) {
    if (b.getF() == 1) {
       BOOM;
    }
}
```

```
class Bar {
   private int f;
   public int getF() { return f; }
   private Bar(int f) {
      this.f = f;
   }
   public static Bar make0() {
      return new Bar(0);
   }
   public static Bar make2() {
      return new Bar(2);
   }
   public static Bar make(int y) {
      return new Bar(y);
   }
}
```

Universal Driver

```
public static void driveFoo(int[] x) {
   int length = x[0];
   int[] y = x[1 … length];
   Bar b = makeBar(y);
   foo(b);
}

public static Bar makeBar(int[] y) {
   switch(y[0]) {
      case 0: return Bar.make0();
      case 1: return Bar.make2();
      case 2: return Bar.make(y[1]);
   }
}
```

SMT: satisfying assignment for driveFoo(): [2, 2, 1]

Generating API-conformant test cases.

Partially evaluate driveFoo()
w.r.t. [2, 2, 1]:

```
public void testFoo() {
   Bar b = Bar.make(1);
   foo(b);
}
```

Universal Driver

```
public static void driveFoo(int[] x) {
   int length = x[0];
   int[] y = x[1 … length];
   Bar b = makeBar(y);
   foo(b);
}

public static Bar makeBar(int[] y) {
   switch(y[0]) {
      case 0: return Bar.make0();
      case 1: return Bar.make2();
      case 2: return Bar.make(y[1]);
   }
}
```

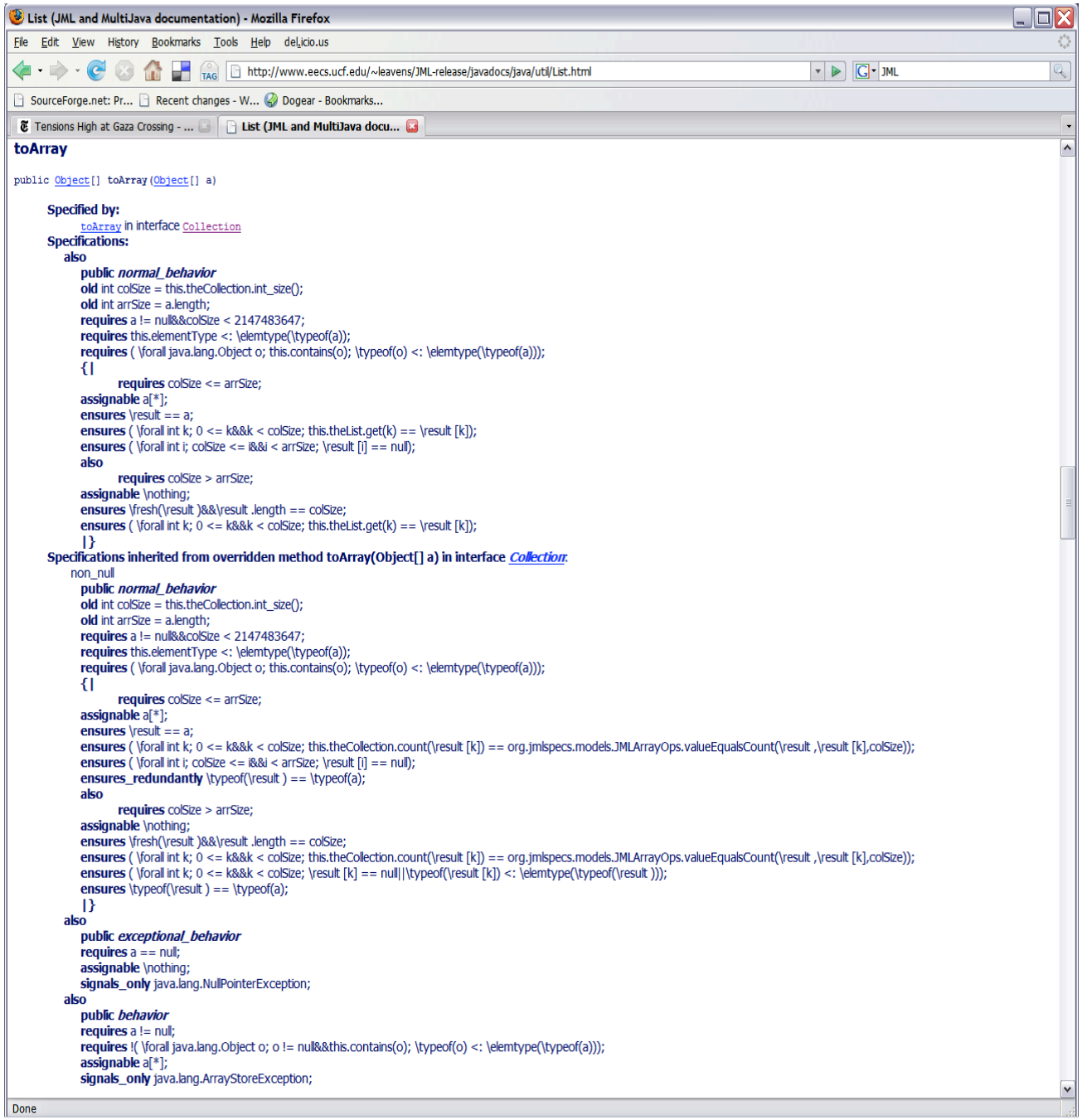
SMT: satisfying assignment for driveFoo(): [2, 2, 1]

Other technologies of interest

- Abstraction to guide search, skip loops/recursion
- Speculation and dynamic checking
- From WP to specifications
  - Requires effective formulae simplification, not just satisfying assignments
  - "lifting" predicates from points to larger scopes (e.g. invariants)
- Lots of ways to improve specification acquisition
  - Tests as specifications
