50.051 Programming Language Concepts

W8-S3 Reminders on Finite State Machines (FSMs)

Matthieu De Mari

Finite State Machine

Definition (Finite State Machine):

A **Finite State Machine (FSM)**, or **finite automaton**, is a mathematical model used to represent systems

- that have a **finite number of possible states**,
- and **can transition between these states based on given inputs**.

An FSM can be represented using a **graph** representation, known as a **state diagram**, which shows the possible states of the system and the transitions between them.

FSMs are used in a wide variety of applications (control systems, communication protocols, digital circuits, etc.). **In our case, FSMs are at the center of the compiling process.**

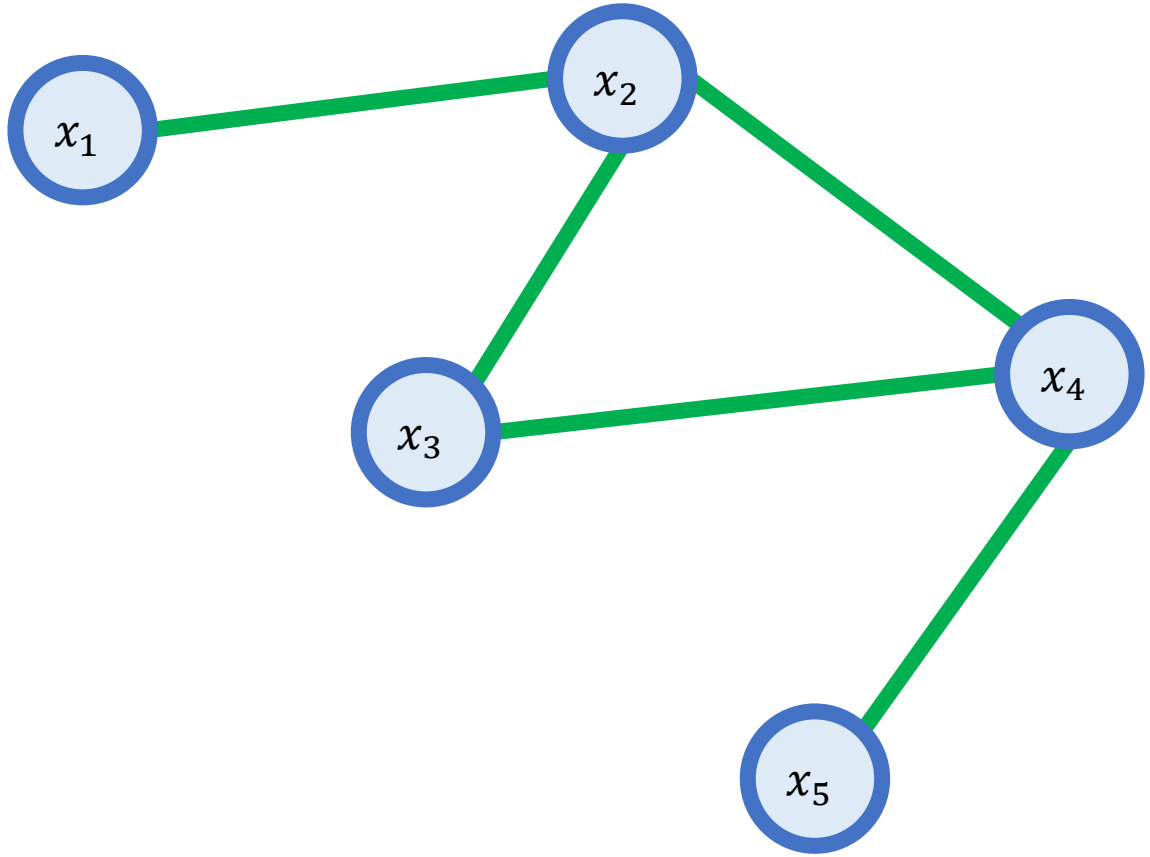
Graph theory

- **Definition (graph theory):**

In mathematics, **graph theory** is the study of graphs objects, which are mathematical structures used to model pairwise relations between objects.

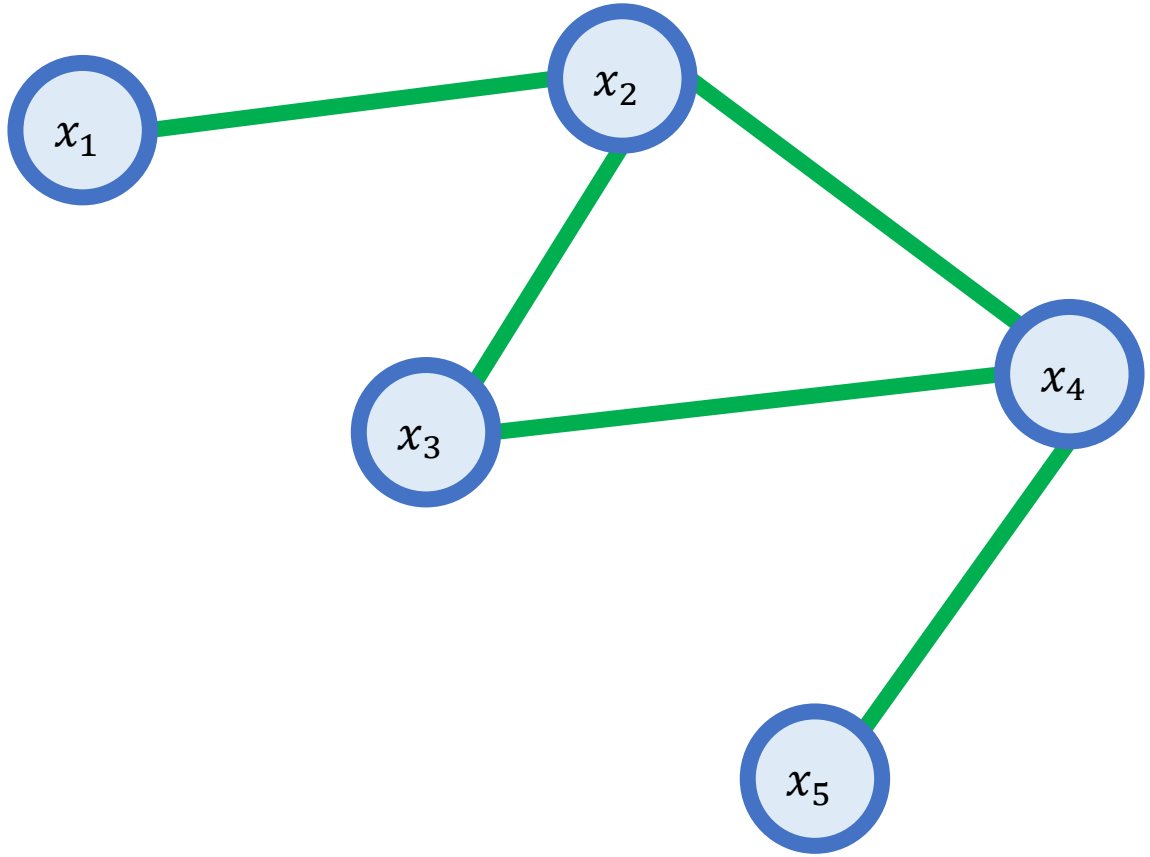
Graphs are one of the principal objects of study in **discrete mathematics**.

(Need a refresher? Check additional materials!)



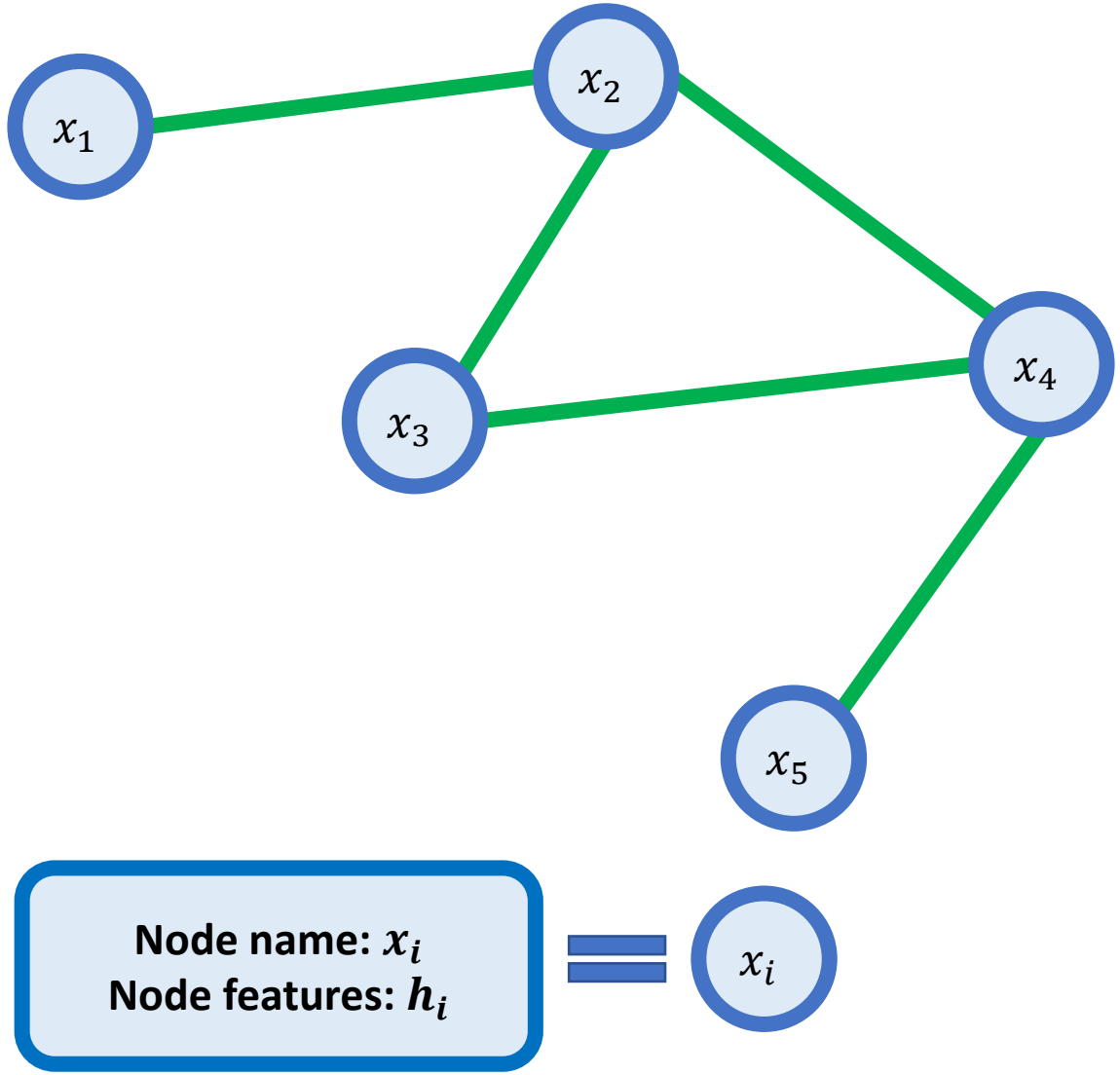
Graphs: a general and minimal definition

- **Definition (graph):** A **graph** is a mathematical object, defined by an ordered pair $\boldsymbol{G = (V, E)}$, with
- $\boldsymbol{V = \{x_1, x_2, \ldots, x_N\}}$ a set of N **vertices** (also called **nodes** or **points**),
- And $\boldsymbol{E}$ a set of **edges** (also called **links** or **lines**), defined as a subset of $\boldsymbol{\{(i, j) \mid \forall i \in [1, N], \forall j \in [1, N]\}}$.



