Exercise solutions

Exercise 1

1.

(a)What is software life cycle?

Answer:

The process of software development and maintenance, which is usually composed of several phases: user requirements analysis and specification, design, implementation, testing, deliver and maintenance.

(b)What is the problem with informal approaches to software development?

Answer:

The requirements specifications may not be accurately and easily understood by the developers carrying out different tasks due to the ambiguity of the natural language.

(c)What are "Formal Methods"?

Answer:

Formal methods for developing computer systems embraces the two techniques: formal specification and formal verification. Both are established based on elementary mathematics, such as set theory, logics, and algebraic theory.

(d)What are the major features of formal engineering methods?

Answer:

Formal engineering methods serve as a bridge between formal methods and their applications by integrating formal methods into the entire software process. It has the following major features:

(1) Adopting specification languages that integrate properly graphical notation, formal notation, and natural language.
(2) Employing rigorous but practical techniques for verifying and validating specifications and programs.
(3) Advocating the combination of prototyping and formal methods.
(4) Supporting evolution rather than strict refinement in developing specifications and programs.
(5) Supporting techniques for constructing, understanding, and modifying specifications.

(e)What is SOFL?

Answer:

SOFL, standing for Structured Object-Oriented Formal Language, is a formal engineering method. It provides a formal but comprehensible language for both

requirements and design specifications, and a practical method for developing software systems.

2. Explain the role of specification in software development.

Answer:

The specification tells “what to do” without the need to tell “how to do it”.

3. Give an example of using the similar principle of “formal methods” to build other kinds of system rather than software systems.

Answer:

For example, building a house usually requires the precise design specification and rigorous verification during and after the construction, although the verification is not very formal.

Exercise 2

1. Explain the notions

[a. proposition]: A proposition is a statement that can be decided to be either true or false. For example, “A tiger is an animal” is a proposition, while “Are you happy?” is not.

[b. conjunction] : A conjunction is a propositional (or predicate) expression whose principal operator is “**and**”. For example, x>5 **and** x<10.

[c. Disjunction] : A disjunction is a propositional (or predicate) expression whose principal operator is “**or**”. For example, x>5 **or** x<10.

[d. negation]: A negation is propositional (or predicate) expression whose principal operator is “**not**”. For example, **not** x > 5.

[e. implication]: An implication is a propositional (or predicate) expression whose principal operator is “=>”. For example, x > 15 => x > 10.

[f. equivalence]: An equivalence is a propositional (or predicate) expression whose principal operator is “<=>”. For example, x>10 < = > not x=10 and not x<10.

[g. tautology]: A tautology is a propositional expression that always evaluates as true. For example, x>10 **or** x<=10.

[h. contradiction]: A contradiction is a propositional expression that always evaluates as false. For example, x>10 **and** x<10.

[i. contingency]: A contingency is a propositional expression that is neither a tautology nor contradiction. For example, x>10 or x< 20.

[j. sequent]: A sequent is an assertion that a conclusion can be deduced from hypotheses and is formally expressed in the form: $P_1, P_2, ..., P_n \vdash Q$,

where P_1, …, P_n are hypotheses and Q is the conclusion.

[k. rule]: An inference rule is composed of two parts: a list of premises and a conclusion. A premise is a propositional expression, and so is a conclusion. A rule means that when the premises are true, the conclusion can be used as a true proposition.

[l. proof]: A proof is a process (or activity or evidence) to show that the conclusion can be established from its hypotheses in a sequent. A proof can be conducted by using two methods: "truth table" and "natural deduction".

2. Give a truth-table proof for the validity of each of the sequents:

a) P, Q ⊢ P and Q

Answer:

P	Q	P **and** Q
T	T	T
T	F	–
F	T	–
F	F	–

b.) P and Q ⊢ Q

Answer:

P	Q	P **and** Q	Q
T	T	T	T
T	F	F	–
F	T	F	–
F	F	F	–

c) P ⊢ P or Q

Answer:

P	Q	P **or** Q
T	T	T
T	F	T
F	T	–
F	F	–

d) P **or** Q, P => R, Q => R ⊢ R

Answer:

P	Q	R	P **or** Q	P =>R	Q => R	R

T	T	T	T	T	T	T
T	T	F	T	F	F	–
T	F	T	T	T	T	T
T	F	F	T	F	T	–
F	T	T	T	T	T	T
F	T	F	T	T	F	–
F	F	T	F	T	T	–
F	F	F	F	T	T	–

e) P ⊢ **not not**P

Answer:

P	**not** P	**not not** P
T	F	T
F	T	–

f) Q ⊢ P => Q

Answer:

P	Q	P => Q
T	T	T
T	F	–
F	T	T
F	F	–

g) P => Q, Q => P ⊢ P <=> Q

Answer:

P	Q	P => Q	Q => P	P <=> Q
T	T	T	T	T
T	F	F	T	–
F	T	T	F	–
F	F	T	T	T

3. Give a boxed proof for each of the properties:

a. P **and** (Q **and** R) ⊢ (P **and** Q) **and** R

Answer:

From	P **and** (Q **and** R)

Infer	(P **and** Q) **and** R	[**and** – ass] (h)

b. P, Q, Q => R ⊢ P **and** R

Answer:

From	P, Q, Q => R	
1	P	[**and**-elim](h)
2	Q, Q => R	[**and**-elim](h)
3	R	[=>-elim](2)
Infer	P **and** R	[**and**-intro](1, 3)

c. **not** (P **or** Q) ⊢ **not** Q

Answer:

From	**not** (P **or** Q)	
1.	**not** P **and not** Q	[**or**-deM](h)
Infer	**not** Q	[**and**-elim2](1)

d. P **or** Q ⊢ **not** (**not** P **and not** Q)

Answer:

From	P **or** Q	
1. from	P	
1.1	P **or** Q	[**or** – intro1]
1.2	**not not** (P **or** Q)	[**not**-intro](1.1)
infer	**not**(**not** P **and not** Q)	[**or**-deM](1.2)
2. from	Q	
2.1	P **or** Q	[**or** – intro2]
2.2	**not not** (P **or** Q)	[**not**-intro](2.1)
infer	**not** (**not** P **and not** Q)	[**or**-deM](2.2)
infer	**not** (**not** P **and not** Q)	[**or**-elim](h,1,2)

4. Transform each of the following propositional expressions into a disjunctive normal form:

a. P **and not** (**not** Q **and** R)

<=> P **and** (**not not** Q **or not** R)

<=> P **and** (Q **or not** R)

<=> P **and** Q **or** P **and not** R

b. P and (Q => R) <=> W

<=> P and (not Q or R) <=> W

<=> P and ((not Q or R) => W) and (W => (not Q or R)

<=> P and (not (not Q or R) or W) and (not W or not Q or R)

<=> P and ((Q and not R) or W) and (not W or not Q or R)

<=> (P and Q and not R or P and W) and (not W or not Q or R)

<=> (P and Q and not R and (not W or not Q or R) or P and W and (not W or not Q or R)

<=> P and Q and not R and not W or P and Q and not R and not Q or
P and Q and not R and R or P and W and not W or P and W and not Q or
P and W and R

<=> P and Q and not R and not W or false or false or false or P and W and not Q or
P and W and R

<=> P and Q and not R and not W or P and W and not Q or P and W and R

c. (P or Q) and (R or W)

<=> (P or Q) and R) or (P or Q) and W)

<=> P and R or Q and R or P and W or Q and W

d. not (P => Q) or (not P and Q)

<=> not (not P or Q) or (not P and Q)

<=> P and not Q or not P and Q

e. P <=> Q and Q <=> R

<=> P => Q and Q => P and Q => R and R => Q

<=> (not P or Q) and (not Q or P) and (not Q or R) and (not R or Q)

<=> ((not P or Q) and not Q or (not P or Q) and P) and
((not Q or R) and not R or (not Q or R) and Q)

<=> (not P and not Q or Q and not Q or not P and P or or Q and P) and
(not Q and not R or R and not R or not Q and Q or R and Q)

<=> (not P and not Q or Q and P) and (not Q and not R or R and Q)

<=> (not P and not Q or Q and P) and not Q and not R or
(not P and not Q or Q and P) and R and Q

<=> not P and not Q and not Q and not R or Q and P and not Q and not R or
not P and not Q and R and Q or Q and P and R and Q

<=> not P and not Q and not R or P and R and Q

Exercise 3

1.Answer the following questions:

a.what is the similarity and difference between a predicate and a function?

Answer: a predicate is a function, but its range must be the boolean type **bool** = {**true**, **false**}.

b.what is the difference between a unversially quantified expression and existentially quanitifed expression?

Answer: A universially quantified expression describes that a property must be satisfied by every element of a given set, while an existentially quantified expression defines that a property must be satified by some elements of a given set.

c.what is a substitution?

Answer: Substitution is an operation that changes a predicate by substituting a variable or expression for a free variable in the predicate.

d.what is a valid predicate?

Answer: A valid predicate is a predicate that evaluates as true for whatever values of the free variables involved.

e.what is a satisfiable predicate?

Answer: A satisfiable predicate is a predicate that evaluates as true for some values of the free variables involved.

f.what is a partial predicate?

Answer: A partial predicate is a predicate that does not yield any truth value for some elements in its domain.

2. Tell which of the following quantified predicate expressions are propositions.

a. **forall**[x: **int**] | x > 5 **and** x < 10

Answer: yes

b. **exists**[x: **int**] | y > x **and** y < x + 10

Answer: no (because of the free variable y)

c. **forall**[x, y: **real**] | x + y > x – y

Answer: yes

d. **forall**[x, y: **real**]**exists**[z: **real**] | x + y > z

Answer: yes

e. **exists**[x: **int**]**forall**[y: **int**] | x * y > z

Answer: no (because of the free variable z)

3. Evaluate the substitutions.

a.(x > y + z => y < x)[t/x]

Answer: (t > y + z => y < t)

b.(**forall**[x, y: **nat0**] | x < z **and** z < y => x < y)[m/y, t/z]

Answer: **forall**[x, y: **nat0**] | x < t **and** t < y => x < y)

c.(**exists**[x, y, z: **nat**] | x * y > z => x > z **and** y > z **and** b > w)[a/x, b/y, c/b]

Answer: **exists**[x, y, z: **nat**] | x * y > z => x > z **and** y > z **and** c > w

4. Give proofs for the properties (assuming all the involved predicates are defined).

a. **forall**[x: X] | P(x) ⊢ **not exists**[x: X] | **not** P(x)

Answer:

From	**forall**[x: X] | P(x)	
1	**not not forall**[x: X] | P(x)	[**not**-intro](h)
infer	**not exists**[x: X] | **not** P(x)	[**forall**-deM](1)

b. x **inset** X, **forall**[y: X] | y > 15 ⊢ **exists**[z: X] | z > 15, where X is a subset of **nat0**.

