CSE 341: Section 2

Tam Dang

University of Washington

October 4, 2018

Outline

Types

- What does `int * int * int` mean?

Types

- What does `int * int * int` mean?
- In HW1, we use it as a **date** type

Types

- What does `int * int * int` mean?
- In HW1, we use it as a **date** type
- Can we make the semantics of this type more explicit?

Types

- What does `int * int * int` mean?
- In HW1, we use it as a **date** type
- Can we make the semantics of this type more explicit?

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

Types vs. DataTypes

A `datatype` introduces a new type distinct from all existing types

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

Types vs. DataTypes

A `datatype` introduces a new type distinct from all existing types

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

A `type` (aka *type synonym*) is just another name

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

Types vs. DataTypes

A `datatype` introduces a new type distinct from all existing types

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

A `type` (aka *type synonym*) is just another name

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

`datatype`s have constructors but type synonyms don't!

Types vs. DataTypes

A `datatype` introduces a new type distinct from all existing types

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

A `type` (aka *type synonym*) is just another name

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

`datatypes` have constructors but type synonyms don't!

Why use type synonyms?

- For now, just for convenience
- Later we will see another use for type synonyms in the `modules` unit

Type Generality

Write a function that appends two **string lists**

Type Generality

Write a function that appends two **string lists**

What happened?

Type Generality

Write a function that appends two **string lists**

What happened?

We thought we needed

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

But the type checker found and used

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

Why is this OK?

Type Generality

More General Types

The type

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

is more general than the type

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

More general types can be used in place of less general types, for example

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

Type Generality

More General Types

The type

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

is more general than the type

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

More general types can be used in place of less general types, for example

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

Is `'a list * 'a list -> 'a list` more general than
`int list * string list -> int list`?

Type Generality

The "Type Generality 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

Write a ”list contains” function

Equality

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”)
- Generality rules work the same, except substitution must be some type which can be compared with =

Equality

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”)
- Generality rules work the same, except substitution must be some type which can be compared with =
- Functions and real are **not** ”equality types”
- You can ignore warnings about “calling polyEqual”

Syntactic Sugar

if-then-else

`if-then-else` is *syntactic sugar* for a case expression:

Syntactic Sugar

if-then-else

`if-then-else` is *syntactic sugar* for a case expression:

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

can be written as

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

Syntactic Sugar

Logical Operators

`andalso` and `orelse` are also forms of *syntactic sugar*!

Syntactic Sugar

Logical Operators

`andalso` and `orelse` are also forms of *syntactic sugar*!

Given

```
val x = true
val y = false
```

Logical "and" can be written as

```
val x_and_y = x andalso y
```

or this

```
val x_and_y = case x of true => y | false => false
```

Adventures in pattern matching!

SML code we write / look at together will be available on the course website