ASYNCHRONOUS ASSERTIONS
What are assertions?

```java
public class ATM {
    ...
    public void withdraw(Account a, int amount) {
        int oldBalance = a.getBalance();
        a.setBalance(oldBalance - amount);
        assert a.getBalance() < oldBalance;
        dispense(amount);
    }
}
```

Assertions allow programmers to verify that their program is in a certain status
What are assertions?

public class ATM {
    ...
    public void withdraw(Account a, int amount) {
        a.addTransaction(-amount, "ATM Withdrawal");
        assert a.findTransaction(-amount, "ATM Withdrawal") != null;
        dispense(amount);
    }
}

What is the problem here?
What are assertions?

- Assertions are used to verify your assumptions about the program
- Evaluating assertions is expensive – especially if they rely on expensive calculations themselves
- Because of this, you may opt to remove them from your production code
- This leads to some implications…
What are assertions NOT?

public class ATM {
    
    public void withdraw(Account a, int amount) {
        assert amount > 0;
        a.addTransaction(-amount, "ATM Withdrawal");
        dispense(amount);
    }
}

Assertions cannot be used to verify user or method inputs
What are assertions NOT?

```java
public class ATM {
    ...

    public void withdraw(Account a, int amount) {
        assert a.addTransaction(-amount, "ATM Withdrawal") == true;
        dispense(amount);
    }
}
```

Assertions cannot have side effects
Idea

- If assertions are used only for debugging, we do not need the control flow to be halted while we evaluate the assertion.
- After all, we are sure that it is true anyway.
- Why not do it asynchronously?
- Problem: By then, object values have probably changed.
Snapshotting

- If we copy the stack and the heap at the time of the assertion, we can make sure we still have the correct data
- That’s expensive...
- Thus, only copy objects that are really modified
- Copy-on-write
Snapshotting

- Copy-on-write automatically guarantees isolation, preservance of identity, and consistent references

- If many assertions are made, objects are copied more than necessary
Snapshotting

- Every assertion defines its own epoch
- Instead of having only a „modified“ flag, objects are checked whether they were changed in a later epoch
- Only then they have to be copied
Of course, if the epochs match, copies can be shared

Objects created after an assertion’s epoch do not have to be copied
In case of an error...

- The user can decide how to handle assertion errors:
  - Either the program terminates, throwing an AssertionError, or
  - The user can handle the situation by using a handle to the asynchronous evaluation
Discussion

- „Handle into the future“ in violation of the JLS
Evaluation

- Microbenchmarks: Simple data structures, synthetic benchmark: No significant improvement
- JBB2000:
Evaluation

- Asynchronous assertions reduce the overhead by approx. 90%.
- They scale good, at least as long as the checker threads are not overloaded.
Discussion

- Fallback to synchronous assertions if checkers are overloaded?
- Profile assertions and execute simple ones synchronously?
Evaluation

- Sharing copies helps:
Are the benchmarks used really meaningful?
Reception

- Only two theses reference the paper
- Not in the Jikes Research Archive
- Not available in other VMs
- Why?
Discussion

- Questions?
Discussion