PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

CSE 341 Section 2

Spring 2020

Adapted from slides by Nick Mooney, Nicholas Shahan, Patrick Larson, and Dan Grossman

1

Today's Agenda

- Testing
- Lists, Let-Expression (Review)
- Options
- Type synonyms
- Type generality
- Equality types
- Syntactic sugar

Reminder: Check out the CSE341 style guide as you work on HW!
Also check out the style guides in section 1 slide!

2

Testing

- You should still test your code!
- We will assign points to your testing file
- Just do something like this:

```
val test1 = ((4 div 4) = 1);
```

"Is expected output = actual output"

3

Section Learning Objectives

- Review building/accessing new types (e.g. **datatypes**)
- Recognize **type synonyms** as "convenient" feature
- Be able to generalize specific types with **polymorphism** (e.g. int list into 'a list) and **equality** types
- Practice using pattern-matching with **case expressions**

4

Lists

- Lots of new types: For any type t, the type t list describes lists where all elements have type t
  - Examples: `int list`, `bool list`, `int list list`, `(int * int) list`, `(int list * int) list`
- So `[]` can have type t list for any type t
  - SML uses type `'a list` to indicate this ("tick a" or "alpha")
- For `e1::e2` to type-check, we need a **t** such that e1 has type **t** and e2 has type **t** list. Then the result type is **t** list
  - `null: 'a list -> bool`
  - `hd: 'a list -> 'a`
  - `tl: 'a list -> 'a list`

5

Let-Expression

- Syntax: `let b1 b2 ... bn in e end`
  - Each **bi** is any binding and **e** is any expression
- Type-checking: Type-check each **bi** and **e** in a static environment that includes the previous bindings.
- Type of whole let-expression is the type of **e**.
- Evaluation: Evaluate each **bi** and **e** in a dynamic environment that includes the previous bindings.

Result of whole let-expression is result of evaluating **e**.

6

Options

t option is a type for any type **t**
- (much like t list, but a different type, not a list)

Building:
- **NONE** has type `'a option` (much like [] has type 'a list)
- **SOME e** has type t option if e has type t (much like e::[])

Accessing:
- **isSome** has type `'a option -> bool`
- **valOf** has type `'a option -> 'a` (exception if given NONE)

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
```


Datatypes

- What if we want something **unique**? A **new** type?
- We can't just use type synonyms because they can only be built from existing types.
- **Datatypes** give us the ability to define **custom types**.

```
datatype foo = bar | baz of int | qux of bool
```


`type` *vs* `datatype`

- **`datatype`** introduces a new type name, distinct from all existing types

```
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

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

- But the type checker found

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

- `'a` are called Polymorphic Types
- Why is this OK?

13

More General Types

- The type

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

is more general than the type

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


14

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?

15

Equality Types

Write a list "contains" function…

16

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"

17

More Syntactic Sugar

- Tuples are just records
- If-then-else is implemented as syntactic sugar for a case statement

18

If-then-else

- We've just covered case statements
- How could we implement if-then-else

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

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


19

val-Pattern Matching

Remember our unit test?

```
(* Neat trick for creating hard-fail tests: *)

val true = ((4 div 4) = 1);
```

Just a pattern match!

"Match the left hand side against the value 'template' true, binding any variables (there aren't any!)"

20

Adventures in pattern matching

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

21