Figure 4.1 Structuring information about the family.

family(
    person( tom, fox, date(7, may, 1960), works(bbc, 15200) ),
    person( ann, fox, date(9, may, 1961), unemployed ),
    [ person( pat, fox, date(5, may, 1983), unemployed ),
      person( jim, fox, date(5, may, 1983), unemployed ) ] ).
Figure 4.2 Specifying objects by their structural properties: (a) any Armstrong family; (b) any family with exactly three children; (c) any family with at least three children. Structure (c) makes provision for retrieving the wife’s name through the instantiation of the variables **Name** and **Surname**.
Example 7.4  To parse the string cab3, you would make the following transitions:

Current state: A  Input: cab3  Scan c and go to B.
Current state: B  Input: ab3  Scan a and go to B.
Current state: B  Input: b3  Scan b and go to B.
Current state: B  Input: 3  Scan 3 and go to B.
Current state: B  Input:  Check for final state.

Because there is no more input and the last state is B, a final state, cab3 is a valid identifier.
Example 7.5  You must make the following decisions to parse +203 with this nondeterministic FSM:

- Current state: A  Input: +203  Scan + and go to B.
- Current state: B  Input: 203  Scan 2 and go to B.
- Current state: B  Input: 03  Scan 0 and go to B.
- Current state: B  Input: 3  Scan 3 and go to C.
- Current state: C  Input:  Check for final state.

Because there is no more input and you are in the final state C, you have proven that the input string +203 is a valid signed integer.
in state I, you can do one of three things: (a) scan + and go to F, (b) scan – and go to F, or (c) scan nothing (that is, the empty string) and go to F.

**Example 7.6** To parse 32 requires the following decisions:

<table>
<thead>
<tr>
<th>Current state: I</th>
<th>Input: 32</th>
<th>Scan $\epsilon$ and go to F.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current state: F</td>
<td>Input: 32</td>
<td>Scan 3 and go to M.</td>
</tr>
<tr>
<td>Current state: M</td>
<td>Input: 2</td>
<td>Scan 2 and go to M.</td>
</tr>
<tr>
<td>Current state: M</td>
<td>Input:</td>
<td>Check for final state.</td>
</tr>
</tbody>
</table>

The transition from I to F on $\epsilon$ does not consume an input character. When you are in state I, you can do one of three things: (a) scan + and go to F, (b) scan – and go to F, or (c) scan nothing (that is, the empty string) and go to F.
Simulating a non-deterministic automaton

Imagine that the representation constructor relations, and representation. In particular, the automaton may choose to make or not to make a silent move, if it is available in the current state. But abstract non-deterministic machines of this kind have a magic property: if there is a choice then they always choose a right move; that is, a move that leads to the acceptance of the input string, if such a move exists. The automaton in Figure 4.3 will, for example, accept the strings ab and aøbaab, but it will reject the strings abb and øbbø. It is easy to see that this automaton accepts any string that terminates with ab, and rejects all others.

Figure 4.3 An example of a non-deterministic finite automaton.
Figure 4.3 An example of a non-deterministic finite automaton.

```
final( s3).
trans( s1, a, s1).
trans( s1, a, s2).
trans( s1, b, s1).
trans( s2, b, s3).
trans( s3, b, s4).
silent( s2, s4).
silent( s3, s1).
```
Simulating a non-deterministic automaton

The rest of the string will be represented when processing a given string. By definition, the automaton begins in its initial state, which defines the start state, accepts the string. They correspond to a final state.

The first symbol in the rest of the string makes a silent move: string from state 1 to state 2.

Figure 4.4 Accepting a string: (a) by reading its first symbol X; (b) by making a silent move.