  - Mining client codes for example specifications
  - Mining the web for specifications
  - Other stuff to be invented


Milestone n: Total world domination. Retire to Tahiti.
Milestone 2: Somebody else judges the snugglebug tool useful enough us to adopt it.
Milestone 1: We judge the snugglebug tool useful enough for us to adopt it into our own daily development.

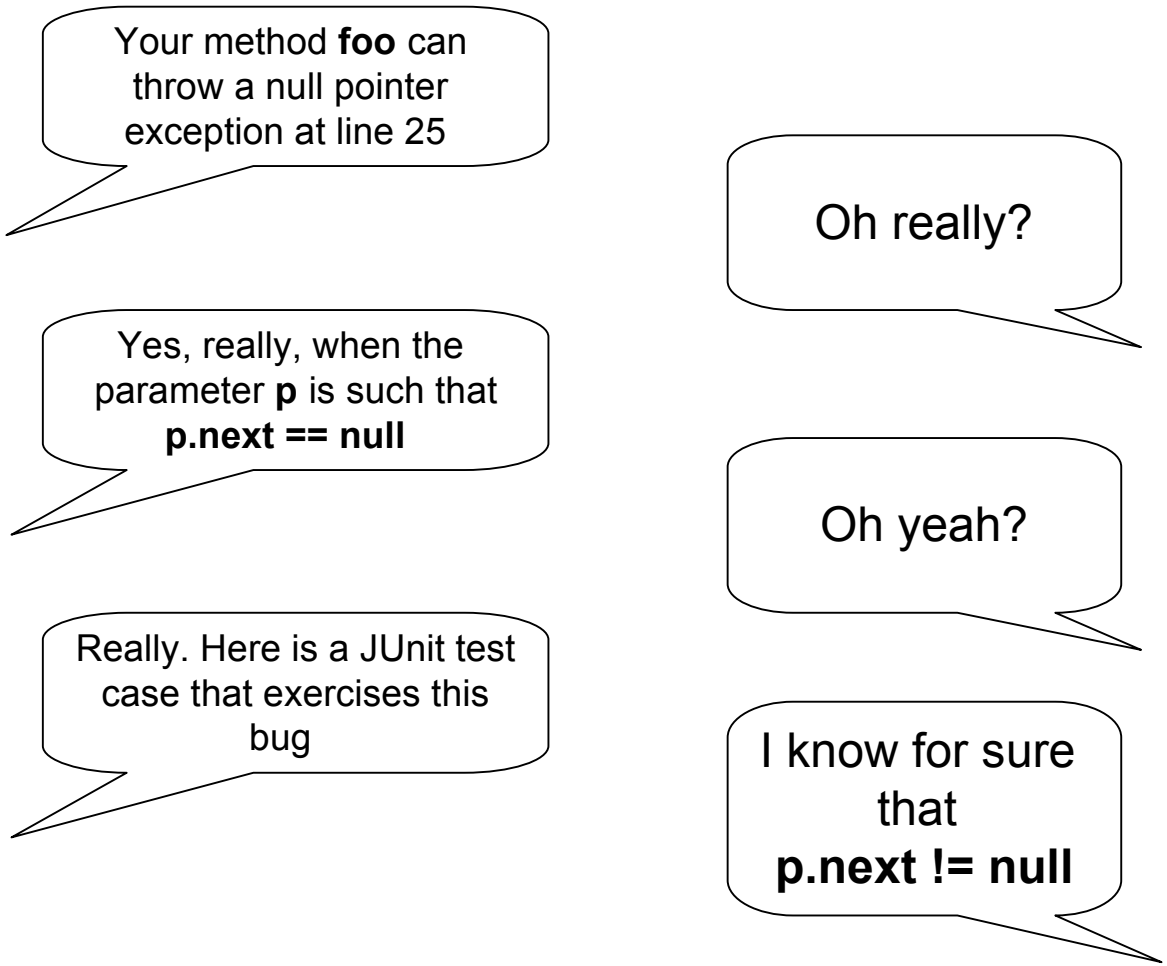
BACKUP SLIDES


List (JML and MultiJava documentation) - Mozilla Firefox
File Edit View History Bookmarks Tools Help del.icio.us
http://www.eecs.ucf.edu/~leavens/JML-release/javadocs/java/util/List.html
JML
SourceForge.net: Pr... Recent changes - W... Dogear - Bookmarks...
Tensions High at Gaza Crossing - ...
List (JML and MultiJava docu...
toArray
public Object[] toArray(Object[] a)
Specified by:
toArray in interface Collection
Specifications:
also
public normal_behavior
old int colSize = this.theCollection.int_size();
old int arrSize = a.length;
requires a != null&&colSize < 2147483647;
requires this.elementType <: \elemtype(\typeof(a));
requires (\forall java.lang.Object o; this.contains(o); \typeof(o) <: \elemtype(\typeof(a)));
{|
requires colSize <= arrSize;
assignable a[*];
ensures \result == a;
ensures (\forall int k; 0 <= k&&k < colSize; this.theList.get(k) == \result [k]);
ensures (\forall int i; colSize <= i&&i < arrSize; \result [i] == null);
also
requires colSize > arrSize;
assignable \nothing;
ensures \fresh(\result)&&\result .length == colSize;
ensures (\forall int k; 0 <= k&&k < colSize; this.theList.get(k) == \result [k]);
|}
Specifications inherited from overridden method toArray(Object[] a) in interface Collection:
non_null
public normal_behavior