Nodes definition and attributes

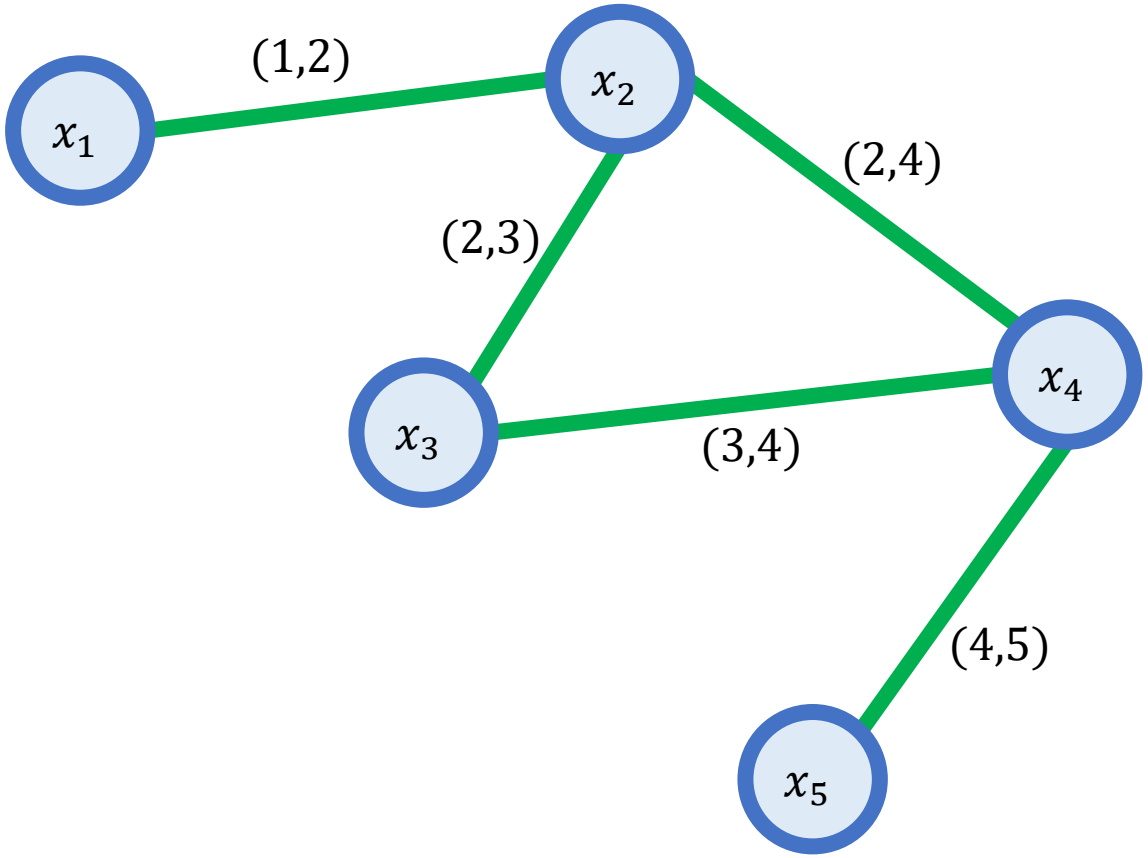
- **Definition (nodes):** A **node x_i** is a point in the graph.
- A **node** has a **name x_i**, which is used for indexing and differs from one node to another.
- A node may also have **attributes**, or **node features**, defined, for instance, as a vector $\boldsymbol{h_i} \in \mathbb{R}^{\boldsymbol{F}}$, with F elements



Edges definition

- **Definition (edges):** An **edge (i, j)** defines a connection from **node x_i** to **node x_j**.
- If **edge $(i, j) \in E$**, then nodes x_i and x_j are connected in the **graph G**.
- In our example, we have

$$E = \{(1,2), (2,3), (2,4), (3,4), (4,5)\}$$



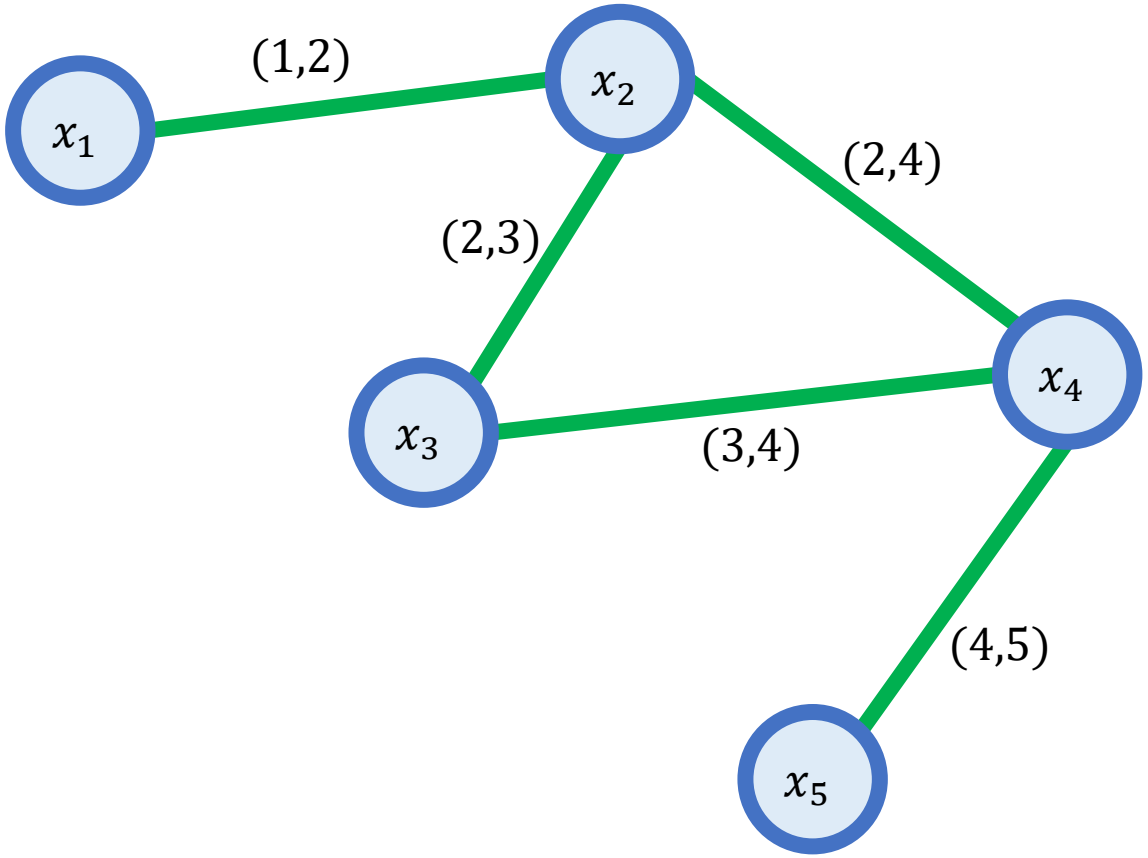
Undirected graph definition

- **Definition (undirected graph):** A graph G is **undirected**, if connections go both ways.
- **Undirected property:** "if node x_i is connected to node x_j, then node x_j is also connected to node x_i".
- Our example is an undirected graph, and the edges set writes as

$$\boldsymbol{E = \{(1,2),(2,1),(2,3),(3,2),(2,4),(4,2),(3,4),(4,3),(4,5),(5,4)\}}$$

Or, to avoid redundancy,

$$\boldsymbol{E = \{(1,2),(2,3),(2,4),(3,4),(4,5)\}}$$

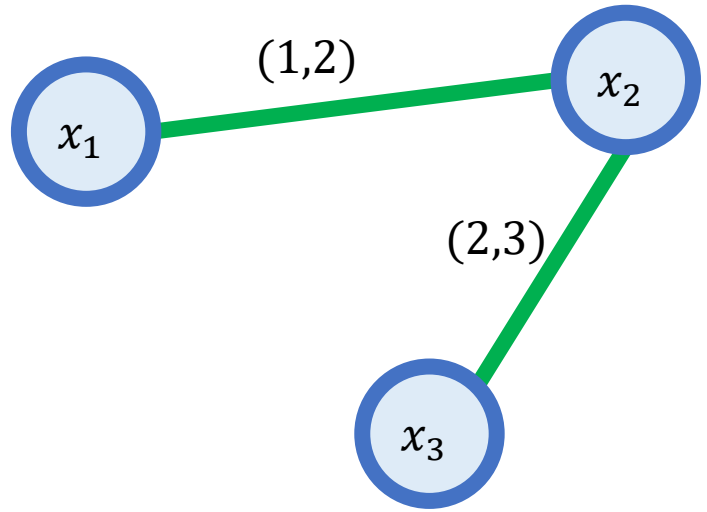


Examples of an undirected graph

- An examples of an undirected graph is… **Facebook!**
- A **node x_i** simply consists of a Facebook user, and its features are user data.
- If two users i and j are friends, then there exist an **edge (i, j)** connecting both users.
- **Undirected property:** "if node x_i is connected to node x_j, then node x_j is also connected to node x_i".

Node name: x_i = User ID

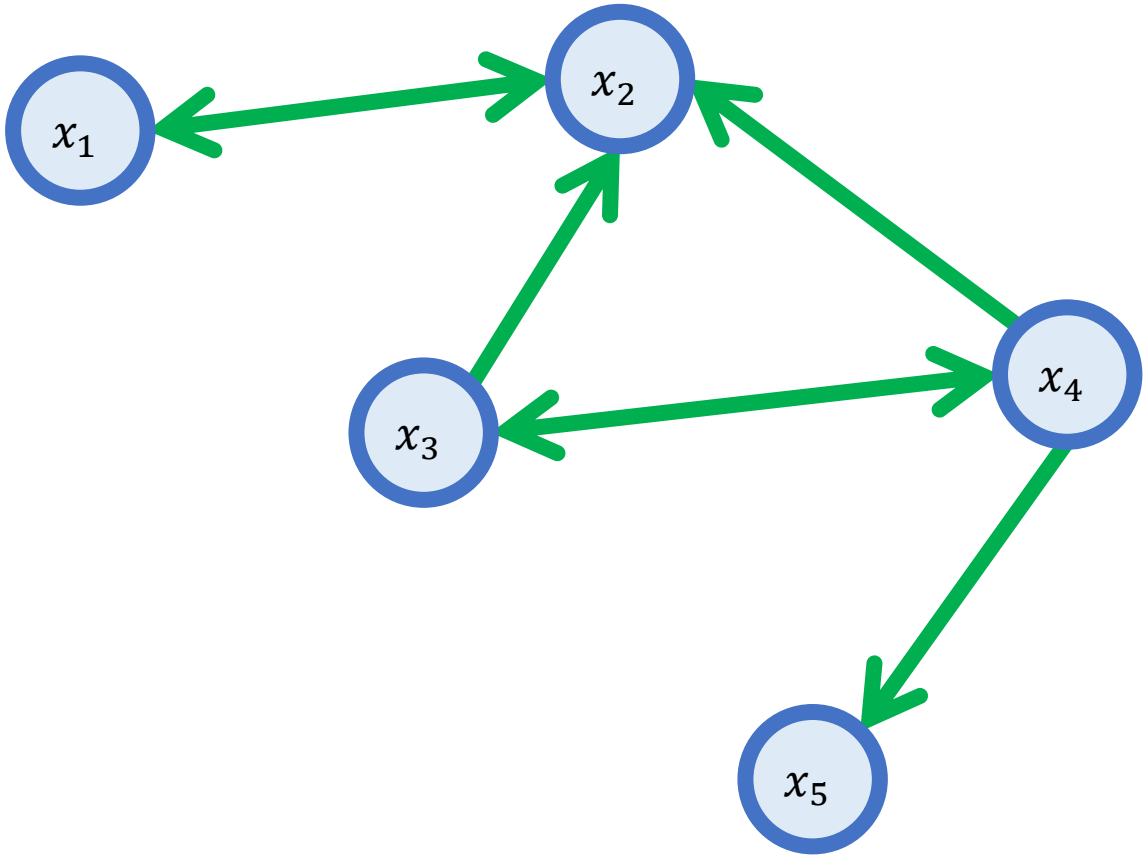
Node features: h_i = (user first name, user family name, date of birth, age, etc.)



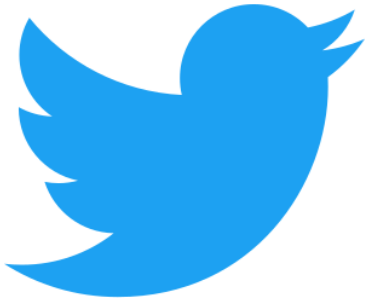
Directed graph definition

- **Definition (directed graph):** A graph G is **directed**, if the undirected property does not hold.
- **Undirected property:** "if node x_i is connected to node x_j, then node x_j is also connected to node x_i".
- In that case, draw arrows for edges.
- Our example is a directed graph, and our edges set writes as

$$\boldsymbol{E = \{(1,2),(2,1),(3,2),(4,2),(3,4),(4,3),(4,5)\}}$$



Examples of a directed graph

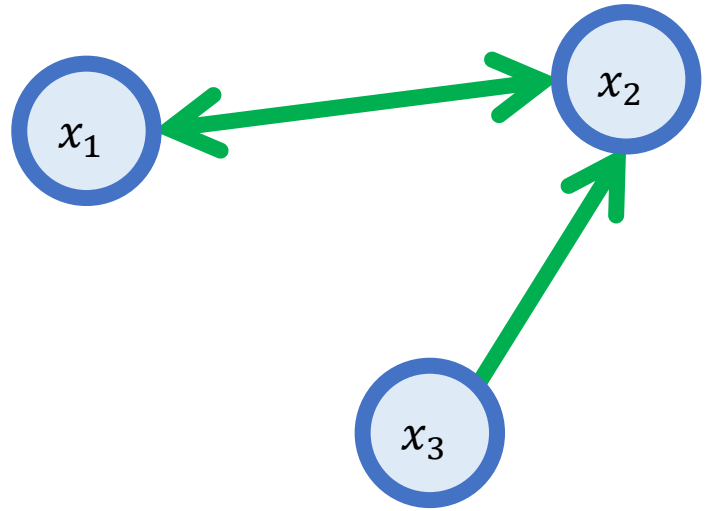


- An examples of a directed graph is… **Twitter/Instagram!**
- As before, a **node x_i** consists of a Twitter user, and its features are user data.

Node name: x_i = User ID

Node features: h_i = (user first name, user family name, date of birth, age, etc.)

- On Twitter, the **undirected property** does not hold: you can follow people, but they do not have to follow you back.



Back to our Finite State Machine

Definition (Finite State Machine):

A **Finite State Machine (FSM)**, or **finite automaton**, is a mathematical model used to represent systems

- that have a **finite number of possible states**,
- and **can transition between these states based on given inputs**.

