Verifikation von Programmen in Java 5

ObjektForum Karlsruhe

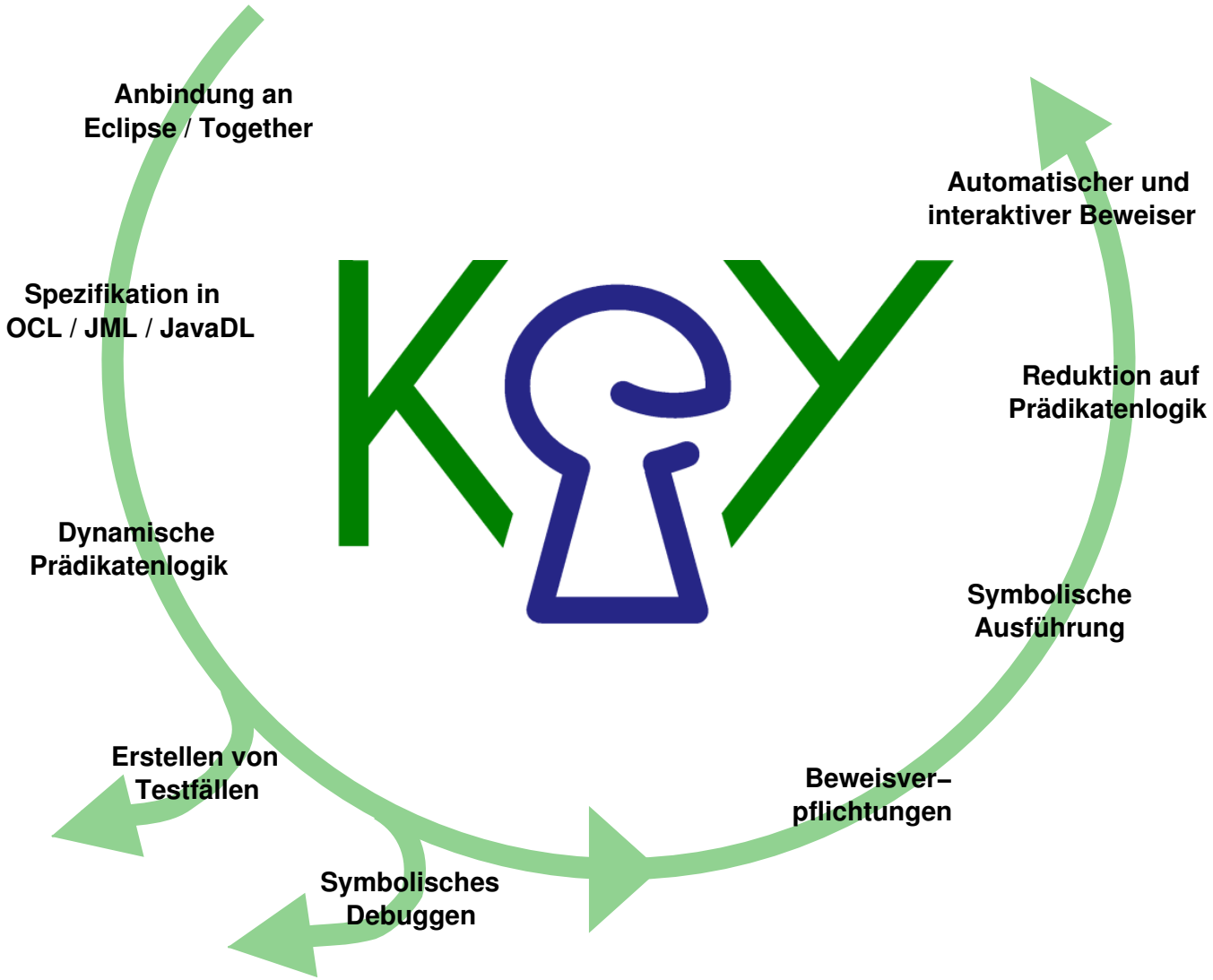
Mattias Ulbrich
Betreuung: Prof. Dr. P. H. Schmitt

Institut für Theoretische Informatik
Universität Karlsruhe (TH)

8. Oktober 2007

Überblick

Formale Verifikation von JavaCard-Programmen



Ein Beispiel

Design by Contract

```
class Account {
  int balance;

  /*@ requires amount >= 0;
    @ ensures balance == \old(balance)-amount;
    @*/
  void debit(int amount) {
     balance = balance - amount;
  }
}
```

Ein Beispiel

Design by Contract

```
class Account {
  int balance;

  /*@ requires amount >= 0;
    @ ensures balance == \old(balance)-amount;
    @*/
  void debit(int amount) {
     balance = balance - amount;
  }
}
```

Daraus entsteht die Beweisverpflichtung

$amount \geq 0 \rightarrow \langle \texttt{debit(amount);} \rangle (balance = balance_{old} - amount)$

Warum  +  ?

1. Schritt halten mit der industriellen Entwicklung
2. Wie **flexibel** und **anpassungsfähig** ist KeY?
3. **Unterstützen** die neuen Sprachelemente die Verifikation?
4. **Benötigen** sie Verifikation?

