2. For the following questions (a)–(c), consider the method FSM for a (simplified) programmable thermostat. Suppose the variables that define the state and the methods that transition between states are:

```
partOfDay : {Wake, Sleep}
temp      : {Low, High}

// Initially "Wake" at "Low" temperature

// Effects: Advance to next part of day
public void advance();

// Effects: Make current temp higher, if possible
public void up();

// Effects: Make current temp lower, if possible
public void down();
```

   (a) How many states are there?
   (b) Draw and label the states (with variable values) and transitions (with method names). Notice that all of the methods are total.
   (c) A test case is simply a sequence of method calls. Provide a test set that satisfies edge coverage on your graph.

2.6 GRAPH COVERAGE FOR USE CASES

UML use cases are widely used to clarify and express software requirements. They are meant to describe sequences of actions that software performs as a result of inputs from the users; that is, they help express the **workflow** of a computer application. Because use cases are developed early in software development, they can be help the tester start testing activities early.

Many books and papers can help the reader develop use cases. As with FSMs, it is not the purpose of this book to explain how to develop use cases, but how to use them to create useful tests. The technique for using graph coverage criteria to develop tests from use cases is expressed with a simple example.

Figure 2.41 shows three simple use cases for an automated teller machine (ATM). In use cases, *actors* are humans or other software systems that use the software being modeled. They are drawn as simple stick figures. In Figure 2.41, the actor is an ATM customer who has three potential use cases; Withdraw Funds, Get Balance, and Transfer Funds.

While Figure 2.41 is technically a graph, it is not a very useful graph for testing. About the best we could do as a tester is to use node coverage, which amounts to "try each use case once." However, use cases are usually elaborated, or "documented" with a more detailed textual description. The description describes the details of operation and includes *alternatives*, which model choices or conditions during execution. The Withdraw Funds use case from Figure 2.41 can be described as follows:

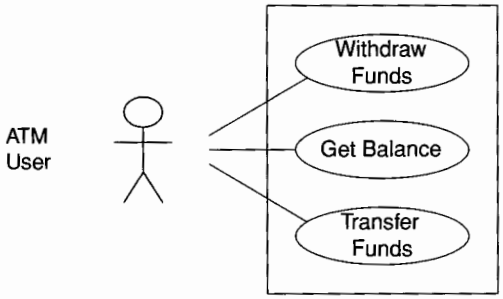

Figure 2.41. ATM actor and use cases.

Use Case Name: Withdraw Funds
Summary: Customer uses a valid card to withdraw funds from a valid bank account.
Actor: ATM Customer
Precondition: ATM is displaying the idle welcome message
Description:

1. Customer inserts an ATM Card into the ATM Card Reader.
2. If the system can recognize the card, it reads the card number.
3. System prompts the customer for a PIN.
4. Customer enters PIN.
5. System checks the expiration date and whether the card has been stolen or lost.
6. If card is valid, the system checks whether the PIN entered matches the card PIN.
7. If the PINs match, the system finds out what accounts the card can access.
8. System displays customer accounts and prompts the customer to choose a type of transaction. Three types of transactions are Withdraw Funds, Get Balance, and Transfer Funds. (The previous eight steps are part of all three use cases; the following steps are unique to the Withdraw Funds use case.)
9. Customer selects Withdraw Funds, selects account number, and enters the amount.
10. System checks that the account is valid, makes sure that the customer has enough funds in the account, makes sure that the daily limit has not been exceeded, and checks that the ATM has enough funds.
11. If all four checks are successful, the system dispenses the cash.
12. System prints a receipt with a transaction number, the transaction type, the amount withdrawn, and the new account balance.
13. System ejects card.
14. System displays the idle welcome message.

Alternatives:

- If the system cannot recognize the card, it is ejected and a welcome message is displayed.
- If the current date is past the card's expiration date, the card is confiscated and a welcome message is displayed.

- If the card has been reported lost or stolen, it is confiscated and a welcome message is displayed.
- If the customer entered PIN does not match the PIN for the card, the system prompts for a new PIN.
- If the customer enters an incorrect PIN three times, the card is confiscated and a welcome message is displayed.
- If the account number entered by the user is invalid, the system displays an error message, ejects the card, and a welcome message is displayed.
- If the request for withdrawal exceeds the maximum allowable daily withdrawal amount, the system displays an apology message, ejects the card, and a welcome message is displayed.
- If the request for withdrawal exceeds the amount of funds in the ATM, the system displays an apology message, ejects the card, and a welcome message is displayed.
- If the customer enters Cancel, the system cancels the transaction, ejects the card, and a welcome message is displayed.
- If the request for withdrawal exceeds the amount of funds in the account, the system displays an apology message, cancels the transaction, ejects the card, and a welcome message is displayed.

Postcondition: Funds have been withdrawn from the customer's account.

At this point, some testing students will be wondering why this discussion is included in a chapter on graph coverage. That is, there is little obvious relationship with graphs thus far. We want to reiterate the first phrase in Beizer's admonition: "testers find a graph, then cover it." In fact, there is a nice graph structure in the use case textual description, which may be up to the tester to express. This graph can be modeled as the transaction flow graphs in Beizer's Chapter 4, or can be drawn as a UML **Activity Diagram**.

An activity diagram shows the flow among activities. Activities can be used to model a variety of things, including state changes, returning values, and computations. We advocate using them to model use cases as graphs by considering activities as *user level steps*. Activity diagrams have two kinds of nodes, action states and sequential branches.[6]

We construct activity graphs as follows. The numeric items in the use case **Description** express steps that the actors undertake. These correspond to inputs to or outputs from the software and appear as **nodes** in the activity diagram as action states. The **Alternatives** in the use case represent decisions that the software or actors make and are represented as **nodes** in the activity diagram as sequential branches.

