Loudon

Data Types

6 Data Types (1)

- data type = (finite) set of values
  - theoretically ∞

ex: int $\in$ −2147483648 ... +2147483647

spec: - explicit enumer.
- subrg of other
- mathematical

subrange

- data type = set of values + set of operations on them
- type equivalence = name | structural
- kinds — strong vs weak typing
  - dynamic | static
  - untyped.

6.2 • Simple (primitive) types.

- predefined: int, float, ~~enum~~
  - short, long. (*types, *types)
  - 6..10
  - FIXED BIN(26)
- enum, subrg
- ordinal types (vs) IEEE754

6.3 • Type Constructors

— Cartesian product

- tuples or records
- new type or compatible w/ existing

(a, b, c) $\{a : T_1; b : T_2; c : T_3\}$

Louden
6 DataTypes (2)

struct Foo { int a; real b; }
class - is just a struct

Foo f → selector operation
f.a or f→a

name equiv
struct equiv — names in diff order
move corresponding.

tuples (products)

record type without names.
ocaml → (3, "abc", 6.8) of type (int * string * real)

- strict structural equiv, positional

Unions (sums)

```
union IR {
    int i
    real r
}
```

discriminated union

```
struct IR {
    enum {Int, Real} which
    union { int i;
            real r
    } val
}
```

```
type ir = Int of int | Real of real | Error;
```

C: union is weak
Ocaml: union (sum) is strong.

data ctors

- alt to OO.

Louden
6 Data types (3)

tag select:

```
switch (s.which) {
    case Int : f(s.val.i);
    case Real : g(s.val.r);
}
```

```
match s with
    | Int i  -> f i
    | Real r -> g r
    | Error  -> raise XXX;
```

Ocaml

Subsets

```
subtype Short = int range 0..255;
```

(Pascal, Ada)

Arrays & Functions

mapping $f : U \to V$

array: indextype → component type.

index: arbit type | only int

lwb..upb or 0..dim-1

array dim: - part of type
- compile time
- creation time
- dynamic. (Perl)

storage: row-major
column-major

multidim: - ragged array of array
- multidim?

Louden
6 Datatypes (4)

slices a[i:j]
substrings a[i:j, k:l]
a[i|len]

substring → start, len
first, last
first, last+1.

Functions

let f x = x + 1;;
⇒ f : int → int

- usu. structural equiv

typedef double (*realfn)(double);
realfn f = sqrt;

fn const vs var. (1st class?)
C – no
ocaml – yes

let f = function x → x + 1;;

integrate (sqrt, 1.0, 10.0)
integrate (function x → x*x, 0.0, 10.0)

Java: ~~can't~~ no functions at all.
can't even do C.

BUT: ~~class~~
→ abstract class Realfn {
double fn(double);
}

or interface

Pointers / Recursive Types

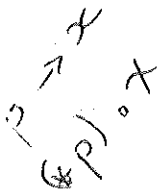
Louden
6 Datatypes (5)

```
typedef int* ptr;
typedef struct tree* treeptr
struct tree{int x; treeptr left, right;}
```

explicit deref

C: p->x vs s.x (p->x (*p).x

Java p.x



C++: ptrs vs references.

can't have ∞ types.

```
type Tree = Nil | Node of int * Tree * Tree;
```

Parameterized?

C: void*

Java: Object

C++: template<class T>

Ocaml:

```
type 'a Tree = Nil | Node of 'a * Tree * Tree;
```

~~let t = int Tree~~

```
# let t = Node(6, Nil, Nil)
  t : int Tree = 6 * Nil * Nil
```

```
type 'a option = None | Some of 'a
```

Louden
6 Datatypes (6)

Type Equiv

structural

name

anonymous types.

Type Checking

- dynamic vs static.
- strong vs weak
- Scheme: type inference = optimization
- Ocaml: always static
  but inference

```
let f n = match n with
        | 0 -> 1
        | n when n>0 -> n* f(n-1)
        | raise error;
```

inference ⟹ unification

C++ / static / dynamic / can be reinterpreted.

Type Compat

assignment compat

e1 = e2 is OK?

lvalue, rvalue

- auto dereferencing:

not Ocaml x := !x + 1

C: *p = *p + 1

↑ only on level.

