I217: Functional Programming

1. Sorts, Operators, Terms & Equations

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Preliminaries
  - Programming paradigms
  - Functional programming & CafeOBJ
  - Sets, Tuples, Functions
- Some examples
- Sorts, Operators, Variables, Terms, Equations
- Some commands and comments
- Exercises

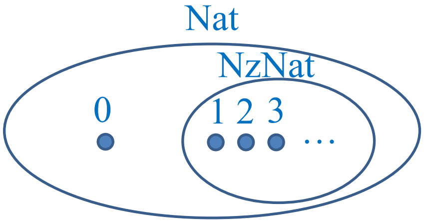
Programming Paradigms

- Imperative (procedural) programming
  - Fortran, Pascal, C, C++, Lisp, Standard ML, Ruby, Python
- Logic programming
  - Prolog, GHC, KL1, Maude, *CafeOBJ*
- Object-oriented programming
  - Smalltalk, C++, Java, Self, Scala, Ruby, ABCL/1, ConcurrentSmalltalk, MultithreadSmalltalk, Python
- *Functional programming*
  - Miranda, Haskell, Erlang, Lisp, Standard ML, Scala, Maude, *CafeOBJ*

Functional Programming & CafeOBJ

- Functional Programming
  - Programs are functions in the *mathematical* sense.
  - To execute programs is to apply functions to arguments.
  - *No destructive assignment*.
  - More amenable to *program verification*.
- CafeOBJ
  - An executable *specification* language.
  - Not a real programming language, but can be used as an educational programming language.
  - CafeOBJ programs (specifications) can be verified, namely that we can prove that programs written in CafeOBJ enjoy (desired) properties with the CafeOBJ system.

Some Example in CafeOBJ

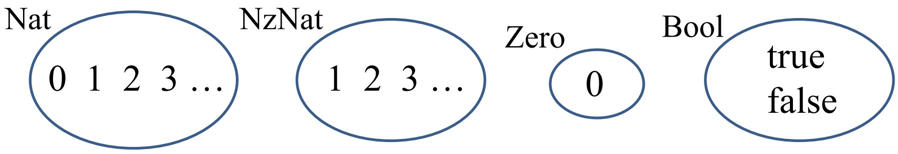



```
open NAT .
  op gcd : Nat Nat -> Nat .
  var X : Nat .
  var NzY : NzNat .
  eq gcd(X,0) = X .
  eq gcd(X,NzY) = gcd(NzY,X rem NzY) .
  -- What follows -- on a line is a comment.
  red gcd(0,0) . -- compute the gcd of 0 & 0
  red gcd(2,0) . -- compute the gcd of 2 & 0
  red gcd(0,2) . -- compute the gcd of 0 & 2
  red gcd(24,36) . -- compute the gcd of 24 & 36
  red gcd(2015,31031) . -- compute the gcd of 2015 & 31031
close
```

```
op _rem_ : Nat NzNat -> Nat .
eq X rem NzY
   = remainder of
     dividing X by NzY .
```


Sets

A set is a collection of similar things such that (1) duplication is irrelevant and (2) the enumeration order is irrelevant.



The set of natural numbers	The set of non-zero natural numbers	The set of zero	The set of Boolean values
{0, 1, 2, 3, …}	{1, 2, 3, …}	{0}	{true, false}

Because of (1) & (2), the following three sets are the same:

{0, 1, 2, 3} {3, 2, 0, 1} {1, 3, 2, 0, 2, 1}

Tuples

A tuple of length n ($\geqq 0$) (or n-tuple) is a collection that consists of n elements such that the enumeration order is relevant and the elements are not necessarily similar.

(110, true, 119) (117) (which is the same as 117)

() (the empty tuple & may be omitted)

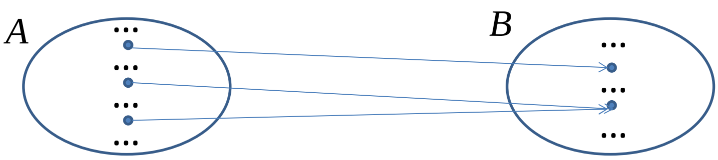
The set of n-tuples such that the ith ($1 \leqq i \leqq n$) element is from a set S_i is represented as $S_1 \times \ldots \times S_n$.

