# Homework 9
# Modules

98-317: Hype for Types

Due: 6 Nov 2018 at 6:30 PM

## 1 Introduction

In class, we talked about the type theory behind modules in SML. This involved discussing how to write the type of a module using singleton kinds and existential types. This homework gives more practice working with the two.

**Note:** If you find yourself spending more than 30 minutes on this homework, stop and turn in what you have. You will get full credit.

**Turning in the Homework** You should submit a PDF with your solutions to Autolab under Modules.

# 2 One of a Kind

In lecture, we talked about singleton kinds, and how both `int :: Type` and `int :: S(int)`. Note that singleton kinds only contain type constructors of kind Type. There is no `S(list)` since `list :: Type` $\rightarrow$ `Type`.

More generally, every type that has kind `S(`$\tau$`)` where $\tau$ :: Type also has kind `Type`. This gives us a sub*kind* relation[1]

In particular, the following rule

$$\frac{\texttt{S}(\tau)\ \texttt{kind}}{\Gamma \vdash \texttt{S}(\tau) <:: \texttt{Type}}$$

tells us that if `S(`$\tau$`)` is a valid kind, then it is a subkind of Type.

The least general kind of a type is one that does not have any subkinds other than itself. For example, `S(int)` is the least general kind of `int`, but Type is not, since it has a subkind, namely, `S(int)`.

For each type constructor, give its least general kind.

**Task 1** `int list`

**Task 2** `option`

For each static component of a signature, give its least general kind.

**Task 3**

```
type t
type s = int
```

**Task 4**

```
type t = int
type s = t
```

[1]Recall from the subtyping lecture that $\tau_1$ is a subtype of $\tau_2$ if all the values that have type $\tau_1$ also have type $\tau_2$. We apply the same thing to kinds now.

# 3 Existential (Crisis) Types

There are multiple signatures S that the following module could ascribe to.

```
structure M :> S = struct
    type t = int list
    type key = int
    fun insert (v : t) (k : key) = k :: v
    fun empty () : t = []
end
```

For each of the signatures given below, state if M ascribes to the signature or not. If it does not, explain why. If it does, write the type of M :> S as an existential type.

**Task 5**

```
signature S = sig
    type t
    type key
    val insert : t -> key -> t
    val empty : unit -> t
end
```

**Task 6**

```
signature S = sig
    type t = int list
    type key
    val insert : int list -> int -> int list
    val empty : unit -> int list
end
```