Answer:

From	x **inset** X, **forall**[y: X] | y > 15	
1	x > 15	[**forall**-elim](h)
infer	**exists**[z: X] | z > 15	[**exists**-intro](h, 1)

5. Tell which predicates are true according to the extended truth tables.

a. x > y **and** y / 0 > 5 <=> **false**

Answer: no

b. x > y and y > x <=> **nil**

Answer: no

c. **true or nil** <=> **nil**

Answer: no

d. **false or nil** <=> **false**

Answer: no

e. **false** => **nil** <=> **nil**

Answer: no

f. **true** => **false** <=> **nil**

Answer: no

g. **true** >= **nil** <=> **false**

Answer: no

h. **true** <=> **false** <=> **nil**

Answer: no

i. **false <=> nil <=> true**

Answer: no

6. Use predicate expressions to describe the following statements (for practice in class).

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

Answer: forall[a: int] | a >= 0 or a < 0

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

Answer: 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.

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

(4) Every student in Xi'an Jiaotong University belongs to a department.

Answer: forall[s: XJTU] exists[d: Dept] | s inset d

(5) Not all people in Sichuan province likes hot food.

Answer: exists[p: Sichuan] | not likes(p, hotFood)

(6) Some students in Xi' an Jiaotong University come from Shanghai, but nobody comes from Beijing.

Answer: exists[s: XJTU] | come_from_Shanghai(s) and forall[s1: XJTU] | not come_from_Beijing(s)

Exercise 4

1. Answer the questions:

a.what is a process?

Answer: A process performs an action, task, or operation that takes input and produces output.

b. what is a data flow?

Answer: A data flow represents a data transmission from one process to another.

c. what is the difference between active data flows and control data flows?

Answer: An active data flow has two functions: enabling processes and provide usable data for processes, while a control data flow has only one function: enabling processes. A control data flow does not provide usable data for processes.

d. what is a data store?

Answer: A data store is a variable that holds data in rest and it can be accessed or

updated by processes.

e. what is the difference between data stores and data flows?

Answer: A data flow represents a moving data, while a store denotes a resting data.

f. what are the conditional structures for?

Answer: The conditional structures are used for describing data transmissions under certain conditions.

g. what are the merging and separating structures for?

Answer: The merging structure describes that several data flows merge into a single composite data flow so that a conditional structure can be used to control the direction or destination of those data flows.

A separating structure describes that a composite data flow is divided into several component data flows. Usually a merging is used together with a separating structure.

h. what are the diverging structures for?

Answer: A diverging structure tranforms a data flow to either one of a set of data flows or a set of data flows, depending on the type of the diverging structure. There are two diverging structures: nondeterministic structure and broadcasting structure.

i. what are the connecting structures for?

Answer: A pair of connecting structures are used together to establish a connection of data flows in a complicated CDFD in order to reduce complexity and potential confusion of data flows.

j. what is a condition data flow diagram (CDFD)?

Answer: A condition data flow diagram is a directed graph that specifies how processes work together to provide functional behaviors.

k. what is a module for?

Answer: A module is a structure for data and functional abstraction. It usually contains data definitions, behavior definition by a CDFD, and process specifications.:

l.what is the general structure of a module?

Answer: A general structure of a module is as follows:

module ModuleName / ParentModuleName;
const ConstantDeclaration;
type TypeDeclaration;
var VariableDeclaration;
inv TypeandStateInvariants;
behav CDFD_no;
InitializationProcess;

Process_1;
Process_2;
...
Process_n;
Function_1;
Function_2;
...
Function_m;
end_module

m. what is an invariant?

Answer: An invariant is a constraint, expressed as a predicate expression, on either types or state variables denoting stores.

n. what is the general structure of a process?

Answer: The general structure of a process is:

process ProcessName(input) output
ext ExternalVariables
pre PreCondition
post PostCondition
decom LowerLevelModuleName
explicit ExplicitSpecification
comment InformalExplanation
end_process

o. how to make a reference to the precondition or postcondition of a process?

Answer: We use the symbols **pre**_A and **post**_A denote the pre and postconditions of process A, respectively.

p. what is a function?

Answer: A function defines a mapping from its domain to its range.

q. what is the difference between a process and a function?

Answer: A process may access or update state variables denoting data stores, while functions have nothing to do with the state variables.

r. what are the general formats of explicit and implicit specifications of a function?

Answer: An explicit specification of a function takes the form:

function Name(InputDeclaration) : Type
== E
end_function

and an implicit specification of a function has the structure:

function Name(InputDeclaration) : Type
pre Pre
post Post(Name)
end_function

s. what is a recursive function, and what are the important points in writing recursive functions?

Answer: A recursive function is a function that applies itself during the computation of its body.

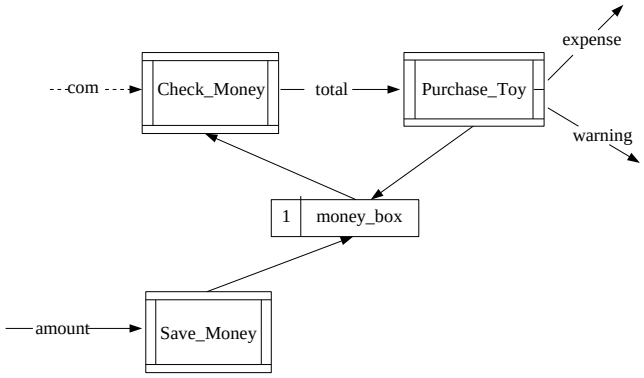
2. Define a calculator as a module. Assume that reg denotes the register that is accessed by various operations. The operations include Add, Subtract, Multiple, and Divide. Each operation is modelled by a process.

Answer:

```
module Calculator;
var reg: int;

process Init()
post reg = 0
end_process;
process Add(x: int)
ext wr reg
post reg = ~reg + x
end_process;
process Subtract(x: int)
ext wr reg
post reg = ~reg – x
end_process;
process Multiple(x: int)
ext wr reg
post reg = ~reg * x
end_process;
process Divide(x: int)
ext wr reg
pre x <> 0
post reg = ~reg / x
end_process
end_module;
```

3. Write a module definning all the data flows, stores, and processes of the CDFD in Figure 4.1 assuming all the data flows and stores are integers, and all the processes perform arithmetic operations.



com: command for checking the total amount of the money in the money-box
amount: the amount of money to be saved in the money_box
total: the total amount of the money in the money_box
expense: the necessary amount for purchasing a toy
warning: a warning message for the shortage of the money in the money_box

Figure 4.1

Answer:

```
module Purchase_Toy;
const toy_price = 500;
var #money_box: nat0;
process Save_Money(amount: nat0)
ext wr money_box
post money_box = ~money_box + amount
end_process;
process Check_Money(com: sign) total: nat0
ext rd money_box
post total = money_box
end_process;
process Purchase_Toy(total: nat0) expense: nat0 | warning: string
ext rd money_box
post if total >= toy_price
        then toy_price <= expense <= total and money_box = total – expense
        else warning = “Money is not enough to buy the toy.”
end_process;
```

end_module

4. Change the following compound expressions into equivalent classical predicate expressions.

a. let a = x + y, b = z + w in a **2 * b + b * y * w > 10

Answer: a = x + y **and** b = z + w **and** a **2 * b + b * y * w > 10

b. if x > 0 then a = x + 1 else a = x + 10

Answer: x > 0 and a = x + 1 or x <= 0 and a = x + 10

c. a = case x of 1, 2, 3 --> x +1; 4, 5 --> x + 2; 6 --> x * x; default --> x end

Answer: ((x = 1 **or** x = 2 **or** x = 3) **and** a = x + 1) **or**

(x = 4 **or** x = 5) **and** a = x + 2) **or**

x = 6 **and** a = x * x **or**

not (x = 1 **or** x = 2 **or** x = 3 **or** x = 4 **or** x = 5 **or** x = 6) **and** a = x

5. Write both 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**.

Answer: (1) Explicit specification

function Fibonacci(n: **nat0**): **nat0**

== **if** n = 0 **or** n = 1 **then** n **else** Fibonacci(n – 1) + Fibonacci(n – 2)

end_function

(2) Implicit specification

function Fibonacci(n: nat0): nat0

post Fibonacci = **if** n = 0 **or** n = 1 **then** n **else** Fibonacci(n – 1) + Fibonacci(n – 2)

end_function

Exercise 5

1. Answer the questions:

a. what is a hierarchy of CDFDs?

Answer: a hierarchy of CDFDs is a leveled CDFDs in which a low level CDFD is derived by decomposing a high

level process.

b. what is a hierarchy of modules?

Answer: a hierarchy of modules is a leveled modules in which a low level module

corresponds to the low level CDFD that is derived by decomposing a high level process in the corresponding high level module.

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

Answer: a hierarchical modules corresponds to a hierarchical CDFDs in the sense that each CDFD is associated with one module and the CDFD describes the behavior of the module.

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

Answer: a CDFD is a refinement of its high level process. In other words, a CDFD is an implementation satisfying the specification of its high level process.

e. what is the condition for a CDFD to be correct with respect to its high level process?

Answer: Let A be a high level process:

process A (x1: Ti_1 | x2: Ti_2) y1: To_1, y2: To_2
ext wr s: Ts
pre pre_A
post post_A
end_process

Let G denote the decomposition of A. Then the correctness of G with respect to A is defined as.

If A and G are structurally consistent, and the following condition

forall[x1: Ti_1, ~s: Ts] | pre_A(x1, x2, ~s) => post_A(x1, x2, G(x1), ~s, s)

or

forall[x2: Ti_2, ~s: Ts] | pre_A(x1, x2, ~s) => post_A(x1, x2, G(x2), ~s, s)

holds, we say that G satisfies A, or G is correct with respect to A.

f. what does it mean by saying that module M1 is the ancestor module of M2?