The above three tuples are elements of the following sets respectively:

Nat × Bool × Nat Nat {()} (which may be omitted)

Functions

A function f from a set A to a set B maps each element of A to an exactly one element of B.



gcd is a function from Nat × Nat to Nat.

gcd maps (0,0), (2,0), (0,2), (24,36) and (2015,31031) to 0, 2, 2, 12 and 403, respectively.

Constants, such as 0, can be regarded as functions from {()} to Nat.

Factorial

```
open NAT .
  op fact : Nat -> Nat .
  var NzX : NzNat .
  eq fact(0) = 1 .
  eq fact(NzX) = NzX * fact(p NzX) .
  --
  red fact(0) .
  red fact(1) .
  red fact(10) .
  red fact(100) .
  red fact(1000) .
  -- red fact(10000) . -- stack overflow
  -- red fact(100000) . -- stack overflow
close
```

```
op p_ : NzNat -> Nat .
eq p NzX  = the previous number of NzX .
```

```
op _*_ : Nat Nat -> Nat .
vars X Y : Nat .
eq X * Y = multiplication of X & Y .
```


Odd-Even Divide & Conquer Factorial

```
open NAT .
  op cond : Bool Nat Nat -> Nat .
  op g : Nat Nat -> Nat .
  op oedc-fact : Nat -> Nat .
  vars X Y : Nat .  var NzX : NzNat .
  -- cond
  eq cond(true,X,Y) = X .
  eq cond(false,X,Y) = Y .
  -- g
  eq g(X,Y) = cond(X > Y, g(X,2 * Y) * g(sd(X,Y),2 * Y), X) .
  -- oedc-fact
  eq oedc-fact(0) = 1 .
  eq oedc-fact(NzX) = g(NzX,1) .
--
  red oedc-fact(10000) . -- can be computed
close
```

```
op _>_ : Nat Nat -> Bool .
eq X > Y = true if so
           false otherwise .
```

```
op sd : Nat Nat -> Nat .
eq sd(X,Y)
   = symmetric difference between X & Y .
```

Fibonacci

```
open NAT .
 op fib : Nat -> Nat .
 op sfib : Nat -> Nat .
 var NzX : NzNat .
 -- fib
 eq fib(0) = 0 .
 eq fib(NzX) = sfib(p NzX) .
 -- sfib
 eq sfib(0) = 1 .
 eq sfib(NzX) = fib(NzX) + fib(p NzX) .
 --
 red fib(10) .
 red fib(20) .
 red fib(30) . -- can be computed, although it takes time.
close
```

```
op _+_ : Nat Nat -> Nat .
vars X Y : Nat .
eq X + Y = addition of X & Y .
```


Ackerman

```
open NAT .
 op ack : Nat Nat -> Nat .
 var Y : Nat .
 vars NzX NzY : NzNat .
 eq ack(0,Y) = Y + 1 .
 eq ack(NzX,0) = ack(p NzX,1) .
 eq ack(NzX,NzY) = ack(p NzX,ack(NzX,p NzY)) .
 --
 red ack(0,0) .
 red ack(1,1) .
 red ack(2,2) .
 red ack(3,2) .
 red ack(3,3) .
close
```

Sorts

Interpreted as sets and correspond to types in programming languages.

Nat, NzNat, Zero and Bool are sorts that are interpreted as the following sets respectively:

0 1 2 3 …

1 2 3 …

0

true
false

May be used as the sets as which the sorts are interpreted.