The activity diagram for the withdraw funds scenario is shown in Figure 2.42. Several things are **expected** but not **required** of activity diagrams constructed from use cases. First, they usually do not have many loops, and most loops they do contain are tightly bounded or determinate. For example, the graph in Figure 2.42 contains a three-iteration loop when the PIN is entered incorrectly. This means that complete path coverage is often feasible and sometimes reasonable. Second, it is very rare to see a complicated predicate that contains multiple clauses. This is because the use case is usually expressed in terms that the users can understand. This means that the logic coverage criteria in Chapter 3 are usually not useful. Third, there are no

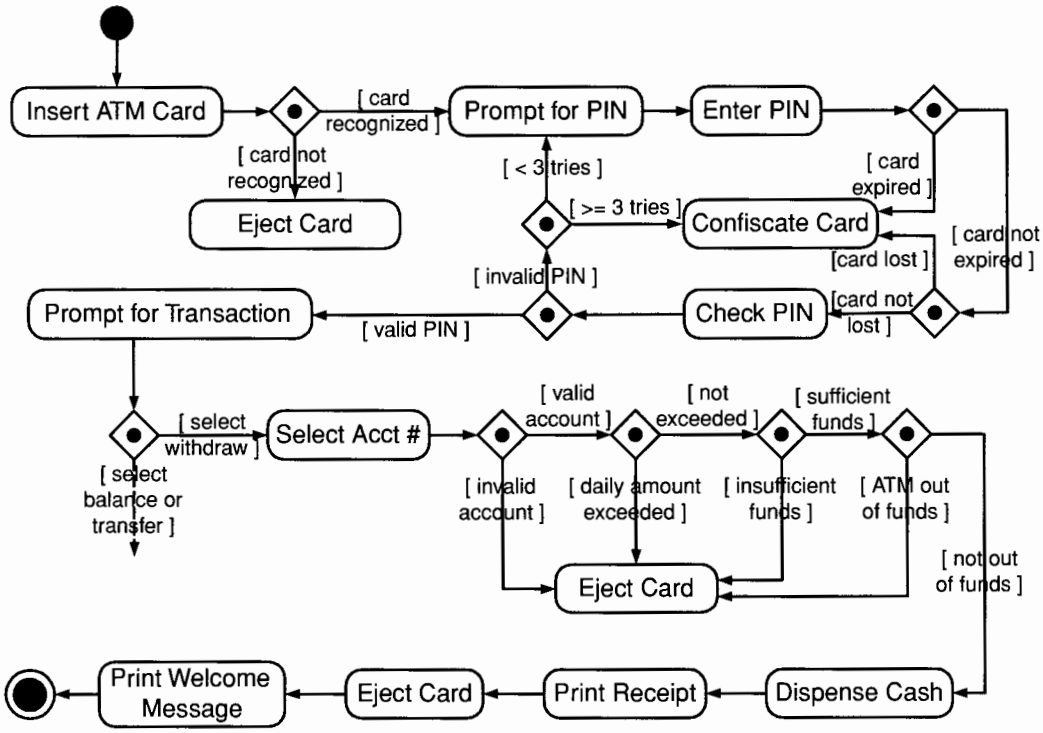

Figure 2.42. Activity graph for ATM withdraw funds.

obvious data definition-use pairs. This means that data flow coverage criteria are not applicable.

The two criteria that are most obviously applicable to use case graphs are node coverage and edge coverage. Test case values are derived from interpreting the nodes and predicates as inputs to the software. One other criterion for use case graphs is based on the notion of "scenarios."

2.6.1 Use Case Scenarios

A use case scenario is an *instance* of, or a complete path through, a use case. A scenario should make some sense semantically to the users and is often derived when the use cases are constructed. If the use case graph is finite (as is usually the case), then it is possible to list all possible scenarios. However, domain knowledge can be used to reduce the number of scenarios that are useful or interesting from either a modeling or test case perspective. Note that *specified path coverage*, defined at the beginning of this chapter, is exactly what we want here. The set S for specified path coverage is simply the set of all scenarios.

If the tester or requirements writer chooses all possible paths as scenarios, then specified path coverage is equivalent to complete path coverage. The scenarios are chosen by people and they depend on domain knowledge. Thus it is **not** guaranteed that specified path coverage subsumes edge coverage or node coverage. That is, it is possible to choose a set of scenarios that do not include every edge. This would probably be a mistake, however. So, in practical terms, specified path coverage can be expected to cover all edges.

EXERCISES

Section 2.6.

1. Construct a
a bank au
of the AT
and what
Desig

2.7 REPRESEN

While we typica
in various nonpr
can be manipula
expressions. Thes
and to answer va

The first requ
names can come
added specifica
unique lower case
nation; if edge a i
written explicitly
re taken, their se
quence of edges is
and zero or more
that an edge label
path product is a sp
ire sometimes rep

Figure 2.43 sh
the loop touring
re 2.43(a) has en
include loops, so
ented using expo

If an edge, pat
with an exponent
inite number of re
epresented by a^*
nerally, $A\lambda = A$

Representing p
ment manipulatio
igure 2.43(b) abo

$$
\begin{aligned}
A &= \\
B &= \\
C &= \\
AB &= \\
C^3 &=
\end{aligned}
$$