Software development problem

- Software construction and maintenance are expensive
- Reliability is costly and difficult to achieve

Vision

- Increased programmer productivity and program reliability through increased rigor

  Record design decisions
  + Utilize automatic checking
  = Detect errors and improve maintainability

User's view

Extended Static Checker for Java (ESC/Java)

- Built at Compaq SRC
- Input: Java + user-supplied annotations
- Annotation language captures programmer design decisions
- Powered by program semantics and automatic theorem proving
- Performs modular checking

public class Bag {
private int[] a;
private int n;
//@ invariant 0 <= n && n <= a.length;
public Bag(int[] initialElements) {
    n = initialElements.length;
a = new int[n];
    System.arraycopy(initialElements, 0, a, 0, n);
}
public void add(int x) {
    if (n == a.length) {
        int[] b = new int[2*(a.length+1)];
        System.arraycopy(a, 0, b, 0, n);
a = b;
    }
a[n] = x;
n++;
}
public int extractMin() {
    int m = Integer.MAX_VALUE;
    int mindex = 0;
    for (int i = 0; i < n; i++) {
        if (a[i] < m) {
            mindex = i;
m = a[i];
        }
    }
    if (0 < n) {
        n--;
a[mindex] = a[n];
    }
    return m;
}
**Program checker design tradeoffs**
- Missed errors
- Spurious warnings
- Annotation overhead
- Performance

**Tool architecture**
- Annotated Java program
  - Translator
  - Verification condition
  - Automatic theorem prover
  - Counterexample context
  - Post processor
  - Warning messages

**Tool architecture, detail**
- Annotated Java program
  - Sugared command
  - Primitive command
  - Passive command
  - Verification condition
  - Automatic theorem prover
  - Counterexample context
  - Post processor
  - Warning messages

**Annotation language**
- Simple
  - non_null
- Method annotations
  - requires E;
  - modifies w;
  - ensures E;
  - ensures (T x) E;
- Object invariants
  - invariant E;

**Annotation language**
- Simple
  - non_null
- Method annotations
  - requires E;
  - modifies w;
  - ensures E;
  - ensures (T x) E;
- Object invariants
  - invariant E;
**Annotated Java**

- **Counterexample**
  - Specification expressions
    - side-effect free Java expressions
    - no ++, no method calls
    - result, old(E)
      - ensures result == old(x);
    - =>
    - (forall T x; E), (exists T x; E)
      - (forall int j; 0 <= j && j < n ==> a[j] > 0);
    - type(E), type(T), <;
      - requires type(x) == type(this);

- **Post processor**
  - Verification
    - messages
    - Automatic command
      - Warning
      - Primitive
        - program
      - Sugared context
    - Passive
      - Warning
      - Primitive
        - program
      - Sugared context
    - Translator
      - Warning
      - Primitive
        - program
      - Sugared context
  - Verification
    - messages
    - Automatic command
      - Warning
      - Primitive
        - program
      - Sugared context
    - Passive
      - Warning
      - Primitive
        - program
      - Sugared context
    - Translator
      - Warning
      - Primitive
        - program
      - Sugared context

- **Translator**
  - Warning
    - Primitive
      - program
    - Sugared context
  - Passive
    - Warning
      - Primitive
        - program
      - Sugared context
  - Translator
    - Warning
      - Primitive
        - program
      - Sugared context

- **Annotations**
  - Concurrency
    - monitored_by lock
      - monitored by lock */ long x;
        - lockset[lock]
          - requires lockset[this];
        - lock0 < lock1
          - max(lockset)
            - requires max(lockset) < this;
  - Ghost variables
    - ghost public T x;
      - ghost public int objectState;
      - ghost public TYPE elementType;
      - set x = E;
        - set objectState = Open;
        - set elementType = type(T);
Annotation language

- Miscellaneous
  - assert E;
  - assume E:
    - assume x > 0; // because x = y * y
    - nowarn
    - x = a[]; // @ nowarn
  - axiom E;
    - axiom (forall int x: x ≥ 2 ≥ 0);

Sugared commands

- S, T ::= assert E
  - assert E
  - assume E
  - x = E
  - raise
  - S ; T
  - S ! T
  - S [] T
  - loop [inv E] S ⇒ T end
  - call x = t.m(E)

Primitive commands

- // @ requires Pre; modifies w; ensures Post:
  - X m(U u);
  - call x = t.m(E)
    - var u in
    - u = E;
    - assert Pre;
    - var w in
    - w0 = w;
    - havoc w;
    - assume Post;
    - x = result
    - end
    - end
Passive commands

- \( S, T ::= \) assert \( E \)
- \( S = E \)
- \( x = E \)
- \( x \) = \( E \)
- \( T \)
- \( S ; T \)
- \( S ! T \)
- \( S \ll T \)

Passive commands

- if \((x < 0)\) \{ \( x = -x; \) \}
- \/*@ assert x >= 0; */
  
  \{ \ass x_0 < 0; \ass x_1 = -x_0; \ass x_2 = x_1
  \}\ass \neg(x_0 < 0);
  \ass x_2 = x_0
  
  assert \( x_3 >= 0 \)

Weakest preconditions

- wp(\ass E, Q) = E \& \& Q
- wp(\ass E, Q) = E \Rightarrow Q
- wp(S;T, Q) = wp(S, wp(T,Q))
- wp(S \ll T, Q) = wp(S, Q) \& \& wp(T, Q)
- wp(S, Q) = wp(S, true) \& \& wp(S, Q)
- wp(S, Q) = wp(S, false) \& \& Q

Verification condition

- Universal background predicate
  - \((\forall t \cdot t < t)\)
- Type-specific background predicate
  - Bag <: java.lang.Object
- Verification condition:
  - BP_univ \& BP_T \Rightarrow VC_method
Theorem prover: “Simplify”
- Nelson-Oppen cooperating decision procedures
- congruence closure
- linear arithmetic
- partial orders
- quantifiers
- Key features:
  - automatic: no user interaction
  - refutation based: searches for counterexamples
  - heuristics tuned for program checking
  - labels
  - time limit

Experience: annotations
- Capture common design decisions
- Suggested immediately by warnings
- Overhead: 4-10% of source code
- ~1 annotation per field or parameter
- Most common annotations:
  - non_null
  - container element types

Experience: performance
- 50% of all methods: < 0.5 s
- 80% of all methods: < 1 s
- time limit: 300 s
- total time for Javafe (∼40kloc): 65 min.

Related work
- ESC/Modula-3
- Full functional specification and verification
  - JML, LOOP, B, Penelope, ...
- Languages and language features
  - Euclid, Eiffel, Escher, Guava, Vault, Cqual, ...
  - LCLint, refinement types, Types against races, ...
- Other checking techniques
  - Abstract interpretation, PREfix, SLAM, Bandera,
  Java PathFinder 2, Canvas, ESP, AST Toolkit, Metal,
Conclusions

- Using program semantics and automatic decision procedures for program analysis works!
- Cost effective?