Today’s Agenda

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

Let’s revisit the “append” function...

```plaintext
fun append (xs, ys) =
  if null xs
  then ys
  else hd xs :: append(tl xs, ys)
```
Type Generality

• We would expect

\[
\text{string list} * \text{string list} \rightarrow \text{string list}
\]

• But the type checker found

\[
\text{\texttt{`a list}} * \text{\texttt{`a list}} \rightarrow \text{\texttt{`a list}}
\]

• Why is this OK?
More General Types

• The type

\`
'a list * 'a list -> 'a list
\`

is **more general** than the type

\`
string list * string list list -> string list
\`

and “can be used” as **any less general** type, such as

\`
int list * int list -> int list
\`

• But it is **not** more general than the type

\`
int list * string list list -> int list
\`
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?
Type Generality

Write a function called Contains that takes a value and a list and returns true if the value is in the list...

What type will Contains have?
Type Generality

Let’s take a look at the `contains` function...

```haskell
fun contains(x, xs) =
    if null xs
    then false
    else (hd xs = x) orelse contains(x, tl xs)
```
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”
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

```plaintext
datatype suit = Club | Diamond | Heart | Spade
datatype rank = Jack | Queen | King | Ace
                   | Num of int
```

• **type** is just another name

```plaintext
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.
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?

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

if x then "apple" else "banana"
```
Adventures in pattern matching

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