Advances in Programming Languages
APL2: Some types and a little OCaml

Ian Stark
School of Informatics
The University of Edinburgh

Monday 14 January 2008
Semester 2 Week 2
Plan

- Types and type systems
- Course timing plan
- A small amount of OCaml
Some types

A selection of types from some languages.

C/C++

int, long, float, unsigned int, char
int[], char*, char&, int(*)(float,char)

OCaml

int, int64, bool, char, string, unit
string*string, int list, bool array
int->int, int->string->char, 'a list -> 'a list

Java

Object, byte[], boolean
StringBuffer, LinkedList, TreeSet, ArrayList<String>
IllegalPathStateException, BeanContextServiceRevokedListener
What do people do with types?

- Type checking
- Static type checking
- Dynamic type checking
- Type annotation
- Type inference
- Subtyping
- Structural typing
- Nominative typing
- Duck typing
- Effect types
What is a type system?

A *type system* is a syntactically defined subset $T$ of programs such that:

$$P \in T \implies \text{Compile}(P) \models \phi$$

(read: “if $P$ is in $T$ then Compile($P$) satisfies $\phi$”)

where Compile($P$) is the object code corresponding to $P$ and $\phi$ is some desired property of its execution.

For example,

$$T = \text{“well-typed Java programs”}$$

$$\phi = \text{“methods are always correctly invoked”}$$

Slogan: *Well-typed programs cannot go wrong.*  
[Robin Milner, 1978]
Java is serious about abstraction

Java works almost entirely through class-based object-oriented programming; it encourages the use of abstract classes through inheritance and interfaces; and it does not expose the private workings of classes and packages.

Java is serious about typing

Java has strong static typing: all programs are checked for type-correctness at compile-time. Bytecode is checked again when classes are loaded, by the bytecode verifier, before execution. The recent introduction of generics extends the power of the type system.

Even so, things do not always go as well as one might hope...
Java has subtyping: a value of one type may be used at any more general type. So `String < Object`, and every `String` is an `Object`.

### Not all is well with Java types

```java
String[] a = { "Hello" }; // A small string array
Object[] b = a; // Now a and b are the same array
b[0] = Boolean.FALSE; // Drop in a Boolean object
String s = a[0]; // Oh, dear
System.out.println(s); // This isn't going to be pretty
```

This compiles without error or warning: in Java, if `S < T` then `S[] < T[]`. Except that it isn’t. So every array assignment gets a runtime check.
## Time plan

<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>7 January</td>
<td>10 January</td>
</tr>
<tr>
<td>Week 2</td>
<td>14 January</td>
<td>17 January</td>
</tr>
<tr>
<td>Week 3</td>
<td>21 January</td>
<td>24 January</td>
</tr>
<tr>
<td>Week 4</td>
<td>28 January</td>
<td>31 January</td>
</tr>
<tr>
<td>Week 5</td>
<td>4 February</td>
<td>7 February</td>
</tr>
<tr>
<td>Week 6</td>
<td>11 February</td>
<td>14 February</td>
</tr>
<tr>
<td>Week 7</td>
<td>18 February</td>
<td>21 February</td>
</tr>
<tr>
<td>Week 8</td>
<td>25 February</td>
<td>28 February</td>
</tr>
<tr>
<td>Week 9</td>
<td>3 March</td>
<td>6 March</td>
</tr>
<tr>
<td>Week 10</td>
<td>10 March</td>
<td>13 March</td>
</tr>
<tr>
<td>Week 11</td>
<td>17 March</td>
<td>20 March</td>
</tr>
</tbody>
</table>

This gives 22 slots.

---

This gives 22 slots
## Time plan

<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 January</td>
<td>10 January</td>
</tr>
<tr>
<td>2</td>
<td>14 January</td>
<td>17 January</td>
</tr>
<tr>
<td>3</td>
<td>21 January</td>
<td>24 January</td>
</tr>
<tr>
<td>4</td>
<td>28 January</td>
<td>31 January</td>
</tr>
<tr>
<td>5</td>
<td>4 February</td>
<td>7 February</td>
</tr>
<tr>
<td>6</td>
<td>11 February</td>
<td>14 February</td>
</tr>
<tr>
<td>7</td>
<td>18 February</td>
<td>21 February</td>
</tr>
<tr>
<td>8</td>
<td>25 February</td>
<td>28 February</td>
</tr>
<tr>
<td>9</td>
<td>3 March</td>
<td>6 March</td>
</tr>
<tr>
<td>10</td>
<td>10 March</td>
<td>13 March</td>
</tr>
<tr>
<td>11</td>
<td>17 March</td>
<td>20 March</td>
</tr>
</tbody>
</table>

This gives 21 slots.
Time plan

Week 1  Monday  7 January  Thursday 10 January
Week 2  Monday 14 January  Thursday 17 January
Week 3  Monday 21 January  Thursday 24 January
Week 4  Monday 28 January  Thursday 31 January
Week 5  Monday  4 February  Thursday  7 February
Week 6  Monday 11 February  Thursday 14 February
Week 7  Monday 18 February  Thursday 21 February
Week 8  Monday 25 February  Thursday 28 February
Week 9  Monday  3 March  Thursday  6 March
Week 10 Monday 10 March  Thursday 13 March
Week 11 Monday 17 March  Thursday 20 March

