CSE 341:
Programming Languages

Section AC with Nate Yazdani

recap

- boolean operators
  - good style to use **`andalso`**, **`orelse`**, and **`not`**
  - syntactic sugar for certain uses of if-then-else

```
(* e1 andalso e2 *)
if e1
then e2
else false
```

```
(* e1 orelse e2 *)
if e1
then true
else e2
```

- style grading will be restrained this quarter

announcement

- sections have typically been like extra lectures
- today, we're trying short coding exercises instead of me live-coding
- hopefully more engaging and useful for you guys
- if not, we'll switch back next time

questions?

agenda

- type synonyms
- type generality
- equality types
- syntactic sugar

type synonyms

- what is the meaning of `int * int * int`?
  - literally, a triple of integers
  - conceptually, it could be a date, a co-ordinate, or some other thing

- it'd sure be nice if our code could reflect the *purpose* of a type in addition to its "literal meaning"

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

`type` vs. `datatype`

- `datatype` defines a *new* type and a name for it
  - different from all existing types

```
datatype suit = Club | Diamond | Heart | Spade
datatype rank = Jack | Queen | King | Ace
              | Number of int (* 2-10 *)
```

- `type` gives a new name to an “existing” type
  - might be built out of smaller types
  - still just a name

```
type card = suit * rank
```

type synonyms: why bother?

- really really good for documentation
  - for this reason, languages without them often have popular conventions for variable names
- doesn't let us do anything we couldn't do before
- later in the course, we'll see how they help with modularity

coding exercise

please work with the people around you to write an SML function to reverse a string list (without type annotations)

type generality

- what type did SML give your function?
- probably **`'a list -> 'a list`**
- why not **`string list -> string list`** ?

```
fun rev xs =
  case xs of
    x::xs' => rev(xs') @ [x]
  | []     => []
```

fast list reverse

```
fun rev xs =
  let fun aux (xs, ys) =
          case xs of
            x::xs' => aux(xs', x::ys)
          | []     => ys
  in
      aux(xs, [])
  end
```

type generality

- the type inferred by SML is *more general* than the one that we had in mind

```
'a list -> 'a list
```

- it works wherever any *less general* type is expected
- so just as good as these other types:

```
string list -> string list
```

```
int list -> int list
```

- but *not* this one:

```
string list -> int list
```

rule for generality

A type t_1 is **more general** than a type t_2
if you can **substitute** the type variables of t_1
consistently to get t_2.

example of generality

The type

```
'a list -> 'a list
```

is more general than the type

```
int list -> int list
```

because you can substitute **`int`** for **`'a`**

more coding!

please work together again and write a list-contains function (without type annotations)

…and without the **`List.exists`** library function :-)

equality types

- a type variable with double quotes (*e.g.*, **`''a`**) can only be substituted with an *equality type*

- an equality type is a type that supports the **`=`** operator, such as **`int`**, **`bool`**, or **`string`**

- function types and **`real`** are *not* equality types

- **you can completely ignore warnings about "calling polyEqual"**

syntactic sugar

- under the hood, the if-then-else syntax form is actually translated into a case statement

```
(* if e1 then e2 else e3 *)
case e1 of
  true  => e2
| false => e3
```

- so the **`andalso`** and **`orelse`** operators are syntactic sugar for if-then-else, which is syntactic sugar for a case statement!
- SML is pretty “sweet” like that!

…yeah okay that was pretty bad

before you go…

some quick feedback

- did the exercises help at all?

- given what you've learned so far, were they…
  - too small?
  - not enough?
  - annoying and/or confusing?
  - distracting from the main point?

- any other suggestions for how to make section work better for you?

thanks!