An FSM can be represented using a **graph** representation, known as a **state diagram**, which shows the possible states of the system and the transitions between them.

FSMs are used in a wide variety of applications (control systems, communication protocols, digital circuits, etc.). **In our case, FSMs are at the center of the compiling process.**

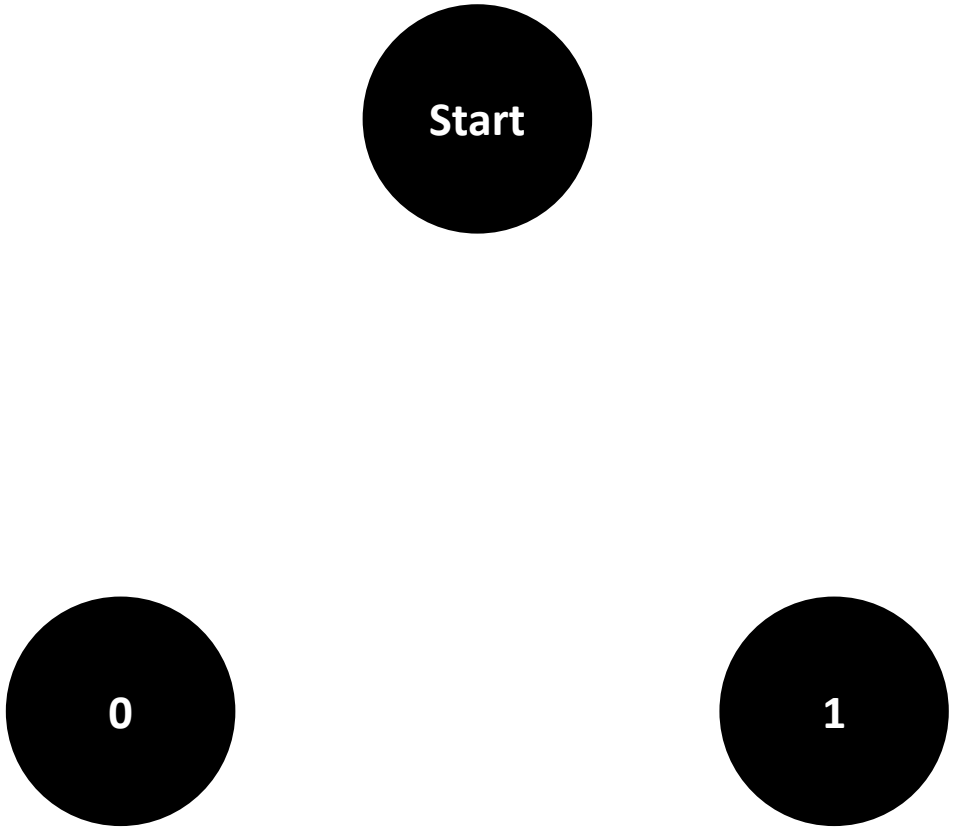
An example of an FSM

Consider the **FSM** and its **state diagram** on the right, represented as a **directed graph**.

- Given a **string**, e.g. *x = "00110"*,
- We will check all elements in the string, one at a time, from left to right, i.e.:

  "0" → "0" → "1" → "1" → "0".

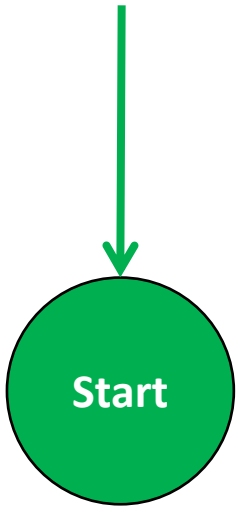
- These five consecutive values will serve as **states** and **inputs**.

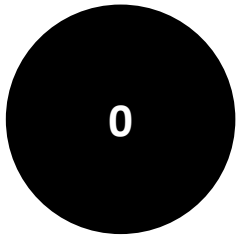


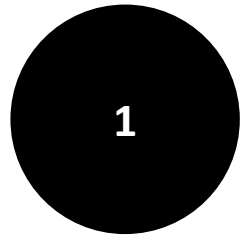
An example of an FSM

Consider the **FSM** and its **state diagram** on the right, represented as a **directed graph**.

- Begin at **start node**, which is defined as a node, receiving an incoming arrow with no source.







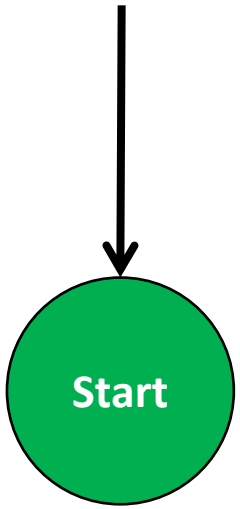
An example of an FSM

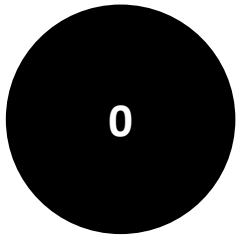
Consider the **FSM** and its **state diagram** on the right, represented as a **directed graph**.

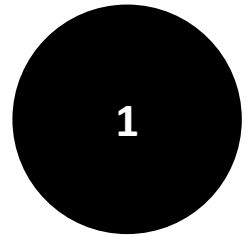
- We will check all elements in the string *x*, one at a time, from left to right, i.e.:

  “0” → “0” → “1” → “1” → “0”.

- We start with “0”, and we are currently on the **start node**.





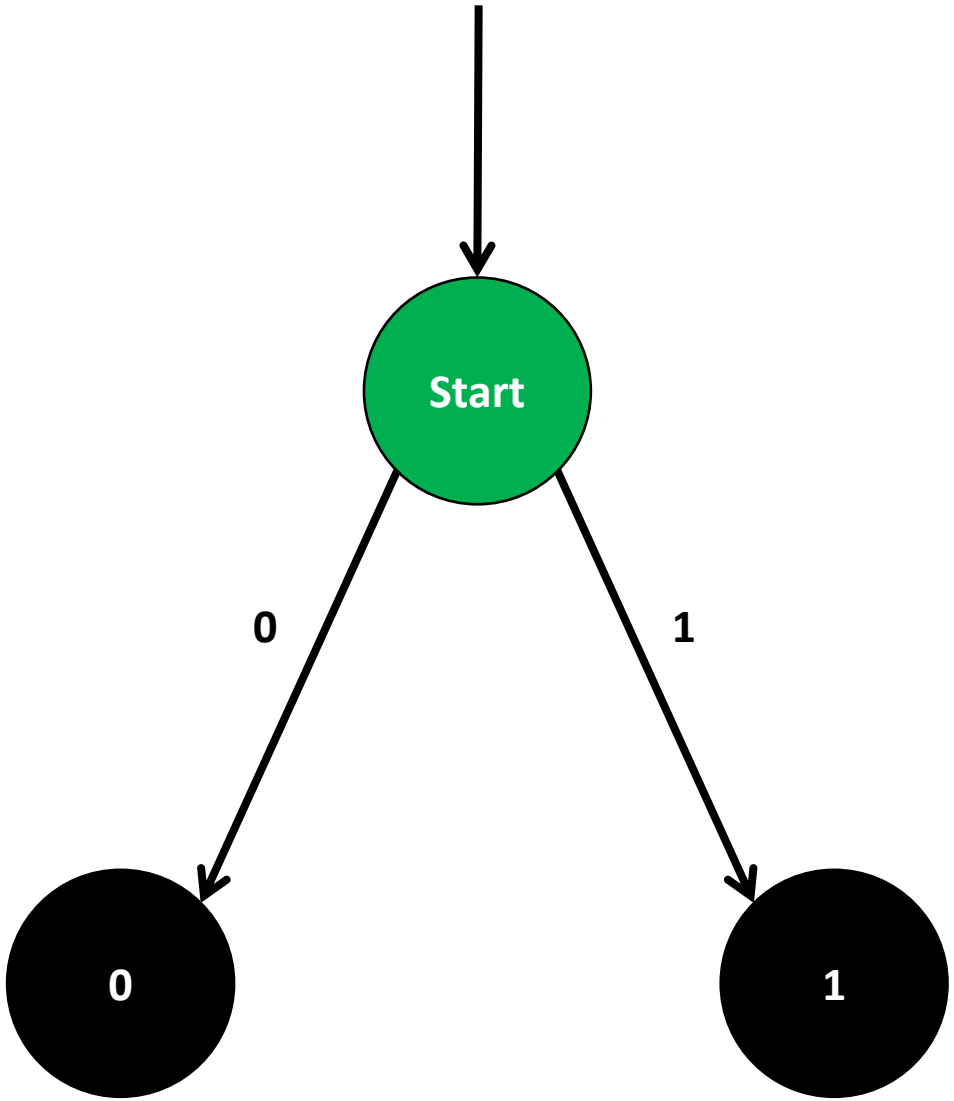


An example of an FSM

An then, move in the FSM state diagram, following a **logic**.

- We first consider that this value "0" serves as **input** (or **action**) for this turn.
- **Logic**, defined by the **directed graph**, **for this FSM (could differ from one FSM to another):**

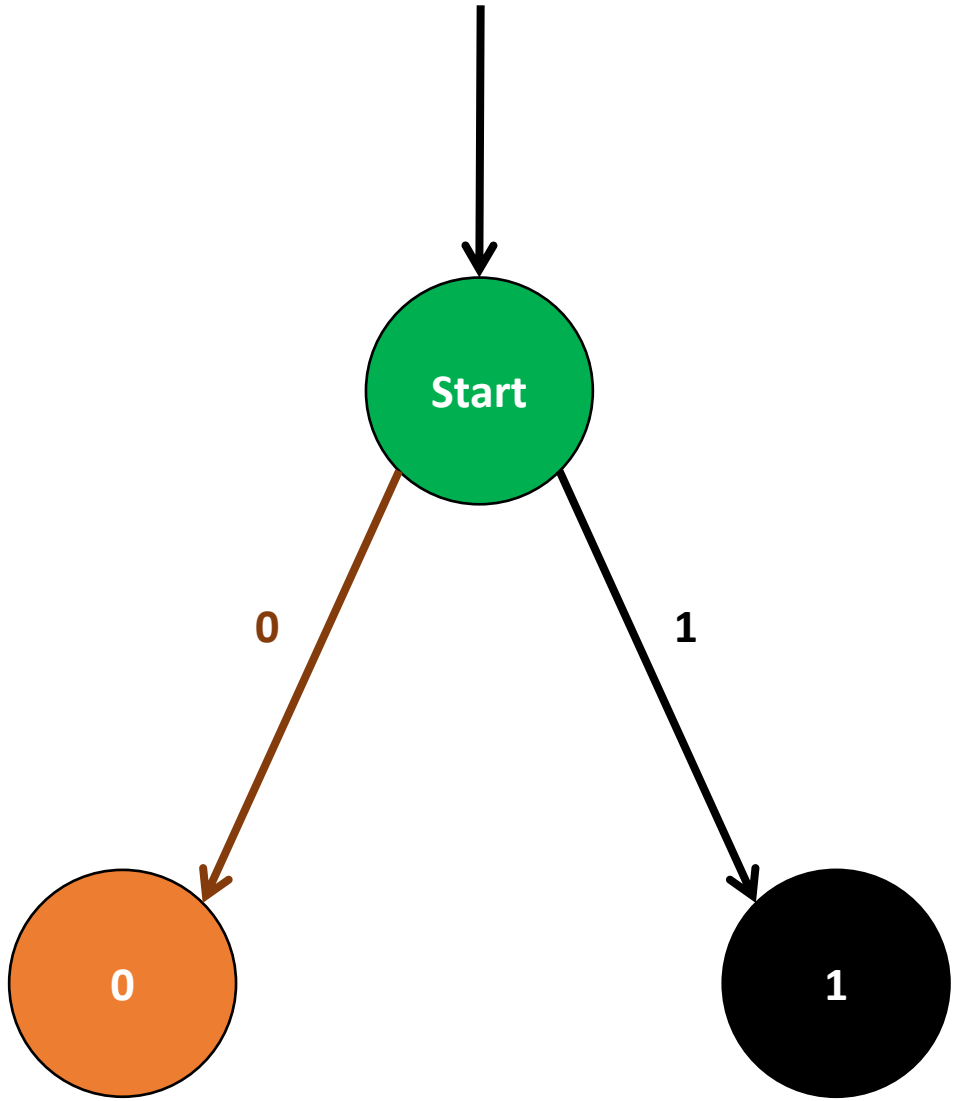
*"the **input** (or **action**) defines the **next** **state** (or **node**) you will move to during this turn."*



An example of an FSM

For instance,

- Following this logic, our first turn sees the **input value** “0”.
- Moves from **our current state**, the **start node**,
- To the **next state**, i.e. **node “0”,**
- Following the **directed edge**, starting from the **start node** and labelled with “0”.

