Exercise

Class Exercise 1

(1) Every integer is greater than 0, equal to 0, or less than 0.

```
  forall[x:nat0] | x>0 or x=0 or x<0
```

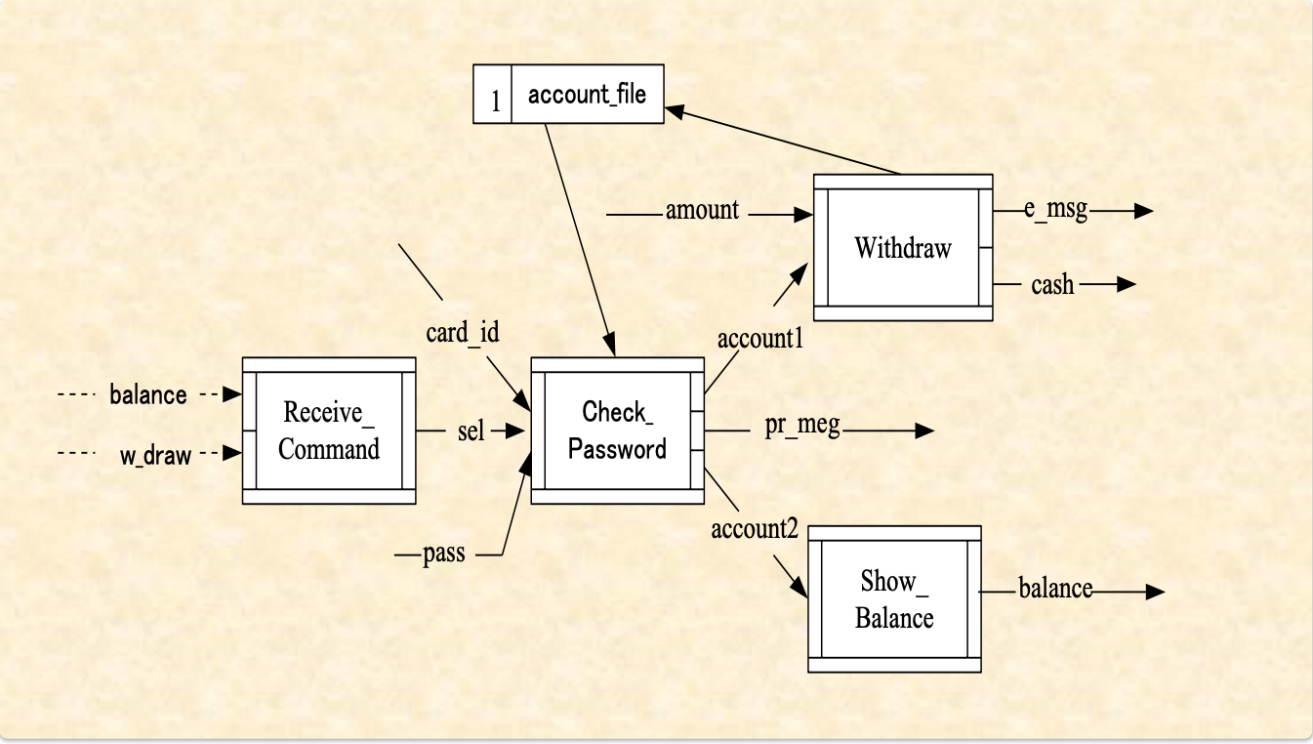
(2) For any three real numbers a, b, and c, if a is greater than b and b is greater than c, then a will be greater than c.

```
  forall[a,b,c:real] | (a>b and b>c)=>(a>c)
```

(3) For any natural number a there must exist another natural number b such that b is greater than a.

```
  forall[a:nat] exists[b:nat] | b>a
```