old int colSize = this.theCollection.int_size();
old int arrSize = a.length;
requires a != null&&colSize < 2147483647;
requires this.elementType <: \elemtype(\typeof(a));
requires (\forall java.lang.Object o; this.contains(o); \typeof(o) <: \elemtype(\typeof(a)));
{|
requires colSize <= arrSize;
assignable a[*];
ensures \result == a;
ensures (\forall int k; 0 <= k&&k < colSize; this.theCollection.count(\result [k]) == org.jmlspecs.models.JMLArrayOps.valueEqualsCount(\result ,\result [k],colSize));
ensures (\forall int i; colSize <= i&&i < arrSize; \result [i] == null);
ensures_redundantly \typeof(\result) == \typeof(a);
also
requires colSize > arrSize;
assignable \nothing;
ensures \fresh(\result)&&\result .length == colSize;
ensures (\forall int k; 0 <= k&&k < colSize; this.theCollection.count(\result [k]) == org.jmlspecs.models.JMLArrayOps.valueEqualsCount(\result ,\result [k],colSize));
ensures (\forall int k; 0 <= k&&k < colSize; \result [k] == null||\typeof(\result [k]) <: \elemtype(\typeof(\result)));
ensures \typeof(\result) == \typeof(a);
|}
also
public exceptional_behavior
requires a == null;
assignable \nothing;
signals_only java.lang.NullPointerException;
also
public behavior
requires a != null;
requires !(\forall java.lang.Object o; o != null&&this.contains(o); \typeof(o) <: \elemtype(\typeof(a)));
assignable a[*];
signals_only java.lang.ArrayStoreException;
Done

What's New?

Everyone wants a piece of the pie … and "Finding Bugs is Easy" …

Typical Interaction between Analysis Tools and Developers



Have we changed the world yet?

Maturity is a bitter disappointment for which no remedy exists, unless laughter can be said to remedy anything.
- Vonnegut