Programmieren in Java

http://proglang.informatik.uni-freiburg.de/teaching/java/2017/

loop-interpreter

Interpreter für LOOP

Woche 08 Aufgabe 2/3

Herausgabe: 2017-06-19
Abgabe: 2017-06-30

Achtung: beachten Sie unbedingt die allgemeinen Hinweise zur Abgabe auf der Homepage.

Project `loop-interpreter`
Package `loopinterpreter`
Klassen

`State extends HashMap<String, Integer>`

`Expression`
`public int eval(State state)`

`Statement`
`public State run(State initial)`

`Statements`
`public static Statement assign(String name, Expression exp)`
`public static Statement seq(Statement s1, Statement s2)`
`public static Statement cond(Expression exp, Statement s1, Statement s2)`
`public static Statement loop(Expression exp, Statement s)`

Aus der Vorlesung kennen Sie bereits das Interface ***`Expression`*** zur Modellierung von Ausdrücken. Das Interface ***`Statement`*** modelliert nun in ähnlicher Weise Kommandos der Minimal-Programmiersprache `LOOP`. `LOOP`-Kommandos werden auf einem initialen Zustand (der Klasse `State`) ausgeführt und verändern diesen Zustand.

Zur Erinnerung: ein Zustqnd ist hier eine Map von Variablennamen auf Integer (implementiert als eine `HashMap<String, Integer>`)).

Es gibt folgende `LOOP` Kommandos:

- `Assign`: Weist einer Variablen den Wert einer Expression zu. Die Expression wird dabei auf dem initialen Zustand ausgewertet.
- `Seq`: Führt zwei Statements, `s1` und `s2`, hintereinander aus. Der Endzustand von `s1` wird dabei zum intialen Zustand von `s2`.
- `Cond`: Gegeben einem Ausdruck `exp` und zwei Kommandos `s1` und `s2`, wertet `Cond` zunächst `exp` auf dem initialen Zustand aus. Ist das Ergebnis ungleich 0, wird `s1` ausgeführt. Andernfalls wird `s2` ausgeführt.
- `Loop`: Gegeben einem Ausdruck `exp` und einem Statement `s` wird `s` so lange ausgeführt, wie das Ergebnis von `exp` ungleich 0 ist. `exp` und `s` werden zunächst auf dem initialen Zustand ausgeführt und dann auf dem Endzustand der vorherigen Ausführung von `s`.

Implementieren Sie außerdem die den Klassen Entsprechenden Konstruktions-Methoden in `Statements`.

```
package loopinterpreter;

import org.junit.Test;

import static loopinterpreter.Expressions.*;
import static loopinterpreter.Statements.*;
import static org.junit.Assert.*;

public class TestExamples {
    /**
     * y := x + 1
     */
    @Test
    public void assignTest() throws Exception {

        State state = new State();
        state.put("X", 42);
        Statement stmt = assign("Y", binop(var("X"), Binary.ADD, constant(1)));
        stmt.run(state);
        assertEquals(Integer.valueOf(42), state.get("X"));
        assertEquals(Integer.valueOf(43), state.get("Y"));

    }

    /**
     * y := x + 1;
     * x := y;
     */
    @Test
    public void seqTest() throws Exception {
        State state = new State();
        state.put("X", 42);
        Statement stmt = seq(assign("Y", binop(var("X"), Binary.ADD, constant(1))),
                             assign("X", var("Y")));
        stmt.run(state);
        assertEquals(Integer.valueOf(43), state.get("X"));
        assertEquals(Integer.valueOf(43), state.get("Y"));
    }

    /**
     * cond(x) {
     * x := x + 1
     * } else {
     * x : = x
     * }
     */
    @Test
    public void condTest() throws Exception {
        State state = new State();
        state.put("X", 42);
        Statement stmt = cond(var("X"),
                              assign("X", binop(var("X"), Binary.ADD, constant(1))),
                              assign("X", var("X")));

        stmt.run(state);
        assertEquals(Integer.valueOf(43), state.get("X"));
```

```
        state = new State();
        state.put("X", 42);
        assertEquals(0, binop(var("X"), Binary.SUB, var("X")).eval(state));
        stmt = cond(binop(var("X"), Binary.SUB, var("X")),
                    assign("X", binop(var("X"), Binary.ADD, constant(1))),
                    assign("X", var("X")));
        stmt.run(state);
        assertEquals(Integer.valueOf(42), state.get("X"));

    }

    /**
     *
     * loop(x) {
     * x := x-1
     * y := y+2
     * }
     */
    @Test
    public void loopTest() throws Exception {
        State state = new State();
        state.put("X", 42);
        state.put("Y", 0);
        Statement stmt = loop(var("X"),
                              seq(assign("X", binop(var("X"), Binary.SUB, constant(1))),
                                  assign("Y", binop(var("Y"), Binary.ADD, constant(2))))
        );
        stmt.run(state);
        assertEquals(state.get("X"), Integer.valueOf(0));
        assertEquals(state.get("Y"), Integer.valueOf(84));
    }

}
```