Answer: Let M1, A_1, ...A_n, M2 be a sequence of modules where n >= 0. If M1 is a parent module of A_1, and A_1 is a parent module of A_2, ..., and A_n is a parent module of M2, then we call M1 ancestor module of M2 and M2 descendant module of M1.

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

Answer: if M1 and M2 belong to the same hierarchy of modules and one of them is not an ancestor module of the other, we say M1 and M2 are relative modules.

h. what is the scope of a variable, type identifier, constant identifier, invariant,

function, and a process?

Answer: the scope of a variable, type identifier and constant identifier is a set of module in which the variable,

type identifier, and constant identifier can be directly used (or referred). The scope of a type invariant is the whole specification where related, likewise for a function. The scope of a process is the module in which it is defined.

2. Explain whether the CDFD in the related Figure 5.1 is structurally consistent with its high level process W. Is the CDFD possible to be correct with respect to process W? If so, explain why.

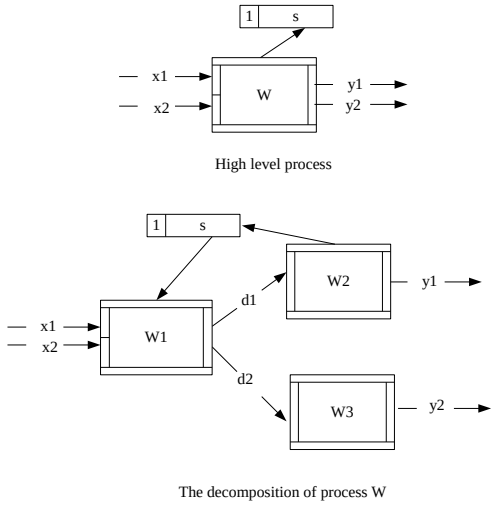


Figure 5.1

Answer: Yes, the CDFD in Figure 5.1 is a structurally consistent decomposition of process W, because both the rules for the access of stores and for the input and output data flows are satisfied. The CDFD is possible to be correct with respect to its high level process specification.

Exercise 6

1. Write explicit specifications for the processes of the ATM given in chapter 4.
   a. Receive_Command
   b. Withdraw
   c. Show_Balance

Answer:

a. process Receive_Command(balance: sign | w_draw: sign) sel: bool
   explicit

```
        if bound(balance)
        then sel := true;
        if bound(w_draw)
        then sel := false
    end_process
```

b.
```
process Withdraw(amount: real, account1: Account)
                                    e_msg: string | cash: real
ext wr account_file
pre account1 inset account_file
        /*input account1 must exist in the account_file*/
explicit
 i, mark: int;
 x: Account;
 account_file_tem: set of Account;
 begin
   account_file_tem := account_file;
   i := 1;
   mark := 0;
   while i <= card(account_file_tem) do
    begin
     x := get(account_file_tem);
     account_file_tem := diff(account_file_tem, {x});
     if x = account1 and x.balance >= amount
     then
       begin
         account_file := diff(account_file, {x});
         cash := amount;
         x.balance := x.balance – amount;
         account_file := union(account_file, {x});
         i := card(account_file_tem) + 1;
         mark := 1
       end
    end; /* the exit of the while loop*/
   if mark = 0
   then write("The requested amount of cash is", cash)
```

else write(“The amount is too big”)

end;

end_process

c. process Show_Balance(account2: Account) balance: real

explicit

balance := account2.balance;

write(“The balance is “, balance

end_process;

Exercise 7

1. Let x = 12, y = 9.8, z = 2, and a = -20. Evaluate the expressions:

Answer

a. - z = -20
b. abs(a) = 20
c. floor(y) =
d. x + z = 14
e. x – y = 2.2
f. a * z = -40
g. x / y = 1.22449
h. a div z = -10
i. a rem x = 8
j. x mod z = 0
k. x ** z = 144

2. Let x = 20, y = 5.5, z = 'd', and a = true. Evaluate the expressions:

Answer:

a. 'a' = z <=> false
b. ')' <> z <=> false
c. x >= y <=> true
d. x < y <= y <=> false
e. a = false <=> false
f. a <> true <=> true

3. 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.

Answer:

```
type
  WeekDay = {<Monday>, <Tuesday>, <Wednesday>, <Thursday>, <Friday>};
  Courses = {<Software Engineering>, <Program Design>, <Discrete
  Mathematics>, <Programming Language>, <Formal Engineering Methods>};

process GetCourse(weekday: WeekDay) course: Courses
post case weekday of
      <Monday> → course = < Software Engineering>;
      <Tuesday> → course = <Program Design>;
      <Wednesday> → course = <Discrete Mathematics>;
      <Thursday> → course = <Programming Language>;
      <Friday> → course = <Formal Engineering Methods>
    end_case
end_process
```

Exercise 8

1. Given a set T = {5, 8, 9}, define a set type based on T, and list all the possible set values in the type.

Answer:

A = set of T;

A = {{ }, {5}, {8}, {9}, {5, 8}, {5, 9}, {8, 9}, {5, 8, 9}}

2. Let T = {5, 8, 9}. Then evaluate the set comprehensions:

Answer:

a. {x | x: nat & x < 8} = {1, 2, 3, 4, 5, 6, 7}

b. {y | y: nat0 & y <= 3} = {0, 1, 2, 3}

c. {x - y | x: int, y: int & -2 < x + y < 3} = {-3, -2, -1, 0, 1, 2, 3, 4, 5}

d. {i | i: set of T & card(i) < 3 and forall[x, y: i] | x + y <= 13} = {{5, 8}}

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

Answer:

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) = inter({15, 30, 50, 2, 8}, 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))) = inter({5, 15, 25, 30, 2, 8}, diff(s2, {5, 15, 25, 30, 2, 8}))

= inter({5, 15, 25, 30, 2, 8}, {50}) = { }

4.Write set comprehensions for the sets:

Answer:

a. a set of natural numbers whose elements are all smaller than 10. {n | n: nat & n < 10}

b. a set of integers whose elements are all greater than 0 and smaller than 10 and cannot be divided by 3.

{i | i: int & 0 < i < 10 and i mod 3 <> 0}

c. a set of prime numbers. {p | p: nat & not exists[x: nat] | x <> p => p mod x = 0}

5. 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.

Answer:

```
module Telephone_Book;

type

 TelephoneNumber = nat;

var

telephone_book: set of TelephoneNumber;

process Add(x: TelephoneNumber)
ext wr telephone_book
pre x notin telephone_book
```

```
post telephone_book = union(~telephone_book, {x})
end_process;

process Find(x: TelephoneNumber) r: bool
ext rd telephone_book
post r = (x inset telephone_book)
end_process;

process Delete(x: TelephoneNumber)
ext wr telephonel_book
pre x inset telephone_book
post telephone_book = diff(~telephone_book, {x})
end_process;

process Update(x, y: TelephoneNumber)
ext wr telephonel_book
pre x inset telephone_book
post telephone_book = union(diff(~telephone_book, {x}), {y})
end_process;
end_module;
```

6. 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 lectured by a different professor, the students from both groups may drop from the merged group (but exactly which students will drop is not known).

Answer:

```
type
 Student = given;

process Merge(s1, s2: set of Student) s: set of Student
post subset(s, union(s1, s2))
end_process
```

Exercise 9

1. Given a set T = {1, 2, 5}, define a sequence type based on T, and give ten possible sequence values in the type.

Answer:

A = seq of T

A = {[ ], [1], [2], [5], [1, 2], [1,5], [1, 1, 2], [1, 5, 2], [5, 2, 5], [5, 2, 2, 1,5, 1], …}

2. Evaluate the sequence comprehensions:

Answer:

a. [x | x: nat & 3 < x < 8] = [4, 5, 6, 7]

b. [y | y: nat0 & y <= 3] = [0, 1, 2, 3]

c. [x - y | x: nat0, y: nat0 & 1 < x + y < 3] = [-1, 0, 1, 2]

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

Answer:

a. hd(s1) = 5

b. hd(s) = s1

c. len(tl(s1)) + len(tl(s2)) + len(tl(s3)) = 2 + 2 + 3 = 7

d. len(s1) + len(s2) - len(s3) = 3 + 3 + 3 = 9

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.

Answer:

```
module Queue;
var
  queue: seq of int;

process Append(e: int)
ext wr queue
post queue = conc(~queue, {e})
end_process;

process Eliminate() t: int
ext wr queue
pre queue <> [ ]
```

```
post t = hd(~queue) and ~queue = conc(t, queue)
end_process;

process Read() t: int
ext rd queue
pre queue <> [ ]
post t = hd(~queue)
end_process;

process Count() no: int
ext rd queue
post no = len(queue)
end_process;
end_module
```

5. Write a specification for a process Search. The process takes an integer and search through a sequence of integers, which is denoted by an external variable of the process. If the input integer is found in the sequence, its indexes (there might be more than one occurrences of the input integer) are given as the result. If the input integer is not found, then empty set is given as the output.

Answer:

```
process Search(x: int) indexes: set of nat
ext rd list: seq of int
pre len(list) > 0
post   indexes = {j | j: inds(list) & list(j) = x}
end_process
```

Exercise 10

1. Explain the similarity and difference between a composite type and product type.

Answer: An object of a composite type is composed of fields and the reference of the fields is done through the fields, while an object of a product type denotes a list of values with a fixed length. The order of the fields in a composite type is not significant, while the order of the values in a tuple of a product type is important.

2. Let a = mk_Account(010, 300, 5000), where the type Account is defined as a composite type (see chapter 10). Then evaluate the expressions:

Answer:

a. a.account_no = 010

b. a.password = 300

c. a.balance = 5000

d. modify(a, password --> 250) = mk_Account(010, 250, 5000)

e. modify(mk_Account(020, 350, 4050), account_no --> 100, balance --> 6000) = mk_Account(100, 350, 4050)

3. Let x be a variable of the type Date defined as a product type in chapter 10, and x = mk_Date(2002, 2, 6). Then evaluate the expressions:

Answer:

a. x(1) = 2002

b. x(2) = 2

c. x(3) = 3

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, data_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 an external variable student_list, which is a sequence of students. Change_Name updates the name of a given student with a given name. Get_Info provides all the available field values for a given student name (assuming that student name is unique).