ATM Module



> *如何判断输入和输出是一起的还是要做选择的?*
> *看 CDFD 的左侧或者右侧的矩形是连着的还是分开的*

```
  module SYSTEM_ATM
type
```

```
        Accout = composed of
                                accout_no:nat
                                password:nat
                                balance:real
                                end
var
        ext #accout_file: set of Account;
inv
        forall[x:Account] | 1000<=x.password<=9999;
behav
        CDFD_1;

process Init()
end process;

process Receive_Command(balance:sign| w_draw:sign) sel:bool
post
        bound(balance) and sel=true or
        bound(w_draw) and sel=false
end_process;

process Check_Password(card_id:nat,sel:bool,pass:nat) account1:Account |
pr_meg:string| account2:Account
ext rd account_file
post
        sel=false and (exists![x:account_file]|x.account_no=card_id and
x.password=pass and account1=x) or
        sel=true and (exists![x:account_file]|x.account_no=card_id and
x.password=pass and account2=x) or
        not (exists![x:account_file] | x.account_no=card_id and
x.password=pass) and pr_meg="Error"
end_process;

process Withdraw(amount:read, account1:Account) e_msg:string|cash:real
ext wr account_file
pre accout1 inset account_file
post
        (exists[x:account_file] | x=account1 and x.balance>=amount and
cash=amount) and account_file=union(diff(~account_file, {account1}),
{modify(account1, balance->accout1.balance-amount)}) or
        not exists[x:account_file] | x=account1 and x.balance >= amount
```

```
and e_meg="Too big"
end process;

process Show_Balance(account2: Account) balance:real
post balance=account2.balance;
end_process;
end_module;
```

Class Exercise 2

1. Define a calculator as a module. Assume that reg denotes the register that should be initialized to 0 and accessed by all the operations defined. The operations include Add, Subtract, Multiply, and Divide. Each operation is modeled by a process.

```
  module Calculator
var
        reg:real;

process Init()
ext wr reg:real
pre bound(reg)
post reg=0
end_process;

process Add(x:real)
ext wr reg:real
pre bound(x)
post reg=~reg+x
end_process;

process Multiply(x:real)
ext wr reg:real
pre bound(x)
post reg=~reg*x
end_process;

process Divide(x:real)
ext wr reg:real
pre x<>0 and bound(x)
post reg = ~reg/x
```

```
end_process;
end_module;
```

Class Exercise 3

Write both the explicit and implicit specifications for the function Fibonacci:
Fibonacci(0) = 0;
Fibonacci(1) = 1;
Fibonacci(n) = Fibonacci(n - 1) + Fibonacci(n - 2)
where n is a natural number of type nat0.

```
  function Fibonacci(n:nat):nat
post
        if n<=1
        then Fibonacci=n
        else Fibonacci=Fibonacci(n-1)+Fibonacci(n-2)
end_function;
```

Class Exercise 4

Assume that the courses to teach on weekdays are: "Software Engineering" on Monday, "Program Design" on Tuesday, "Discrete Mathematics" on Wednesday, "Programming Language" on Thursday, and "Formal Engineering Methods" on Friday. Write a formal specification for the process that gives the corresponding course title for an input weekday.
(Hint: define a type Course as an enumeration type)

```
  process CorrespondingCourse(weekday:{<Mon>, <Tue>, <Wed>,<Thu>,
<Fri>})title:{<C1>,<C2>,<C3>,<C4>,<C5>}
post
        title=case weekday of
                        <Mon>-> <C1>;
                        <Tue>-> <C2>;
                        <Wed>-> <C3>;
                        <Thu>-> <C4>;
                        <Fir>-> <C5>;
                        end_case;
end_process;
```

Class Exercise 5

1. Let s1 = {5, 15, 25}, s2 = {15, 30, 50}, s3 = {30, 2, 8}, and s = {s1, s2, s3}. Evaluate the expressions:

```
  a. card(s1)=3
b. card(s)=3
c. union(s1, s2)={5,15,25,30,50}
d. diff(s2, s3)={15,50}
e. inter(union(s2, s3), s1)={15}
f. dunion(s)={5,15,25,30,50,2,8}
g. dinter(s)={}
h. inter(union(s1, s3), diff(s2, union(s1, s3))={}
```

