Events:

1. Open()
2. Login()
3. IncorrectLogin()
4. IncorrectPin(int max)
5. CorrectPinAboveMin()
6. CorrectPinBelowMin()
7. Logout()
8. Deposit()
9. DepositAboveMin()
10. DepositBelowMin()
11. Balance()
12. Withdraw()
13. WithdrawAboveMin()
14. WithdrawPenalty()
15. NoFound()
16. Lock()
17. IncorrectLock()
18. IncorrectUnlock()
19. UnlockAboveMin()
20. UnlockBelowMin()
21. Suspend()
22. Activate()
23. Close()

Actions:

A1:
```
StoreData(){
    dop.pin = dop.temp_p;
    dop.balance = dop.temp_a
    dop.userID = dop.temp_y
}
```

A2:
```
IncorrectIDMsg(){
    Display the message for incorrect ID.
}
```

A3:
```
PromptPin(){
    Display and prompt the PIN.
}
```

A4:
```
IncorrectPinMsg(){
    Display the message for incorrect pin.
}
```

A5:
```
TooManyAttemptsMsg(){
    Display the message for too many attempts of entering pin.
}
```

A6:
```
DisplayMenu(){
    Display the menu.
}
```

A7:
```
MakeDeposit(){
    dop.balance = dop.balance + dop.temp_d;
}
```

A8:
```
DisplayBalance(){
    Display the value of dop.balance;
}
```

A9:

```
BelowMinMsg(){
    Display the message that the current balance is below the minimum required balance.
}
```

A10:

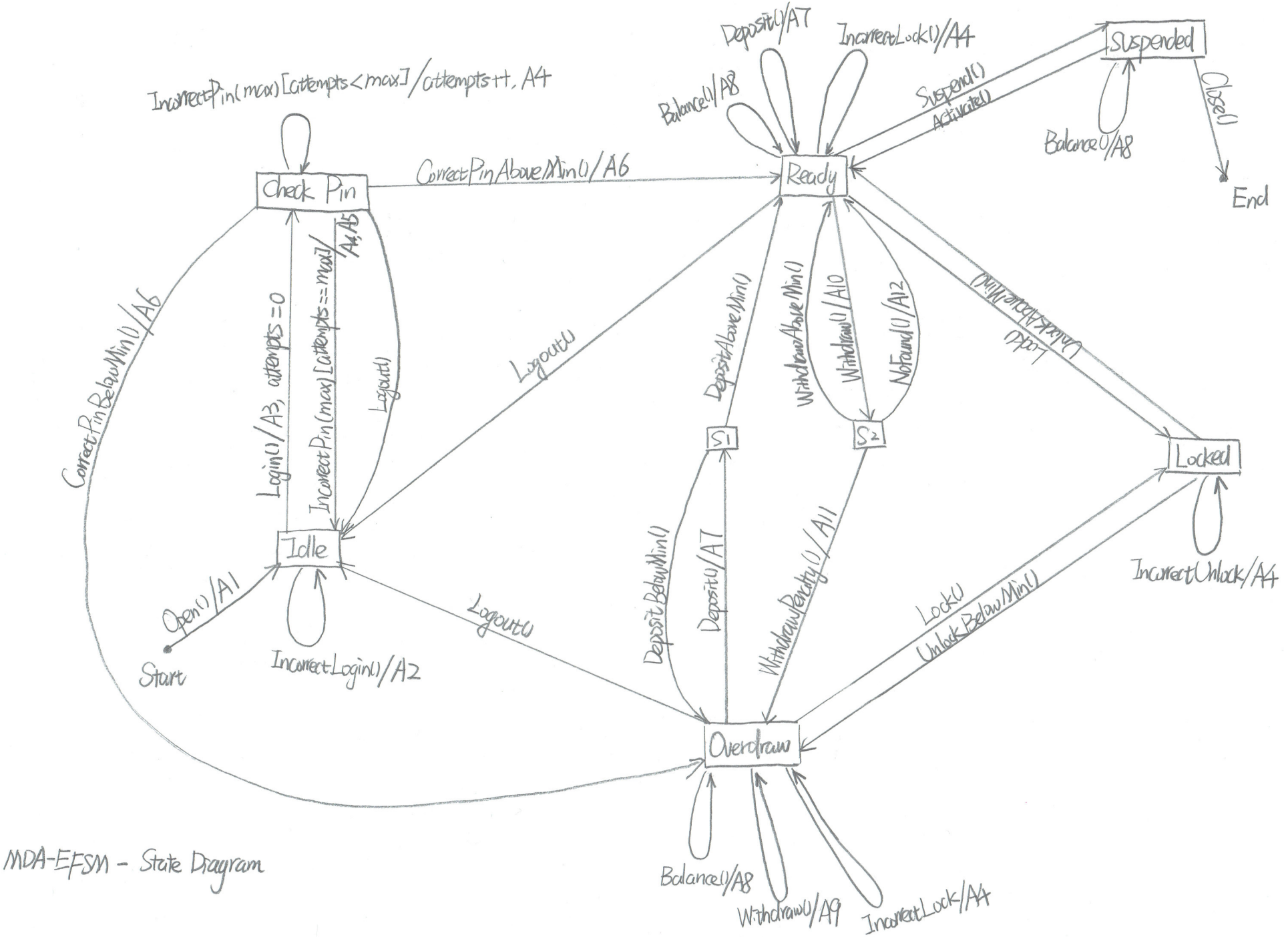
```
MakeWithdraw(){
    dop.balance = dop.balance - dop.temp_w;
}
```