Declared by enclosing them with [and], such as

[NatList]

Operators

Interpreted as functions and declared as follows:

sorts

$$\textbf{op}\ f\ : \underbrace{\underbrace{S_1 \ \ldots\ S_n}_{\text{arity}} \text{ -> } \underset{\text{co-arity}}{S}}_{\text{rank}}\ . \qquad \text{where } n \geqq 0$$

f is interpreted as a function from $S_1 \times \ldots \times S_n$ to S.

When $n = 0$, f is called a constant of S.

op cond : Bool Nat Nat -> Nat .
op g : Nat Nat -> Nat .
op oedc-fact : Nat -> Nat .

are interpreted as functions from Bool × Nat × Nat to Nat, Nat × Nat to Nat and Nat to Nat.

May be used as the functions as which they are interpreted.

Variables

Declared as follows:

sorts

$$\textbf{var } V \;:\; S \;.$$

Multiple variables of a same sort can be declared as follows:

$$\textbf{vars } V_1 \ldots V_n \;: S \;.$$

vars X Y : Nat .
var NzX : NzNat .

X & Y are variables of Nat, and NzX is a variable of NzNat.

Terms

Constructed from operators and variables.

Have sorts.

gcd(X,0)	a term of sort Nat
gcd(X,NzY)	a term of sort Nat
gcd(NzY,X rem NzY)	a term of sort Nat
X	a term of sort Nat
0	a term of sort Zero
NzY	a term of sort NzNat
X rem NzY	a term of sort Nat

Note that a term of sort Zero is also a term of sort Nat and a term of sort NzNat is also a term of sort Nat.

Terms

Inductively defined as follows:

(1) A variable of sort S is a term of sort S.

(2) For an operator f : $S_1 \ldots S_n$ -> S, if $t_1, \ldots, t_n$ are term of sorts $S_1, \ldots, S_n$, then $f(t_1, \ldots, t_n)$ is a term of sort S.

Note that when $n = 0$, $f()$ is a term of sort S and because () may be omitted (must be omitted in CafeOBJ), f itself is a term of sort S (called a constant of sort S).

Note that terms of sort Zero or NzNat are also terms of sort Nat.

Terms

```
open NAT .
  op gcd : Nat Nat -> Nat .
  var X : Nat .
  var NzY : NzNat .
  ...
```

X is a term of sort Nat because X is a variable of sort Nat.

0 is a term of sort Nat because 0 is a term of sort Zero.

gcd(X,0) is a term of sort Nat because gcd is an operator whose rank is Nat Nat -> Nat, X is a term of sort Nat and 0 is a term of sort Nat.

Equations

eq gcd(X,0) = X .

says that for all natural numbers X, gcd(X,0) equals X.

gcd(0,0) = 0
gcd(1,0) = 1
gcd(2,0) = 2

eq gcd(X,NzY) = gcd(NzY,X rem NzY) .

says that for all natural numbers X & all non-zero natural numbers NxY, gcd(X,NzY) equals gcd(NzY,X rem NzY).

gcd(3,1) = gcd(1,3 rem 1) .
gcd(31031,2015) = gcd(2015, 31031 rem 2015) .

Equations

Declared as follows:

eq *LeftTerm* = *RightTerm* .

where *LeftTerm* and *RightTerm* are terms of a same sort.

If variables X_1, X_2, … of sorts S_1, S_2, … occur in the equation, then the equation says that for all X_1 of S_1, all X_2 of S_2, … *LeftTerm* equals *RightTerm*.

(Precisely, the least sort of *RightTerm* is the same as or a sub-sort of the lest sort of *LeftTerm*, or equivalently the least sort of *LeftTerm* is a sort of *RightTerm*. See lecture note 2 for the least sort and lecture note 3 for the condition.)

