Table of Contents

Introduction

This lab outlines specifications for travel requisition system. To make our specifications simple, we introduce some assumptions. Those are

1. A traveler fills out a form after going on a trip.
2. A manager must approve or reject a form.
3. A manager may give a form to manager's manager.
4. The top manager cannot give a form to anyone.

After the assumptions, we identify relevant sets of entities and relationships. Then we define functions relevant to this system. Finally, we included state machines diagrams for travelers and approvers. In addition, we used the Z notation for describing specifications of the system.

Set of entities

We identify the following entities.

[FORMS, EMPLOYEE, STATUS, NAME]

We assume that "FORMS" include travel forms and other forms. "EMPLOYEE" includes a top manager, managers and travelers. "STATUS" refers to status of a travel form and "NAME" is a name of an employee.

"STATUS" has the following values.

STATUS ::= "*filled*" | "*submitted*" | "*approved*" | "*rejected*"

These values are changed as the travel expense reimbursement process proceeds.

Schema

EmployeeDatabase schema

-------- EmployeeDatabase ----------

employee_member, traveler_member : ℙ NAME
manager_member, top_manager_member: ℙ NAME
department_db : ℙ (NAME x NAME)

---

top_manager_member = 1
top_manager_member ∉ manager_member
traveler_member ∩ top_manager_member = {}
top_manager_member ∉ employee_member

traveler_member = employee_member ∪ manager_member
employee_member ⊆ dom (department_db)
manager_member ⊆ ran (department_db)

Declaration :

- *employee_member*, *traveler_member*, *manager_member* and *top_manager* are subsets of EMPLOYEE.
- *department_db* has a type of power set of Cartesian product of NAME.

Invariants :

- We assume that we have only one top manager who doesn't have any supervisor and cannot be a traveler.
- Some managers can be travelers.

FormSystem schema

−−−−−−−− TravelForm −−−−−−−−−−−−−−−−−

EmployeeDatabase

traveler_name : NAME

approvedBoxChecked, rejectedBoxChecked : BOOL

totalExpenditure : ℕ
status : STATUS

---

totalExpenditure > 0

(∀x : traveler_name • x ∈ traveler_member)

Declarations:

- *traveler_name* has a type of NAME. Traveler can be employee or manager, but not top manager.
- *approvedBoxChecked* and *rejectedBoxChecked* are boxes to be checked depending on the status of the travel form, which have a type of BOOL.
- *totalExpenditure* is the total money expensed for a travel, which has a type of natural numbers.
- *status* is the status of the travel form.

Invariants:

- The total money expensed should be more than zero.
- All names that will be put in this form have to be the names of traveler.

FilledOutForms schema

−−−−−−−− FilledOutForms −−−−−−−−−−

EmployeeDatabase
Δ TravelForm

fillOut : NAME -> FORMS

travel_form : ℙ FORMS

formDatabase : ℙ (NAME x FORMS)

---

dom (fillOut) ⊆ traveler_member
ran (fillOut) ⊆ traveler_form

traveler_name ⊆ dom (formDatabase)

traveler_form ⊆ ran (formDatabase)

status' = "filled"

Declarations:

- *fillOut* is a function which is a map from natural numbers to FORMS.

Invariants:

- The domain of *fillOut* should be a subset of *traveler_member*.
- The range of *fillOut* should be a subset of *traveler_form*.
- *traveler_name* should be a subset of or equal to the domain of *formDatabase*.
- *traveler_form* should be a subset of or equal to the range of *formDatabase*.
- The status observed after *fillOut* performed should be "*filled*"

SubmitForm schema

−−−−−−−− SubmitForm −−−−−−−−−−−−−−

EmployeeDatabase
Δ TravelForm

submit : (ℕ x FORMS) -> formDatabase

∃x : NAME ; ∃y : FORMS |

x ∈ traveler_member ∧ y ∈ travel_form •

submit (x, y) = formDatabase ∪ { x ↦ y }

∃m : NAME ; ∃n : FORMS • submit (m, n) ⇔ status' = "submitted"

status ≠ status'

Declaration:

- *submit* is a function which is a map from Cartesian product between natural numbers and FORMS to FormDatabase.

