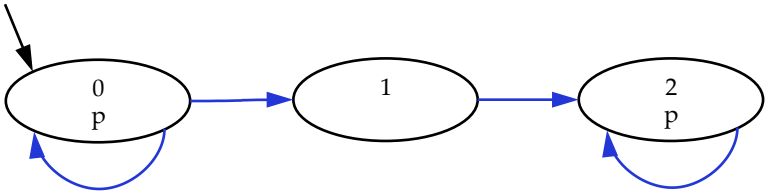

Fig. 4. Additional Automaton A.

— $q_1 = q_0$, the initial state of LA.
— For all i, $q_i \xrightarrow{a} q_{i+1}$

A path π in LA starting at state q is a path where $q_1 = q$.

To not confuse notation, we have paths start at index 1. (Recall that q_0 is the notation for the initial state of the automaton). In the definition, to handle an automaton with a finite path, we utilize the assumption that every state has a transition to allow an infinite sequence. Because every state can take some sequence, any finite sequence can be extended into an infinite sequence by taking that transition. The motivation behind this modeling choice is the intention that desirable models have no dead states.

Let us explore some paths.

Example 3.3 (*Gate*). Consider the gate in Figure 1. It has one path from state up, which is the path $up \longrightarrow lower \longrightarrow down \longrightarrow raise \longrightarrow up \ldots$.

Example 3.4 (*Additional Example*). Consider the automaton in Figure 4. From state 0 there are a variety of paths, but of two kinds. The first path stays in state 0 forever. The second kind stays in state 0 for a finite number of steps (perhaps no transitions) and then goes to state 1, then to state 2, and then stays in state 2 forever. One such path is $\pi = 0 \longrightarrow 0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 2 \ldots$.

Paths are **infinite**. However, when illustrating if a labelled automaton satisfies a formula or not, I will use path segments to illustrate the point. These path segments can be extended to form paths in the labelled automata.

3.3. Semantics

Now with the definition of paths, we can give the formal $CTL_{F,G,X}$ semantics.

Definition 3.5 ($CTL_{F,G,X}$ *semantics*). Given a labelled automaton LA, a $CTL_{F,G,X}$ formula ϕ, and a state $q \in Q$, we say that a state q satisfies ϕ, written as $q \models \phi$ if and only if:

— $q \models p$ iff $p \in L(q)$.
— $q \models \neg\phi$ iff $q \not\models \phi$.
— $q \models \phi_1 \wedge \phi_2$ iff $q \models \phi_1$ and $q \models \phi_2$.
— $q \models \phi_1 \vee \phi_2$ iff $q \models \phi_1$ or $q \models \phi_2$.
— $q \models EX\,(\phi)$ iff there exists a state q' such that $q \longrightarrow q'$ and $q' \models \phi$.
— $q \models EF\,[\phi]$ iff there exists a path π starting at q and there is some index $j \geq 1$ such that $q_j \models \phi$.
— $q \models EG\,[\phi]$ iff there exists a path π starting at q and for all $j \geq 1$, $q_j \models \phi$.
— $q \models AX\,(\phi)$ iff for every state q' such that $q \longrightarrow q'$, then $q' \models \phi$.
— $q \models AF\,[\phi]$ iff for all paths π starting at q, there is some index $j \geq 1$ such that $q_j \models \phi$.
— $q \models AG\,[\phi]$ iff iff for all paths π starting at q, and for all $j \geq 1$, $q_j \models \phi$.

For model checking purposes, we will often consider $AX(S)$ and $EX(S)$, where $S \subseteq Q$ is a set of states. We say that $q \in AX(S)$ if and only if for every q' such that $q \longrightarrow q'$, then $q' \in S$; and we say that $q \in EX(S)$ if and only if there is some q' such that $q \longrightarrow q'$ and $q' \in S$.

4. VERIFYING THE PROPERTIES: CTL MODEL CHECKING ALGORITHM

Now we want to verify that a model satisfies (or does not satisfy) a given CTL property. Do to this, we need an algorithm. The algorithm is recursive. To handle a temporal operator such as $AF[\phi]$, we first label all the states that satisfy ϕ (not labelling a state means that the state does not satisfy ϕ). Then we verify the temporal operator.

Verifying the temporal operators are a bit trickier. The easiest are $EX(\phi)$ and $AX(\phi)$, since we either check one next state, or all next state. However, to verify $AF[\phi]$ and $AG[\phi]$, we need to work more carefully.

The benefit of using CTL to specify properties is that the verification is polynomial in the size of the automaton and the size of the formula (linear in the product of them).

4.1. Representing the Formulas Recursively

The key to verifying these formulas is to get **recursive representations** of the formulas.

Definition 4.1 (*Fixpoint*). A **fixpoint** of a function f is an input x such that $f(x) = x$.

Fixpoints are valuable because they provide termination to recursion. In our case, we will make the formulas recursive and terminate over a fixpoint. The recursive formulas we use are:

$$EF[\phi] \stackrel{\mu}{=} \phi \vee EX(EF[\phi]) \tag{1}$$

$$EG[\phi] \stackrel{\nu}{=} \phi \wedge EX(EG[\phi]) \tag{2}$$

$$AF[\phi] \stackrel{\mu}{=} \phi \vee AX(AF[\phi]) \tag{3}$$

$$AG[\phi] \stackrel{\nu}{=} \phi \wedge AX(AG[\phi]) \tag{4}$$

Here, we label a least fixpoint with μ and a greatest fixpoint with ν. If we read these equations, the first one ($EF[\phi]$) reads as, "Possibly ϕ" means that either ϕ is true now or there is some next state where Possibly ϕ is true. For the time being, think of $\equiv$ as equiv, and the symbols ν, μ as keys for initializations for the algorithm.

4.2. Global Algorithm: Examining All States

Rather than using a local on-the-fly algorithm, one can model check a formula for all states of the automaton. This also utilizes the recursive nature. The key is to write the formula using a **variable** to store the current set of states that satisfy

(1) Take a recursive formula (see previous equations) and use a variable Y to store the set of states that currently satisfy the formula
(2) If the formula has a least fixpoint μ, we initialize $Y = \emptyset$. Else, if the formula has a greatest fixpoint ν, we initialize $Y = Q$.
(3) Iterate the formula, changing Y to be the set of states that currently satisfies the formula.
(4) Repeat iterations until we have a fixpoint (Y does not change value within an iteration).