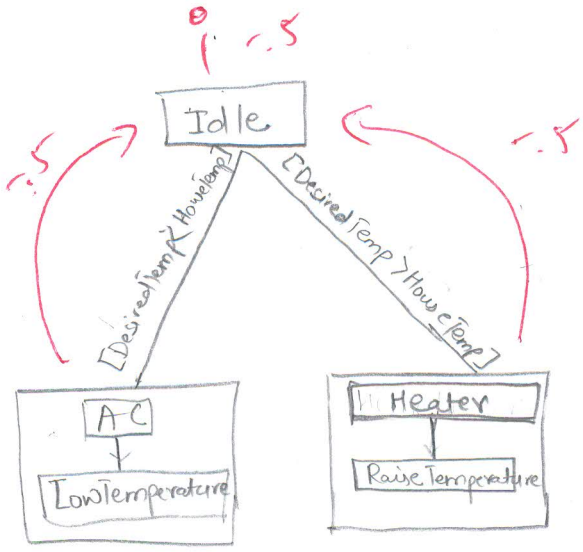
(4pts) 1. Consider the following specification for a heating control system:

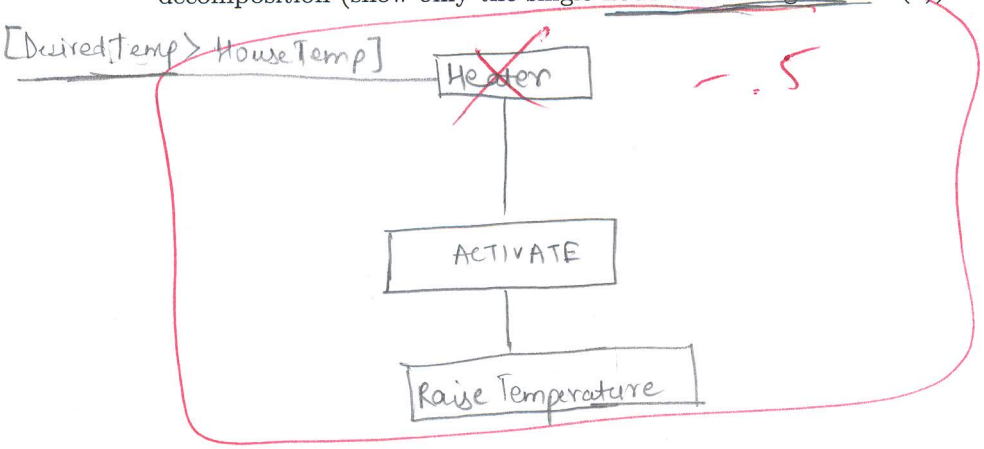
2
4 pts

Consider the life of your home's thermostat on one crisp fall day. In the wee hours of the morning, things are pretty quiet for the thermostat because the temperature of the house is stable. Toward dawn, however, the ambient temperature raises slightly, and/or a family member might twist the thermostat's dial. In case the temperature of the house is higher than the desired temperature, the thermostat starts behaving by commanding the home's air conditioner to lower the insider temperature. In case the temperature of the house is lower than the desired temperature, the thermostat starts behaving by commanding the home's heater to raise the inside temperature. In the evening, with the heat from cooking, the thermostat has a lot of work to do t keep the temperature even when it runs the heater and cooler efficiently. Finally, at night, things return to quiet state.

(a) (3 pts) Draw a UML state machine diagram to specify thermostat behaviour:



(b) (1 pt) Now additionally consider the requirement that the heater needs *activating* before it can be actively heating. Show this with a hierarchical state using sequential decomposition (show only the single node that changed from (a)):



(1pt) **2.** Name two methods for identifying *conceptual classes* in domain modeling:

0 / 1 pt

1. _____

2. _____

(1pt) **3.** Select the software engineering activities where traceability links are helpful *(check all correct answers)*:

0 / 1 pt

- [x] Compliance Checking
- [x] Defect Tracking
- [x] Software Evolution
- [x] Coverage Analysis
- [x] Project Tracking

(2pts) **4.** *True* or *False*? ***Note:*** Each correct answer gives you $\frac{1}{2}$ points, each wrong answer $-\frac{1}{2}$, "don't know" 0 (however, you will never get a negative score for this question).

