15-150 Fall 2024

Review

Dilsun Kaynar

What is on the final exam?

Advice for the final

Review

Lecture slides, notes, labs, homeworks

Sleep

If languages were cars …

C was the great all-arounder: compact, powerful, goes everywhere, and reliable in situations where your life depends on it.

http://crashworks.org/if_programming_languages_were_vehicles/

If languages were cars …

C++ is the new C — twice the power, twice the size, works in hostile environments, and if you try to use it without care and special training you will probably crash.

http://crashworks.org/if_programming_languages_were_vehicles/

If languages were cars …

Java is another attempt to improve on C. It sort of gets the job done, but it's way slower, bulkier, spews pollution everywhere, …

http://crashworks.org/if_programming_languages_were_vehicles/

If languages were cars …

Python is great for everyday tasks: easy to drive, versatile, comes with all the conveniences built in.

http://crashworks.org/if_programming_languages_were_vehicles/

If languages were cars …

ML … a beautiful car.

Can only be driven on properly typed roads.

It's possible to drive conventionally, if the dealer activates some under-the-hood enhancements.

ENSUREd to reach the REQUIREd destination. It *never breaks down.**

***When the car does break down an *exception* light flashes.**

Thanks to Stephen Brookes

What is SML?

- A functional programming language

Computation = evaluation

- A typed language

Only well-typed expressions are evaluated

- A polymorphic typed language

well-typed expressions have a most general type

- A call-by-value language

Function calls evaluate their arguments first

Benefits

- **Referential transparency**
  - Equivalent code is interchangeable, in all contexts
  - Simple compositional reasoning
- **Mathematical foundations**
  - Can use math and logic to prove correctness
  - Use induction to analyze recursive code and data
- **Functions are values**
  - Can be used as data in lists, tuples, …
  - and argument or result of other functions

- **Parallelism**
  - Expression evaluation has *no side-effects*
  - Evaluation order makes no difference to the value obtained
  - Can evaluate *independent* code *in parallel*

Principles

- **Expressions must be well-typed.**
  Well-typed expressions don't go wrong.
- **Every function needs a specification.**
  Well-specified programs are easy to understand.
- **Every specification needs a proof.**
  Well-proven programs do the right thing.
- **Large programs should be designed as *modules*.**
  Well-interfaced code is easier to maintain.

- **Data structures, algorithms.**
  Good choice of data structure leads to better code.
- **Exploit parallelism.**
  Parallel code may run faster.
- **Strive for simplicity.**
  Programs should be as simple as possible, but no simpler.

Functions are values

Some values are -- integers, lists of integers, …

Some values do -- functions, streams, …

Functions can be used to represent
graphs, dictionaries, …

Higher order functions

- Functions can take functions as arguments
- Functions can return functions as results

```
List.map : (’a -> ’b) -> (’a list -> ’b list)
Seq.map : (’a -> ’b) -> (’a seq -> ’b seq)

List.foldl, foldr : (’a * ’b -> ’b) -> ’b -> ’a list -> ’b
Seq.reduce : (’a * ’a -> ’a) -> ’a -> ’a seq -> ’a
```

- Allow uniform solutions to parameterized problems
- Write once, use many ways

```
ins : (’a * ’a -> order) -> (’a * ’a list -> ’a list)
foldr : (’a * ’b -> ’b) -> ’b -> ’a list -> ’b
```

```
fun isort cmp = foldr (ins cmp) [ ]
```

- Can represent patterns of computation
- Can express control flow such as continuations
- Let you delay, manipulate, ignore a computation

Staging

- A curried function may do useful work before getting all of its arguments
- May improve efficiency by doing this work once, early, rather than in every function call
  - Choose argument order wisely