A11:

```
BelowMinPenalty(){
    dop.balance = dop.balance – dop.penaly;
}
```

A12:

```
NoFoundMsg(){
    Display the message that there is no found in the account.
}
```



MDA-EFSM - State Diagram

ACCOUNT-1

```
open(string p, string y, float a){
    d.temp_p = p
    d.temp_y = y
    d.temp_a = a
    m.open()
}

pin(string x){
    if(x == d.pin){
        if(d.balance > 500){
            m.CorrectPinAboveMin()
        }else{
            m.CorrectPinBelowMin()
        }
    }else{
        m.IncorrectPin(3)
    }
}

deposit(float d){
    d.temp_d = d
    m.Deposit()
    if(d.balance > 500){
        m.DepositAboveMin()
    }else{
        m.DepositBelowMin()
    }
}

balance(){
    m.Balance()
}

login(string y){
    if(y == d.userID){
        m.Login()
    }else{
        m.IncorrectLogin()
    }
}
```

```
logout(){
    m.Logout()
}

withdraw(float w){
    d.temp_w = w
    d.penalty = 20
    m.Withdraw()
    if(d.balance > 500){
        m.WithdrawAboveMin()
    }else{
        m.WithdrawPenalty()
    }
}

lock(string x){
    if(x == d.pin){
        m.Lock()
    }else{
        m.IncorrectLock()
    }
}

unlock(string x){
    if(x == d.pin){
        if(d.balance > 500){
            m. UnlockAboveMin()
        }else{
            m. UnlockBelowMin()
        }
    }else{
        m.IncorrectUnlock()
    }
}
```

ACCOUNT-2

```
OPEN(int p, int y, int a){
    d.temp_p = p
    d.temp_y = y
    d.temp_a = a
    m.open()
}

PIN(int x){
    if(x == d.pin){
        m.CorrectPinAboveMin()
    }else{
        m.IncorrectPin(2)
    }
}

DEPOSIT(int d){
    d.temp_d = d
    m.Deposit()
}

WITHDRAW(int w){
    d.temp_w = w
    m.Withdraw()
    if(d.balance > 0){
        m.WithdrawAboveMin()
    }else{
        m.NoFound()
    }
}

BALANCE(){
    m.Balance()
}

LOGIN(int y){
    if(y == d.userID){
        m.Login()
    }else{
        m.IncorrectLogin()
    }
}
```

```
LOGOUT(){
    m.Logout()
}

suspend(){
    m.Suspend()
}

activate(){
    m.Activate()
}

close(){
    m.Close()
}
```