

## Lab Session 06

Practice Formal Method's tool to achieve preciseness in Software Specification Document.





1. Provide the given operational schemas in the given BirthdayBook Example using CZT Eclipse Tool.  
Also attach print outs.

- RemindBirthday
- AlreadyKnown
- NotKnown
- NewAddBirthday
- NewFindBirthday (Incorporate free type definitions in part "d" and "e")

```
*birthday.zed16
-- [ NAME,DATE ]
--
-- RESULT ::= Ok | Already_known |Not_Known
--
-- BirthdayBook
-- known:P NAME
-- birthday:NAME→DATE
-- |
-- | known=dom birthday
-- |
--
-- InitBirthdayBook
-- BirthdayBook
-- |
-- | known = ∅
-- |
```

```
*birthday.zed16
--
-- RemindBirthday
-- ⊆ BirthdayBook
-- todaysdate?:DATE
-- card!:P NAME
-- |
--
-- AlreadyKnown
-- ⊆ BirthdayBook
-- name?:NAME
-- result!:RESULT
-- |
-- | name? ∈ known
-- | result!=Already_known
-- |
--
-- NotKnown
-- ⊆ BirthdayBook
-- name?:NAME
-- result!:RESULT
-- |
-- | name? ∉ known
-- | result!=Not_Known
-- |
```

```
*birthday.zed16
--
-- NewAddBirthday
-- ΔBirthdayBook
-- name?:NAME
-- date?:DATE
-- result!:RESULT
-- |
-- | (name? ∉ known ∧ birthday!=birthday ∪ {name? ↦ date?} ∧ result!=Ok) ∨
-- | (name? ∈ known ∧ birthday!=birthday ∧ result!=Already_known)
-- |
--
-- NewFindBirthday
-- ⊆ BirthdayBook
-- name?:NAME
-- date!:DATE
-- result!:RESULT
-- |
-- | (name? ∈ known ∧ date!=birthday(name?)) ∧ result!=Already_known ∨
-- | (name? ∉ known ∧ result!=Not_Known)
-- |
```




 Z Characters  Z Conversion  Problems  Z Info
 

```
File birthday.zed16 converted to LaTeX (originally Unicode)
\begin{zsection}          \SECTION Specification \parents~standard\_toolkit
\end{zsection}

\begin{zed}[ NAME , DATE ]
\end{zed}

\begin{zed}RESULT ::= Ok | Already\_ known | Not\_ Known
\end{zed}

\begin{schema}{BirthdayBook}
\\
  known : \power NAME \\
  birthday : NAME \pfun DATE
\where
  known = \dom birthday
\end{schema}
```

 Z Characters  Z Conversion  Proble

File birthday.zed16 converted to LaTeX (originally U

```
\begin{schema}{InitBirthdayBook}
\\
  BirthdayBook
\where
  known =~\emptyset
\end{schema}
```

```
\begin{schema}{RemindBirthday}
\\
  \Xi BirthdayBook \\
  todaysdate? : DATE \\
  card! : \power NAME
\end{schema}
```

```
\begin{schema}{AlreadyKnown}
\\
  \Xi BirthdayBook \\
  name? : NAME \\
  result! : RESULT
\where
  name? \in known \\
  result! = Already\_ known
\end{schema}
```

Z Characters Z Conversion Problems Z Info

file birthday.zed16 converted to LaTeX (originally Unicode)

```
\begin{schema}{NotKnown}
\\
\Xi BirthdayBook \\
name? : NAME \\
result! : RESULT
\where
name? \notin known \\
result! = Not\_Known
\end{schema}
```

```
\begin{schema}{NewAddBirthday}
\\
\Delta BirthdayBook \\
name? : NAME \\
date? : DATE \\
result! : RESULT
\where
( name? \notin known \land birthday' = birthday \cup \{ name? \mapsto date? \} \land result! = Ok ) \lor ( name? \in known \land birthday' = birthday \land result!
```

```
\begin{schema}{NewFindBirthday}
\\
\Xi BirthdayBook \\
name? : NAME \\
date! : DATE \\
result! : RESULT
\where
( name? \in known \land date! = birthday ( name? ) \land result! = Already\_known ) \lor ( name? \notin known \land result! = Not\_Known )
\end{schema}
```

2. A login sub-system maintains a set of accounts, one for each user of the system. Each account consists of the username and password. It is required that names of the users must be unique in the system. A user can have multiple accounts with different names. Provide the type definition, state-space schema, and operational schemas for the following: (Also attach printouts)

