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

```haskell
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...
Type Generality

• We would expect

\[
\text{string list } \times \text{ string list } \to \text{ string list}
\]

• But the type checker found

\[
\text{'a list } \times \text{ 'a list } \to \text{'a list}
\]

• Why is this OK?

More General Types

• The type

\[
\text{'a list } \times \text{ 'a list } \to \text{'a list}
\]

is more general than the type

\[
\text{string list } \times \text{ string list } \to \text{ string list}
\]

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

\[
\text{int list } \times \text{ int list } \to \text{ int list}
\]

• But it is not more general than the type

\[
\text{int list } \times \text{ string list } \to \text{ int list}
\]

The Type Generality Rule

The “more general” rule

A type \( t1 \) is more general than the type \( t2 \) if you can take \( t1 \), replace its type variables consistently, and get \( t2 \)

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"

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

Adventures in pattern matching

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