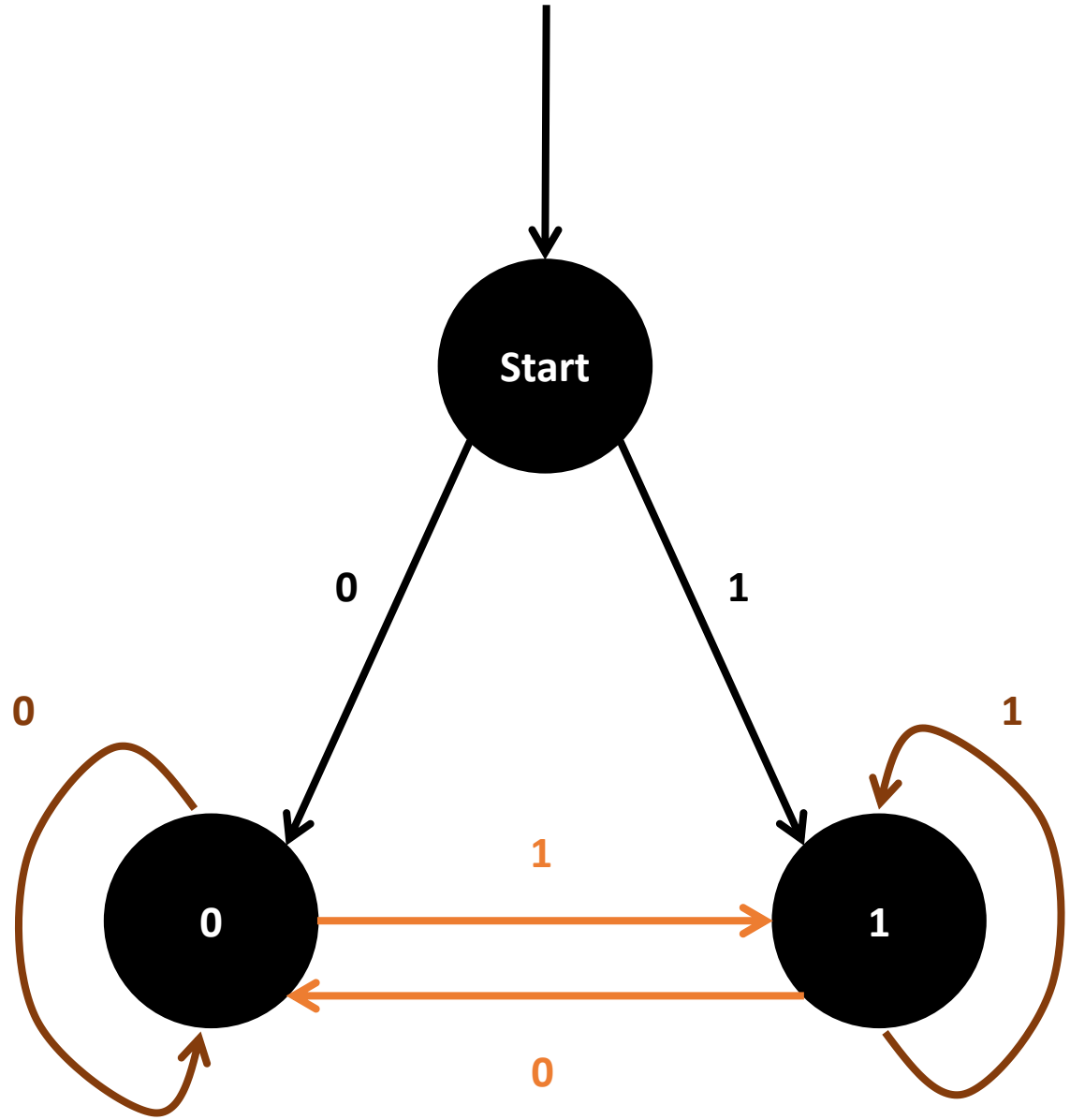


An example of an FSM

And then, we repeat this logic for each element in our string *x*,

"0" → "0" → "1" → "1" → "0".

- To do so, we will need to draw **additional links**.
- In general, we want to have exactly two links departing from each state node, one for each possible action.
- Some of these links **might point to themselves**!



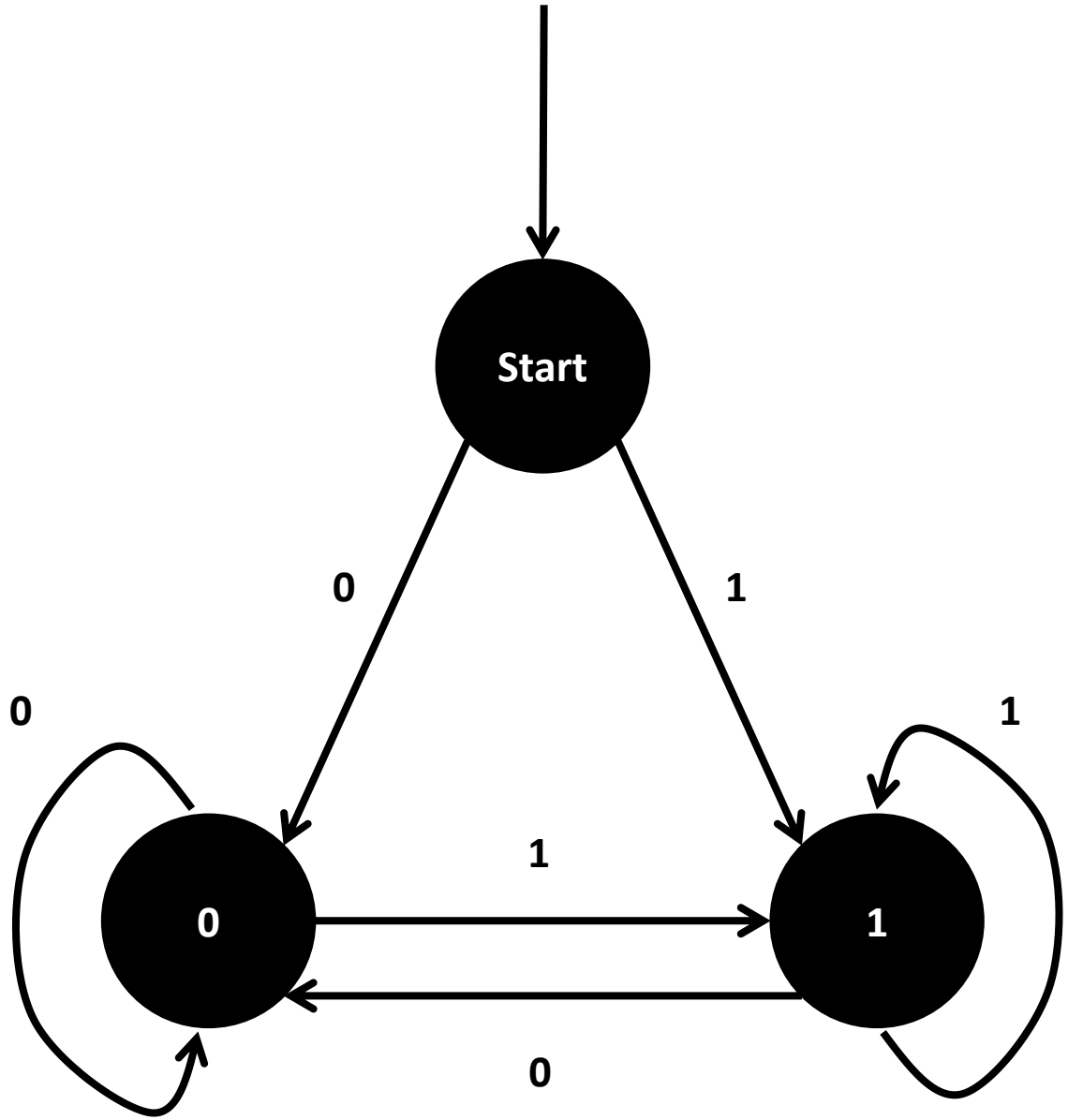
Elements of an FSM

Based on the FSM definition from earlier, we understand that several elements must be defined.

1. A **finite set of states *S***.

In our example, we have

$$S = \{start, 0, 1\}$$



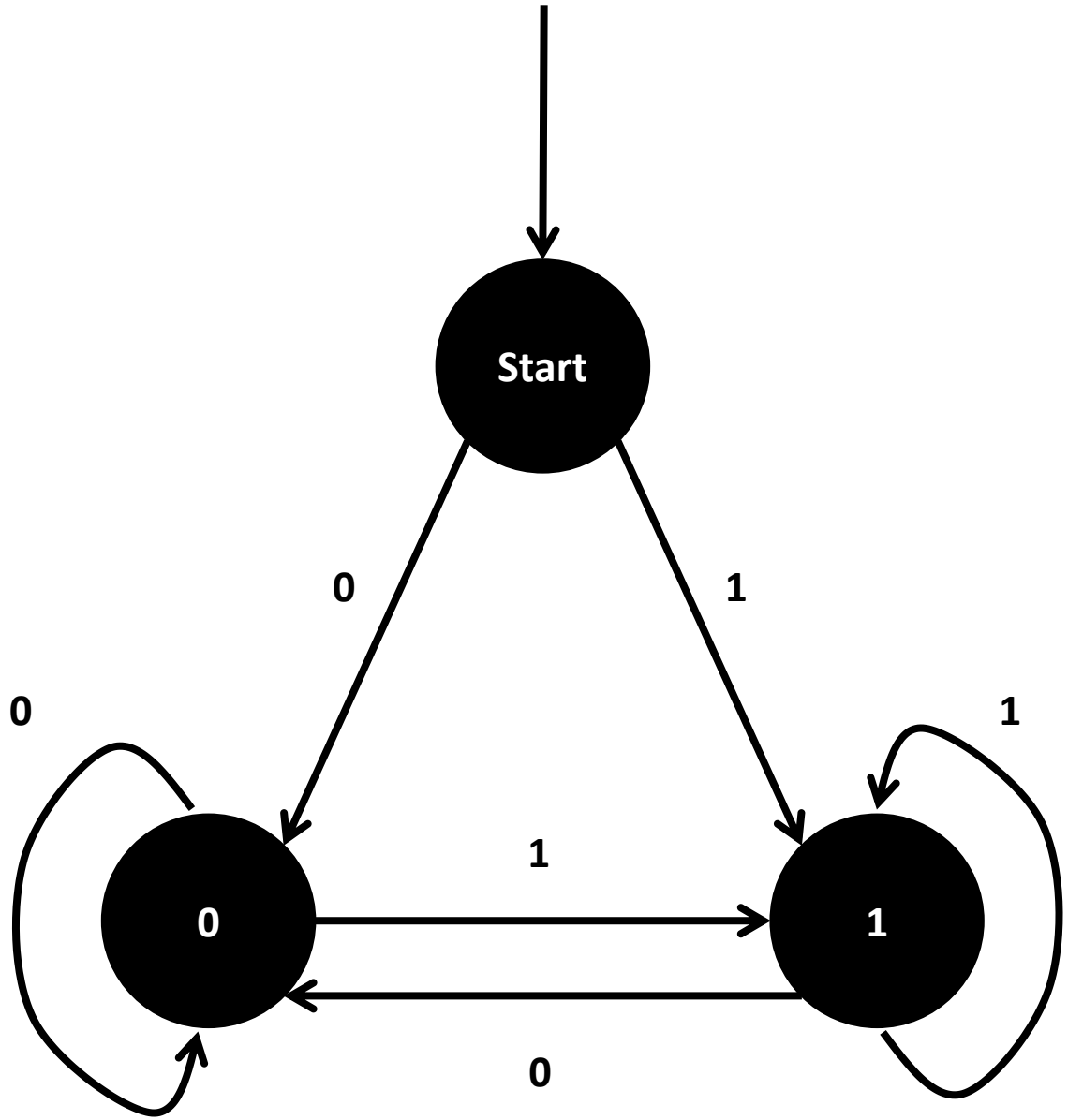
Elements of an FSM

Based on the FSM definition from earlier, we understand that several elements must be defined.