Answer:

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

var
student_list: seq of Student;
```

```
process Register(s: Student)
ext wr student_list
pre s notin elems(student_list)
post student_list = conc(~student_list, [s])
end_process;

process Change_Name(s: Student, new_name: string)
ext wr student_list
pre s inset elems(student_list)
post exists[i: inds(~student_list)] | ~student_list(i) = s and
        len(student_list) = len(~student_list) and
        student_list(1, i – 1) = ~student_list(1, i – 1) and
        student_list(i + 1, len(student_list)) = ~student_list(i + 1, len(~student_list)) and
        student_list(i) = modify(~student_list(i), name → new_name)
end_process;

process Get_Info(name: string) name1: string, date_of_birth: Date, college: string,
grade: nat0
ext rd student_list
pre exists[s: elems(student_list)] | s.name = name
post exists[s: elems(student_list)] | s.name = name and
        name1 = s.name and
        date_of_birth = s.date_of_birth and
        college = s.college and
        grade = s.grade
end_process;
```

Exercise 11

1. Tell the similarity and difference between a map and function.

Answer: The similarity of a map and function is that both defines an association between domain and range. The difference is that map is a finite function, while a function can be infinite (i.e., describing an association between infinite domain and infinite range).

2. Given two sets T1 = {1, 2}, T2 = {10, 11}, construct a map type with T1 being its

domain type and T2 being its range type, and enumerate all the possible maplets of the map type.

Answer:

A = map T1 to T2

A = {{ → }, {1 → 10}, {1 → 11}, {2 → 10}, {2 → 11}, {1 → 10, 2 → 11}, {1 → 11, 2 → 10}, {1 → 10, 2 → 10}, {1 → 11, 2 → 11}}

3. 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:

Answer:

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) = { → }

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, 3 → 30}

l. rngrb(m2, s) = {2 → 40, 4 → 80}

m. override(m1, m2) = {1 → 10, 2 → 40, 3 → 1, 4 → 80}

n. override(m2, m1) = {2 → 3, 3 → 30, 4 → 80, 1 → 10}

o. inverse(m1) = {10 → 1, 3 → 2, 30 → 3}

p. inverse(m2) = {40 → 2, 1 → 3, 80 → 4}

q. comp(m1, m2) = {2 → 1}

r. comp(m2, m1) = {3 → 10}

s. m1 = m2 <=> false

t. m1 <> m2 <=> true

4. Give a concrete example to explain that comp(m1, m2) is defined whereas comp(m2, m1) is undefined.

Answer:

Let m1 = {1 → 'a', 2 → 'b', 3 → 'c'}, m2 = {'a' → "Hosei", 'c' → "University"}. Then

comp(m1, m2) = {1 → "Hosei", 3 → "University"}, but

comp(m2, m1) = undefined due to the type incompatibility.

m2: map **char** to **string**

m1: map **nat** to **char**

Where the type of the intermediate value, **string**, is incompatible with that of the domain **nat**.

5. Define BirthdayBook as a map type from the type Person 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 for a person from birthday_book; and Update replaces the wrong birthday existing in birthday_book with a correct birthday.

Answer:

```
type
  BirthdayBook = map Person to Birthday;
  Person = given;
  Birthday = given;
var
  birthday_book: BirthdayBook;

  process Register(person: Person, birthday: Birthday)
  ext wr birthday_book
  pre person notin dom(birthday)
  post birthday_book = override(~birthday_book, {person → birthday})
  end_process;

  process Find(person: Person) birthday: Birthday
  ext rd birthday_book
  pre person inset dom(birthday_book)
  post birthday = birthday_book(person)
  end_process;

  process Delete(person: Person)
  ext wr birthday_book
  pre person inset dom(birthday_book)
  post birthday_book = domrb({person}, ~birthday_book)
  end_process;
```

process Update(person: Person, birthday: Birthday)
ext wr birthday_book
post birthday_book = override(~birthday_book, {person → birthday})
end_process;
end_module

Exercise 12

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

Answer:

School = ElementarySchool | JuniorHighSchool | HighSchool | 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:

Answer:

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>, 3}

3. Let a, b, c: Identifier. Evaluate the expressions:

Answer:

a. is_Identifier(a) <=> true
b. is_Digit(b) <=> false
c. is_EnglishLetter(c) <=> false

Exercise 13

1. Answer the questions:

a. what is a class?

Answer: A class is a user-defined type, which defines a collection of objects with the same features.

b.what is an object?

Answer: An object of a class is an instance (or value) of the class (as a type).

c. what is inheritance?

Answer: Inheritance is a mechanism for building a new class based on an existing class, which allows the reuse of the attributes and behaviors (methods) of the existing class by the new class.

d. what is superclass and subclass?

Answer: Let class B inherits directly from class A. Then we say that A is the superclass of B and B is a subclass of A.

e. what is polymorphism?

Answer: Polymorphism is a mechanism by which a single method or attribute variable may be defined upon more than one class and may take on different implementations in each of those classes.

f. what is a generic class?

Answer: A class is a generic class if it allows parameters that will be bound to concrete types (or type identifiers).

2. Define the class Polygon as the superclass of the classes Triangle and Rectangle. Define an attribute variable area and a method Compute_Area in each of the classes, but with different specifications, depending on the specific shapes.

Answer:

```
class Polygon;
  var
    edges: seq of nat;

    method Init()
    post edges = [ ]
    end_class;

    method Update_Edges(new_edges: seq of nat)
    ext wr edges
    post edges = new_edges
    end_method;

    method Compute_Area() s: real
    ext rd edges
    end_method
end_class;

class Triangle / Polygon;
```

```
  var
   area: real;

  method Init()
  post area = 0
  end_method;

  method Compute_Area()
  ext rd edges
       wr area
  pre len(edges) = 3
  post let x = (edges(1) + edges(2) + edges(3)) / 2 in
            area ** 2 = x * (x – edges(1)) * (x – edges(2)) * (x – edges(3))
  end_method;
  end_class;

class Rectangle / Polygon;
var
  area: real;

method Init()
post area = 0
end_method;

method Compute_Area()
ext wr area
ext rd edges
pre len(edges) = 4 and edges(1) = edges(3) and edges(2) = edges(4)
post area = edges(1) * edges(2)
end_method
end_class;
```

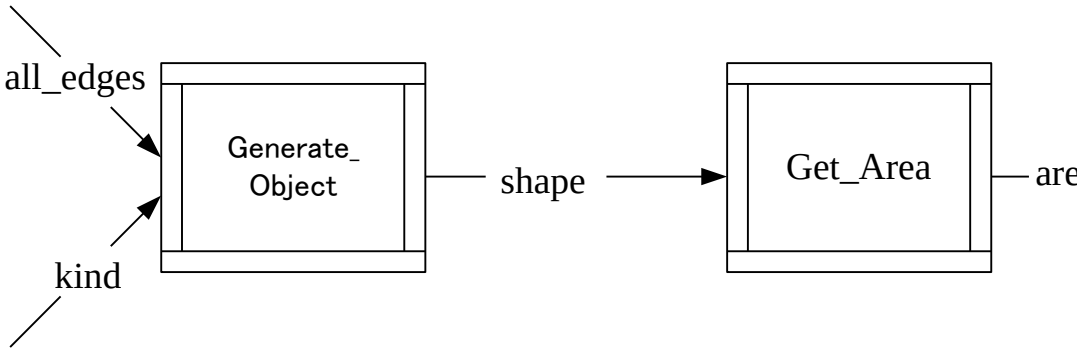
3. Specify a module whose CDFD creates a required shape that can be one of the objects of the two classes Triangle and Rectangle, and compute its area.

Answer:

```
  module Create_Objects;
```

```
  process Generate_Object(all_edges: seq of nat, kind: bool) shape: Polygon
  explicit
   begin
     shape := new Polygon;
     shape.Update_Edges(all_edges);
     if kind
     then shape := (Triangle) shape;
     else shape := (Rectangle) shape;
   end
  end_process;

  process Get_Area(shape: Polygon) area: real
  explicit
   begin
      shape.Compute_Area();
      area := shape.area
   end
  end_process
end_module
```



Exercise 14

1. Give an example to explain the difference between evolution and refinement of processes.

Answer:

Let process A be defined as

process A(x: int) y: int

pre x >= 0

post y > x + 1

end_process

Then process B is a refinement of A:

process B(x: int) y: int

pre x >= 0

post y = x + 2

end_process

because pre_A => pre_B and pre_A and post_B => post_A.

However, process C given below is not a refinement of A, but its evolution:

process C(x: int) y: int

post y < x + 2

end_process

Where C is a modification of A.

2. Construct a formal design specification of library system by taking the three steps: informal, semi-formal, and formal specification. The system is required to provide the services: Borrow, Return, and Search. Each of these services should be implemented by a process. The process Borrow registers the data of the borrowed book; Return removes the registered information about the borrowed book; and Search provides the requested information of the wanted book, if it is available.

Answer:

(1) Informal specification:

The library system provides the following functions:

Borrow book

Return book

Search book

The necessary data resources:

(a) a set of books, each having a unique title and book number

(b) a set of borrowers, each having borrowed a set of books.

The policy for the library system:

A person cannot borrow more than 10 books.

(2) Semi-formal specification:

```
module SYSTEM_LibrarySystem;
 type
  BookInformation = composed of
                      title: string
                      author: Name
                      publisher: string
                      year: nat0
                      book_no: nat0
                     end;
  Name = string * string * string;
  BookBase = map Title to BookInformation;
  Title = string;
  Person = composed of
             id: nat0
             name: Name
             address: string
             affiliation: string
            end;
  Borrowers = map BookInformation to Person;
  ID = nat0;

  var
  ext #borrower_list: Borrowers;
  ext #book_base: BookBase;

  process Borrow(title: Title, person: Person) book: BookInformation |
error_message: string
  ext wr borrower_list
      wr book_base
  post if the book with the wanted title is available in the library and
          the person's borrowed books are less than 10
```

```
        then (1) add the book and person information to the borrower_list and
                (2) remove the borrowed book from the library book_base
        else issue an error message indicating that the requested book in not available.
    end_process;

    process Return(book: BookInformation, person: Person)
    ext wr borrower_list
        wr book_base
    post remove the borrowed book and the person information from the borrower_list
   and
        add the book information to the library book_base
    end_process;

    process Search(title: Title) book: BookInformation
    ext rd book_base
    pre the book with the requested title is available in the library book_base
    post get the book information
    end_process;
   end_module
```

(3) Formal specification:

