# Aflevering 8

Jens Kristian R. Nielsen, 201303862 Thomas D. Vinther , 201303874

1. april 2019

## Opgave 95

### Kode

```
/ case BlockExp(vals, vars, defs, exps) =>
      var env1 = env
      var sto1 = sto
      trace("Calculating variable values and adding to variable environment")
      for (d <- vals) {
        val (v, sto2) = eval(d.exp, env1, sto1)
        env1 = env1 + (d.x -> v)
        sto1 = sto2
        checkValueType(v,d.opttype,e)
      }
      for (d <- vars) {
        val (v, sto2) = eval(d.exp, env1, sto1)
        val loc = nextLoc(sto2)
        env1 = env1 + (d.x -> RefVal(loc, d.opttype))
        sto1 = sto2 + (loc -> v)
        checkValueType(v,d.opttype,e)
      }

      //Defs
      env1 = defs.foldLeft(env1)((en: Env, d: DefDecl) => {
        en + (d.fun -> ClosureVal(d.params, d.optrestype, d.body, en, defs))
      })
      var res: Val = unitVal
      for (exp <- exps) {
        val (res1, sto2) = eval(exp, env1, sto1)
        res = res1
        sto1 = sto2
      }
      (res, sto1)

case AssignmentExp(x, exp) =>
      val (v, sto1) = eval(exp, env, sto)
      env(x) match {
        case RefVal(loc, opttype) =>
          checkValueType(v, opttype, e)
          (unitVal, sto1 + (loc -> v))
        case _ => throw new InterpreterError("Not a var", e)
      }

case WhileExp(cond, body) =>
      eval(cond, env, sto) match {
        case (BoolVal(true), st) =>
          val (_, st1) = eval(body, env, st)
          eval(WhileExp(cond, body), env, st1)
        case (BoolVal(false), st) => (unitVal, st)
        case _ => throw new InterpreterError("Not a boolean", cond)
      }
```

### Tests

Se opgave 96

## Opgave 96

### Kode

```
case BlockExp(vals, vars, defs, exps) =>
      var tenv_updated = tenv //ValDecl
      for (d <- vals) {
        val t = typeCheck(d.exp, tenv_updated)
        checkTypesEqual(t, d.opttype, d)
        tenv_updated += (d.x -> d.opttype.getOrElse(t))
      }
      //VarsDecl
      for(d <- vars){
        val dType = typeCheck(d.exp,tenv_updated)      //theta e : tau
        checkTypesEqual(dType,d.opttype,BlockExp(vals,vars,defs,exps))     //tau =
            type(tau)
        tenv_updated += (d.x->RefType(dType))          //theta' x -> ref(tau)
      }
      //DefDecl with mututal recursion, via lecture 6, slide 36
      for (d <- defs) {
        tenv_updated += (d.fun -> getFunType(d))
      }
      for (d <- defs) {
        var tenvy = tenv_updated
        for (p <- d.params) {
          tenvy += (p.x -> p.opttype.getOrElse(throw new TypeError("Some error",
              BlockExp(vals,vars, defs, exps)))) //tau_1 = type(t_1) paramtype
        }
                        //Theta' [x ->tau-1] |-e:tau_2
        checkTypesEqual(typeCheck(d.body, tenvy), d.optrestype, BlockExp(vals,vars,
            defs, exps))  //tau_2 = type(t_2) resttype
      }

      //Block2 and block empty
      var res : Type = unitType
      for (exp <- exps){
        res = typeCheck(exp,tenv_updated)
      }
      res

case AssignmentExp(x, exp) =>
      tenv(x) match{
        case RefType(a) => checkTypesEqual(typeCheck(exp,tenv),Some(a),e)
          return unitType
        case _ => throw new TypeError("Not a var",e)
      }
      return unitType

case WhileExp(cond, body) =>
      typeCheck(cond,tenv) match{
        case BoolType() => typeCheck(body,tenv)
          return unitType
        case _ => throw new TypeError("Not a boolean expression",e)
      }
```

