What Do Program Logics and Type Systems Have in Common?

Martin Hofmann*

Department of Informatics, University of Munich,
Oettingenstraße 67, 80538 München
Germany
mhofmann@informatik.uni-muenchen.de

This talk tries to contribute to a discussion started by John Reynolds' in his short presentation ("five minute madness talk") at the SPACE workshop 2004 (http://www.diku.dk/topps/space2004/).

Program logics such as Hoare logic or indeed any formalisation of operational semantics allow one to specify properties of programs and to formally prove them. In particular, simple safety properties such as absence of "method not understood" or non-violation of array bounds have been successfully established using program logics for substantial pieces of code.

Type systems seem to serve a similar purpose; they, too, promise to ensure safety properties of programs starting from R Milners celebrated slogan "well-typed programs do not go wrong."

The big advantage of a type system is its low (practical) complexity and its guaranteed success. A disadvantage of a more sociological nature is the enormous quantity of mutually incompatible type systems that have been developed and also the syntactic nature of the properties they purport to establish which makes it sometimes difficult to compare their merits.

The main advantage of type systems over program logics seems to dwindle in view of impressive recent progress in the area of automatic inductive theorem proving and software model checking. Will type systems therefore die out? I will argue in the talk that the answer is no and propose a useful synthesis between type systems and program logics that would also help addressing the aforementioned compatibility problem.

In a nutshell, the synthesis is that a type system can help to automatically generate invariants to be used in a proof based on program logic. In other words, the type system provides a high-level front end to a program logic.

One concrete example that I will use in the talk is a Java version of insertion sort using a free list. The goal is to prove (as automatically as possible!) that the line marked /*DON'T*/ is never executed.

* Partial support by the EU-funded project "Mobile Resource Guarantees" (IST-2001-33149) is herewith gratefully acknowledged.

J. Diaz et al. (Eds.): ICALP 2004, LNCS 3142, pp. 4–7, 2004.

```
class List{ static List freelist;

    int head;
    List tail;

    List(int head,List tail){
        this.head=head;
        this.tail=tail;
    }

    public static void init(int n) {
    // initialise freelist
        for(int i = 1;i<=n;i++){
            freelist = new List(0,(List)freelist);
        }
    }

    public static void free(List l) {
        l.tail = freelist;
        freelist = l;
    }

    public static List make(){
        if (freelist != null) {
            List res = freelist;
            freelist = freelist.tail;
            return res;
        } else {
/* DON'T */ System.out.println("Called new within make");
            return new List(0,(List)null);
        }
    }

    public static List nil(){
        return null;
    }

    public static List cons(int i,List l){
        List res = make();
        res.head = i;
        res.tail = l;
        return res;
    }
```

```
    public static boolean isnil(List l){
        return (l==null);
    }

    public static int head(List l){
        return l.head;
    }

    public static List tail(List l){
        return l.tail;
    }
}



class InsSort{
    public static void main(String[] args){
        List.init(2 * (args.length-1));

        List l = List.nil();

        for(int i=1;i<args.length;i++) {
            l = List.cons(Integer.valueOf(args[i]).intValue(),l);
        }
        printlist(sort(l));
    }

    public static void printlist(List l){
        if (List.isnil(l))
            System.out.println();
        else {
            System.out.print(List.head(l)+" ");
            printlist(List.tail(l));
        }
    }

    public static List insert(int i, List l){
        if (List.isnil(l))
            return List.cons(i,List.nil());
        else {
            int h = List.head(l);
            List t = List.tail(l);
            List.free(l);
            if (h <= i){
                return List.cons(h, insert(i,t));
```

```
            } else {
                return List.cons(i,List.cons(h,t));
            }
        }
    }

    public static List sort(List l){
        if (List.isnil(l))
            return List.nil();
        else {
            int h = List.head(l);
            List t = List.tail(l);
            return insert(h,sort(t));
        }
    }
}
```