Recursion

- ML supports recursive function definition

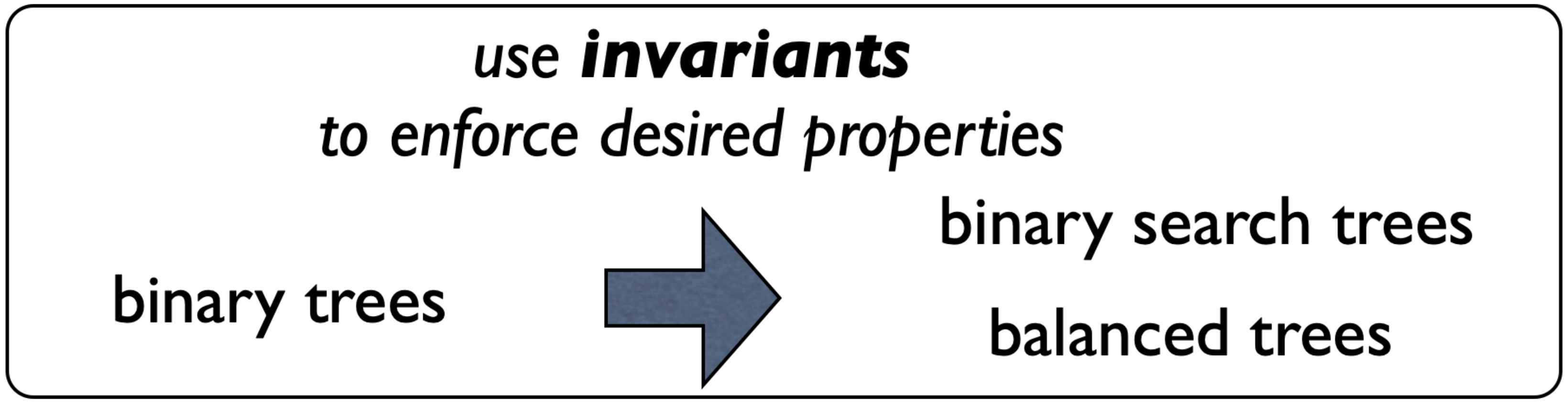
```
fun f(x:t1):t2 = e
```

- Use *induction* to prove properties

Datatypes

- Represent your problem, your way
- Extend the type discipline, seamlessly
- Can be recursive and parametric

```
’a list          ’a tree
```



*use **invariants***
to enforce desired properties

binary trees → binary search trees

balanced trees

Structural induction

- The set of values of a recursive datatype can be characterized *inductively*
- For every recursive datatype definition there is a *principle of structural induction*
  - Use to prove properties of values...

For all types `t` and values `T : t tree`, `inord(T)` evaluates to a value...

Modules

- Signatures as interfaces
- Structures as implementations

```
signature DICT =
sig
  structure Key : ORDER
  type ’a dict
  val empty : ’a dict
   ...
end
```

```
structure Bst : DICT =
struct
  structure Key = ...
  datatype ’a dict = Empty | ...
  val empty = Empty
   ...
end
```

hide information …

- Users of a structure can only see what’s visible in the signature

support abstract code design

- ***abstract data types*** with limited operations

binary search trees,
sequences,
dictionaries, …

invariants guide design and
support localized reasoning

- ***type classes***: types with operations

Functors

- Build implementations from implementations
- Encapsulate common constructions
- Allow code re-use

Work and Span

- Can reason abstractly about both sequential and parallel complexity

  W = sequential complexity
  S = parallel complexity

  - Can extract ***recurrence relations*** for W and S from a recursive function definition

  - Can ***solve*** or find ***asymptotic*** approximation

- Can use a *cost graph* for an expression evaluation

  W = size
  S = depth

Abstracts away from scheduling details

Functional Programming in Practice

- Theorem provers, hardware/software verification
- Companies in finance and telecommunications
- Compilers for most functional languages are implemented in themselves

You might also like

- 15-210: Parallel Data Structures and Algorithms
- 15-312: Principles of Programming Languages
- 15-317: Constructive Logic
- 15-411: Compiler Design
- 15-451: Algorithms
- 80-413: Category Theory

Two Sources of Beauty In Programs

- **Structure:** code as an expression of an idea
- **Efficiency:** code as instructions for a computer

Bob Harper’s talk at John Mitchell’s birthday celebration, 2016

It has been a pleasure to have you as my students!

Thanks to our awesome course staff!