1 / 2 pts

Traceability links between (requirements) documents and source code can be partially generated by automated methods:

- [x] True
- [ ] False
- [ ] Don't know

In traceability, one artifact can have at most a single incoming and a single outgoing link:

- [ ] True
- [x] False
- [ ] Don't know

A feature diagram is one way to capture and represent traceability links:

- [x] True
- [ ] False
- [ ] Don't know

In requirements evolution, when classifying requirements by change likelihood, we use as many stability levels as possible:

- [ ] True
- [x] False
- [ ] Don't know

(1pt) **5.** Consider a requirement represented as a propositional logic formula r that is *valid* (written $\models r$). This means *(check only one answer)*:

1 / 1 pt

- [ ] the requirement is *inconsistent*
- [x] the requirement is *redundant*
- [ ] the requirement is *unsatisfiable*
- [ ] the requirement is *ambiguous*
- [ ] the requirement is *illogical*
- [x] nothing – has no particular impact for the requirements specification

(5pts) **6.** Consider the following requirements statements for a library:

2
5 pts

1. A book can have a printed or electronic edition (or both).
2. If a book has a printed edition, then we need bookshelves.
3. If a book has an electronic edition, then it is never unavailable.
4. A book is unavailable.

1½

(a) (1 pt) Translate these requirements into propositional logic, using the abbreviations:
P: *book has printed edition*
E: *book has electronic edition*
B: *bookshelves needed*
U: *book unavailable*

1. $P \vee E$ ✓
2. $P \rightarrow B$ ✓
3. $E \rightarrow U$ ✓
4. ~~$(P \wedge E) \leftrightarrow U$~~

1½

(b) (4 pts) Using a *proof by resolution*, show that the statement
We need bookshelves.
logically follows from the requirements:

Statement: we need bookshelves (B) ✓
Proof by contradiction
Statement: we donot need bookshelves ($\neg B$) ✓

$P \rightarrow B \Rightarrow \neg P \vee B$ (-)

$\neg(\neg P \vee B)$

$\Rightarrow P \wedge \neg B$

(Book has printed edition and bookshelves not needed)
contradiction

2

not a proof by resolution!