2. A **finite set of inputs or actions *A***. Note that it does not have to be the same as ***S***!

In our example, we have

$$A = \{0, 1\}$$



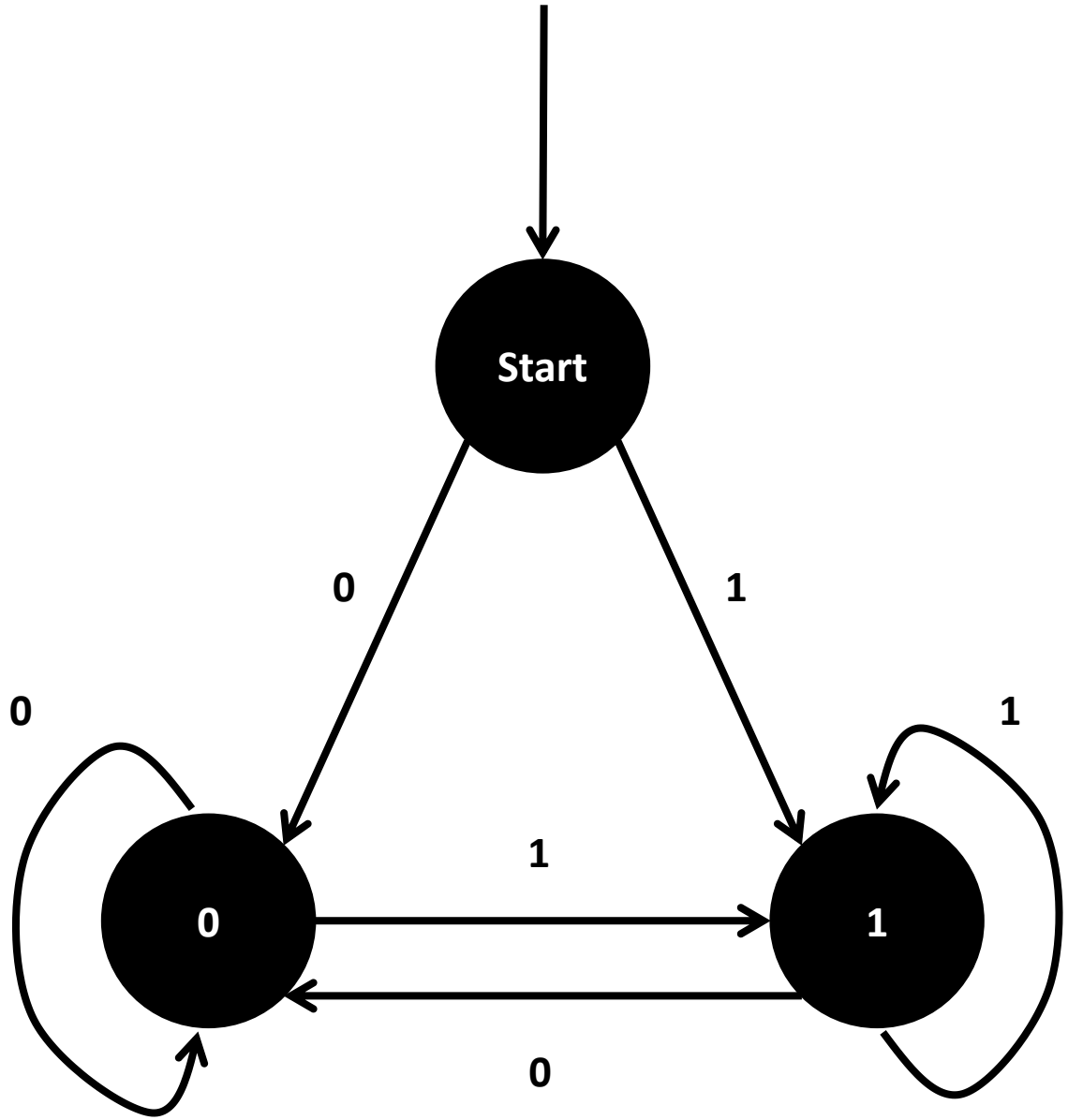
Elements of an FSM

Based on the FSM definition from earlier, we understand that several elements must be defined.

3. A **starting state s_0** ∈ ***S***.

In our example, we have

$$s_0 = "start"$$



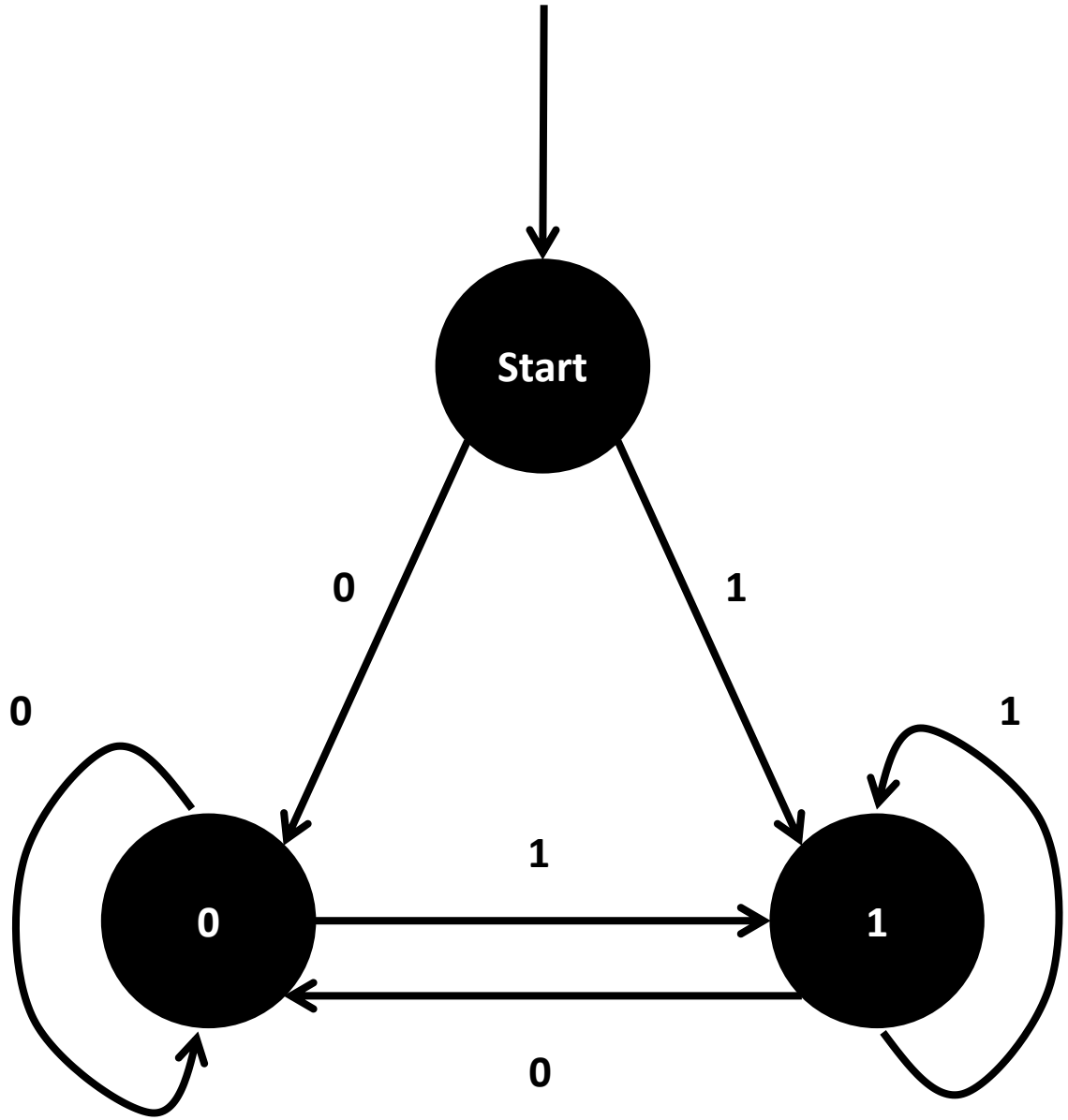
Elements of an FSM

Based on the FSM definition from earlier, we understand that several elements must be defined.

4. A **transition function** f, which describe the **transition logic** in the FSM.

This function takes a **current state** $s \in S$, and a **current action** $a \in A$ as parameters. It returns a **new state** $s' \in S$ as output.

$$f: (s, a) \rightarrow s'$$



Elements of an FSM

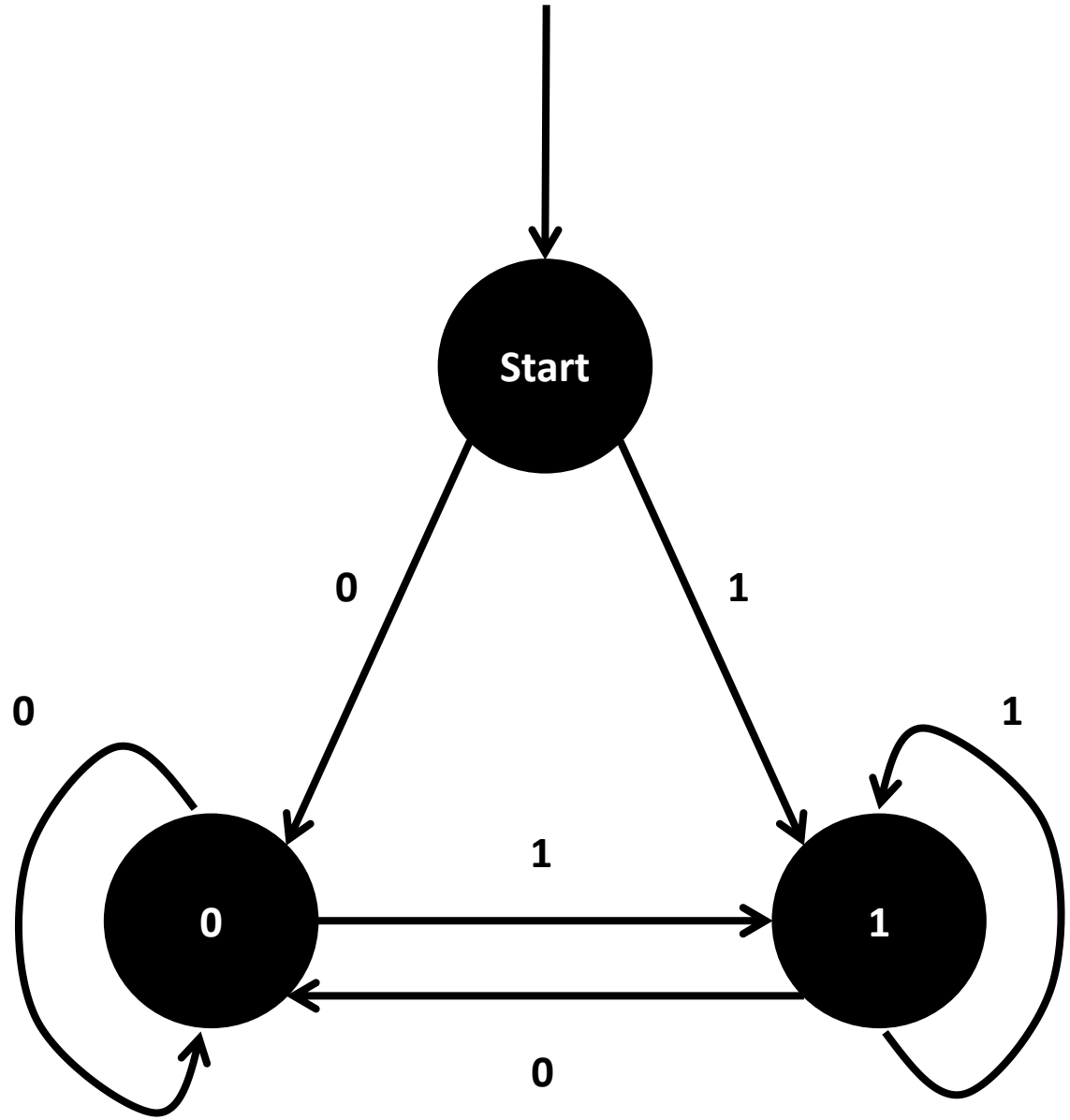
Based on the FSM definition from earlier, we understand that several elements must be defined.

4. A **transition function f**, which describe the **transition logic** in the FSM.

In our case, the function is simply:

$$f: S \times A \rightarrow S$$
$$f(s, a) = a$$

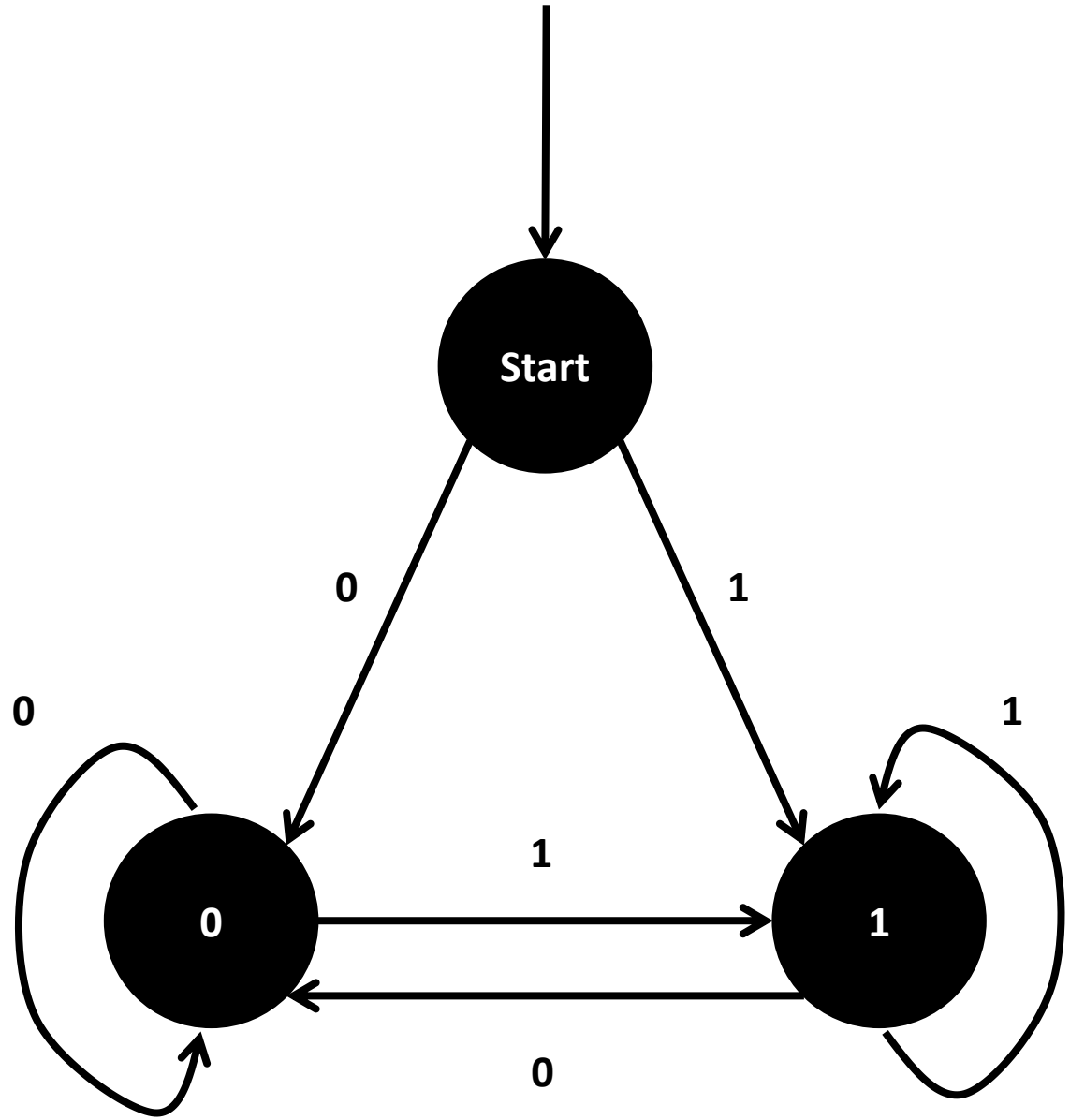
(Other logics can be implemented)



Elements of an FSM

Note: the **transition function** can also be expressed in the form of a **transition table**, as shown below.

