Types, Part I

Programming Languages
CS 214

Types

A *type* is set V of values, and a set O of operations onto V.

Examples from C++:

- The *int* type:

  V = {INT_MIN, ... -1, 0, 1, ..., INT_MAX-1, INT_MAX}

  O = {<<, >>, +, -, *, /, %, =, ++, --, ...}

- The *char* type:

  V = {NUL, ..., '0', …, '9', …, 'A', ..., 'Z', ..., ‘a', ..., 'z', DEL}

  O = {<<, >>, =, ++, --, isupper(), islower(), toupper(), … }

- The *string* type:

  V = {"", "A", "B", "C", ..., "AA", "AB", "AC", ..., "AAA", ...}

  O = {<<, >>, +, +=, [], find(), substr(), … }

Fundamental Types

Let's assume the existence of some basic types:

Name	V	C++	Ada	Smalltalk	Lisp
bool	false, true	*bool*	*boolean*	*Boolean*	*boole*
char	the set of chars	*char*	*character*	*Character*	*character*
int	the integers	*int*	*integer*	*Integer*	*integer*
real	the reals	*double*	*float*	*Float*	*real*

Given these, new types can be created via *type constructors*.

Each constructor has 3 components:

- The syntax used to denote that constructor;
- The set of elements produced by that constructor; and
- The operations associated with that constructor.

Set Constructor I: *Product*

The product constructor is the basis for A cross B (*A x B)*.

- The product of two sets A and B is denoted A x B.

- A· B consists of all ordered pairs (a, b): $a \in A$, $b \in B$.

  A· B· C consists of all ordered triples (a, b, c): $a \in A$, $b \in B$, $c \in C$.

  A· B· … · N consists of all ordered n-tuples (a, b, …, n):
  $a \in A$, $b \in B$, …, $n \in N$.

  Example: the set bool x char has 256 elements:
  { …, (true, 'A'), (false, 'A'), (true, 'B'), (false, 'B'), …, }.

- Operations associated with product are the *projection* operations:
  first, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns s1.
  second, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns s2.
  nth, applied to an n-tuple $(s_1, s_2, \ldots, s_n)$ returns sn.

Product Example: C++ structs

```
struct Student
{
   int id;
   double gpa;
   char gender;
};
Student aStudent;
```

Formally, a Student consists of:

Int x real x char

Formally, a particular Student:

```
aStudent.id = 12345;
aStudent.gpa = 3.75;
aStudent.gender = 'F';
```

is the 3-tuple: (12345, 3.75, ‘F’).

The C++ “dot operator” is a projection operation:

```
cout << aStudent.id          // extract id
     << aStudent.gpa         // extract gpa
     << aStudent.gender      // extract gender
     << endl;
```


Set Constructor II: *Function*

The function constructor is the basis for *subprograms*.

- The set of all functions from a set A to a set B is denoted (A) → B.
- A particular function *f* mapping A to B is denoted *f(A)* → *B* .

Examples:

- The set (char) → bool contains all functions that map char values into bool values, some C examples of which include:

```
IsUpper('A') → true          isLower('A') → false
IsAlpha('A') → true          isDigit('A') → false
isAlnum('A') → true          isSpace('A') →  false
```

- The set (char) → char contains all functions that map char values into char values, some C examples of which include:

```
toLower('A') → 'a'           toUpper('a') → 'A'
```


Function and Product

What does this set contain? (int · int) → int

– All functions that map pairs of integers onto an integer.

Examples?

```
+((2,3)) -> 5          -((2,3)) -> -1
*((2,3)) -> 6          /((2,3)) -> 0
```

Suppose we define an aggregate named *IntPair*:

```
struct IntPair {
   int a,
       b;
};
```

and then define a function named add():

```
int add(IntPair ip) {
  return ip.a + ip.b;
};
```

add() is a member of the set: (int x int) → int

– The function constructor let us create new operations for a type.

Function Arity

Product serves to denote an aggregate or an argument-list.

What does this set contain? (int · int) → bool

– All functions that map pairs of integers onto a boolean.

Examples? `==, !=, <, >, <=, >=...`

Definition:

The number of operands an operation requires is its *arity*.

– Operations with 1 operand are *unary* operations, with *arity-1*.

– Operations with 2 operands are *binary* operations, with *arity-2*.

– Operations with 3 operand are *ternary* operations, with *arity-3*.

– ...

Example Ternary Operation

The C/C++ conditional expression has the form:

```
<expr>0 ? <expr>1 : <expr>2
```

producing <expr>$_1$ if <expr>$_0$ is true, and producing <expr>$_2$ if <expr>$_0$ is false.

