1、输入参数 a 输出一个函数 b->c。

A type signature like 'a -> 'b -> 'c should be read as 'a -> ('b -> 'c).

如果类型是 a->b->c->d，那就是 curry 型的函数接受 abc 三个类型的参数，返回 d 类型的参数。

如果类型是(a->b)->c 就是接受一个函数作为参数，返回类型为 c 的参数。

2、关于 fold map filter 等内置函数，如果我们想要调用的函数 g 和 map 内置的 f 要求的参数不同，那么完全可以在 f 里面调用 g，这也是 ok 的。比如 hold 的 f 要求输入两个参数 f(a,b)，但是我们想要的函数 g 接受的函数的两个字符串的长度，那么可以这么写：fn f(a,b) => g(String.size a, String.size b)这样，在匿名函数里面处理参数，然后调用我们想要的函数。

3、Map 和 filter 是列表到列表的映射，fold 是对列表每个元素做操作然后叠加起来，适用于求整个列表的某种值，比如最大值，最小值，累积，累加，整个数组中符合某种规则的数的和等等。

4、还有就是 listpair.zip 函数，将两个列表压缩为一个二元元组的列表。好处，比如有个函数 match 接受(p,v)两个参数，而现在我有一个 p 的列表 pl，v 的列表 vl，他们之间是相互对应的，我想知道对应位置的 p 和 v 是否 match。这里可以先用 zip 将 pl vl 压缩为一个二元列表，然后调用 fold 函数即可。比如我想知道是否都 match，是的话就返回 true 那么可以这么写

```
Foldl (fn (a,acc)=>(match a)andalso acc) true
ListPairs.zip pl vl
```

这样返回值就是整个列表是否符合要求了。

5、关于挑战问题：

首先明确问题：

给定一系列类型，一个一系列 pattern，要求推导出这些 pattern 的类型。

已知类型如下

```
datatype typ = Anything (* any type of value is okay *)
             | UnitT (* type for Unit *)
             | IntT (* type for integers *)
             | TupleT of typ list (* tuple types *)
             | Datatype of string (* some named datatype *)
```

已知的 pattern 如下：

```
datatype pattern = Wildcard | Variable of string | UnitP | ConstP of int
                 | TupleP of pattern list | ConstructorP of string * pattern
```

type (string*string*typ) 相当于是(constructor, datatype, typ)

constructor 将 typ 转化为 datatype 类型。实际上第一个参数 type 列表是针对 constructorP 这个 pattern 的！！其他的 pattern 直接用 typ 其他的类型对应起来就好了。

所以问题转化为，判断 pattern list，如果不是 construtorP 类型，那么直接一一对应就好了，如果是 construtorP 类型，那么在第一个参数 type list 找到对应的元组，也就是说返回 constrctor 相互对应，typ 相互对应的那一项的 datatype。

此外题目还有一个要求就是最宽松的类型。

首先说一下思路：

将前两个 pattern 类型比较找到这个 pzattern 最宽松的类型，然后和第三个比较，以此类推迭代下去。

所以我们需要两个 helper 函数：

第一个：找到一个 pattern 对应的类型的函数。

第二个：找到两个 pattern 对应的最宽松的类型的函数。

关于第二个函数，其中两个 pattern 都是 tuple 要处理的时候比较复杂，看起来有好多情况啊，不等长啊什么的，但是实际上我们只要关注两种主要的情况就好了，首先是两个 tuple 都是空的，那返回空的就行。假如都不是空的，那么我们只要同时抽出两个 tuple 的第一个 pattern 进行比较就好了。至于其他的情况，不等长啊什么的，直接返回错误就行了！