decimal, corresponding to 110 in binary. Because of periodic boundary conditions, the index to the left of $i = 0$ is $L - 1$. The expression `(automaton.getValue(L-1,t-1)<<1)` yields `001 << 1 = 010` because `<<` shifts all bits to the left. (Only 3 bits are needed to describe the neighborhood.) The statement

```
int neighborhood = (automaton.getValue(L-1,t-1)<<1) +
     automaton.getValue(0,t-1);
```

yields `010 + 001 = 011`. The effect of the statement

```
neighborhood = neighborhood & 3;
```

is to clear the third bit of the neighborhood but to keep the second and first bits: 011 & 011 = 011. In this case nothing is changed. We then shift the second and first bits of the neighborhood to the third and second bits

```
neighborhood = neighborhood << 1;
```

and obtain `neighborhood = 110`. Finally the statement

```
neighborhood += automaton.getValue((i+1)%L, t-1);
```

gives `neighborhood = 011 + 000 = 011`, which is 2 in decimal.

***Problem 14.2 Whose time is more important?**

(a) Work out another example to make sure that you understand the nature of the bit manipulations that are used in Listing 14.2 and in the more efficient version of method `iterate`.

(b) Which version of method `iterate` would you use, the more efficient but more difficult to understand (and debug) version or the less efficient but easier to understand version? What is more important, computer time or programmer time? In general, the answer depends on the context. ■

The dynamical behavior of many of the 256 one-dimensional Boolean cellular automata is uninteresting, and hence we also consider one-dimensional Boolean cellular automata with larger neighborhoods (including the site itself). Because a larger neighborhood implies that there are many more possible update rules, we place some reasonable restrictions on the rules. First, we assume that the rules are symmetrical; for example, the neighborhood 100 produces the same value for the central site as 001. We also require that the zero neighborhood 000 yields 0 for the central site, and that the value of the central site depends only on the sum of the values of the sites in the neighborhood; for example, 011 produces the same value for the central site as 101 (see Wolfram, 1984).

A simple way of coding the rules that is consistent with these requirements is as follows. Each rule is labeled by a sequence of 0s and 1s such that the sequence indicates which sums set the central site equal to 1. If the lowest order digit is 1, then the central site is set to 1 if the sum is 0. If the next digit is 1, then the central site is set to 1 if the sum is 1, etc. For example, the rule 10110 indicates that the central site will be set to 1 if the number of neighbors equal to 1 is 1, 2, or 4.

Problem 14.3 More one-dimensional cellular automata

(a) Modify class `OneDimensionalAutomatonApp` so that it incorporates the possible rules discussed in the text based on the number of sites equal to 1 in a neighborhood of $2z + 1$ sites. How many possible rules are there for $z = 1$? Choose $z = 1$ and a random initial configuration and determine if the long time behavior for each rule belongs to one of the following categories:

(i) A homogeneous state where every site has the same value. An example is rule 1000.

(ii) A pattern consisting of separate stable or periodic regions. An example is rule 0100.

(iii) A chaotic, aperiodic pattern. An example is rule 1010.

(iv) A set of complex, localized structures that may not live forever. There are no examples for $z = 1$.

(b) Modify your program so that $z = 2$. Wolfram (1984) claims that rules 010100 and 110100 are the only examples of complex behavior (category 4). Describe how the behavior of these two rules differs from the behavior of the other rules. Find at least one rule for each of the four categories. ■

The results of Problem 14.3 suggests that an important feature of cellular automata is their capability for self-organization. In particular, the class of complex localized structures is distinct from regular as well as aperiodic structures.

An important idea of complexity theory is that simple rules can lead to complex behavior. This complex behavior is not random but has structure. Are there "coarse grained" descriptions that can predict the dynamical behavior of these systems, or do we have to implement the model on a computer using the dynamical rules at the lowest level of description? For example, our understanding of the flow of a fluid through a pipe would be very limited if the only way we could obtain information about the behavior of fluids was to solve the equations of motion for all the individual particles. In this case there is a coarse grained description of fluids where the fundamental fluid variables are not the individual positions and velocities of the molecules, but rather a velocity field which can be interpreted as a spatial average over the velocities of many particles. The resultant partial differential equation of fluid mechanics, known as the Navier–Stokes equation, provides the coarse grained description, which can be solved, in principle, to predict the motion of the fluid.

Is there an analogous coarse grained description of a cellular automaton? Israeli and Goldenfeld have found some examples for which a coarse grained description exists. We first simulate a cellular automaton that produces complex structures. Then we start with the same initial state and create a coarse grained lattice such that each of its cells is a coarse grained description of a group of cells on the original lattice. The idea is to determine a different update rule to evolve the coarse grained lattice such that the configurations of the coarse grained lattice are identical to the coarse grained configurations of the original lattice that were obtained using the original update rule. If it is possible to implement this procedure in general, we would be better able to develop theories of complex macroscopic systems without needing to know the details of the dynamics of the microscopic constituents that make up these systems. We explore two examples in Problem 14.4.

***Problem 14.4 Coarse graining one-dimensional cellular automata**

(a) Add methods to `OneDimensionalAutomatonApp` that create a coarse grained lattice such that groups of three cells are coarse grained to 1 if all three cells are 1 and coarse grained to 0 otherwise. Allow the coarse grained lattice to evolve separately using