This gives 19 slots.
<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 January</td>
<td>10 January</td>
</tr>
<tr>
<td>2</td>
<td>14 January</td>
<td>17 January</td>
</tr>
<tr>
<td>3</td>
<td>21 January</td>
<td>24 January</td>
</tr>
<tr>
<td>4</td>
<td>28 January</td>
<td>31 January</td>
</tr>
<tr>
<td>5</td>
<td>4 February</td>
<td>7 February</td>
</tr>
<tr>
<td>6</td>
<td>11 February</td>
<td>14 February</td>
</tr>
<tr>
<td>7</td>
<td>18 February</td>
<td>21 February</td>
</tr>
<tr>
<td>8</td>
<td>25 February</td>
<td>28 February</td>
</tr>
<tr>
<td>9</td>
<td>3 March</td>
<td>6 March</td>
</tr>
<tr>
<td>10</td>
<td>10 March</td>
<td>13 March</td>
</tr>
<tr>
<td>11</td>
<td>17 March</td>
<td>20 March</td>
</tr>
</tbody>
</table>

This gives 19 slots = 5 topics × 3 regular lectures + 4 interstitial lectures.
<table>
<thead>
<tr>
<th>Week</th>
<th>Monday Date</th>
<th>Thursday Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 January</td>
<td>10 January</td>
</tr>
<tr>
<td>2</td>
<td>14 January</td>
<td>17 January</td>
</tr>
<tr>
<td>3</td>
<td>21 January</td>
<td>24 January</td>
</tr>
<tr>
<td>4</td>
<td>28 January</td>
<td>31 January</td>
</tr>
<tr>
<td>5</td>
<td>4 February</td>
<td>7 February</td>
</tr>
<tr>
<td>6</td>
<td>11 February</td>
<td>14 February</td>
</tr>
<tr>
<td>7</td>
<td>18 February</td>
<td>21 February</td>
</tr>
<tr>
<td>8</td>
<td>25 February</td>
<td>28 February</td>
</tr>
<tr>
<td>9</td>
<td>3 March</td>
<td>6 March</td>
</tr>
<tr>
<td>10</td>
<td>10 March</td>
<td>13 March</td>
</tr>
<tr>
<td>11</td>
<td>17 March</td>
<td>20 March</td>
</tr>
</tbody>
</table>

This gives 19 slots = 5 topics \times 3 regular lectures + 4 interstitial lectures.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.
## Time plan

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Monday 7 January</th>
<th>Thursday 10 January</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 2</td>
<td>Monday 14 January</td>
<td>Thursday 17 January</td>
</tr>
<tr>
<td>Week 3</td>
<td>Monday 21 January</td>
<td>Thursday 24 January</td>
</tr>
<tr>
<td>Week 4</td>
<td>Monday 28 January</td>
<td>Thursday 31 January</td>
</tr>
<tr>
<td>Week 5</td>
<td>Monday 4 February</td>
<td>Thursday 7 February</td>
</tr>
<tr>
<td>Week 6</td>
<td>Monday 11 February</td>
<td>Thursday 14 February</td>
</tr>
<tr>
<td>Week 7</td>
<td>Monday 18 February</td>
<td>Thursday 21 February</td>
</tr>
<tr>
<td>Week 8</td>
<td>Monday 25 February</td>
<td>Thursday 28 February</td>
</tr>
<tr>
<td>Week 9</td>
<td>Monday 3 March</td>
<td>Thursday 6 March</td>
</tr>
<tr>
<td>Week 10</td>
<td>Monday 10 March</td>
<td>Thursday 13 March</td>
</tr>
<tr>
<td>Week 11</td>
<td>Monday 17 March</td>
<td>Thursday 20 March</td>
</tr>
</tbody>
</table>

This gives 19 slots = 5 topics × 3 regular lectures + 4 interstitial lectures.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3;
<table>
<thead>
<tr>
<th>Week</th>
<th>Monday</th>
<th>Thursday</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7 January</td>
<td>10 January</td>
</tr>
<tr>
<td>2</td>
<td>14 January</td>
<td>17 January</td>
</tr>
<tr>
<td>3</td>
<td>21 January</td>
<td>24 January</td>
</tr>
<tr>
<td>4</td>
<td>28 January</td>
<td>31 January</td>
</tr>
<tr>
<td>5</td>
<td>4 February</td>
<td>7 February</td>
</tr>
<tr>
<td>6</td>
<td>11 February</td>
<td>14 February</td>
</tr>
<tr>
<td>7</td>
<td>18 February</td>
<td>21 February</td>
</tr>
<tr>
<td>8</td>
<td>25 February</td>
<td>28 February</td>
</tr>
<tr>
<td>9</td>
<td>3 March</td>
<td>6 March</td>
</tr>
<tr>
<td>10</td>
<td>10 March</td>
<td>13 March</td>
</tr>
<tr>
<td>11</td>
<td>17 March</td>
<td>20 March</td>
</tr>
</tbody>
</table>

