

# Extended Static Checking

---

Das ESC/Java Projekt eine Übersicht  
*24.1.2002 Jens Wilke*

TU-München, Lehrstuhl IV  
Prof. Dr. Manfred Broy  
Prof. Tobias Nipkow, Ph.D.

Betreuer: Martin Strecker



# Vortragsgliederung

---

- Übersicht und Projektziele
- Komponenten
- Prüfungsmöglichkeiten
- Ein einfaches Beispiel
- komplexeres Beispiel / Experiment
- Zusammenfassung

*JW*

# Das ESC/Java Projekt

---

- ESC = „Extended Static Checking“
- mehr als statische Typprüfung
- typische Programmierfehler entlarven
- Keine vollständige Programm Verification



Handwritten signature

# Ziele des Projektes

---

- Erforschung unterschiedlicher Entwicklungs-  
"Trade-Offs":
  - Anzahl nicht gefundener Fehler
  - Anzahl von falschen Fehlermeldungen
  - Einarbeitungsaufwand
  - Aufwand für Programm Annotation
  - Laufzeit des Prüfprogramms
  - Entwicklungsaufwand für das Prüfprogramm



Handwritten signature

## Ziele des Projektes (2)

---

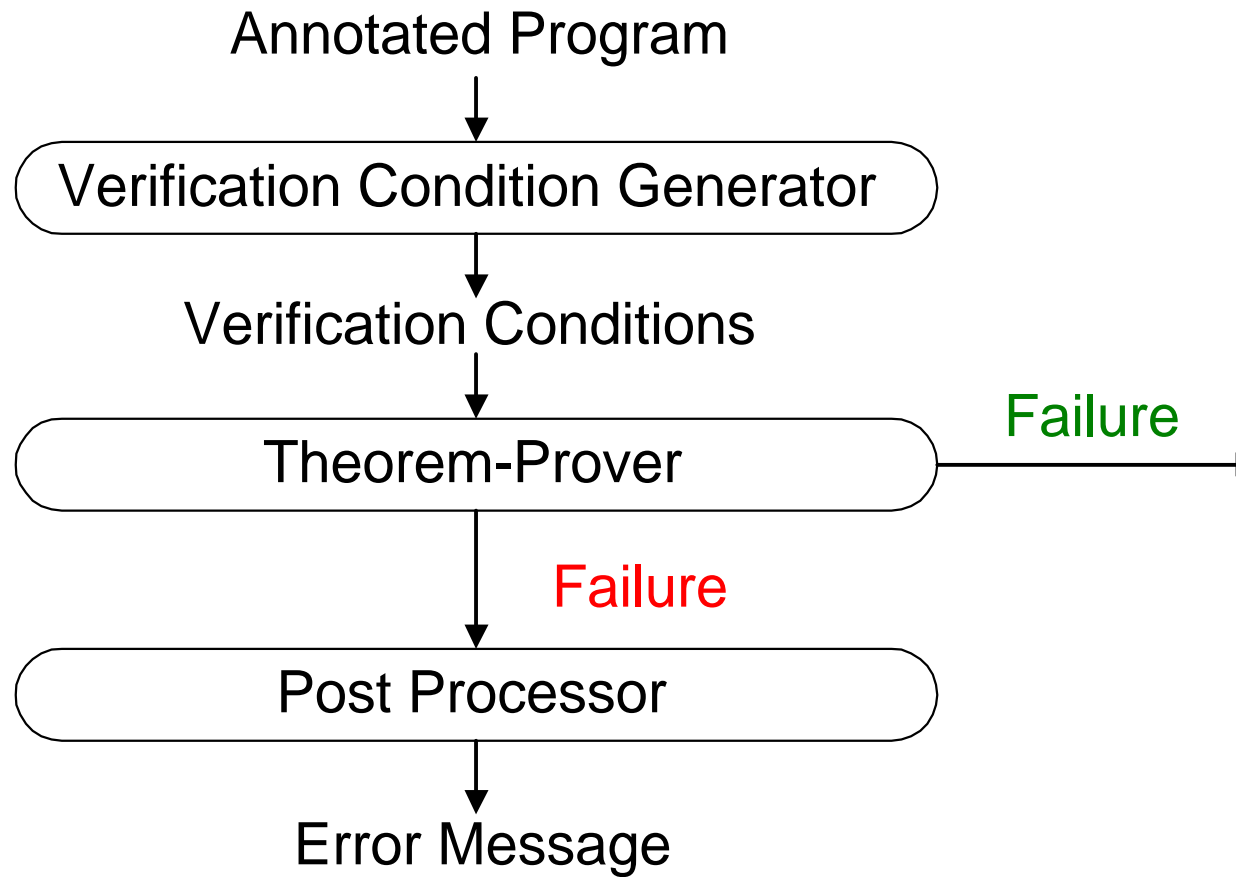
- Modulare Prüfung
- Source muß für notwendige Programmbibliotheken nicht vorhanden sein
- Unterstützung von Java 1.2
- Benutzung wie Java-Compiler
- Fehlermeldungen ähnlich wie bei einem Java-Compiler