2. Construct a module to model a telephone book containing a set of telephone numbers. The necessary processes are Add, Find, Delete, and Update. The process Add adds a new telephone number to the book; Find tells whether a given telephone number is available or not in the book; Delete eliminates a given telephone number from the book; and Update replaces an old telephone number with a new number in the book.

```
  module telephone_book
type
        TelephoneNumber = given;
var
        telephone_file: set of TelephoneNumber;
behav: CDFD_1;

process Add(t:TelephoneNumber)
ext wr telephone_file
pre t notin ~telephone_file
post telephone_file=union(~telephone_file, t)
end_process;

process Find(t:TelephoneNumber) r:bool
ext rd telephone_file
post r=t inset telephone_file
end_process;

process Delete(t:TelephoneNumber)
ext wr telephone_file
post telephone_file=diff(~telephone_file, t)
end_process;

process Update(t_old, t_new: TelephoneNumber)
ext wr telephone_file
```

```
pre t_old inset ~telephone_file
post telephone_file=union(diff(~telephone, t_old), t_new)
end_process;
end_module;
```

3. Write a specification for a process Merge. The process takes two groups of students and merge them into one group. Since the merged group will be taught by a different professor, the students from both groups may drop from the merged group (but exactly which students will drop is unknown).

```
  process Merge(s1: Class, s2: Class) merged_class: Class
pre ture
post subset(merged_class, union(s1, s2))
end_process;
```

Class Exercise 6

1. Given a set T = {1, 2, 5}, declare a sequence type based on T, and list up to 5 possible sequence values in the type.

```
  A = seq of T
```

2. Evaluate the sequence comprehensions:

```
  a.[x|x:nat &3<x<8]=[4,5,6,7]
b. [y | y: nat0 & y <= 3]=[0,1,2,3]
c. [x * x | x: nat, y: nat & 1 <= x <= 3]=[1,4,9]
```

3. Let s1 = [5, 15, 25], s2 =[15, 30, 50], s3 = [30, 2, 8], and s = [s1, s2, s3]. Evaluate the expressions:

```
  a. hd(s1)=5
b. hd(s)=s1
c. len(tl(s1)) + len(tl(s2)) + len(tl(s3))=6
d. len(s1) + len(s2) - len(s3)=3
e. union(elems(s1), elems(s2))={5,15,25,30,50}
f. inter(union({hd(s2)}, elems(s3)), elems(s1))={15}
g. union(inds(s1), inds(s2), inds(s3))={1,2,3}
h. elems(conc(s1, s2, s3))={5,15,25,30,50,2,8}
i. dconc(s)=[5,15,25,15,30,50,30,2,8]
```

4. Construct a module to model a queue of integers with the processes: Append, Eliminate, Read, and Count. The process Append adds a new element to the

queue; Eliminate deletes the top element of the queue; Read tells what is the top element; and Count yields the number of the elements in the queue.

```
  module Queue
type
        Q = given;
var
        q:seq of Q;

process Append(s:Q)
ext wr q
post q = conc(~q, [s])
end_process;

process Eliminate()
ext wr q
post q = ~q(1,len(~q)-1)
end_process;

process Read()
ext rd q
post q(len(q))
end_process

process Count()
ext rd q
post len(q)
end_process;
end_module;
```

Class Exercise 7

1. Let a = mk_Account(010, 300, 5000), where the type Account is defined as follows:
   Account = composed of
   account_no: nat1
   password: nat1
   balance: real
   end
   Then evaluate the expressions:

```
  a. a.account_no = 010
b. a.password = 300
c. a.balance = 5000
d. modify(a, password -> 250) = mk_Accout(010,250,5000)
e. modify(mk_Account(020, 350, 4050), account_no -> 100,
balance -> 6000) = mk_Account(100, 350, 6000)
```