Current state	Input	Next state
Start	0	0
Start	1	1
0	0	0
0	1	1
1	0	0
1	1	1



Adding a stopping state

Definition (stopping states or accepting states):

In a finite state machine (FSM), a **stopping state** or **accepting state** is a **special type of state node** in which the FSM can terminate.

Stopping states can be represented in the state transition diagram by drawing a double circle around the state.

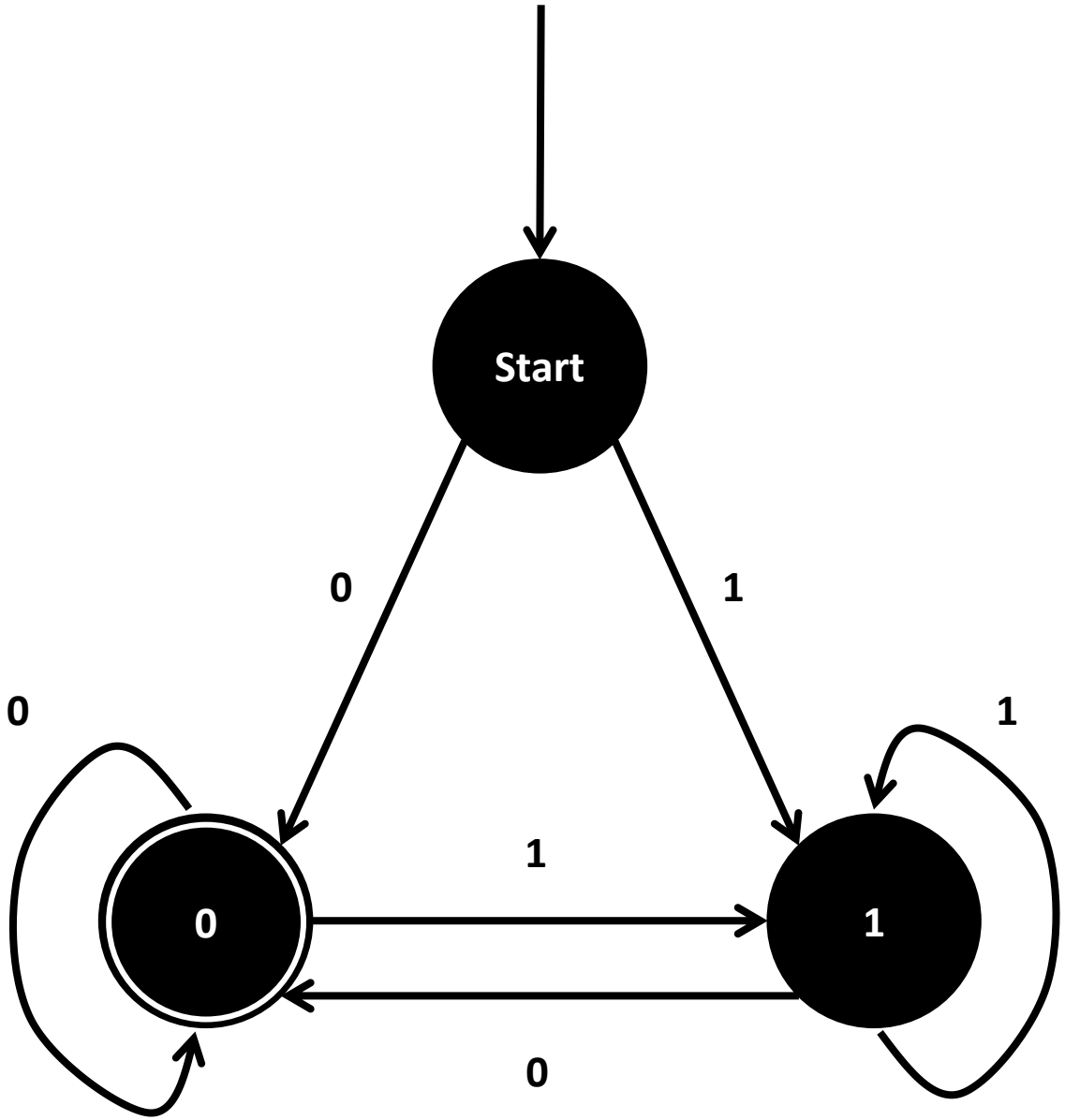
Definition (acceptable input):

In general, we will consider that the inputs string *x* (e.g. our “00110” from earlier) is an **acceptable input**, if and only if the **final state** after processing the string *s* with the FSM happens to be a **stopping state** or **accepting state**.

Adding a stopping state

Stopping states can be represented in the state transition diagram by **drawing a double circle** around the state, or by adding a label to the state to indicate that it is a stopping state.

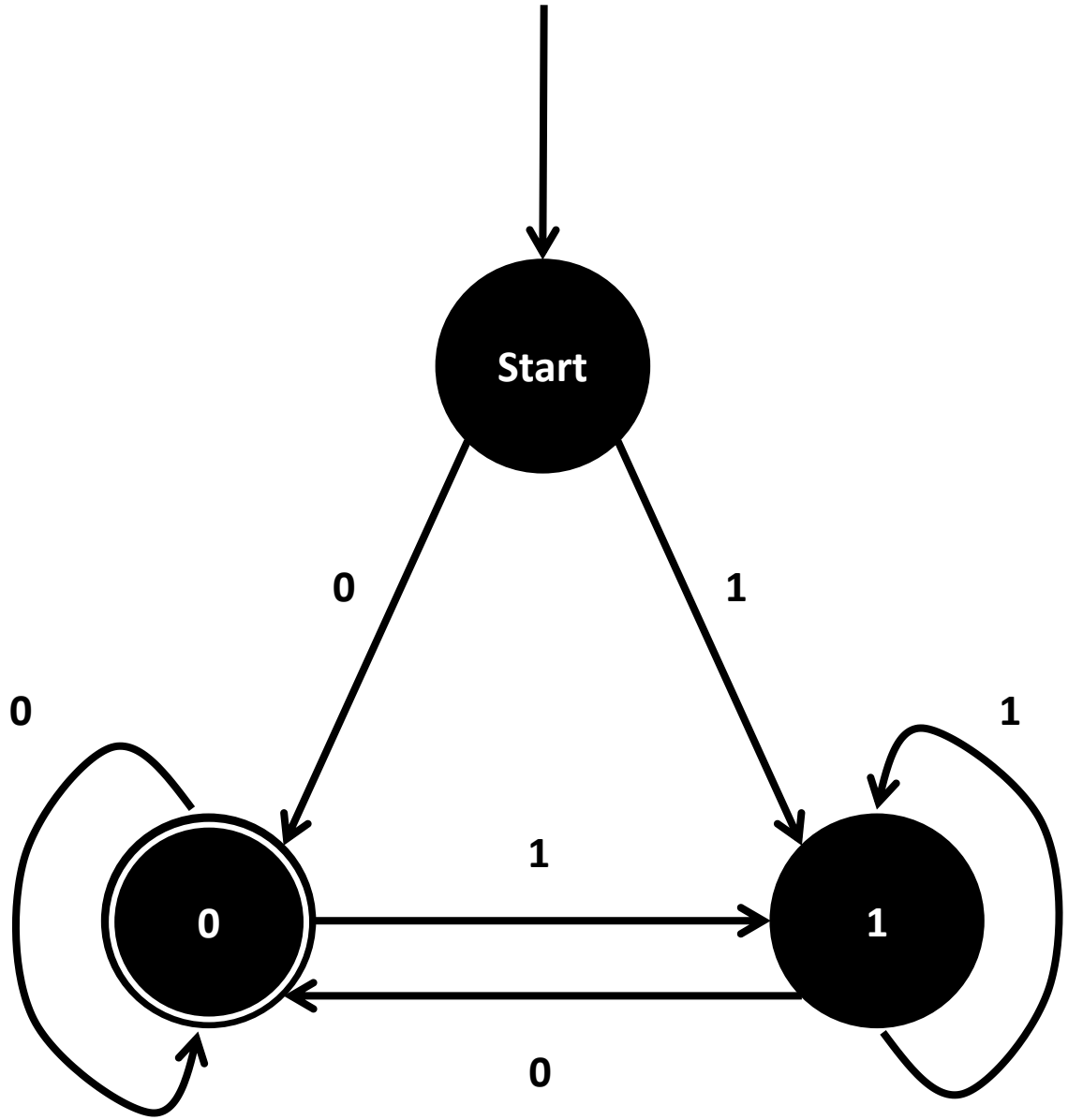
In our example, **the state 0 is now a stopping state**, but 1 and Start are not.



Adding a stopping state

As we will see later through practice, the FSM with stopping states can be used to perform specific tasks, such as recognizing patterns or processing data.

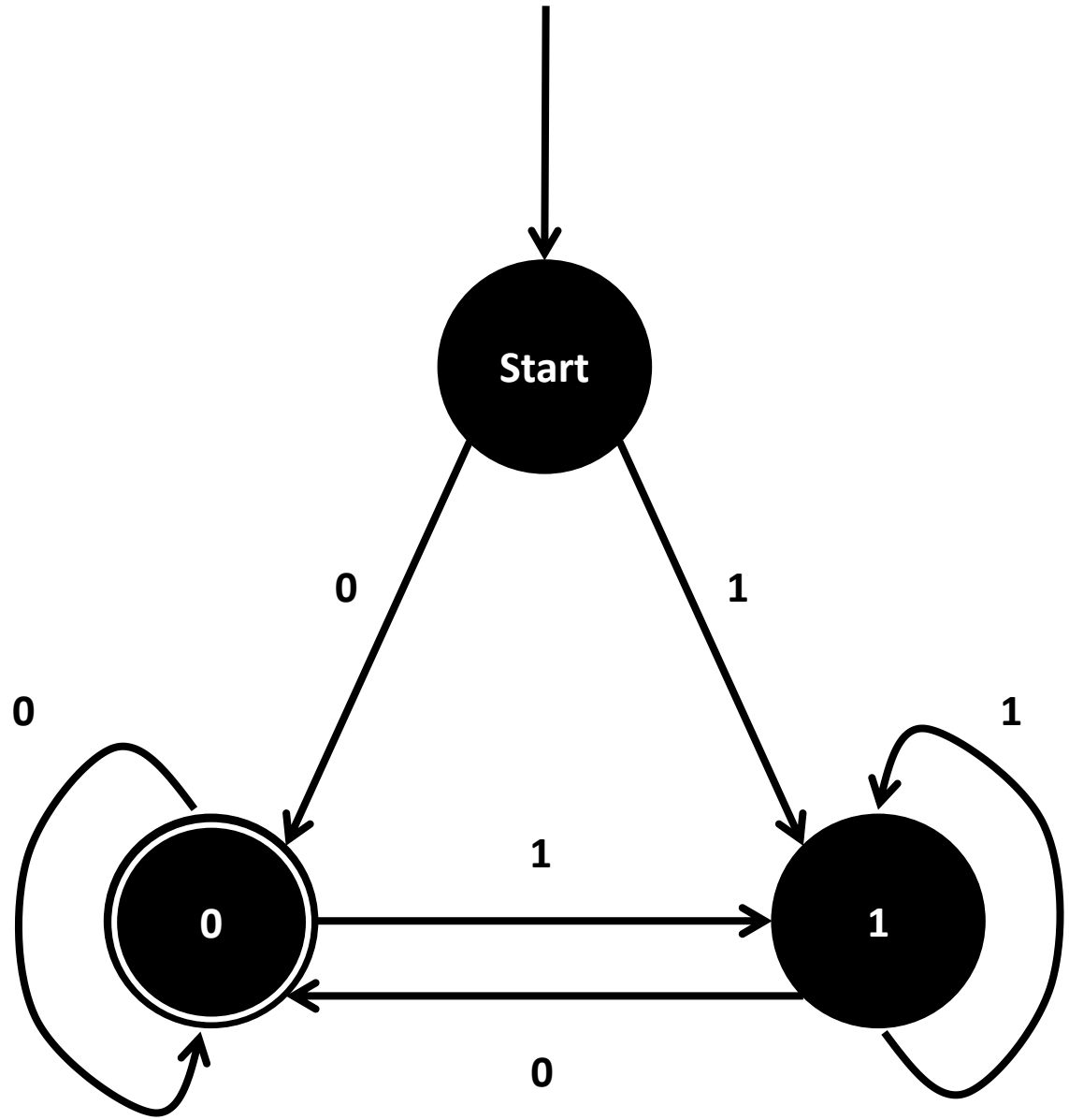
In a language recognizer FSM, the stopping state might be reached when the FSM has recognized a certain substring in the input string *x*.