### Tests

```
// test 95
    test("{ def f(x: Int): Int = x; f(2) }", IntVal(2), IntType())
    testFail("{ def f(x: Int): Int = x; f(2, 3) }")
    test("2",IntVal(2),IntType())

    testVal("{ var z: Int = 0; { var t: Int = x; while (y <= t) { z = z + 1; t = t
        - y }; z } }", IntVal(3), Map("x" -> IntVal(17), "y" -> IntVal(5)))
    testType("{ var z: Int = 0; { var t: Int = x; while (y <= t) { z = z + 1; t = t
         - y }; z } }", IntType(), Map("x" -> IntType(), "y" -> IntType()))
    testVal("{ var x: Int = 0; def inc(): Int = { x = x + 1; x }; inc(); inc() }",
        IntVal(2))
    testType("{ var x: Int = 0; def inc(): Int = { x = x + 1; x }; inc(); inc() }",
         IntType())
    testVal("""{ def make(a: Int): Int => Int = {
              |    var c: Int = a;
              |    def add(b: Int): Int = { c = c + b; c };
              |    add
              |  };
              |  { val c1 = make(100);
              |    val c2 = make(1000);
              |    c1(1) + c1(2) + c2(3) } }""".stripMargin, IntVal(101 + 103 +
                 1003))
//test 49 & 68
    test("{def get(x: Int): Int = x; get(2) }", IntVal(2), IntType())
    test("{def f(x: Int) : Int = x; if(true) f(5) else f(3)}",IntVal(5),IntType())
    test("{def dyt(x: Int): Int = x*2; dyt(21)}",IntVal(42),IntType())
    test(" {def fac(n: Int) : Int = if (n == 0) 1 else n * fac(n - 1); fac(2)} ",
        IntVal(2),IntType())
    test("{def f(y: Int): Boolean = (y == y); f(2)}",BoolVal(true),BoolType())
    testFail("{ def f(x: Int): Int = x; f(2, 3) }")
    testFail("{def f(y: Int): Int = (y == y); f(2)}")
    testFail(" {def fac(n: Int) : Boolean = if (n == 0) 1 else n * fac(n - 1); fac
        (2)} ")
    testFail("{ def f(x: Float): Int = x; f(2f) }")
    test("{ def te(u: Int => Int): Int = u(u(4)); te((u: Int) => u % 4) }",IntVal
        (0),IntType())
    testFail("{ def te(u: Boolean => Int): Int = u(u(4)); te((u: Int) => u % 4) }")
    testVal("{ def isEven(x) = if (x == 0) true else isOdd(x-1);" +
      " def isOdd(x) = if (x == 0) false else isEven(x-1);isEven(2) }",BoolVal(true
        ))
//test slides week 10
    test("{ var x = 10;{ x = x - 10; x } + { x = x * 7; x }}",IntVal(0),IntType())
    test("{var x = 1;var y = 2;{var x = 3;x = 4;y = 5;x + y};x = x + 10;x}",IntVal
        (11),IntType())
    test("{def fac(n: Int): Int = if (n == 0) 1 else n * fac(n - 1);fac(4)}",IntVal
        (24),IntType())
    test("{def fac(n: Int): Int = {var r = 1 ; var i = n ; while ( 0 < i){r = r * i
        ;i = i - 1};r};fac(4)}",IntVal(24),IntType())
    test("{def fib(n: Int): Int = if (n <= 1) n else fib(n - 1) + fib(n - 2);fib(4)
        }",IntVal(3),IntType())
    test("{def fib(n: Int): Int = {if (n <= 1) n else {var pp = 0 ; var p = 1 ; var
         r = 0 ; var i = 2 ; while (i <= n) {r = p + pp ; pp = p ; p = r ; i = i +
        1};r}};fib(4)}",IntVal(3),IntType())
```