3. Let x be a variable of the type Date defined as follows: Date = nat0 *nat0* nat0 Let x = mk_Date(2002, 2, 6). Then evaluate the expressions:

```
  a. x(1) = 2002
b. x(2) = 2
c. x(3) = 6
d. modify(x, 1 -> 2003)=mk_Date(2003,2,6)
e. modify(x, 2 -> 5, 3 -> 29)=mk_Date(2002,5,29)
f. modify(x, 1 -> x(1), 2 -> x(2))=mk_Date(2002,2,6)
```

4. Define a composite type Student that has the fields: name, date_of_birth, college, and grade. Write specifications for the processes: Register, Change_Name, Get_Info. The Register takes a value of Student and adds it to the external variable student_list, which is a sequence of students. Change_Name updates the name of a given student with a new name in student_list. Get_Info provides all the available field values of a given student in student_list.

```
  type
       Student = composed of
                              name: string
                              date_of_birth: nat0*nat0*nat0
                              college: string
                              grade: nat
                         end

process Register(s:Student)
ext wr student_list:seq of Student
post student_list = conc(~student_list, [s])
end_process;

process Change_Name(s:Student, new_name:string)
ext wr student_list:seq of Student
pre exists[i:inds(student_list)]|student_list(i)=s
post
       len(student_list) = len(~student_list) and
```

```
        forall[i:inds(~student_list)]|
        (~student_list(i)=s => student_list(i)=modify(~student_list(i),
1->name)) and
        (~student_list(i)<>s => student_list(i)=~student_list(i)))
end_process;

process Get_Info(s:Student) name:string, date_of_birth:nat0*nat0*nat0,
college:string, grade:nat
ext rd student_list:seq of Student
pre exists[i:inds(student_list)]|student_list(i)=s
post
        exists[i:inds(student_list)]|student_list(i)=s and
        name=s.name and date_of_birth=s.date_of_birth and
college=s.college and grade=s.grade
end_process;
```

Class Exercise 8

> *Domrt (s1, m1): s 1 与 m1 的 key 的交集在 m1 中的映射关系*
> *rngrt (m 1，s 1): s 1 与 m1 的 value 的交集在 m1 中的映射关系*

> *Domrb (s 1, m 1) : m 1 的 key 与 s1 的差集在 m1 中的映射关系*
> *rngrb (m 1，s 1)： m 1 的 value 与 s1 的差集在 m1 中的映射关系*

> *Override (m 1, m 2): 拿到所有的 key，然后 value 取值优先 m2，其次 m1*

1. Let m1 and m2 be two maps of the map type from
   nat0 to nat0;
   m1 = {1 -> 10, 2 -> 3, 3 -> 30},
   m2 = {2 -> 40, 3 -> 1, 4 -> 80}, and s = {1, 3}.

Then, evaluate the expressions:

```
  a. dom(m1) = {1,2,3}
b. dom(m2) = {2,3,4}
c. rng(m1) = {10,3,30}
d. rng(m2) = {40,1,80}
e. domrt(s, m1) = {1->10,3->30}
f. domrt(s, m2) = {3->1}
g. rngrt(m1, s) = {2->3}
h. rngrt(m2, s) = {3->1}
i. domrb(s, m1) = {2->3}
j. domrb(s, m2) = {2->40,4->80}
```

```
k. rngrb(m1, s) = {1->10,30->30}
l. rngrb(m2, s) = {2->40,4->80}
m. override(m1, m2) = {2->40,3->1,4->80,1->10}
n. override(m2, m1) = {1->10,2->3,3->30,4->80}
o. inverse(m1) = {10->1,3->2,30->3}
p. inverse(m2) = {40->2,1->3,80->4}
q. comp(m1, m2) = {2->1,3->1}
r. comp(m2, m1) = {3->10}
s. m1 = m2 <=> false
t. m1 <> m2 <=> true
```