Mix-fix Operators

The operators used in 3 + 4, p 1 and 10 ! are called infix, prefix and postfix operators, which are declared as follows:

```
op _+_ : Nat Nat -> Nat .
op p_ : NzNat -> Nat .
op _! : Nat -> Nat .
```

Moreover, the operator (called a *mix-fix operator*) used in
if X > Y then { g(X,2 * Y) * g(sd(X,Y),2 * Y) } else { X }
can be used and declared as follows:

```
op if_then {_} else {_} : Bool Nat Nat -> Nat .
```


Some Commands of CafeOBJ

Command in

Programs written in CafeOBJ are saved as text files whose names have the extension .cafe, such as gcd.cafe and fact.cafe. The command takes a file name and loads the programs in it.

```
CafeOBJ> in gcd.cafe
```

The extension can be omitted.

```
CafeOBJ> in gcd
```

Some Commands of CafeOBJ

Command open & close

open takes a module (a unit of programs in CafeOBJ) and makes it available.

```
CafeOBJ> open NAT .
```

where NAT is the built-in module in which natural numbers and functions over them are described and the prompt becomes %NAT> after opening NAT. Note that a period is needed.

close makes the currently open module close and the prompt back to CafeOBJ>.

```
%NAT> close
```


Some Commands of CafeOBJ

Command red

red takes a term and computes it with the equations available.

```
%NAT> red 3 + 4 .
```

Its result is (7):NzNat, where the sort NzNat is the important part of the result.

Note that a period is needed.
Note that spaces around + are needed.

A typical mistake is to forget a period (red 3 + 4) and spaces (red 3+ 4 . or red 3 +4 . or red 3+4 .).

Some Commands of CafeOBJ

Command full-reset

It fully resets the system.

```
CafeOBJ> full-reset
```

Command ?

It displays a list of commands available.

```
CafeOBJ> ?
```


Comments

A comment starts with the first occurrence -- on a line and continues to the end of the line.

```
-- This is a comment.
```

A segment that starts with a double quotation mark and ends with a double quotation mark is also a comment.

```
"This is another way
to write a comment."
```

When you want to use a double quotation mark in the segment, a backslash should be used in front of it.

```
"You can use \"double quotation
marks\" in a comment."
```

Exercises

1. Type each piece of programs in the slides as one file, feed it into CafeOBJ, and do some testing. Note that the extension of a file name in which CafeOBJ programs are written is .cafe, such as fact.cafe.

2. Explain in which way fact(5) is computed.

3. Explain in which way oedc-fact(5) is computed.

4. Write two versions of programs computing the summation $0+1+2+\ldots+n$ for a given number n, where one corresponds to fact and the other corresponds to oedc-fact and do some testing for both versions.

Exercises

5. Write a program in CafeoBJ that corresponds to the following and do some testing

$$\text{ext-fib}(n) = \begin{cases} 0 & \textbf{if } n = 0 \\ 1 & \textbf{if } n = 1 \\ 2 & \textbf{if } n = 2 \\ \text{ext-fib}(n-1) + \text{ext-fib}(n-2) + \text{ext-fib}(n-3) & \textbf{otherwise} \end{cases}$$

You should not use sd. You should not use any user defined function that does the same thing as sd. When you use p t, you should explicitly guarantee that the least sort of t is NzNat. Hint: you can consult the program of fib, where one sub-function is used to split the natural numbers into (1) 0, (2) 1 and (3) 2 or larger ones. You can use two sub-functions to split natural numbers into (1) 0, (2) 1, (3) 2 and (4) 3 or larger ones.

Exercises

6. Revise the program in which the factorial function is defined such that instead of fact the following postfix operator is used:

op _! : Nat -> Nat .

7. Revise the program in which the odd-even divide & conquer factorial function is defined such that instead of oedc-fact and cond the following postfix and mix-fix operators are used:

op _! : Nat -> Nat .

op if_then {_} else {_} : Bool Nat Nat -> Nat .

Exercises

8. Investigate why it takes times to compute Ackerman function with even small natural numbers, such as ack(5,5) and ack(10,10).

9. Find the definition of Takeuchi function, describe it in CafeOBJ and run it with some natural numbers. Investigate why it takes time to compute Takeuchi function with even small natural numbers.

10. Write the programs found in the lecture note in another functional programming language, such as Standard ML, and run them with its processor, such as SML#.