Elements of an FSM with stopping states

In order to define a FSM with stopping state, we keep the previous FSM elements:

1. A **finite set of states *S***.
2. A **finite set of inputs or actions *A***.
3. A **starting state $s_0 \in S$**.
4. A **transition function f**, or **transition table**, which describe the **transition logic** in the FSM.



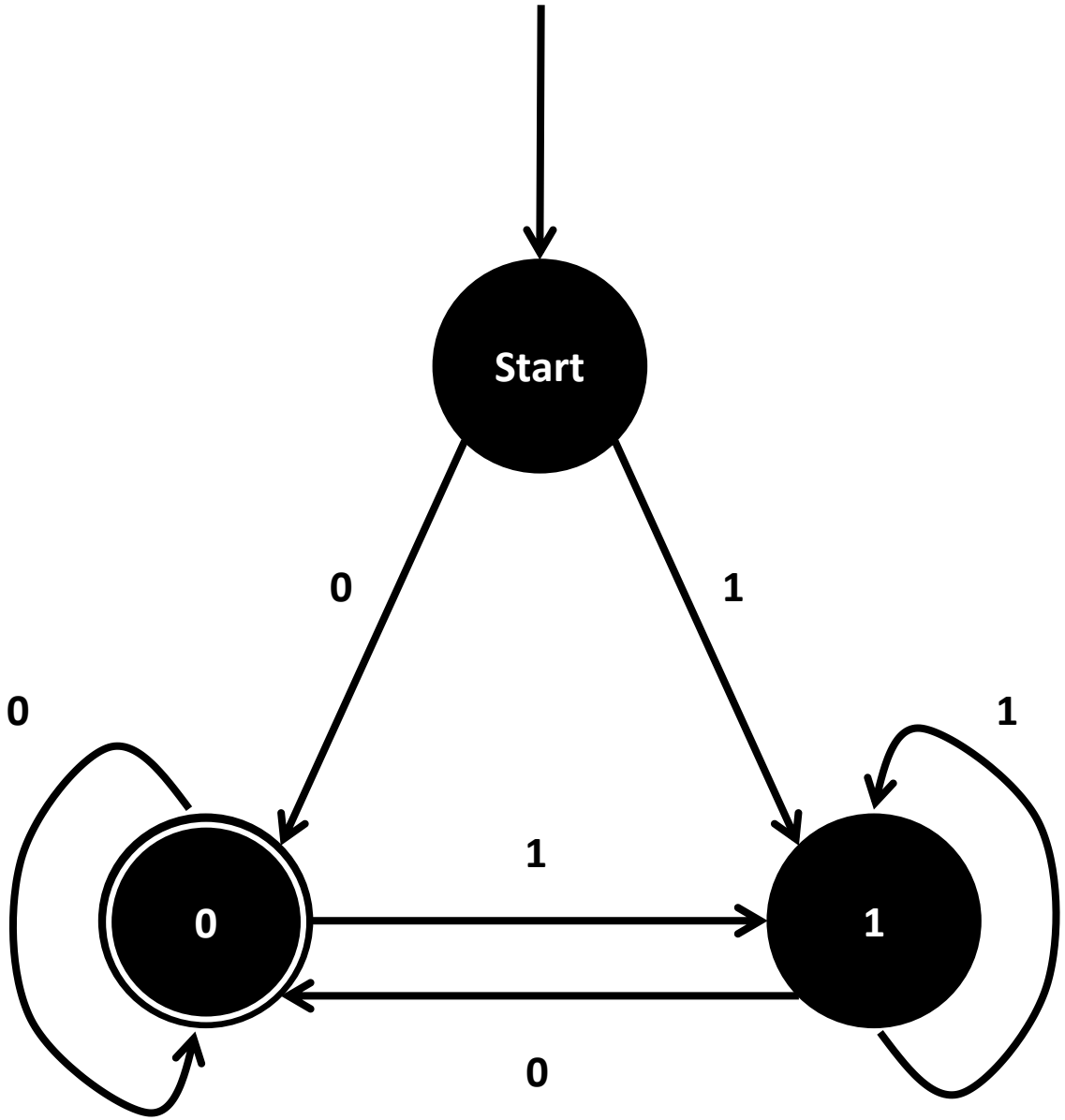
Elements of an FSM with stopping states

In order to define a FSM with stopping state, we keep the previous FSM elements.

5. And we add **a finite set of stopping states F**, defined as a subset of **all possible states S**, i.e. $F \subseteq S$.

In our example, we simply have

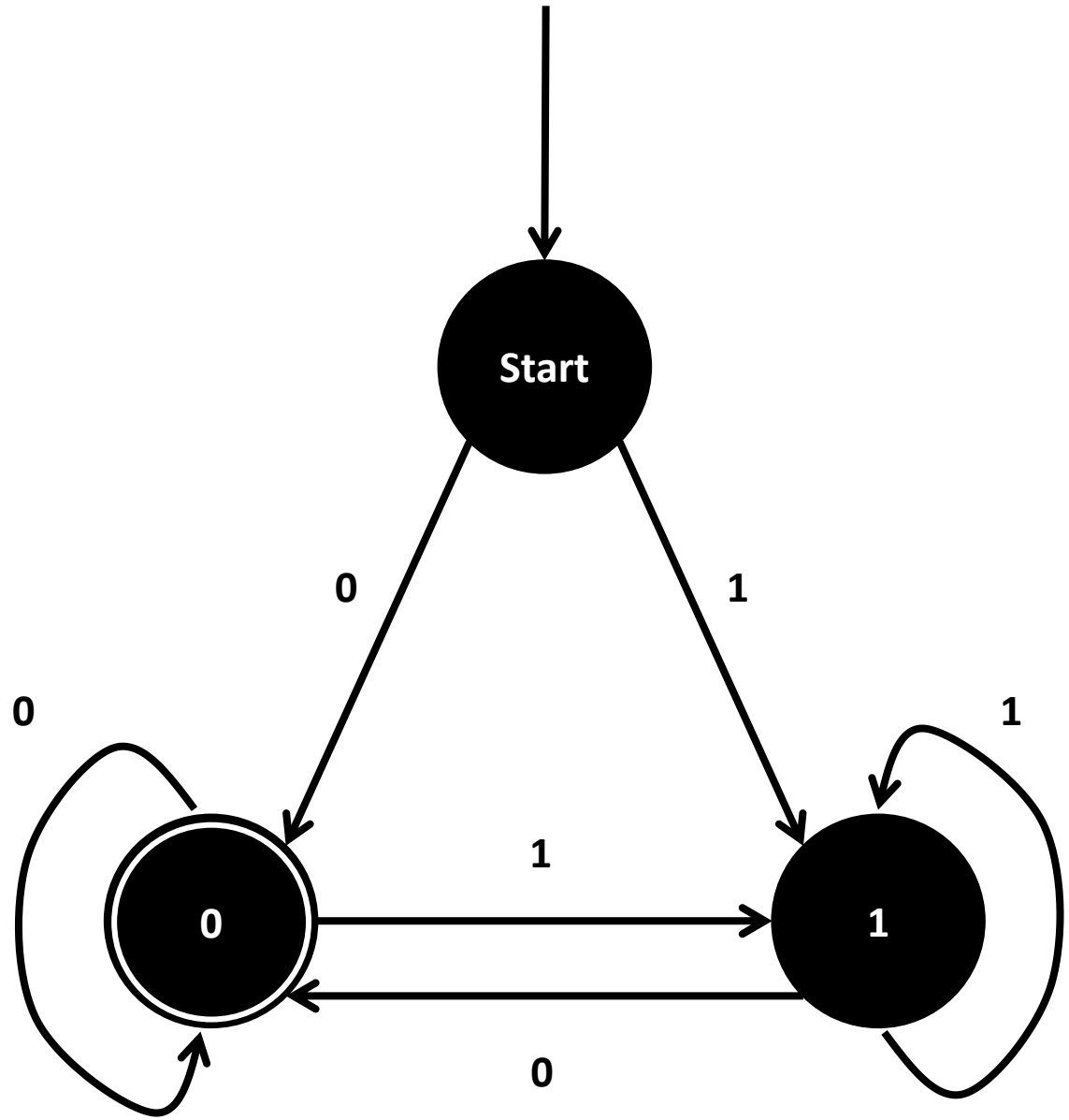
$$F = \{0\}$$



Practice 1: Analysing our example FSM

Question 1: Looking at the FSM state diagram on the right, what are the **acceptable inputs *x*** that our FSM is checking?

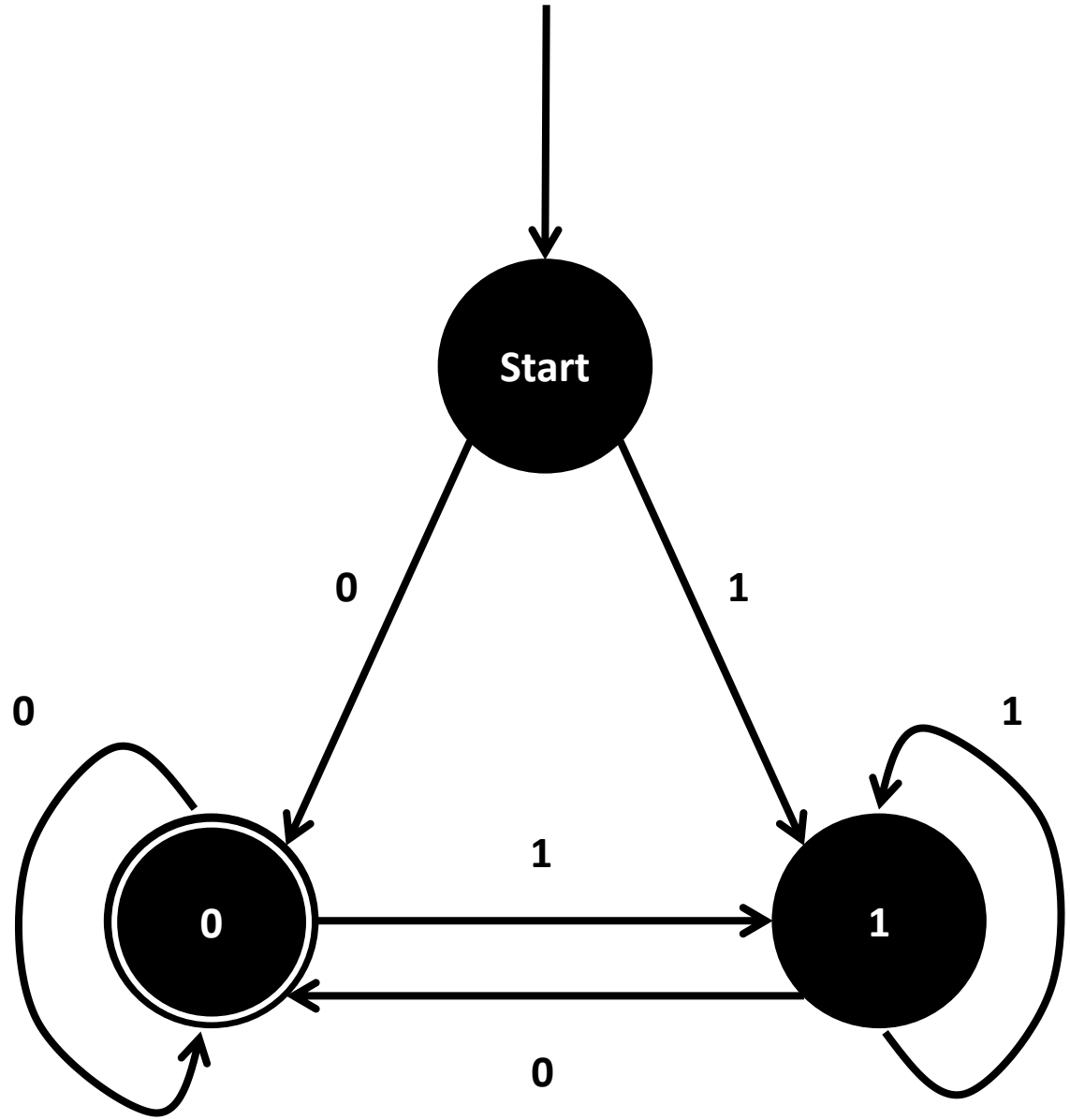
Question 2: If these binary strings now encode an int number, what **numbers** does this FSM consider as **acceptable inputs**?