This gives 19 slots = 5 topics × 3 regular lectures + 4 interstitial lectures.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3; choice of topic must be made by the end of Week 4;
### Time plan

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Monday 7 January</th>
<th>Thursday 10 January</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 2</td>
<td>Monday 14 January</td>
<td>Thursday 17 January</td>
</tr>
<tr>
<td>Week 3</td>
<td>Monday 21 January</td>
<td>Thursday 24 January</td>
</tr>
<tr>
<td>Week 4</td>
<td>Monday 28 January</td>
<td>Thursday 31 January</td>
</tr>
<tr>
<td>Week 5</td>
<td>Monday 4 February</td>
<td>Thursday 7 February</td>
</tr>
<tr>
<td>Week 6</td>
<td>Monday 11 February</td>
<td>Thursday 14 February</td>
</tr>
<tr>
<td>Week 7</td>
<td>Monday 18 February</td>
<td>Thursday 21 February</td>
</tr>
<tr>
<td>Week 8</td>
<td>Monday 25 February</td>
<td>Thursday 28 February</td>
</tr>
<tr>
<td>Week 9</td>
<td>Monday 3 March</td>
<td>Thursday 6 March</td>
</tr>
<tr>
<td>Week 10</td>
<td>Monday 10 March</td>
<td>Thursday 13 March</td>
</tr>
<tr>
<td>Week 11</td>
<td>Monday 17 March</td>
<td>Thursday 20 March</td>
</tr>
</tbody>
</table>

This gives 19 slots = 5 topics × 3 regular lectures + 4 interstitial lectures.

Coursework is to research a novel language feature, from a list provided; making a written report on this, with your own working code examples.

The topic list will be presented at the start of Week 3; choice of topic must be made by the end of Week 4; report due by the end of Week 9.
Objective Caml (OCaml) is:

- A strongly-typed functional language, a version of ML; with
- high-performance native-code compilers for many processors;
- as well as a portable bytecode compiler;
- and an interactive execution environment.

Features include:

- First-class higher-order functions;
- Objects, classes, multiple inheritance;
- Parametric polymorphism, exceptions;
- Records, variants, and general algebraic datatypes.
Simple statements

```ocaml
# let x = 3 in x+x;;
- : int = 6

# let square x = x*x;;
val square : int -> int = <fun>

# let rec factorial n = if n < 1 then 1 else n*(factorial(n-1));;
val factorial : int -> int = <fun>

# factorial (square 3);;
- : int = 362880
```
Type constructions

(”Monday”,9,10) : string * int * int

[ 2. ; 2.5 ; 3. ] : float list

[| ’a’; ’b’ |] : char array

fun x y -> x+y/2 : int -> int -> int

type day = { month:string; date:int }
{ month = ”Jan”; date = 14 } : day

type shape = Circle of int | Rectangle of int*int

type ’a tree = Node of ’a | Leaf
A *region quadtree* is a structure for representing two-dimensional data, such as images. Where the data is constant across large areas it can be more space-efficient than the comparable two-dimensional array.

```
type quadtree = Clear
| Black | White | Red | Green | Blue
| Tree of quadtree * quadtree * quadtree * quadtree
```

```
type picture = { title : string; image: quadtree }
```
let rec isclear : quadtree -> bool
    = fun qt ->
        match qt with
        Clear -> true
        | Tree (a,b,c,d) -> isblank a && isblank b && isblank c && isblank d
        | _ -> false

(* nonblank : picture -> bool *)
let nonblank pic = not (isclear pic.image)
let rec chop : int -> quadtree -> quadtree
    = fun n qt ->
      if n <= 0 then Clear
      else
        match qt with
          Tree (a,b,c,d) -> Tree (chop (n-1) a, chop (n-1) b,
                                   chop (n-1) c, chop (n-1) d)
          | colour -> colour

(* simpler : picture -> picture *)
let simpler { title = t; image = i } = { title = t; image = chop 5 i }

(* summary : pictures list -> picture list *)
let summary pics = List.map simpler (List.filter nonblank pics)
Homework

By the next lecture, on Thursday:

- Test out the Java array subtyping example, and confirm that (a) it compiles, and (b) there is a type error when run.
- Read Gilad Bracha’s articles on his blog “Computational Theology” about Java type annotations and the idea of pluggable types.
- Read the Java fable Execution in the Kingdom of Nouns.

If you are uncertain about OCaml programming, try these online guides:

- The Objective Caml Tutorial
- Chapter 1 of OCaml for Scientists
- Developing Applications with Objective Caml
- For those who already know Standard ML, Andreas Rossberg has written a handy conversion guide.
Languages use types and type systems for several reasons.

- A *type system* is a syntactically defined subset of programs which are certain to have some desired property.

- Objective Caml (OCaml) is a functional programming language with a rich type system.

- We saw some example OCaml code for manipulation quadtrees, a 2-dimensional data representation.