2. Define BirthdayBook as a map type from the type Person (with the fields: id, name, and age) to the type Birthday, and specify the processes: Register, Find, Delete, and Update. All the processes access or update the external variable birthday_book of the type BirthdayBook. The process Register adds a person's birthday to birthday_book. Find detects the birthday for a person in birthday_book. Delete eliminates the birthday of a person from birthday_book. Update replaces the wrong birthday registered in birthday_book with a correct birthday.

```
  type
        Person = composed of
                                id: string
                                name: string
                                age: nat0
                        end
        Birthday = nat0*nat0*nat0
        BirthdayBook = map Person to Birthday
process Register(p:Person, b:Birthday)
ext wr bir_book:BirthdayBook
post bir_book = override(~bir_book, {p->b})
end_process;

process Find(p: Person) b:Birthday
ext rd bir_book:BirthdayBook
pre bound(bir_book(p))
post b=bir_book(p)
end_process;

process Delete(p: Person)
ext wr bir_book:BirthdayBook
```

```
pre bound(~bir_book(p))
post bir_book = domrb(~bir_book, {p})
end_process;

process Update(p:Person, b1:Birthday, b2:Birthday)
ext wr bir_book:BirthdayBook
pre p inset dom(~bir_book) and bir_book(p) = b1
post bir_book = override(~bir_book, {p->b2})
end_process;
```

Class Exercise 9

1. Define a union type School with the constituent types ElementarySchool, JuniorHighSchool, HighSchool, and University, assuming that all the constituent types are given types.

```
  type
        School = EleSchool | JuniorHighSchool | HighScholl | University
```

2. Let s1 and s2 be two variables of the type set of Hybrid. Let s1 = {<Red>, 3, 'b'} and s2 = {<Blue>, 'a', 'b', 9}. Evaluate the expressions:

```
  a. card(s1) = card(s2) <=> false
b. union(s1, s2) = {<Red>, 3, 'b', <Blue>, 'a', 9}
c. inter(s1, s2) = {'b'}
d. diff(s1, s2) = {<Red>, 'b'}
```

Class Exercise 10

Answer the questions:
a. what is a hierarchy of CDFDs?

b. what is a hierarchy of modules?

c. what is the relation between module hierarchy and CDFD hierarchy?

d. what is the relation between a CDFD and its high level process in a CDFD hierarchy?

e. what does it mean by saying that modules M1 and M2 are relative modules?

f. what is the scope of a variable, type identifier, constant identifier, invariant, function, and a process?

Class Exercise 11

Derive the pre-assertion for each of the following two assignment statements based on their post-assertion:

```
  (1)
{x>0}
{x+y>y}
x: = x + y;
{x > y}
```

```
  (2)
{x=2 and (x+y*(x+5))*x=10}
y := x + y * (x + 5)
{x = 2 and y * x = 10}
```

Class Exercise 12

Prove the validity of the following Hoare triple:

```
  {x = 5 and y = 2}
x: = x * y;
y := x + y * (x + 5)
{x = 10 and y = 40}
```

```
  {x=5 and y=2}
{x*y=10 and y=2}
{x*y=10 and 10+y*10+y*5=40}
{x*y=10 and x*y+y*(x*y+5)=40}
x:=x*y
{x=10 and x+y*(x+5)=40}
y:=x+y*(x+5)
{x=10 and y=40}
```

Class Exercise 13

Prove the validity of the following Hoare triple:

```
  {x > 0}
if x > y + 10
then x := y + 20 else y := x + 10
{x = y + 20 v x < y}
```

Proof:

```
  {x>0 and x>y+10}
X:=y+20
{x=y+20 or x<y}
==>
{y+20=y+20 or y+20<y}
X:=y+20
{x=y+20 or x<y}

X>0 and x>y+10 => y+20=y+20 or y+20<y
==>
{x>0 and x>y+10}
X:=y+20
{x=y+20 or x<y}
```

```
  {x>0 and x<=y+10}
Y:=x+10
{x=y+20 or x<y}
==>
{x=x+10+20 or x<x+10}
Y:=x+10
{x=y+20 or x<y}

x>0 and x<=y+10 => x=x+30 or x<x+10
==>
{x>0 and x<=y+10}
Y:=x+10
{x=y+20 or x<y}
```

```
  {x>0}
If x > y + 10
Then x := y+20
Else y := x+10
{x=y+20 or x<y}
```