C++ refs p = p + 1

Louden
Ch Data types (7)

Implicit Types

Fortran: I-N → int
Else real

C: f(m1x); → ret int

Ocaml: inference (no défaut)

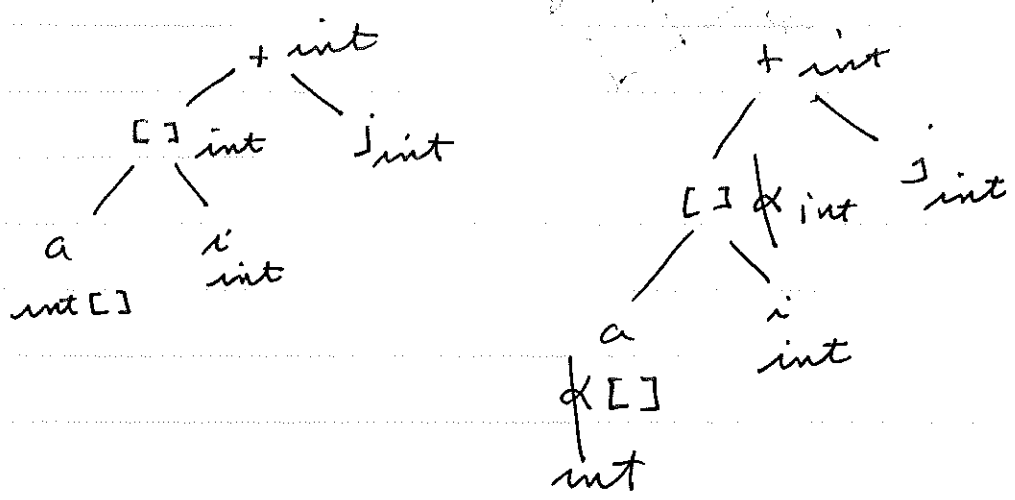
Type Conversion

implicit? prohibited?
widening, narrowing int ⇄ real
base class ⇆ derived?
explicit with cast
or conv fn.
casts: static
dynamic
reinterpret (unsafe)

char ⊂ int?
bool ⊂ int?

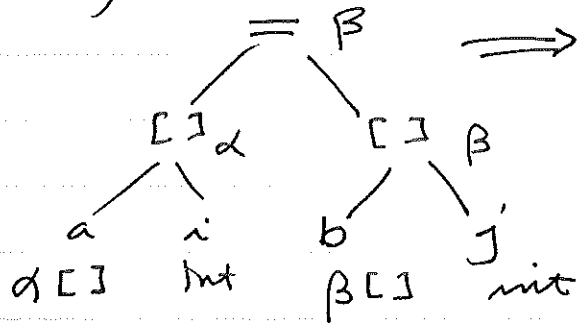
Polymorphic Type Checking

non-explicit types



Louden
6 Datatypes (8)

unification



$\Longrightarrow \alpha \equiv \beta$ but what type.

Hindley-Miller

polymorphic = poly(many) morphic(forms)

politics?

@lists – parametric polymorphism
overloading – ad hoc polymorphism.
("toward this"
subtype polymorphism — overriding
monomorphic.

let swap f x y = f y x;

~~swap : ('a → 'b → 'c) → 'b~~

⇒ ('a → 'b → 'c) → 'b → 'a → 'c

?

Explicit Polymorph

type 'a stack = EmptyStack | Stack of 'a * ('a stack)

user-defined type ctors

type ctor = fn from type t type

Louden
6 Datatype (9)

```
let swap f x y = f y x;;
  ('a -> 'b -> 'c) -> 'b -> 'a -> 'c
(-) 3 4;;
   -1;
swap (-) 3 4;;
     +1
let rsub = swap (-);
     int -> int -> int
(-);
   int -> int -> int
rsub 5 6;
     1;
let sub1 = swap (-)
let f = (-) 3;;
     int -> int
f 5;
  -2
sub1 6;;
     5
```

let compose f g x = f (g x)

6.9 Explicit Polymorphism — Louden 6 Datatype (10)

- need to define data structs.

Void *

Object

```
type 'a Stack = Empty | Stack of 'a * ('a Stack)
let empty = Empty;;
let x = Stack (3, Empty);;
```

▷ Templates in C++

```
template<class T>
T max(T x, T y) { ret x > y ? x : y }
```

↑ error if > not def.

~~template<class T>~~