Invariants:

- All travelers can submit the travel form to *formDatabase*.
- After submission, the status of form will be changed to "submitted".
- The previous status should be different form the status observed.

JudgeForm schema

−−−−−−−− JudgeForm −−−−−−−−−−−−−−

EmployeeDatabase
Δ TravelForm

submitted_form : ℙ (travel_form)
judge : (NAME x FORMS) -> STATUS

---

submitted_form ⊆ ran (formDatabase)

∃x : NAME; ∃y : travel_form |

x ∈ manager_name ∧ y ∈ submitted_form ∧ status = "submitted" ∧

approvedBoxChecked = FALSE ∧ rejectedBoxChecked = FALSE •
judge (x, y) = ((status' = "approved" ∧ approvedBoxChecked' = TRUE) ∨
(status' = "rejected" ∧ rejectedBoxChecked' = TRUE))

approvedBoxChecked' ≠ rejectedBoxChecked'

Declarations:
- *submitted_form* has a type of a power set of travel form.
- *judge* is a function which is a map from Cartesian product between natural numbers and forms to STUTASE

Invariants:
- *submitted_form* should be a range of FormDatabease.
- *approvedBoxCheck* should not equal *rejectedBoxChecked*.

State machines

We come up with three roles: traveler, approver, and secretary.

Traveler is an employee who travels to a trip. A trip here is defined as a business trip from one place to another place. For example, when traveler departs from company to one destination, it is a trip. When traveler has to travel to another destination. It is called another strip.
Approver is manager who that traveler is working for. His position has to be at least manger or above. Top manager, who is the highest in organization chart, is not considered to be a traveler, but (s)he can be an approver.
Each level of management can authorize to a certain limit amount of money.
Manager allows to authorize up to $5,000.
Senior manager allows to authorize unto $10,000.
Director allows to authorize up to $50,000.
Vice President allows to authorize unto $100,000.
President (top manager) allows to authorize unlimited.

When traveler finishes one trip, (s)he has to submit to his approver, who is his/her immediate manager, for approval. If the reimbursed amount is beyond the limit (s)he can authorize, that travel form is passed to the approver's manager. The form can pass to top manager as the last approver person, if necessary.

When receiving the travel form, approver judges to approve or reject this form. If approver is confused, it means that travel form is filled out correctly but approver is not sure whether (s)he should approve or reject it. In this case, the approver has to ask another manager to pickup and judge it.

If the travel form is rejected, it will send back to traveler. The traveler will modify the form and resubmit to his approver.

From above, we use Labeled Transition Systems (LTS) to formalize our model.

State Machine ::= (S, A, I, T) where

- S is a finite set of states
- A is set actions
- I is set of initial states
- $T \subseteq S \times S$ is a finite set of transitions

Although LTS supports the action transition from one state to another state, it doesn't allow us to define each state in formal language. The state is sequentially enumerated in integer. We will start to define the LTSs of each user roles, its state machine, and some event-based traces.

Traveler:

```
TRAVELER ==     (
                {0, 1, 2, 3, 4, 5},
                {       t_travel, t_fillOut, t_submitForm,
                        t_receiveRejectForm, t_getReimburse, t_modifyForm
                },
                {0},
                {       (0, t_travel, 1), (1, t_fillOut, 2), (2, t_submitForm, 3),
                        (3, t_receiveRejectForm, 4), (3, t_getReimburse, 5),
                        (4, t_modifyForm, 2), (5, t_travel, 1)
                }
                )
```