```
    module SYSTEM_LibrarySystem;
    type
      BookInformation = composed of
                          title: string
                          author: Name
                          publisher: string
                          year: nat0
                          book_no: nat0
                        end;
      Name = string * string * string;
      BookBase = map Title to BookInformation;
      Title = string;
      Person = composed of
                 id: nat0
                 name: Name
```

```
        address: string
        affiliation: string
      end;
  Borrowers = map BookInformation to Person;
  ID = nat0;

  var
  ext #borrower_list: Borrowers;
  ext #book_base: BookBase;

  process Borrow(title: Title, person: Person) book: BookInformation |
error_message: string
  ext wr borrower_list
      wr book_base
  post if title inset dom(book_base) and
          card({y | y: dom(borrower_list) & borrower_list(y) = person}) < 10
       then book = ~book_base(title) and
            book_base = domrb({title}, ~book_base) and
            borrower_list = override(~borrower_list, {~book_base(title) → person})
       else error_message = "No book can be borrowed"
  end_process;

  process Return(book: BookInformation, person: Person)
  ext wr borrower_list
      wr book_base
  post borrower_list = domrb({book}, ~borrower_list) and
       book_base = override(~book_base, {book.title → book})
  end_process;

  process Search(title: Title) book: BookInformation
  ext rd book_base
  pre title inset dom(book_base)
  post book = book_base(title)
  end_process;
end_module
```

3. .Refine the implicit specifications of all the three processes in the library system into explicit specifications.

Answer:

```
process Borrow(title: Title, person: Person) book: BookInformation | error_message: string
ext wr borrower_list
    wr book_base
explicit
 book_tem: BookInformation
 begin
  if    title inset dom(book_base) and
        card({y | y: dom(borrower_list) & borrower_list(y) = person}) < 10
  then
      begin
        book := book_base(title);
        book_tem := book;
        book_base := domrb({title}, book_base);
        borrower_list := override(borrower_list, {book_tem → person})
      end
    else error_message := "No book can be borrowed"
end_process;

process Return(book: BookInformation, person: Person)
ext wr borrower_list
    wr book_base
explicit
 begin
  borrower_list := domrb({book}, borrower_list);
  book_base = override(book_base, {book.title → book})
end_process;

process Search(title: Title) book: BookInformation
ext rd book_base
pre title inset dom(book_base)
explicit
 book := book_base(title)
```

end_process;

Exercise 15

1. Explain the advantages and weaknesses of the top-down and middle-out approaches to building specifications.

Answer:

The top-down approach is usually effective and intuitive in providing sub-goals or sub-tasks to support the current goal or task, and in developing ideas with little information into ideas with more information. It also provides a good global view of data flows and stores that may be used across CDFDs at different levels, thus the consistency in using data flows and stores can be well managed during the decomposition of high level processes. However, the difficulty in applying this approach may be caused by frequent modifications of high level processes, data flows, stores, and even the entire CDFDs, as with the progress of decomposition of high level processes. Modifications are necessary because creating accurate components of a high level CDFD in the first place is usually challenging due to the lack of sufficient knowledge about what data flows and stores will be used or produced by the processes in the lower level CDFDs. To reduce the effect of this problem, the approach of using the top-down approach for introducing processes and the bottom-up approach for introducing data flows and stores can be helpful.

In contrast to the top-down approach, the middle-out approach may be more effective and natural, because it always starts with modeling the most familiar and crucial functions. It also takes a flexible way to utilize the top-down and the bottom-up approaches, and taking which approach usually stems from natural demands during the construction of the entire specification. However, by using this approach the developer may not be easy to take a global view of the specification in the early stage, thus data flows, stores, and processes created in different CDFDs may overlap or defined inconsistently.

2. What is the difference between the CDFD-module-first strategy and the CDFD-hierarchy-first strategy.

Answer:

The CDFD-module-first strategy emphasizes the importance of completing a CDFD and its module in an interactive manner before proceeding to decompositions of its processes, while CDFD-hierarchy-first strategy emphasizes the importance of first

creating the architecture of the entire specification and then concentrating on the definition of all the components of each level CDFD.

3. Build a "Personal Expense Management System" using both top-down and middle-out approaches, respectively. The management system provides the services: (1) record the expense of an item, (2) search the expense for a specific item, (3) search for the expense for a kind of items (e.g., cloth, book), (4) update the record of the expense for a specific item, and (5) show the total expense of all the items purchased in a specific month.

   Answer: leave this to the reader.

4. Rebuild the same "Personal Expense Management System" using both the CDFD-module-first and the CDFD-hierarchy-first approaches, respectively, and compare the advantages and disadvantages of the two different approaches.

   Answer: leave this to the reader.

Exercise 16

1.Give a semi-formal specification for the module Manage_Savings_Account_Decom.

Answer:

```
module Manage_Savings_Account_Decom / SYSTEM_ATM;
var
ext #savings_accounts: SavingsAccountFile;
process Savings_Authorization(savings_inf: SavingsAccountInf)
                                        permission1: sign | e_mesg2: sign
ext rd savings_accounts
post if the input account_no and pass match those of the customer in the
        store savings_accounts
    then generate permission1
    else generate an error message e_mesg2
end_process;

process Savings_Deposit(permission1: sign, savings_inf: SavingsAccountInf, deposit_amount: nat0)
                                        notice1: Notice | warning3: string
ext wr savings_accounts
post if the deposit_amount is less than or equal to the maximum_deposit_once
```

then

(1) add the deposit_amount to the savings account

(2) give the customer a notice showing the amount of deposit and the updated balance

(3) update the transaction histroy of the savings account

else give a warning message to indicate that the deposit amount is over the limit

end_process;

process Apply_Withdraw(applied_amount: nat0)

a_notice: ApplicationNotice | warning4: string

ext wr savings_accounts

post if applied_amount is less than or equal to the maximum_withdraw_application amount

then

(1) set the withdraw_application_amount to the applied_amount

(2) set the application_status as true

(3) give a notice to show the success of application.

else give a warning message to indicate that the applied_amount is over limit.

end_process;

process Savings_Withdraw(permission2: sign, withdraw_amount: nat0)

notice2: Notice | warning5: string

ext wr savings_accounts

post if the customer has applied for withdrawing in advance and the withdraw_amount

is less than or equal to the withdraw_application_amount

then

(1) provide cash of the requested withdraw amount

(2) give a notice to the customer to indicate the withdraw amount and the updated balance

else

give a warning message to the customer

end_process;

process Savings_Show_Balance(permission3: sign) savings_balance: nat0
ext rd savings_accounts
post Display the balance of the savings account
end_process;

process Savings_Print_Transaction_Records(permission4: sign, date: Date)
transaction_records1:
TransactionRecords
ext rd savings_account
post print out the transaction records since the input date
end_process;

end_module;

2.Give a formal abstract design specification for the module Manage_Savings_Account_Decom.

Answer:

module Manage_Savings_Account_Decom / SYSTEM_ATM;
type
Notice = Manage_Current_Account_Decom.Notice;
OutputDevice = Manage_Current_Account_Decom.OutputDevice;
InputDevice = Manage_Current_Account_Decom.InputDevice;
SavingsServiceCollection = {<1>, <2>, <3>, <4>, <5>};
var
ext #current_accounts: CurrentAccountFile;
ext #output_device: OutputDevice;
ext #InputDevice: InputDevice;
inv
forall[x, y: {s_deposit, apply, s_withdraw, s_s_balance, s_p_transactions}] |
bound(x) and bound(y) = false;
behav CDFD_No3;

process Select_Savings_Services(s_deposit, a1: sign |
b1, apply: sign |
c1, s_withdraw: sign |
d1, s_s_balance: sign |

```
                                                    e1, s_p_transactions: sign)
                                                    sel: SavingsServiceCollection
post bound(s_deposit) and sel = <1> or
        bound(apply) and sel = <2> or
        bound(s_withdraw) and sel = <3> or
        bound(s_s_balance) and sel = <4> or
        bound(s_p_transactions) and sel = <5>
comment
The output data flow sel is decided to take different value depending on
the availability of the input data flows.
end_process;

process Savings_Authorization(sel: SavingsServiceCollection,
                                                    savings_inf: CustomerInf)
                                                    savings_inf1: CustomerInf |
                                                    savings_inf2: CustomerInf |
                                                    savings_inf3: CustomerInf |
                                                    savings_inf4: CustomerInf |
                                                   savings_inf5: CustomerInf |
                                                    e_mesg2: string
ext rd savings_accounts
post if savings_inf inset dom(savings_accounts)
        then case sel of
                <1> --> savings_inf1 = savings_inf;
                <2> --> savings_inf2 = savings_inf;
                <3> --> savings_inf3 = savings_inf;
                <4> --> savings_inf4 = savings_inf;
               <5> --> savings_inf5 = savings_inf;
                end
        else e_mesg2 = "Your password or account number is incorrect."
comment
if the input account_no and password match those of the customer's
            savings account in the store savings_accounts
 then generate output data flows based on the value of variable sel
 else output an error message.
end_process;
```

```
process Savings_Deposit(d_amount: nat0, savings_inf1: CustomerInf)
                                        notice1: Notice | warning3: string
ext wr savings_accounts;
post if d_amount <= maximum_deposit_once
        then
            savings_accounts =
             override(~savings_accounts,
                {savings_inf1 -->
                    modify(~savings_accounts(savings_inf1),
                      balance -->
                       ~savings_accounts(savings_inf1).balance + d_amount,
                      transaction_history -->
                       conc(~savings_accounts(savings_inf1).transaction_history,
                         [Get_Savings_Transaction(savings_accounts, today,
current_time, 0, d_amount, savings_inf1)]
                                )
                               )
                }
                ) and
                notice1 = mk_Notice(d_amount,
savings_accounts(savings_inf1).balance))
        else warning3 = "Your amount is over 1000000 yen limit."
    comment
     if the input d_amount is less than or equal to the maximum_deposit_once
     then
          (1) add the d_amount to the savings_account
          (2) give the customer a notice showing the amount of deposit and the
updated balance
          (3) update the transaction history of the account
     else give a warning message to indicate that the amount is over the limit.
    end_process;

    process Apply_Withdraw(savings_inf2: CustomerInf, a_amount: nat0)
                                        a_notice: string | warning4: string
    ext wr savings_accounts
```