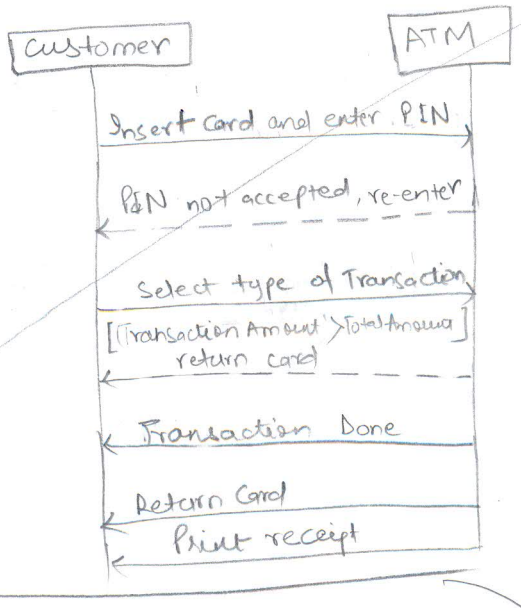
0
4 pts

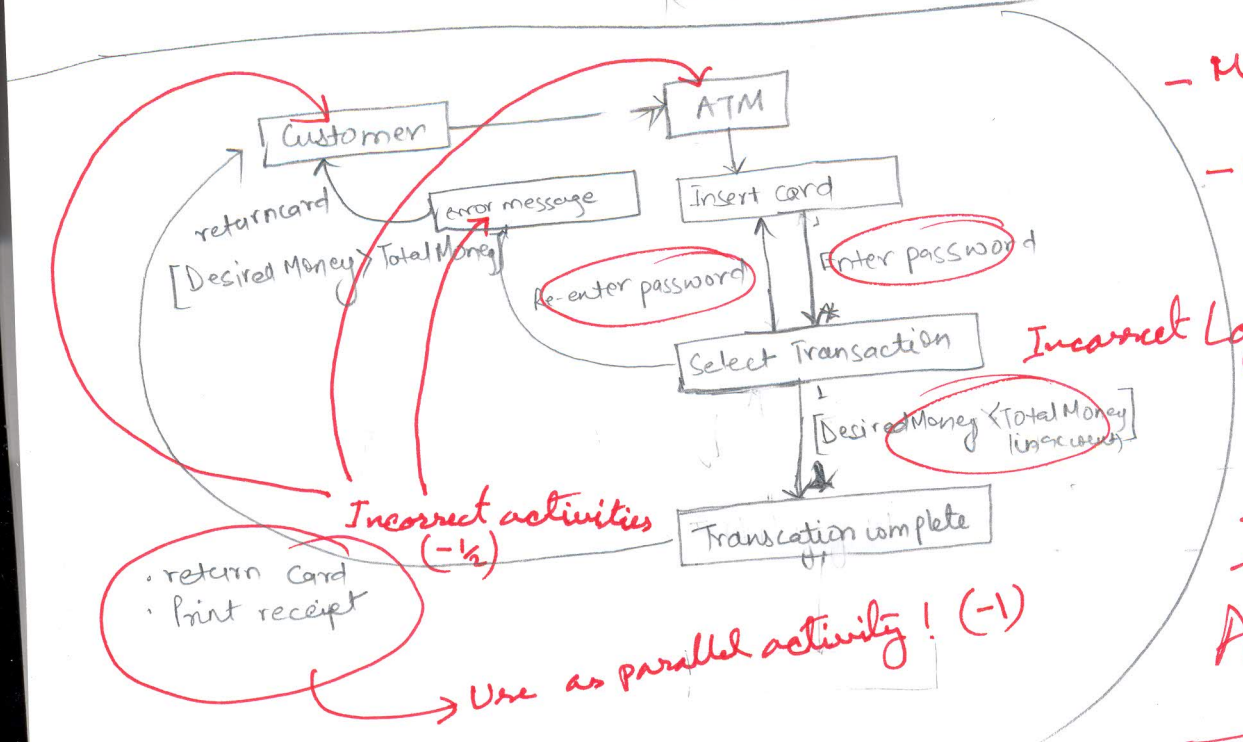
(4pts) **7.** Consider the following specification for an automated teller machine (ATM):

A customer arrives at an ATM. He insert his card and then enters the password (PIN). If the password is not accepted he has to re-enter it. If the password is accepted the customer selects the type of transaction he wishes to conduct and then the ATM performs the transaction. If the ATM detects that it has not enough money to fulfil the request it prints an error message, returns the card, and ends the transaction.

Once the transaction is complete the customer has the option to perform more transactions. If no further transactions are selected, the ATM concurrently returns the card and prints a receipt of all transactions and the interaction with the ATM terminates.

Draw a UML *activity diagram* for this description. Show explicitly the allocation of the activities among the customer and the ATM (use swimlanes).





(7pts) **8.** Consider this specification for a simple document control system:

5
7 pts

> If a user wants to check out a document in order to change the document and the user has the permission to change it, and nobody else is changing it at the moment, then that user may check the document out.
>
> As soon as a user has checked out a document for editing, everyone else is disallowed from checking it out (of course people with read permission can read it).
>
> When the user is done editing the document, it should be checked in, allowing another user to check it out.

Your task is to define the requirements formally in Z, based on the following schema:

$[PERSON, DOCUMENT]$

Documents

$checked\_out : DOCUMENT \pfun PERSON$
$permission : DOCUMENT \rel PERSON$

$checked\_out \subseteq permission$

Note: the relation *permission* is just a set of pairs of the form (*document*, *person*).