Here is a simple *minimum()* function using it:

```
int minimum(int first, int second) {
  return (first < second) ? first : second;
};
```

The C/C++ conditional expression is a ternary operation, which in this case is a member of the set:

$$?:(\text{bool} \times \text{int} \times \text{int}) \rightarrow \text{int}$$

Operator Positioning

Operators are also categorized by their position relative to their operands:

- *Infix* operators appear *between* their operands: `1 + 2`
- *Prefix* operators appear *before* their operands: `+ 1 2`
- *Postfix* operators appear *after* their operands: `1 2 +`

`* + 2 3 - 4 2   ≡≡  (2+3) * (4-2)   ≡≡ - 2 3 + 4 2 - *`

Prefix, infix, and postfix notation are different conventions for the same thing; a language may choose any of them:

C++ Expr	Category	Value	Lisp Expr	Category	Value
x < y	Binary, infix	true, false	(< x y)	Binary, prefix	true, false
++x	Unary, prefix	x+1	(incf x)	Unary, prefix	x+1
11 + 12	Binary, infix	23	(+ 11 12)	Unary, prefix	23
!*flag*	Unary, prefix	neg. of *flag*	(not *flag*)	Unary, prefix	neg. of *flag*
cout << x	Binary, infix	cout	(princ x *str*)	Binary, prefix	x
x++	Unary, postfix	x	None		

Set Constructor III: *Kleene Closure*

Kleene Closure is the basis for representing *sequences*.

- The Kleene Closure of a set A is denoted A*.
- The Kleene Closure of a set is the set of all tuples that can be formed using elements of that set.

  Example: The Kleene Closure of bool -- bool* -- is the infinite set:

  `{ (), (false), (true), (false, false), (false, true), (true, fa`
  `}`

- For a tuple t ∈ A*, the operations include:

null(A*) → bool

```
null(())          → true
null((false))     → false
null((true))      → false
```

first(A*) → A

```
first((true, false))   → true
first((false, true))   → false
```

rest(A*) → A*

```
rest((true, true, false)) → (true, false)
rest((false, true, true)) → (true, true)
```


Kleene Closure Examples

If *char* is the set of ASCII characters, what is *char** ?

– The infinite set of all tuples formed from ASCII characters. (AKA the set of all Strings).

The C/C++ notation: "Hello"
is just a different syntax for: ('H', 'e', 'l', 'l', 'o')

Thus, *int** denotes a sequence (array, list, …) of integers;

```
int intStaticArray[32];
int * intDynamicArray = new int[n],
vector<int> intVec;
list<int> intList;
```

*real** denotes a sequence (array, list, …) of reals; and so on.

Sequence Operations

Sequence operations can be built via *null()*, *first()*, and *rest()*

- An output operation can be defined like this (pseudocode):

```
void print(ostream out, int * a) {
  if ( !null(a) ) {
    out << first(a) << ' ';
    print(out, rest(a));
  }
};
```

- A subscript operation can be defined like this (pseudocode):

```
char & operator[](int * a, int i) {
  if (i > 0)
    return operator[](rest(a), i-1);
  else
    return first(a);
};
```

In Lisp:

first is called *car*

rest is called *cdr*.

Practice Using Constructors

Give formal descriptions for:

- The *logical and* operation (&&):
  - How many operands does it take?
  - What types are its operands?
  - What type of value does it produce?

  So && is a member of

<Takes 2 operands
- Uses bool, bool operands
- Produces a bool
- Therefore is a member of (bool x bool) -> bool>

- The C++/*STL substring* operation (*str*.substr(*i*,*n*)):
  - How many operands does it take?
  - What types are its operands?
  - What type of value does it produce?

  So substr() is a member of:

<Takes 3 parameters (this, i, n)
- Types are string (char*), int, int
- Produces a string (char*)
-

Therefore is a member of (string x int x int) -> string>

- For you: The *logical negation* operation (!):

More Practice

- For you: this *C++ record:*

```
struct Student {
  int myID;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

- For you: an *accessor* method:

```
struct Student {
  int myID;
  int id() const ;
  string myName;
  bool iAmFullTime;
  double myGPA;
};
```

- How does this affect our *Student* description?

More Practice (ii)

- For you: A “complete” class:

```
class Student {
 public:
  Student();
  Student(int, string, bool, double);
  int getId() const;
  string getName() const;
  bool getFullTime() const;
  double getGPA() const;
  void read(istream &);
  void print(ostream &) const;
 private:
  int    myID;
  string myName;
  bool   iAmFullTime;
  double myGPA;
};
```

Summary

A type consists of *data* and *operations*.

The set constructors:

- *product*,
- *function*, and
- *Kleene Closure*

provide a formal way to represent types:

→ Use the *product* and *Kleene closure* to represent the *data*;

→ Use the *function* constructor to represent the *operations* on the type.

