Einführung in die Programmierung
Introduction to Programming

Prof. Dr. Bertrand Meyer
(Nadia Polikarpova)

Verification tools demo
Verification Tools for Eiffel

**AutoTest:**
Automated contract-based testing

**EveProofs:**
Automated proofs
Contract-based testing

Input generation → Test case execution → Result validation

Preconditions

Contracts

Postconditions
Contract-based testing

As long as we know what the software is supposed to do, we do not need any human intervention to test it.

When testing a certain routine:
- We try to satisfy its **precondition** (so that we can execute it).
- We hope it will not fulfill its **postcondition**.

For every test case 3 possible outcomes:
- **Pass** - all ok
- **Fail** - postcondition/invariant violation or other exception
- **Invalid** - precondition violation
Inputs generation

Random input generation:

- **Primitive values**: random selection
  - 25% probability: select from 0, ±1, ±2, ...
  - 75% probability: randomly choose a value

- **Objects**: constructor calls + other (state-changing) methods
  - 20% probability: create a new instance
  - 80% probability: reuse old objects
Random testing strategy

Workflow of the random testing strategy

Select next routine to test

Select objects randomly

Invoke routine

Sample test cases

create \{LINKED\_LIST[\text{INTEGER}]\} v1.make

v2 := 1

v1.extend (v2)

v3 := 125

v1.wipe\_out

v4 := v1.has (v3)

v5 := v1.count

Object pool
Testing vs. proofs

- **Testing** helps us find and correct bugs, but cannot guarantee that there are no more bugs.
- We would like to prove formally that our programs are faultless.
- **Proofs** can give us a guarantee, but they either require human intervention or will sometimes say “I don’t know.”
- Note that both testing and proofs require a specification (contracts).
  - you can ask whether your code is correct with respect to some requirements, not by itself.
EveProofs

- Fully automatic proof tool for Eiffel
- Uses Microsoft Boogie program verifier

**Workflow:**
- Translate Eiffel code and contracts into Boogie programming language
- Run Boogie tool
- Interpret Boogie output in terms of original classes and routines