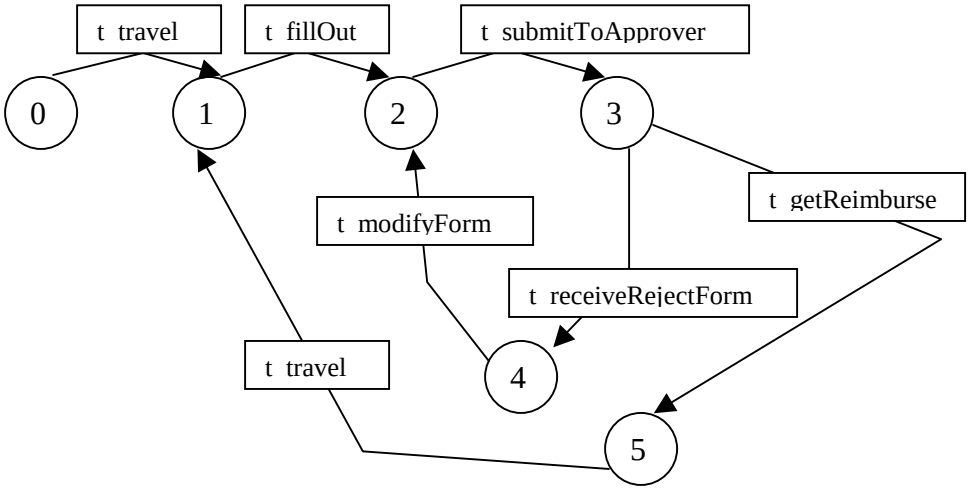

Figure 2.1 State machine of Traveler

The event-based traces of traveler is such as

- <t_travel, t_fillOut, t_submitToApprover, t_getReimburse>
- <t_travel, t_fillOut, t_submitToApprover, t_receiveRejectForm, t_modifyForm, t_ submitToApprover, t_getReimburse >

Approver:

```
APPROVER == (
                {0, 1, 2, 3, 4},
                {       a_receiveTravelForm, a_confused,
                        a_notConfused, a_approve,
                        a_pickUp, a_passFormToApproveManager,
                        a_rejectForm, a_reimburse
                },
                {0},
```

{ (0, a_receiveTravelForm, 1), (1, a_confused, 2),
(1, a_notConfused, 3), (2, a_pickUp, 1),
(3, a_passFormToApproverManager, 1),
(3, a_approve, 4), (3, a_rejectForm, 0),
(4, a_reimburse, 0)
}
)

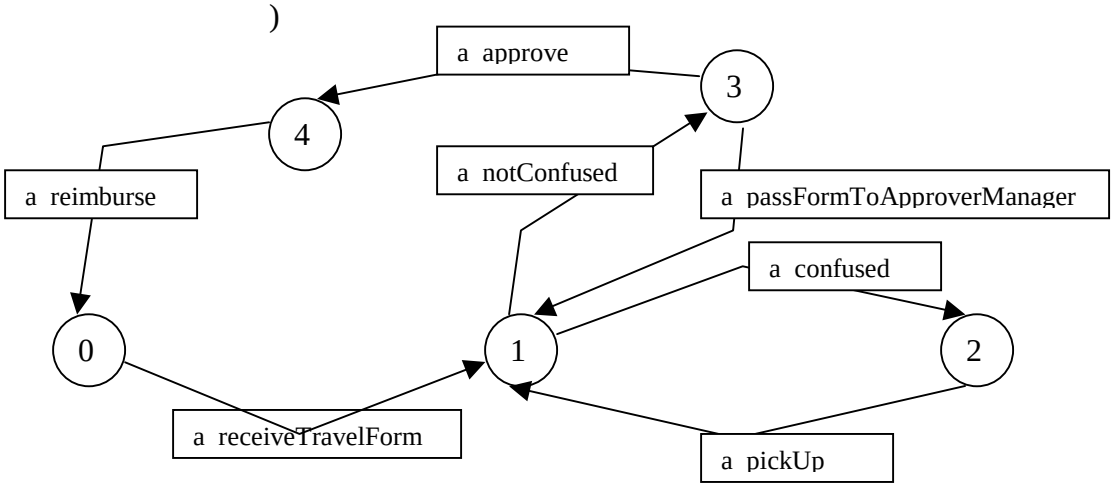

Figure 2.2 State machine of Approver

The event-based traces are such as:

- <a_receiveTravelForm, a_notConfused, a_rejectForm>
- <a_receiveTravelForm, a_Confused, a_pickUp, a_notConfused, a_approve, a_reimburse>

Secretary:
It is important to note that this user role is mainly to link between the traveler and approver user roles.