*JW*

# Komponenten

---



*JW*

# Annotierter Java-Source

- Programm wird mit sog. „Pragmas“ versehen
- Annotationen kompatibel zu Java Modeling Language (JML)

```
class Bag {  
    /*@ non_null */ int[] a;  
    int n;  
    /*@ invariant 0 <= n && n <= a.length;  
  
    /*@ requires input != null;  
    Bag(int[] input) {  
        n = input.length;  
  
    . . .
```



# Der Theorem Beweiser (Simplify)

---

- entwickelt um Beweise ohne manuelle Hilfe durchzuführen
- Zusätzliche Routinen für Funktionen und Prädikate die für die Programmierung wichtig sind (Gleichheit, Arithmetik)
- Generierung von nützlichen Fehlermeldungen
- Lücken:
  - Keine eingebaute Semantik für Multiplikation
  - Keine Unterstützung für mathematische Induktion



# Gefundene Fehler: Runtime Exceptions

---

- Exceptions

- ArrayOutOfBounds

- NullPointer

- ClassCast

- DivisionByZero

- Mögliches Auftreten dieser Exceptions

- => Warnmeldung



# Weitere Prüfungsmöglichkeiten

---

- Im Methodenrumpf
  - Assertions z.B. `//@ assert n>2`
  - Unreachable Pragma
  - Schleifeninvarianten
- Objekt-Invarianten
- Vorbedingungen für Methodenparameter
- Tests für korrekte Synchronisation



# Beispiel: Bag.java

## ■ Programm mit potenziellen Fehlern

```
class Bag {
    int[] a;
    int n;

    Bag(int[] input) {
        n = input.length;
        a = input;
    }

    int extractMin() {
        int m = Integer.MAX_VALUE;
        int mindex = 0;
        for (int i = 1; i <= n; i++) {
            if (a[i] < m) {
                mindex = i;
                m = a[i];
            }
        }
        n--;
        a[mindex] = a[n];
        return m;
    }
}
```



# Fehlerausgabe

## Ausgabe von:

```
escjava -quiet -notrace Bag.java
```

```
Bag.java:6: Warning: Possible null dereference (Null)
    n = input.length;
           ^
Bag.java:15: Warning: Possible null dereference (Null)
    if (a[i] < m) {
           ^
Bag.java:15: Warning: Array index possibly too large (IndexTooBig)
    if (a[i] < m) {
           ^
Bag.java:21: Warning: Possible null dereference (Null)
    a[mindex] = a[n];
           ^
Bag.java:21: Warning: Possible negative array index (IndexNegative)
    a[mindex] = a[n];
           ^

5 warnings
```



# Beseitigung der NullPointerException Fehler

- `/*@non_null*/`

Feld darf nie null sein

- `//@requires ...;`

Vorbedingungen für den Methodenaufruf

```
class Bag {
    /*@non_null*/ int[] a;
    int n;

    //@ requires input!=null;
    Bag(int[] input) {
        n = input.length;
        a = input;
    }
}
```



# Beseitigung der Index Fehler

■ **//@invariant ...;**

Gibt Object-Invariante an

Hier: stellt Beziehung zw. *n* und *a.length* her

```
//@invariant 0 <= n & n <= a.length;  
  
int extractMin() throws Exception {  
    if (n<=0) {  
        throw new Exception("Nix mehr drin!");  
    }  
    int m = Integer.MAX_VALUE;  
    int mindex = 0;  
    for (int i = 0; i < n; i++) {  
        if (a[i] < m) {  
            mindex = i;  
        }  
    }  
}
```

. . .



# Experimentalteil



■ ...

A handwritten signature in black ink, located in the bottom right corner of the slide. The signature appears to be 'JW'.

# Zusammenfassung

---

## ■ Gut:

- Programmierer werden diszipliniert ausführlicher (logischer?) zu dokumentieren
- Flüchtigkeitsfehler bzw. Fehler von unerfahrenen Programmierern werden automatisch erkannt
- Zusätzlicher Arbeitsaufwand hält sich in Grenzen

## ■ Aber:

- Kein Ersatz für richtige Funktionstests
- Kein Ersatz für vollständige Verification, falls diese erforderlich ist



Handwritten signature