```
post if a_amount is maximum_withdraw_application
        then
         savings_accounts =
         override(~savings_accounts,
                        {savings_inf2 -->
                          modify(~savings_accounts(savings_inf2),
                              withdraw_application_amount: nat0 -->
                              a_amount,
                              application_status --> true)
                        }
                      ) and
         a_notice = "Your application is successful"
         else warning4 = "Your application amount is over the limit."
      comment
      If the applied withdraw amount a_amount is within the fixed limit,
      then change the withdraw_application_amount to the applied withdraw
      amount and the application_status to true, indicating the application is
      made, and issue a notice to tell the customer that the application is
      successful. Otherwise, give a warning message to indicate that the
      applied withdraw amount is over the fixed limit.
      end_process;

       process Savings_Withdraw(savings_inf3: CustomerInf, w_amount: nat0)
                                        notice2: Notice | warning5: string
      ext wr savings_accounts
      post                    if                    w_amount                    <=
~savings_accounts(savings_inf3).withdraw_application_amount and
                w_amount <= ~savings_accounts(savings_inf3).balance
         then
          savings_accounts =
           override(~savings_accounts,
                 {savings_inf3 -->
                     modify(~savings_accounts(savings_inf3),
                       balance -->
                         ~savings_accounts(savings_inf3).balance - w_amount,
                       transaction_history -->
```

```
                    conc(~savings_accounts(savings_inf3).transaction_history,
                     [Get_Savings_Transaction(savings_accounts, today,
current_time, w_amount, 0, savings_inf2)]
                        )
                    )
                }
                ) and
                notice2 = mk_Notice(w_amount,
savings_accounts(savings_inf3).balance))
            else warning5 = "Your withdraw amount is over the limit."
                                    comment
        if the input w_amount is less than or equal to the applied withdraw amount
           and the balance of the savings account
         then
            (1) output the cash of the requested amount
            (2) reduce the withdraw amount from the balance
            (3) update the transaction history of the account
            (4) give a notice
        else
            generate a warning message
        end_process;

        process Savings_Show_Balacnce(savings_inf4: CustomerInf)
                                    s_balance: nat0
        ext rd savings_accounts
        post s_balance = savings_accounts(savings_inf4).balance
        comment
        Display the balance of the customer's savings account
        end_process;

        process Savings_Print_Transaction_Records(savings_inf5: CustomerInf, date:
Date)
                                    transaction_records:
TransactionRecords
        ext rd savings_accounts
        post let transactions = savings_accounts(savings_inf5).transaction_history
```

```
        in let i = get({i | i: inds(transactions) & transactions(i).date = date})
             in
               transaction_records = transactions(i, ..., len(transactions))
comment
Print out the transaction records since the input date
end_process;

process Savings_Display_Information(notice1: Notice |
                                    a_notice: string |
                                    notice2: Notice |
                                    s_balance: nat0 |
                                    transaction_records: TransactionRecords)
ext wr output_device
post bound(notice1) and output_device = conc(~output_device, [notice1]) or
     bound(a_notice) and output_device = conc(~output_device, [a_notice]) or
      bound(notice2) and output_device = conc(~output_device, [notice2]) or
      bound(s_balance) and output_device = conc(~output_device, [s_balance]) or
      bound(transaction_records) and output_device =
          conc(~output_device, [transactions_records])
comment
Display the input data flows onto the output device based on their
availability.
end_process;

process Savings_Display_Message(warning3: string |
                                warning4: string |
                                warning5: string |
                                e_mesg2: string)
ext wr output_device
post bound(warning3) and output_device = conc(~output_device, [warning3]) or
       bound(warning4) and output_device = conc(~output_device, [warning4]) or
     bound(warning5) and output_device = conc(~output_device, [warning5]) or
      bound(e_mesg2) and output_device = conc(~output_device, [e_mesg2])
comment
Display the input data flows onto the output device based on their
availability.
```

```
end_process;

function Get_Savings_Transaction(savings_accounts: SavingsAccountFile,
                                 to_day: Date,
                                 time: Time,
                                 pay_amount: nat0,
                                 deposit_amount: nat0,
                                 customer_inf: CustomerInf): Transaction
== mk_Transaction(to_day, time, pay_amount, deposit_amount,
                  savings_accounts(customer_inf).balance
                  ATM_no)
end_function
end_module;
```

4. Write a formal detailed design specification for the module Manage_Savings_Account_Decom.

Answer:

```
module Manage_Savings_Account_Decom / SYSTEM_ATM;
type
 Notice = Manage_Current_Account_Decom.Notice;
OutputDevice = Manage_Current_Account_Decom.OutputDevice;
InputDevice = Manage_Current_Account_Decom.InputDevice;
SavingsServiceCollection = {<1>, <2>, <3>, <4>, <5>};
var
 ext #current_accounts: CurrentAccountFile;
ext #output_device: OutputDevice;
ext #InputDevice: InputDevice;
inv
forall[x, y: {s_deposit, apply, s_withdraw, s_s_balance, s_p_transactions}] |
        bound(x) and bound(y) = false;
behav CDFD_No3;

process Select_Savings_Services(s_deposit, a1: sign |
                                                b1, apply: sign |
                                                c1, s_withdraw: sign |
                                                d1, s_s_balance: sign |
                                                e1, s_p_transactions: sign)
```

```
                                                    sel: SavingsServiceCollection
post bound(s_deposit) and sel = <1> or
        bound(apply) and sel = <2> or
        bound(s_withdraw) and sel = <3> or
        bound(s_s_balance) and sel = <4> or
        bound(s_p_transactions) and sel = <5>
explicit
begin
  if bound(s_deposit)
  then sel := <1>
  else if bound(apply)
        then sel := <2>
        else if bound(s_withdraw)
              then sel := <3>
              else if bound(s_s_balance)
                    then sel := <4>
                   else sel := <5>
end
comment
The output data flow sel is decided to take different value depending on
the availability of the input data flows.
end_process;

process Savings_Authorization(sel: SavingsServiceCollection,
                                                    savings_inf: CustomerInf)
                                                    savings_inf1: CustomerInf |
                                                    savings_inf2: CustomerInf |
                                                    savings_inf3: CustomerInf |
                                                    savings_inf4: CustomerInf |
                                                   savings_inf5: CustomerInf |
                                                    e_mesg2: string
ext rd savings_accounts
post if savings_inf inset dom(savings_accounts)
      then case sel of
            <1> --> savings_inf1 = savings_inf;
            <2> --> savings_inf2 = savings_inf;
```

```
                <3> --> savings_inf3 = savings_inf;
                <4> --> savings_inf4 = savings_inf;
                <5> --> savings_inf5 = savings_inf;
                end
        else e_mesg2 = "Your password or account number is incorrect."
explicit
        if savings_inf inset dom(savings_accounts)
        then case sel of
                <1> --> savings_inf1 := savings_inf;
                <2> --> savings_inf2 := savings_inf;
                <3> --> savings_inf3 := savings_inf;
                <4> --> savings_inf4 := savings_inf;
               <5> --> savings_inf5 := savings_inf
                end
        else e_mesg2 := "Your password or account number is incorrect."
comment
if the input account_no and password match those of the customer's
            savings account in the store savings_accounts
 then generate output data flows based on the value of variable sel
 else output an error message.
end_process;

process Savings_Deposit(d_amount: nat0, savings_inf1: CustomerInf)
                                        notice1: Notice | warning3: string
ext wr savings_accounts;
post if d_amount <= maximum_deposit_once
        then
            savings_accounts =
             override(~savings_accounts,
                 {savings_inf1 -->
                     modify(~savings_accounts(savings_inf1),
                        balance -->
                          ~savings_accounts(savings_inf1).balance + d_amount,
                        transaction_history -->
                          conc(~savings_accounts(savings_inf1).transaction_history,
                            [Get_Savings_Transaction(savings_accounts,          today,
```

```
current_time, 0, d_amount, savings_inf1)]
                                    )
                                   )
                        }
                        ) and
                        notice1                    =                    mk_Notice(d_amount,
savings_accounts(savings_inf1).balance))
              else warning3 = "Your amount is over 1000000 yen limit."
      explicit
      account_inf: savingsAccountInf;
      transaction: Transaction;
      begin
            account_inf := new savingsAccountInf;
            transaction := new Transaction;
            if d_amount <= maximum_deposit_once
              then
                begin
                  account_inf := savings_accounts(savings_inf1);
                  account_inf.Increase_Balance(d_amount);
                  account_inf.Update_Transaction_History(
                    transaction.Get_Savings_Transaction(savings_accounts,          today,
current_time,
                                                          0, d_amount, savings_inf1));
                   savings_accounts :=
                    override(savings_accounts,
                        {savings_inf1 --> account_inf});
                  notice1 := new Notice;
                  notice1.Make_Notice(d_amount,
                                              savings_accounts(savings_inf1).balance)
                  end
              else warning3 := "Your amount is over 1000000 yen limit."
          end
      comment
        if the input d_amount is less than or equal to the maximum_deposit_once
        then
              (1) add the d_amount to the savings_account
```

(2) give the customer a notice showing the amount of deposit and the updated balance

(3) update the transaction history of the account

else give a warning message to indicate that the amount is over the limit.

end_process;

```
process Apply_Withdraw(savings_inf2: CustomerInf, a_amount: nat0)
                                        a_notice: string | warning4: string
ext wr savings_accounts
post if a_amount <= maximum_withdraw_application
       then
        savings_accounts =
        override(~savings_accounts,
                    {savings_inf2 -->
                     modify(~savings_accounts(savings_inf2),
                         withdraw_application_amount: nat0 -->
                         a_amount,
                         application_status --> true)
                    }
                  ) and
        a_notice = "Your application is successful"
        else warning4 = "Your application amount is over the limit."
explicit
  begin
      account_inf := new SavingsAccountInf;
      if a_amount <= maximum_withdraw_application
       then
        begin
          account_inf := savings_accounts(savings_inf2);
           account_inf.Set_Application_Amount(a_amount);
           savings_accounts :=
              override(~savings_accounts,
                          {savings_inf2 --> account_inf)}
                          );
           a_notice := "Your application is successful"
        end
```