SECRETARY == (
{0, 1, 2, 3},
{ t_submitToApprover, t_receiveRejectForm,
t_getReimburse, a_receiveTravelForm,
a_rejectForm, a_reimburse
},
{0},
{ (0, t_submitToApprover, 1), (1, a_receiveTravelForm, 0),
(0, a_rejectForm, 2), (2, t_receiveRejectForm, 0),
(0, a_reimburse, 3), (3, t_getReimburse, 0)
}
)

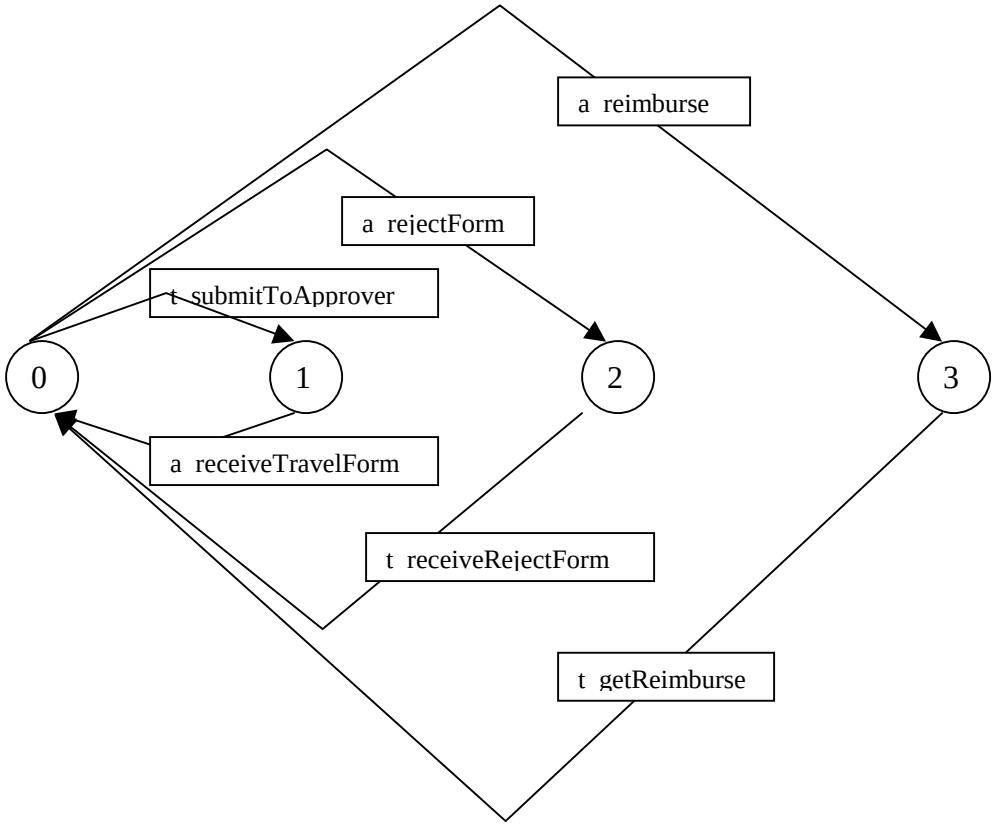

Figure 2.3 State machine of Secretary

The event-based traces are such as

- <t_submitToApprover, a_receiveTravelForm>
- <a_rejectForm, t_receiveRejectForm>
- <a_reimburse, t_getReimburse>