- Initialize the system
- Add a new account
- Delete an existing account
- Change password of an account

```
*login.zed16
⊢ Credentials = —[ USERIDs , PASSWORDS ] ⊢
⊢ —
  MSGS ::= Login_Succeed | Login_Failed | Account_Added | Account_Deleted
⊢ —
⊢ Login
  User_name : P USERIDs
  Passwords : P PASSWORDS
  |
  ( User_name u Passwords ) ∈ P Credentials
  ∀ C1,C2 : Credentials | C1 ∈ Credentials ∧ C2 ∈ Credentials
  C1 = C2 ⇔ C1.User_name = C2.User_name
⊢ —
```

```
⊢ InitLogin
  Login
  |
  Credentials = ∅
⊢ —
⊢ AddAccount
  Δ Login
  User_name? : USERIDs
  Passwords? : PASSWORDS
  Msg! : MSGS
  |
  (User_name ∉ USERIDs) ∧ (Passwords ∉ PASSWORDS)
  USERIDs' = USERIDs u {User_name?}
  PASSWORDS' = PASSWORDS u {Passwords?}
  Msg! = Account_Added
⊢ —
```

```
⊢ DeleteAccount
  Δ Login
  User_name? : USERIDs
  Passwords? : PASSWORDS
  Msg! : MSGS
  |
  (User_name ∈ USERIDs) ∧ (Passwords ∈ PASSWORDS)
  USERIDs' = USERIDs \ {User_name?}
  PASSWORDS' = PASSWORDS \ {Passwords?}
  Msg! = Account_Deleted
⊢ —
⊢ Sign_In
  ≡ Login
  User_name? : USERIDs
  Passwords? : PASSWORDS
  Msg! : MSGS
  |
  (User_name ∈ USERIDs) ∧ (Passwords ∈ PASSWORDS)
  Msg! = Login_Succeed
  User_name ∉ USERIDs
  Msg! = Login_Failed
⊢ —
```

3. Consider the example of Block Handler for managing files in the OS. Provide schema specification using Z-notations. Include the operational schemas for “AddBlocks” and “RemoveBlocks” functions. Also provide its corresponding Latex code.

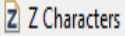
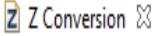
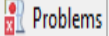

```

*block.zed16
[ BLOCKS ]
┌ BlockHandler
  used, free: P BLOCKS
  BlockQueue: seq P BLOCKS
  AllBlocks: P BLOCKS
  |
  used n free = ∅ ∧
  used u free = AllBlocks ∧
  ∀i : dom BlockQueue • BlockQueue i ⊆ used ∧
  ∀i,j: dom BlockQueue • i ≠ j ⇒ BlockQueue i ∩ BlockQueue j = ∅
└

┌ RemoveBlocks
  ΔBlockHandler
  |
  #BlockQueue > 0
  used' = used \ head BlockQueue ∧
  free' = free u head BlockQueue ∧
  BlockQueue' = tail BlockQueue
└

┌ AddBlocks
  ΔBlockHandler
  Ablocks? : P BLOCKS
  |
  Ablocks? ⊆ used
  BlockQueue' = BlockQueue ∧ (Ablocks?) ∧
  used' = used ∧
  free' = free
└

```

 Z Characters
  Z Conversion
  Problems
  Z Info

File block.zed16 converted to LaTeX (originally Unicode)

```

\begin{zsection}      \SECTION Specification \parents~standard\_toolkit
\end{zsection}

```

```

\begin{zed}[ BLOCKS ]
\end{zed}

```

```

\begin{schema}{BlockHandler}
\\
used , free : \power BLOCKS \\
BlockQueue : \seq \power BLOCKS \\
AllBlocks : \power BLOCKS
\where
used \cap free =~\emptyset \land used \cup free = AllBlocks \land ( \forall i : \dom BlockQueue @ BlockQueue~i \subs
\end{schema}

```

```

\begin{schema}{RemoveBlocks}
\\
\Delta BlockHandler
\where
\# BlockQueue > 0 \\
used' = used \setminus head~BlockQueue \land free' = free \cup head~BlockQueue \land BlockQueue' = tail~BlockQueue
\end{schema}

```

```

\begin{schema}{AddBlocks}
\\
\Delta BlockHandler \\
Ablocks? : \power BLOCKS
\where
Ablocks? \subseteqq used \\
BlockQueue' = BlockQueue \land ( Ablocks? ) \land used' = used \land free' = free
\end{schema}

```