(a) (1 pt) Show one possible, *legal* system state for *Documents* with exactly two persons {*Joe*, *Anne*} and three documents {*Req*, *Design*, *Test*}:

1

$checked\_out = \{Req \mapsto Joe, Design \mapsto Joe, Test \mapsto Joe, Req \mapsto Anne, Design \mapsto Anne, Test \mapsto Anne\}$

$permission = \{(Req, Joe), (Design, Joe), (Test, Joe), (Req, Anne), (Design, Anne), (Test, Anne)\}$ ✓

(b) (3 pts) Write a non-robust Z schema for the *CheckOut* operation. It has two input parameters, the person $p?$ and the document $d?$. Note that a document can only be checked out if (1) the person checking it out has permission to do so and (2) the document is not currently checked out.

2

CheckOut

$\Delta Documents$

$p? : \cancel{P}PERSON$ (−½)

$d? : DOCUMENT$ ✓

$(d?, p?) \in permission$ ✓

$d? \in domchecked\_out$ ✓

$checkout' = (d? \cancel{\rightarrow} p?) \land (d? \rightarrow p?)$ (−½)

⇒ ***Continued on next page!***

(c) (1 pt) Now make the operation *robust* by using the following schemas for error handling:

$REPORT ::= ok \mid unauthorized \mid unavailable$

Success

$result! : REPORT$

$result! = ok$

Unauthorized

$\Xi Documents$
$p? : PERSON$
$d? : DOCUMENT$
$result! : REPORT$

$(d?, p?) \notin permission$
$result! = unauthorized$

Unavailable

$\Xi Documents$
$d? : DOCUMENT$
$result! : REPORT$

$d? \in \text{dom } checked\_out$
$result! = unavailable$

Define a robust version of *CheckOut* that returns *ok* in case of success and *unauthorized* or *unavailable* in case of an error:

$RCheckOut =$ (checkout ∧ Sucess) ∨ (unauthorized ∨ unavailable)

(d) (2 pts) Now show the combined schemas for the *RCheckOut* operation:

RCheckOut

ΔCheckout
p?: PERSON
d?: DOCUMENT
result!: REPORT

(d?, p?) ∉ permission
checkout' = [{(d? → p?) ∧ (d? ↔ p?) ∧ result! = ok} ∨
{result! = unauthorized ∨ result! = unavailable}]

Additional Information on Propositional Logic

Truth tables for $\neg$, $\wedge$, and $\vee$

p	$\neg p$
T	F
F	T

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Truth tables for $\leftrightarrow$ and $\rightarrow$

p	q	$p \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Equivalence Rules

Equivalence Rule	Name
$p \Leftrightarrow \neg\neg p$	double negation
$p \rightarrow q \Leftrightarrow \neg p \vee q$	implication
$\neg(p \wedge q) \Leftrightarrow \neg p \vee \neg q$ $\neg(p \vee q) \Leftrightarrow \neg p \wedge \neg q$	De Morgan's laws
$p \vee q \Leftrightarrow q \vee p$ $p \wedge q \Leftrightarrow q \wedge p$	commutativity
$p \wedge (q \wedge r) \Leftrightarrow (p \wedge q) \wedge r$ $p \vee (q \vee r) \Leftrightarrow (p \vee q) \vee r$	associativity

Inference Rules

Inference Rule	Name
$\left.\begin{array}{r} p \\ q \end{array}\right\} \Rightarrow p \wedge q$	conjunction
$\left.\begin{array}{r} p \\ p \rightarrow q \end{array}\right\} \Rightarrow q$	*modus ponens*
$\left.\begin{array}{r} \neg q \\ p \rightarrow q \end{array}\right\} \Rightarrow \neg p$	*modus tollens*
$\left.\begin{array}{r} p \rightarrow q \\ q \rightarrow r \end{array}\right\} \Rightarrow p \rightarrow r$	chaining
$\left.\begin{array}{r} p \vee q \\ \neg p \vee r \end{array}\right\} \Rightarrow q \vee r$	resolution
$p \wedge q \Rightarrow p$	simplification
$p \Rightarrow p \vee q$	addition