Spracherweiterungen in Java 5

- Typsichere Aufzählungstypen
- Iterationsschleifen
- Automatische Typ-Wandlung (Auto-Boxing)
- Generische Klassen
- Kovariante Ergebnistypen
- Annotationen
- Statische “Import”-Direktiven
- Variable Argumentenzahl

Spracherweiterungen in Java 5

- Typsichere Aufzählungstypen
- Iterationsschleifen
- Automatische Typ-Wandlung (Auto-Boxing)
- Generische Klassen

- ~~Kovariante Ergebnistypen~~
- ~~Annotationen~~
- ~~Statische "Import"-Direktiven~~
- ~~Variable Argumentenzahl~~

Irrelevant für die Verifikation

Spracherweiterungen in Java 5

- Typsichere Aufzählungstypen
- Iterationsschleifen
- Automatische Typ-Wandlung (Auto-Boxing)
- Generische Klassen

- Kovariante Ergebnistypen
- Annotationen
- Statische "Import"-Direktiven
- Variable Argumentenzahl

Irrelevant für die Verifikation

Spracherweiterungen in Java 5

- Typsichere Aufzählungstypen
- Iterationsschleifen
- Automatische Typ-Wandlung (Auto-Boxing)
- Generische Klassen

- ~~Kovariante Ergebnistypen~~
- ~~Annotationen~~
- ~~Statische "Import"-Direktiven~~
- ~~Variable Argumentenzahl~~

Irrelevant für die Verifikation

Generische Klassen – *wie sie sein sollten*

Java ohne Generika

```
Stack st = new Stack();

st.push("String");
String s = (String)st.pop();
```

- Allgemeinster Typ wird verwendet
- Typüberprüfung erst zur Laufzeit
- Typwandel-Ausnahmen möglich!

Java 5

```
Stack<String> st =
    new Stack<String>();
st.push("String");
String s = st.pop();
```

- Konkreter Typ als Parameter festgelegt
- Typüberprüfung bereits durch den Übersetzer
- Keine Typwandel-Ausnahme möglich

Generische Klassen – *wie sie wirklich sind*

“Type Erasure”

Typ-Parameter werden vom Übersetzer verworfen.
Zur Laufzeit gibt es nur unparametrisierte Klassen (*raw types*).

```
Stack<String> stack1 = new Stack<String>();
stack1.push("String");

Object object = stack1;
Stack<Integer> stack2 = (Stack<Integer>)object;

Integer i = stack2.pop();
```

throws ClassCastException

Generische Klassen – *wie sie wirklich sind*

Generische Klassen können den „Speicher verschmutzen" (*heap pollution*).

Generische Klassen – *wie sie wirklich sind*

Generische Klassen können den „Speicher verschmutzen" (*heap pollution*).

Abhilfe

- Formal beweisen, dass keine Speicherverschmutzung vorliegt.
- KeY mit parametrisierter Typhierarchie kann das
- KeY benutzen, um Generika **wirklich typsicher** zu machen.

Generische Klassen – *wie sie wirklich sind*

Generische Klassen können den „Speicher verschmutzen" (*heap pollution*).

Abhilfe

- Formal beweisen, dass keine Speicherverschmutzung vorliegt.
- KeY mit parametrisierter Typhierarchie kann das
- KeY benutzen, um Generika **wirklich typsicher** zu machen.

Herausforderung: Drastische Änderungen der Logik

- Unendliche Typhierarchie
- Typen als Objekte in der Logik
- Typvariablen als Typen
- Existenzielle und universelle Typen

Zusammenfassung

Zur Erinnerung: Untersuchungsziele

1. Wie unterstützen die Neuheiten die Verifikation,
2. wie benötigen sie Verifikation und
3. wie flexibel und anpassungsfähig ist KeY als System.

Zusammenfassung

Zur Erinnerung: Untersuchungsziele

1. Wie unterstützen die Neuheiten die Verifikation,
2. wie benötigen sie Verifikation und
3. wie flexibel und anpassungsfähig ist KeY als System.

Feature	braucht Verif.	unterstützt Verif.	„Passt"
Aufzählungstypen	JA	JA	JA
Neue Schleifen			
Auto-Boxing			
Generika			

Zusammenfassung

Zur Erinnerung: Untersuchungsziele

1. Wie unterstützen die Neuheiten die Verifikation,
2. wie benötigen sie Verifikation und
3. wie flexibel und anpassungsfähig ist KeY als System.

Feature	braucht Verif.	unterstützt Verif.	„Passt"
Aufzählungstypen	JA	JA	JA
Neue Schleifen	JA	JA	JA
Auto-Boxing			
Generika			

Zusammenfassung

Zur Erinnerung: Untersuchungsziele

1. Wie unterstützen die Neuheiten die Verifikation,
2. wie benötigen sie Verifikation und
3. wie flexibel und anpassungsfähig ist KeY als System.

Feature	braucht Verif.	unterstützt Verif.	„Passt"
Aufzählungstypen	JA	JA	JA
Neue Schleifen	JA	JA	JA
Auto-Boxing	JA	NEIN	NEIN
Generika			

Zusammenfassung

Zur Erinnerung: Untersuchungsziele

1. Wie unterstützen die Neuheiten die Verifikation,
2. wie benötigen sie Verifikation und
3. wie flexibel und anpassungsfähig ist KeY als System.

Feature	**braucht Verif.**	**unterstützt Verif.**	**„Passt"**
Aufzählungstypen	JA	JA	JA
Neue Schleifen	JA	JA	JA
Auto-Boxing	JA	NEIN	NEIN
Generika	JA	JA	**NEIN**

Vielen Dank für Ihre Aufmerksamkeit!

www.key-project.org