CSE 341 AC

Yuma & Taylor
University of Washington

Section Overview

- Check-in on Homework 1
- Type synonyms
- Type generality
- Equality types
- Syntactic sugar

Type Synonyms

- In Homework 1, we represented a date as: `(int * int * int)`
- Now, we can tell SML that that *is* a date:

```
type date = (int * int * int)
```

- This is *not* a datatype:
  - No constructors.
  - No variants.
  - Completely interchangeable for `(int * int * int)`.

Type Synonyms

```
type date = (int * int * int)

fun tomorrow (d : date) : date = ...
```

- What is `tomorrow`'s type?
  - `val tomorrow = fn : date -> date`
  - `val tomorrow = fn : (int * int * int) -> date`
  - `val tomorrow = fn : date -> (int * int * int)`
  - `val tomorrow = fn : (int * int * int) -> (int * int * int)`

Type Generality

Write a function that appends two `string lists`:

```
fun append (xs, ys) = ...
```

Type Generality

```
fun append (xs, ys) =
  case xs of
      [] => ys
    | x::xs' => x :: (append (xs', ys))
```

Type Generality

The type checker just told us that append's type is:

```
val append = fn : 'a list * 'a list -> 'a list
```

Why is it not:

```
val append = fn : string list * string list -> string list
```

More General Types

- `'a` is "more general" than `string`
- `t1` is more general than `t2` if:
  - You can replace its type variables **consistently**, and
  - You get `t2`

Example (replace `'a` with `string`):

- `t1 : 'a list * 'a list -> 'a list`
- `t2 : string list * string list -> string list`

Equality Types

Write a function that determines if one element is contained in a list:

```
fun contains (x, xs) = ...
```

Equality Types

```
fun contains (x, xs) =
  case xs of
      [] => false
    | x'::xs' => x = x' orelse (contains (x, xs'))
```

Equality Types

The type checker just told us that `contains`'s type is:

```
val contains = fn : ''a * ''a list -> bool
```

Why is it not:

```
val contains = fn : 'a list * 'a list -> 'a list
```

Equality Types

- ‘ ‘a is a type variable that is equipped with equality
- Another way to think about this: “*on what types is equality well defined*”?
- Some examples:
  - `string`, `int`, `datatypes` where all members are equality types
- Some counter-examples:
  - `real`, `datatypes` where *not* all members are equality types
- *Note*: ignore warnings about `polyEqual`

Fun fact: `if then else` is syntactic sugar

`if then else` is syntactic sugar for a `case` expression!

Write the following as a `case` expression:

```
if x then 5 else 10
```

Fun fact: `if then else` is syntactic sugar

`if then else` is syntactic sugar for a `case` expression!

Write the following as a `case` expression:

```
if x then 5 else 10
```

```
case x of
    true => 5
  | false => 10
```

Pattern matching example:

1. Let’s write a datatype shape which represents some 2D shapes, and
2. A function `val area = fn : shape -> real` which computes a shape’s area.

datatype shape

```
datatype shape = square of real (* side length *)
               | rectangle of real * real (* dimensions *)
               | circle of real (* radius *)
```

val area = fn : shape -> real

```
fun area (s : shape) : real =
  case s of
      (Square l) => l * l
    | (Rectangle (x, y)) => x * y
    | (Circle r) => 3.14 * r * r
```

val area = fn : shape -> real

```
fun area (Square l) = l * l
  | area (Rectangle (x, y)) = x * x
  | area (Circle r) = 3.14 * r * r
```