Today’s Agenda

- Type synonyms
- Type generality
- Equality types
- Syntactic sugar

Type Synonyms

- What does `int * int * int` represent?
- In HW1 we called it a date
- Wouldn’t it be nice to reflect this representation in the source code itself?

```
type date = int * int * int
```

`type` vs `datatype`

- `datatype` introduces a new type name, distinct from all existing types

```
datatype suit = Club | Diamond | Heart | Spade

datatype rank = Jack | Queen | King | Ace | Num of int
```

- `type` is just another name

```
type card = suit * rank
```
Type Synonyms

Why?
• For now, just for convenience
• It doesn’t let us do anything new

Later in the course we will see another use related to modularity.

Type Generality

Write a function that appends two string lists...

More General Types

• The type
  
  \texttt{\lstinline!'a list * 'a list -> 'a list}

  is more general than the type
  
  \texttt{\lstinline!string list * string list -> string list}

  and “can be used” as any less general type, such as
  
  \texttt{\lstinline!int list * int list -> int list}

• But it is not more general than the type
  
  \texttt{\lstinline!int list * string list -> int list}

• We would expect
  
  \texttt{\lstinline!string list * string list -> string list}

• But the type checker found
  
  \texttt{\lstinline!'a list * 'a list -> 'a list}

• ‘a are called Polymorphic Types
• Why is this OK?
The Type Generality Rule

The “more general” rule

A type $t_1$ is more general than the type $t_2$ if you can take $t_1$, replace its type variables consistently, and get $t_2$

What does consistently mean?

Equality Types

Write a list “contains” function...

Equality Types

• The double quoted variable arises from use of the = operator
  • We can use = on most types like int, bool, string, tuples (that contain only “equality types”)
  • Functions and real are not “equality types”
• Generality rules work the same, except substitution must be some type which can be compared with =
  • You can ignore warnings about “calling polyEqual”

Syntactic Sugar

• If-then-else is implemented as syntactic sugar for a case statement
If-then-else

• We’ve just covered case statements
• How could we implement if-then-else?

```haskell
case x of
  true => "apple"
  | false => "banana"
```

```haskell
if x then "apple" else "banana"
```

Adventures in pattern matching

• Shape example
• Function-pattern syntax if we get to it