```
        else warning4 := "Your application amount is over the limit."
    end
    comment
    If the applied withdraw amount a_amount is within the fixed limit,
    then change the withdraw_application_amount to the applied withdraw
    amount and the application_status to true, indicating the application is
    made, and issue a notice to tell the customer that the application is
    successful. Otherwise, give a warning message to indicate that the
    applied withdraw amount is over the fixed limit.
    end_process;

    process Savings_Withdraw(savings_inf3: CustomerInf, w_amount: nat0)
                                notice2: Notice | warning5: string
    ext wr savings_accounts
    post if w_amount <=
~savings_accounts(savings_inf3).withdraw_application_amount and
            w_amount <= ~savings_accounts(savings_inf3).balance
        then
          savings_accounts =
            override(~savings_accounts,
                {savings_inf3 -->
                    modify(~savings_accounts(savings_inf3),
                      balance -->
                        ~savings_accounts(savings_inf3).balance - w_amount,
                      transaction_history -->
                        conc(~savings_accounts(savings_inf3).transaction_history,
                          [Get_Savings_Transaction(savings_accounts, today,
savings_time, w_amount, 0, savings_inf3)]
                                )
                               )
                }
                ) and
                notice2 = mk_Notice(w_amount,
savings_accounts(savings_inf3).balance))
        else warning5 = "Your withdraw amount is over the limit."
    explicit
```

```
        account_inf: SavingsAccountInf;
        transaction: Transaction;
        begin
          account_inf := new SavingsAccountInf;
          transaction := new Transaction;
          if                                    w_amount                                    <=
~savings_accounts(savings_inf3).withdraw_application_amount and
              amount <= savings_accounts(savings_inf3).balance
           then
           begin
             account_inf := savings_accounts(savings_inf3);
             account_inf.Decrease_Balance(w_amount);
             account_inf.Update_Transaction_History(
             transaction.Get_Savings_Transaction(savings_accounts,          today,
current_time,
                                                  w_amount, 0, savings_inf3));
              savings_accounts :=
               override(savings_accounts,
                   {savings_inf3 --> account_inf});
             notice2 := new Notice;
             notice2.Make_Notice(w_amount,
                                          savings_accounts(savings_inf3).balance)
             end
          else warning5 := "Your amount is over 1000000 yen limit."
       end
    comment
     if the input w_amount is less than or equal to the applied withdraw amount
         and the balance of the savings account
     then
           (1) output the cash of the requested amount
           (2) reduce the withdraw amount from the balance
           (3) update the transaction history of the account
           (4) give a notice
    else
           generate a warning message
    end_process;
```

```
process Savings_Show_Balacnce(savings_inf4: CustomerInf)
                                        s_balance: nat0
ext rd savings_accounts
post s_balance = savings_accounts(savings_inf4).balance
explicit
s_balance := savings_accounts(savings_inf4).balance
comment
Display the balance of the customer's savings account
end_process;

process Savings_Print_Transaction_Records(savings_inf5: CustomerInf, date:
Date)
                                                transaction_records:
TransactionRecords
ext rd savings_accounts
post let transactions = savings_accounts(savings_inf5).transaction_history
        in let i = get({i | i: inds(transactions) & transactions(i).date = date})
            in
              transaction_records = transactions(i, ..., len(transactions))
explicit
transactions: seq of Transaction;
index: nat0;
begin
  transactions := savings_accounts(savings_inf5).transaction_history;
  index := get({i | i: inds(transactions) & transactions(i).date = date});
  transaction_records := transactions(index, ..., len(transactions))
end
comment
Print out the transaction records since the input date
end_process;

process Savings_Display_Information(notice1: Notice |
                                    a_notice: string |
                                     notice2: Notice |
                                     s_balance: nat0 |
```

```
                                                transaction_records: TransactionRecords)
    ext wr output_device
    post bound(notice1) and output_device = conc(~output_device, [notice1]) or
            bound(a_notice) and output_device = conc(~output_device, [a_notice]) or
            bound(notice2) and output_device = conc(~output_device, [notice2]) or
            bound(s_balance) and output_device = conc(~output_device, [s_balance]) or
            bound(transaction_records) and output_device =
                conc(~output_device, [transactions_records])
    explicit
     if bound(notice1)
     then output_device := conc(~output_device, [notice1])
     else if bound(a_notice)
              then output_device := conc(~output_device, [a_notice])
              else if bound(notice2)
                       then output_device := conc(~output_device, [notice2])
                       else if bound(s_balance)
                                then output_device := conc(~output_device, [s_balance])
                                else output_device := conc(~output_device,
[transactions_records])
    comment
    Display the input data flows onto the output device based on their
    availability.
    end_process;

    process Savings_Display_Message(warning3: string |
                                                    warning4: string |
                                                   warning5: string |
                                                    e_mesg2: string)
    ext wr output_device
    post bound(warning3) and output_device = conc(~output_device, [warning3]) or
            bound(warning4) and output_device = conc(~output_device, [warning4]) or
           bound(warning5) and output_device = conc(~output_device, [warning5]) or
            bound(e_mesg2) and output_device = conc(~output_device, [e_mesg2])
    explicit
     if bound(warning3)
     then output_device := conc(~output_device, [warning3])
```

```
    else if bound(warning4)
        then output_device := conc(~output_device, [warning4])
        else if bound(warning5)
            then output_device := conc(~output_device, [warning5])
            else output_device := conc(~output_device, [e_mesg2])
comment
  Display the input data flows onto the output device based on their availability.
end_process;
end_module;
```

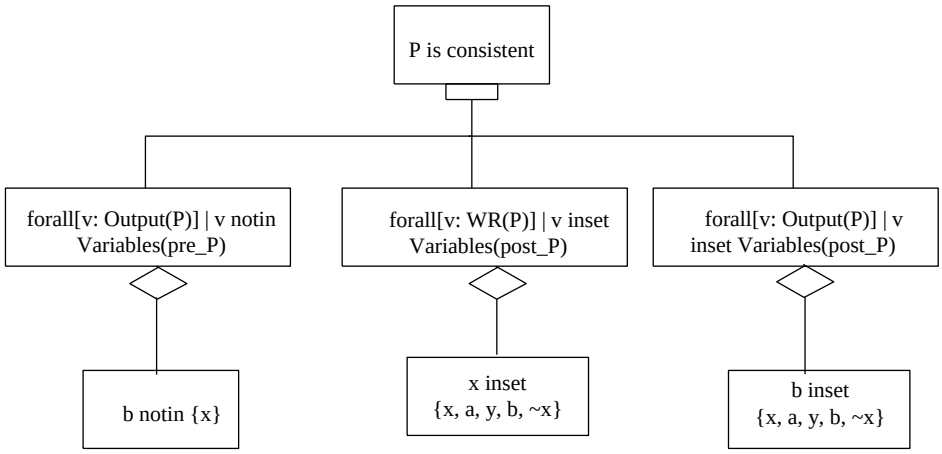
Exercise 17

1. Suppose the process P is defined as follows:

```
process P(a: int) b: set of int
ext wr x: set of int
      rd y: int
pre card(x) <> 0
post inter(x, b) = union({a, y}, ~x)
end_process
```

Build a review task tree for the reviewing the internal consistency of process P, and tell whether the process is internally consistent.

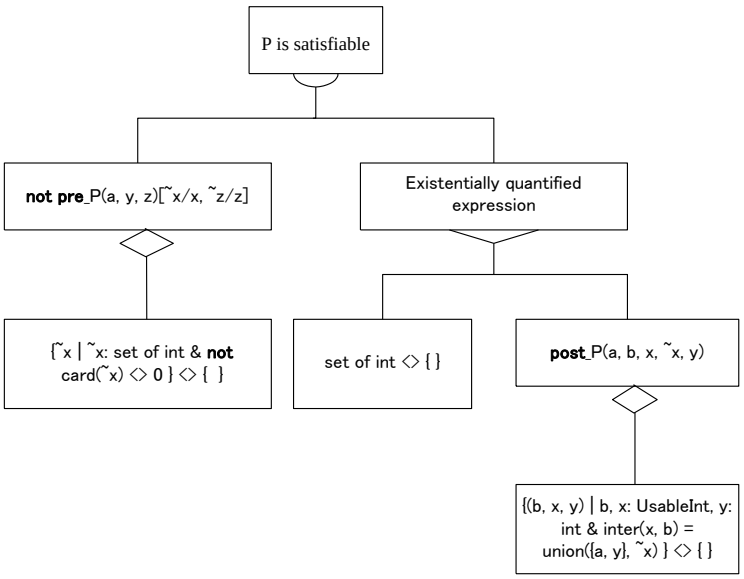
Answer:



The process is internally consistent because all the bottom-level tasks are confirmed to be true.

2. Build a review task tree for both constructive review and critical review of the satisfiability of process P given in problem 1, and tell if process P is satisfiable.

Answer:



The process is satisfiable because all the bottom-level tasks are true.

3. Construct review task trees for the "library system" required in exercise 2 of chapter 14 to review the properties: internal consistency of each process involved, the consistency between each process and the invariants (if any), satisfiability of each process, and the internal consistency of all the CDFDs involved in the specification.

Answer: leave this for the reader.

Exercise 18

1. Answer the questions:

   a. what is a test case ?

   Answer: Let P(x_1, x_2, ..., x_n) be a predicate expression. A test case for this expression is a group of values v_1, v_2, ..., v_n bound to x_1, x_2, ..., x_n, respectively.

   b. what is a test set ?

   Answer: A test set is a set of test cases.

   c. what is a test suite ?

   Answer: A test suite is a test set with expected test results.

   d. what is a test target ?

   Answer: A test target is an object to be tested. Such an object is a property of the

specification under testing in SOFL specification testing method.

e. what are possible ways of generating test cases ?

Answer: A test case can be generated based on either the user requirements (informal) or the formal specification to be tested.

f. what are the three steps for testing a specification ?

Answer: (1) Generation of test cases, (2) evaluation of the specification, (3) analysis of test results.

g. what is a failed test, successful test, and uncertain test ?

Answer: A failed test is a test that does not detect any fault. A successful test is a test that detects a fault. An uncertain test is a test that is unable to tell whether the specification has a fault or not.

h. is it possible to have a successful test for a process ? If so, give an example. If not, explain why.

Answer: Yes, it is possible to have a successful test for a process. For example, let process P be defined as follows:

```
process P(x: int) y: bool
pre x > 0
post y and not y
end_process
```

In this case, we can generate the following test to detect the fault in the postcondition (postcondition is a contradiction).

x	y	pre	post
1	true	true	false
2	false	true	false

Since true and false constitute the type of variable y (i.e., the range of the possible values that can be taken by y), there is no other possible values for y. Examining the results of postcondition, we understand that all the results are false when the precondition is true. This has proved that the test is a successful test.

2. Generate a test based on Criterion 2 for process A1 …. Where A1 is defined as

```
process A1(x: seq of nat0) d1, d2: seq of nat0
pre   len(x) > 0
post forall[a: elems(d1)] a < 60 and
        forall[b: elems(d2)] | b >= 60 and
        union(elems(d1), elems(d2)) = elems(x)
end_process
```

Answer: We consider to test the proof obligation of the process: pre => post, which is the same as: not pre or post. A test for this predicate expression is as follows:

x	d1	d2	not pre	post	not pre or post
[50, 79]	[50]	[79]	false	true	true
[ ]	[ ]	[ ]	true	true	true
[60, 23]	[60]	[23]	false	false	false

3. Generate a test based on Criterion 3 for process A2 …. Where process A3 is defined as

process A2(d1: seq of nat0) d3: seq of nat0
post d1 = [ ] and d3 = [ ] or
d1 <> [ ] and subset(elems(d3), elems(d1)) and
forall[e: elems(d3)] | e >= 40
end_process

Answer: We consider to test the proof obligation: not pre or post. The specific proof obligation for process A2 is:

not true or d1 = [ ] and d3 = [ ] or d1 <> [ ] and subset(elems(d3), elems(d1)) and forall[e: elems(d3)] | e >= 40

We abstract this expression and let it be denoted by P:

P = not pre or Q1 or Q2

Then a test for P is as follows:

d1	d3	not pre	Q1	Q2	P
[0, 1, 5]	[0, 5]	false	false	false	false
[ ]	[ ]	false	true	false	true
[5, 50]	[50]	false	false	true	true

This test has detected a fault in the postcondition due to the false value of P (the first line). The fault is that for the input d1 containing no natural numbers greater than or equal to 40, there does not exist any output d3 that can meet the postcondition.

5. Generate a different test from the one given in this chapter for the verification of the consistency between process A and its decomposition …. Where process A is defined

as:

process A(x: seq of nat0) y, z: nat0
pre len(x) > 0
post y < len(x) and z < len(x)
decom G
end_process;

And all the processes in the decomposition of process A are defined in section Testing decompositions. The necessary predicate expressions for testing are:

Con1: pre_A => pre_A1

Con2: pre_A1 and post_A1 => pre_A2 and pre_A3

Con3: pre_A2 and post_A2 => pre_A4

Con4: (pre_A1 and post_A1) and (pre_A3 and post_A3) and
(pre_A4 and post_A4) => post_A

Answer:

The tests for each of the conditions is given below, respectively.

For Con1:

x	pre_A	pre_A1	Con1
[35, 50, 85, 39]	true	true	true
[35, 50, 95]	true	true	true
[ ]	false	false	true
[28, 40]	true	true	true

For Con2:

x	d1	d2	conj1	conj2	Con2
[35, 50, 85, 39]	[35, 50, 39]	[85]	true	true	true
[35, 85, 95]	[35]	[85, 95]	true	true	true
[ ]	[ ]	[ ]	false	true	true
[28, 60]	[28]	[60]	true	true	true

For Con3:

d1	d3	pre_A2	post_A2	pre_A4	Con3
[35, 50, 39]	[85]	true	true	true	true
[35]	[ ]	true	true	true	true
[ ]	[ ]	true	true	true	true

[28]	[ ]	true	true	true	true

For Con4:

x	d1	d2	d3	z	y
[35, 90, 85, 39]	[35, 39, 85]	[90]	[85]	1	1
[35, 85, 95]	[35]	[35]	[ ]	1	0
[ ]	[ ]	[ ]	[ ]	0	0
[28, 60]	[28]	[28]	[ ]	1	0

conj3	conj4	conj5	post_A	condition 4
false	true	true	true	true
true	true	true	true	true
false	true	true	false	true
true	true	true	true	true

Exercise 19

1. Give another way of transforming a source module and class that differs from that of the one given in the section Transformation of modules and classes.

Answer:

Another possible way to transform a source module is as follows. Let M be a module, and P1, P2, and P3 be its processes. Then

(1) Transform M into a target class S.

(2) Transform each process of P1, P2, and P3 into a target class in which a method is created to implement its function specification.

(3) Transform the CDFD associated with module M into a method of the target class S in which all the methods of the objects of the classes generated from the processes are integrated in accordance with the semantics of the CDFD of module M.

2. Give a transformation of process A that is different from the one given in section Transformation of one-port processes in the sense that the target method A produces an error message when the precondition is not satisfied by the inputs.

Answer:

```
class Transformation1 {
To_1 y_1;
To_2 y_2;
```

```
    ...
    To_m y_m;
    ...
    public void A(Ti_1 x_1, Ti_2 x_2, ..., Ti_n x_n) {
     if (pre_A)
     {
        Tran(post_A)
     }
     else
      println("The precondition is violated");
    ...
  }
```

3. Give another different transformation of process B with two input and output ports, whose format is given in section Transformation of multiple-ports processes.

Answer:

Another different transformation strategy is to transform process B into two separated methods in the target class corresponding to the module in which process B is defined. For example, consider the module:

```
    module A;
      …
    process B (x_1: Ti_1 | x_2: Ti_2) y_1: To_1 | y_2: To_2
    pre pre_B
    post post_B
    end_process
     …
  end_module;
```

Then we transform module A into the following target class:

```
class A;
  …
public To_1 B1(Ti_1 x_1) {
 implementation of the related part in the pre_B and post_B
}

public To_2 B2(Ti_2 x_2) {
 implementation of the related part in the pre_B and post_B
```

```
 }
 …
}
```

5. Suppose process A is decomposed into a CDFD. Give a transformation of A that utilizes the CDFD in defining the body of the target method.

Answer:

```
public void A(…) {
  CDFD d = new CDFD();   //CDFD is the class generated from the CDFD of process A.

  Algorithm (d.A1(…), d.A2(…); d.A3(…));      //The algorithm implementing the
```

decomposed CDFD based on the invocations of the three methods A1(), A2(), and A3() which are supposed to be derived from the processes A1, A2, and A3 of the decomposed CDFD. Also, we assume that the CDFD contains only three processes.

Exercise 20

1. Describe the major differences between a traditional software engineering environment and an intelligent software engineering environment.

Answer:

The major differences are:

(1) The intelligent software engineering environment (ISEE) is able to guide the developer to follow the well-established process to develop his or her software systems, while a traditioanl software engineering environment (TSEE) offers only a collection of inter-related tools that can be freely used for development of software systems.

(2) The ISEE usually have knowledge on either domain or the method or both, while the TSEE usually has little such knowledge.

(3) The human developer is treated as a software tool under control of the ISEE, but as a person who can control the development activities.

2. Give some idea about building an intelligent office environment by simulating the idea of intelligent software engineering environment.

Answer:

A traditional office environment offers a collection of tools that can be used by people, while an intelligent office uses the available tools automatically in

accordance with the activities of the people. For example, when a person enters the office and it is too dark, the light will be automatically turn on, and when the person has left the office for a certain time, the light will be automatically swiched off. In the similar principle, air conditioner, computer, chairs, windows, and so on can also provide automatic services.

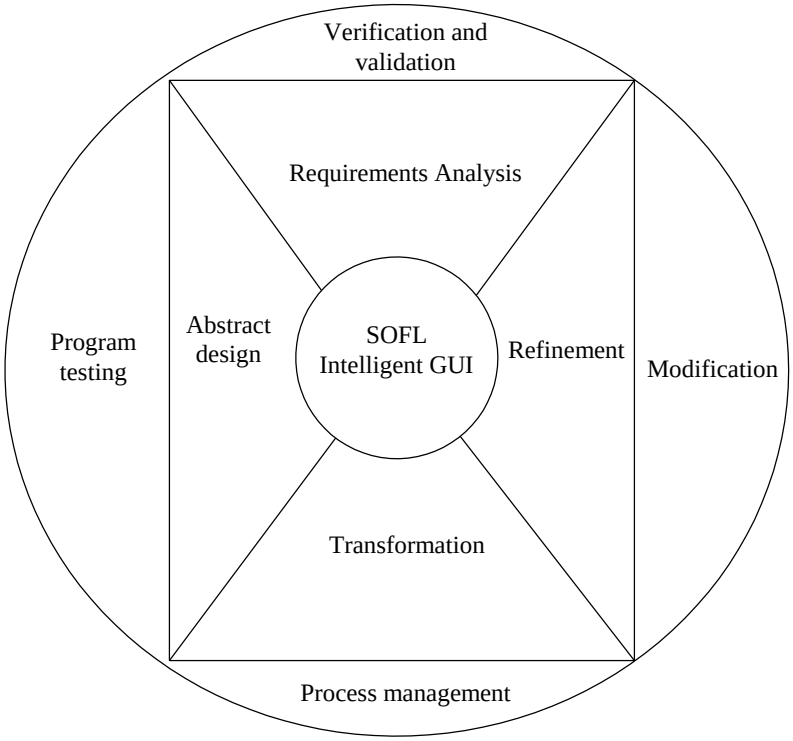
3. Explain why it is important that human developers be treated as a "software tool" in an intelligent software engineering environment.

Answer:

It is important because only in that way the ISEE can control the entire software development process and reduce the opportunities of creating faults occurred due to the human mistakes.

4. Draw a diagram to depict an intelligent software engineering environment for SOFL that provides all the functions presented in section ISEE for SOFL. Those functions need to be arranged

Answer: One arrangement of the ISEE for SOFL is shown in the following figure that emphasizes the layer of the tools. The fundamental tool is the intelligent GUI. Then the second layer of the tools that can be built on the basis of the GUI include those supporting requirements analysis, abstract design, refinement, and transformation. On the basis of these tools, the next layer of the tools can be built, they include those supporting verification and validation, program testing, system modification, and process management.



ISEE for SOFL