2018-2019学年第1学期

**实 验 报 告**



* 课程名称:编程语言原理与编译
* 实验项目: 类型检查,一阶函数语言实现
* 专业班级\_\_\_
* 学生学号\_\_\_
* 学生姓名\_\_\_
* 实验指导教师:郭鸣

实验内容

1. **阅读 pku 目录里面的pdf 文件理解类型系统**

* **类型检查**
* **类型等价和类型相容**
* **基本类型**
* **用户定义类型**
* **多态性**
* **类型推理**

1. **阅读 3.monotype.p18-43.pdf**

* **理解各个语言语法构造类型检查的过程.**

1. **阅读4.unification.pdf**

* **理解 多态类型推理的类型归并unify方法**

1. **阅读 1.typecheck.ustc.pdf**

* **运行 p39- 45 例子程序,说明结果**
* **在gcc 里面 编译 p58 的程序,根据运行结果回答问题**
* 写出下面p56 对应的 fsharp 代码 ,构造该类型的值,查看其类型.

typedef struct cell {  
int info;  
struct cell \*next;  
} cell;

* **学习p60多态函数部分 请给出一个多态的Tree 的定义,写出计算Tree 深度的多态函数 depth : a Tree -> int**
* **理解类型协变与类型逆变**

1. **阅读**

**http://10.66.27.234/typeFunL/**

**<http://10.66.27.234/highOrderFunL/>**

**阅读 lab/type.funL/ 里面的MicroML源代码,理解**

* **函数定义/函数调用的实现**
* **静态作用域,动态作用域的实现**
* **类型检查的实现**
* **高阶函数的实现方法**
* **多态函数的类型归并unify 的实现**

1. **理解程序设计语言的类型系统**

* **查看并运行ExprType.fs 查看 a b c e1 e2 e3 eval 的类型并加以说明**
* **请定义一个笛卡尔类型 string \* string \* int，表达学生信息(姓名，学号，成绩) 请构造几个合法的值**
* **请定义一个函数，使得函数类型为： ‘a ->’a->’a**
* **查看并运行ExprEnv.fs 理解什么是求值环境 env**
* **fsc.exe编译 .fs文件 ，注意 当前目录需要有合适版本的 FSharp.Core.dll**

1. **请设计你大作业的类型系统,在大作业最终报告里面说明**
2. **提供哪些基本类型 int/double/char/string/bool ?**
3. **提供哪些类型复合机制 pair list array tuple ?**
4. **完成哪些类型检查/类型推理机制**

**强类型还是弱类型/静态类型还是动态类型 等.**

1. **修改microML 的函数定义,支持多参数的函数**

**Exercise 4.3** For simplicity, the current implementation of the functional language  
requires all functions to take exactly one argument. This seriously limits the programs that can be written in the language (at least it limits what that can be written  
without excessive cleverness and complications).  
Modify the language to allow functions to take one or more arguments. Start by  
modifying the abstract syntax in Absyn.fs to permit a list of parameter names in  
Letfun and a list of argument expressions in Call.  
Then modify the eval interpreter in file Fun.fs to work for the new abstract  
syntax. You must modify the closure representation to accommodate a list of parameters. Also, modify the Letfun and Call clauses of the interpreter. You will  
need a way to zip together a list of variable names and a list of variable values, to  
get an environment in the form of an association list; so function List.zip might  
be useful.  
**Exercise 4.4** In continuation of Exercise 4.3, modify the parser specification to  
accept a language where functions may take any (non-zero) number of arguments.  
The resulting parser should permit function declarations such as these:

let pow x n = if n=0 then 1 else x \* pow x (n-1) in pow 3 8 end  
let max2 a b = if a<b then b else a  
in let max3 a b c = max2 a (max2 b c)  
in max3 25 6 62 end  
end

1. **大作业分组登记,期末翻译任务下发.**
2. **以下为附加题 (完成数量不限,需要单独提交)**

**Exercise 4.6** Extend the abstract syntax, the concrete syntax, and the interpreter for  
the untyped functional language to handle tuple constructors (...) and component  
selectors #i (where the first component is #1):  
type expr =  
| ...  
| Tup of expr list  
| Sel of int \* expr  
| ...  
If we use the concrete syntax #2(e) for Sel(2, e) and the concrete syntax  
(e1, e2) for Tup[e1; e2] then you should be able to write programs such as  
these:  
let t = (1+2, false, 5>8)  
in if #3(t) then #1(t) else 14 end  
and  
let max xy = if #1(xy) > #2(xy) then #1(xy) else #2(xy)  
in max (3, 88) end  
This permits functions to take multiple arguments and return multiple results.  
To extend the interpreter correspondingly, you need to introduce a new kind of  
value, namely a tuple value TupV(vs), and to allow eval to return a result of  
type value (not just an integer):  
type value =  
| Int of int  
| TupV of value list  
| Closure of string \* string list \* expr \* value env  
let rec eval (e : expr) (env : value env) : value = ...  
4.12 Exercises 75  
Note that this requires some changes elsewhere in the eval interpreter. For instance, the primitive operations currently work because eval always returns an  
int, but with the suggested change, they will have to check (by pattern matching)  
that eval returns an Int i.  
**Exercise 4.7** Modify the abstract syntax tyexpr and the type checker functions  
typ and typeCheck in TypedFun/TypedFun.fs to allow functions to take  
any number of typed parameters.  
This exercise is similar to Exercise 4.3, but concerns the typed language. The  
changes to the interpreter function eval are very similar to those in Exercise 4.3  
and can be omitted; just delete the eval function.

**Exercise 4.8** Add lists (CstN is the empty list [], ConC(e1,e2) is e1::e2),  
and list pattern matching expressions to the untyped functional language, where  
Match(e0, e1, (h,t, e2)) is match e0 with [] -> e1 | h::t  
-> e2.  
type expr =  
| ...  
| CstN  
| ConC of expr \* expr  
| Match of expr \* expr \* (string \* string \* expr)  
| ..