Finally, there is the problem of spurious actions such as $\text{Fly}(P_1, JFK, JFK)$, which should be a no-op, but which has contradictory effects (according to the definition, the effect would include $\text{At}(P_1, JFK) \land \neg \text{At}(P_1, JFK)$). It is common to ignore such problems, because they seldom cause incorrect plans to be produced. The correct approach is to add inequality preconditions saying that the from and to airports must be different; see another example of this in Figure 10.3.

10.1.2 Example: The spare tire problem

Consider the problem of changing a flat tire (Figure 10.2). The goal is to have a good spare tire properly mounted onto the car's axle, where the initial state has a flat tire on the axle and a good spare tire in the trunk. To keep it simple, our version of the problem is an abstract one, with no sticky lug nuts or other complications. There are just four actions: removing the spare from the trunk, removing the flat tire from the axle, putting the spare on the axle, and leaving the car unattended overnight. We assume that the car is parked in a particularly bad neighborhood, so that the effect of leaving it overnight is that the tires disappear. A solution to the problem is $[\text{Remove} (\text{Flat}, \text{Axle}), \text{Remove} (\text{Spare}, \text{Trunk}), \text{PutOn} (\text{Spare}, \text{Axle})]$.

```
Init(\text{Tire}(\text{Flat}) \land \text{Tire}(\text{Spare}) \land \text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Spare}, \text{Trunk}))
Goal(\text{At}(\text{Spare}, \text{Axle}))
Action(\text{Remove}(\text{obj}, \text{loc}))
  \text{PRECOND}: \neg \text{At}(\text{obj}, \text{loc})
  \text{EFFECT}: \neg \text{At}(\text{obj}, \text{loc}) \land \text{At}(\text{obj}, \text{Ground}))
Action(\text{PutOn}(t, \text{Axle}))
  \text{PRECOND}: \text{Tire}(t) \land \text{At}(t, \text{Ground}) \land \neg \text{At}(t, \text{Axle})
  \text{EFFECT}: \text{At}(t, \text{Ground}) \land \neg \text{At}(t, \text{Axle})
Action(\text{LeaveOvernight})
  \text{PRECOND}: \text{At}(\text{Spare}, \text{Ground}) \land \neg \text{At}(\text{Spare}, \text{Axle}) \land \neg \text{At}(\text{Spare}, \text{Trunk})
  \land \neg \text{At}(\text{Flat}, \text{Ground}) \land \neg \text{At}(\text{Flat}, \text{Axle}) \land \text{At}(\text{Flat}, \text{Trunk}))
```

Figure 10.2 The simple spare tire problem.

10.1.3 Example: The blocks world

One of the most famous planning domains is known as the blocks world. This domain consists of a set of cube-shaped blocks sitting on a table. The blocks can be stacked, but only one block can fit directly on top of another. A robot arm can pick up a block and move it to another position, either on the table or on top of another block. The arm can pick up only one block at a time, so it cannot pick up a block that has another one on it. The goal will always be to build one or more stacks of blocks, specified in terms of what blocks are on top.

---

2 The blocks world used in planning research is much simpler than SHRDLU's version, shown on page 20.
Section 10.1. Definition of Classical Planning

**Figure 10.3** A planning problem in the blocks world: building a three-block tower. One solution is the sequence `[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]`.

We use `On(b, x)` to indicate that block `A` is on `x`, where `x` is either another block or the table. The action for moving block `b` from the top of `x` to the top of `y` will be `Move(b, x, y)`. Now, one of the preconditions on moving `b` is that no other block is on it. In first-order logic, this would be \(\neg\forall x On(x, b)\) or, alternatively, \(\forall x \neg On(x, b)\). Basic PDDL does not allow quantifiers, so instead we introduce a predicate `Clear(x)` that is true when nothing is on `x`. (The complete problem description is in Figure 10.3.)

The action `Move` moves a block `b` from `x` to `y` if both `x` and `y` are clear. After the move is made, `b` is still clear but `y` is not. A first attempt at the `Move` schema is

\[
\text{Action(Move(b, x, y),}
\]

\[
\text{Precond: On(b, x) A Clear(b) A Clear(y),}
\]

\[
\text{Effect: On(b, y) A Clear(x) A Clear(y).}
\]

Unfortunately, this does not maintain `Clear` properly when `x` or `y` is the table. When `x` is the `Table`, this action has the effect `Clear(Table)`, but the table should not become `clear`, and when `y = Table`, it has the precondition `Clear(Table)`, but the table does not have to be clear