Practice 1: Analysing our example FSM

Question 1: Looking at the FSM state diagram on the right, what are the **acceptable inputs *x*** that our FSM is checking?

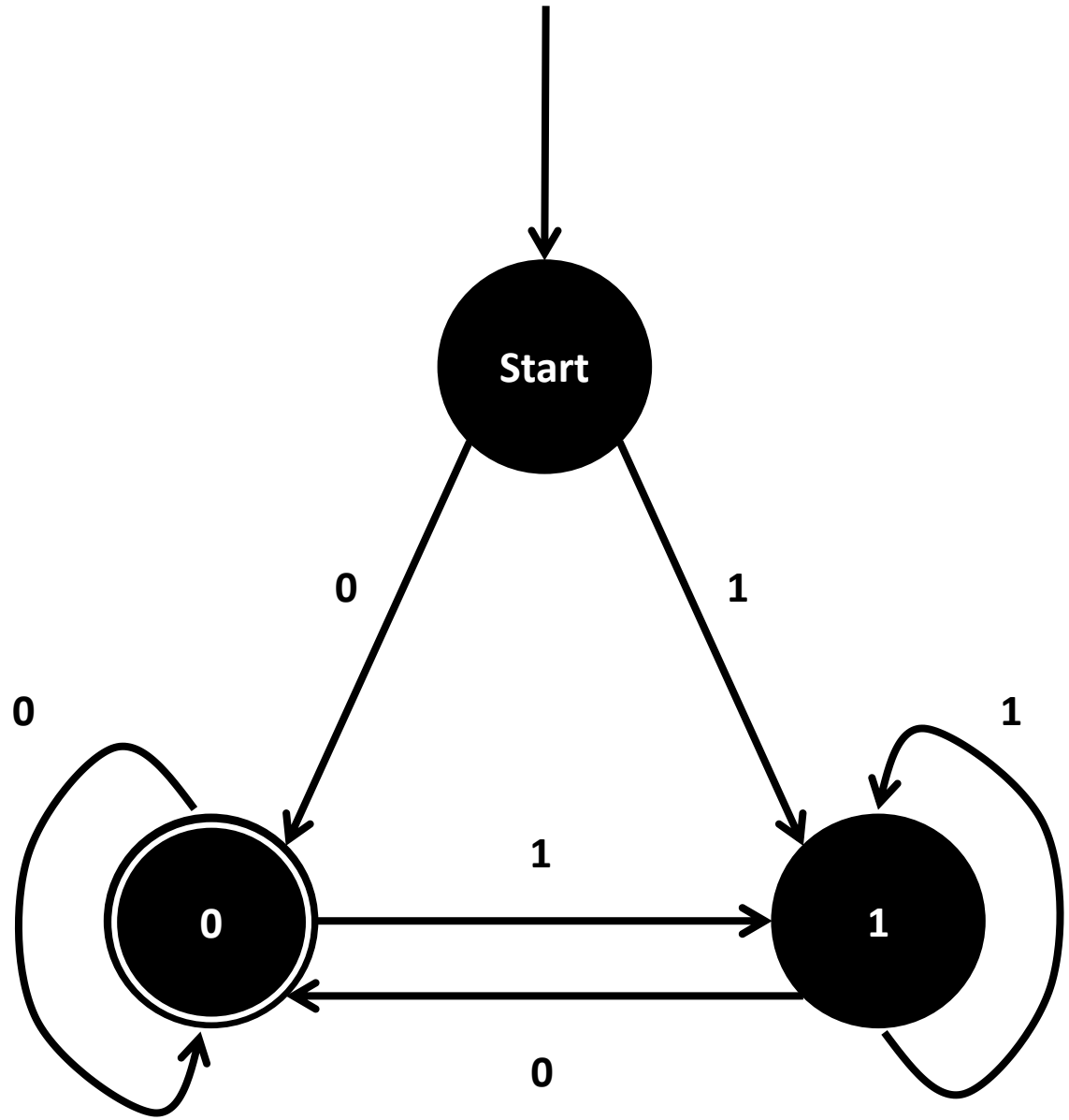
Answer: We need to end on a final state being 0. In our case, this means that an acceptable string s simply consists of any binary string that ends with 0.



Practice 1: Analysing our example FSM

Question 2: If these binary strings now encode an int number, what **numbers** does this FSM consider as **acceptable inputs**?

Answer: We need the bit of least importance to be 0. This means that the integer number input represented by the string *s* needs to be even to be acceptable.



Practice 2: a simple FSM for word recognition

We would like to write an FSM with stopping states that will take strings *x* consisting of combinations of four characters: S, U, T and D.

Possible combinations for the string *x* include, among many others, “USD”, “SUUUUTD”, and the only acceptable input “SUTD”.

Draw a FSM state diagram, which:

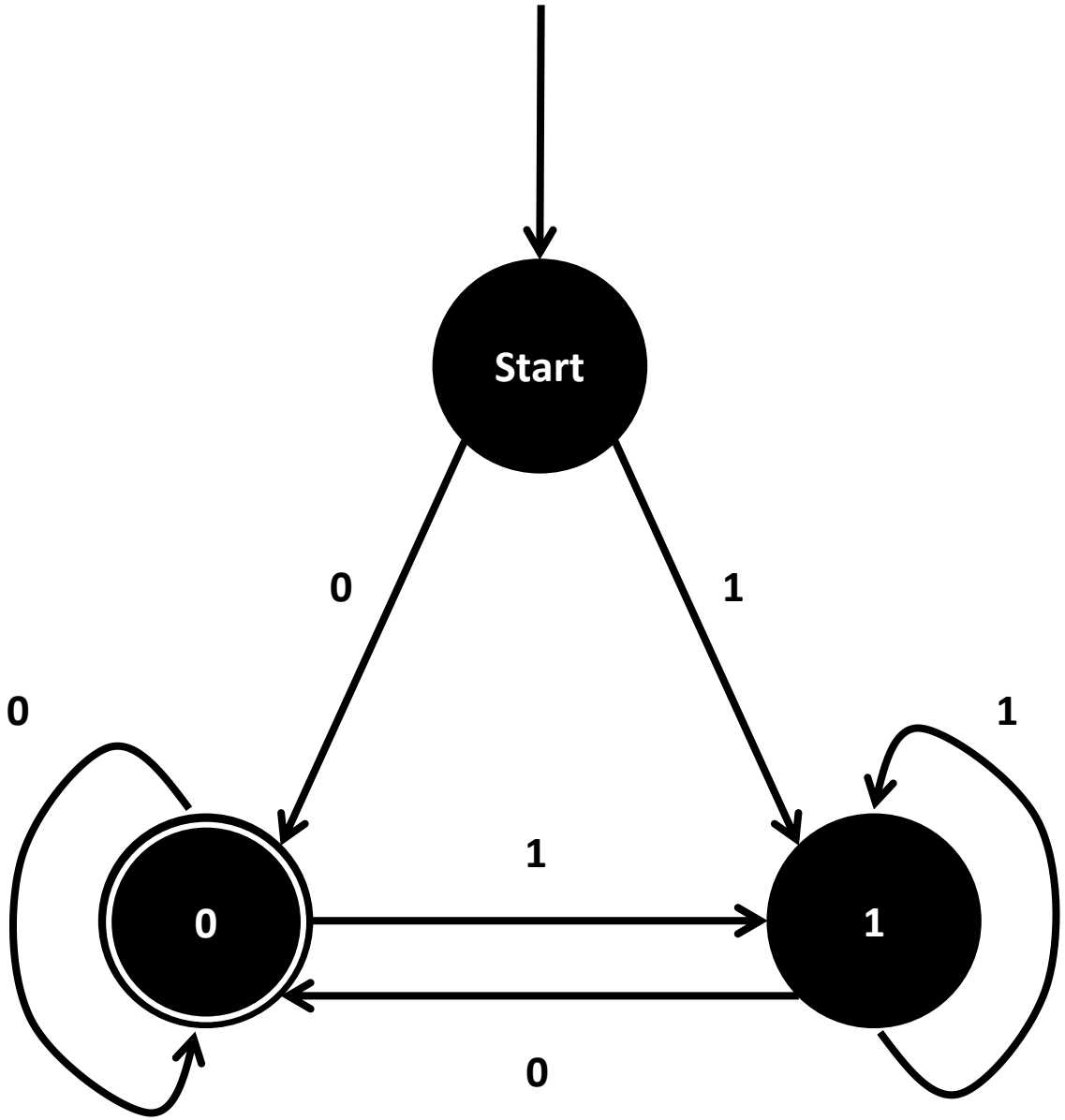
- Has 7 possible States (Start, S, U, T, D, Valid, Invalid),
- Has 4 possible Actions (S, U, T, D),
- Has the Start state defined as the starting state,
- Has the Valid state defined as the only stopping state,
- Has the FSM stop in this state, if and only *x* is SUTD; otherwise, it stops in another state (Invalid or something else).

Elements of an FSM with outputs

In general, outputs with stopping or accepting states are useful, but limited in terms of applications.

A stronger version of the FSM consists of the FSM with **outputs**.

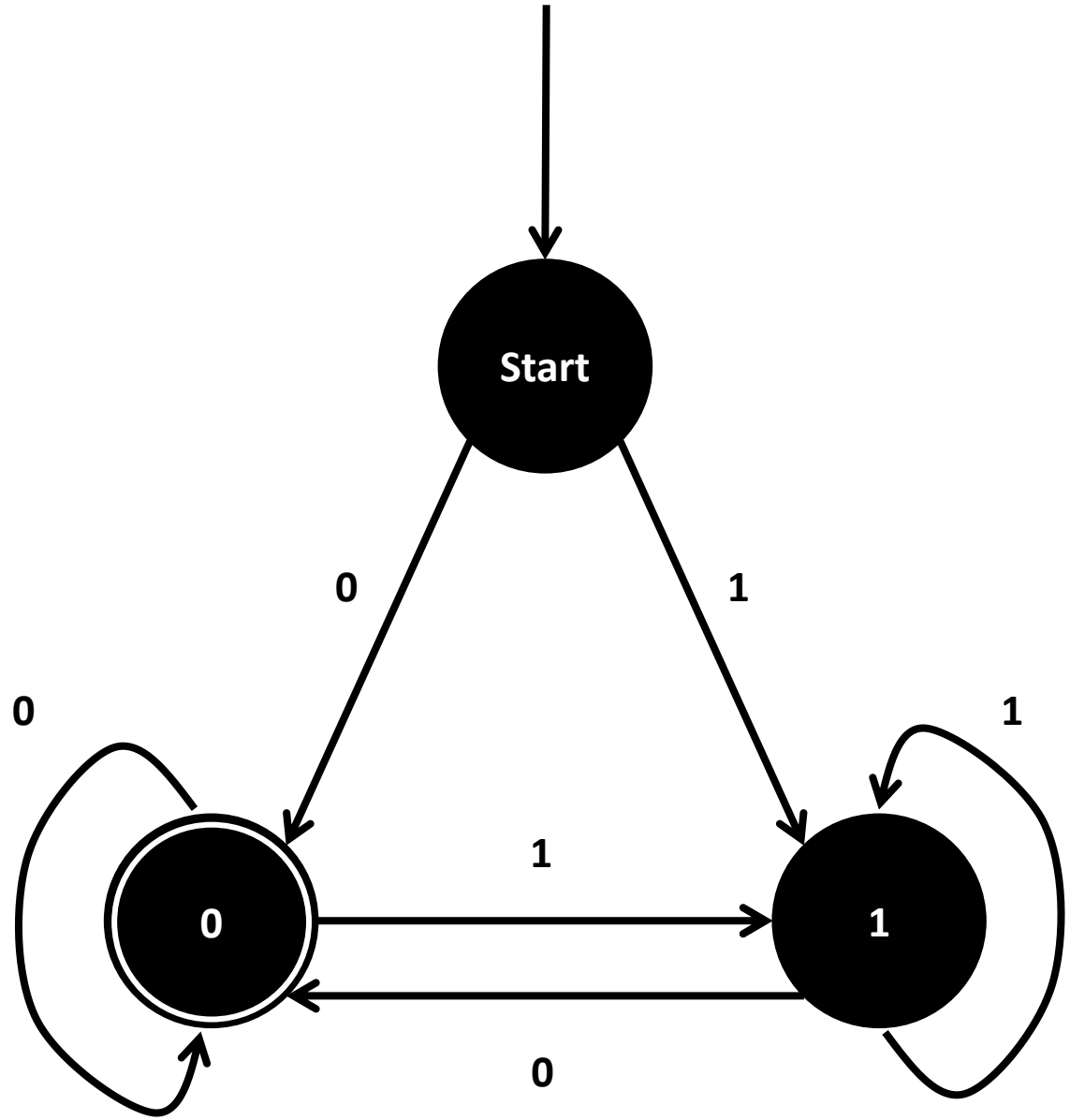
It simply replaces the stopping states with outputs being produced every time an action is taken.



Elements of an FSM with outputs

In order to define a FSM with outputs, we keep the previous FSM elements:

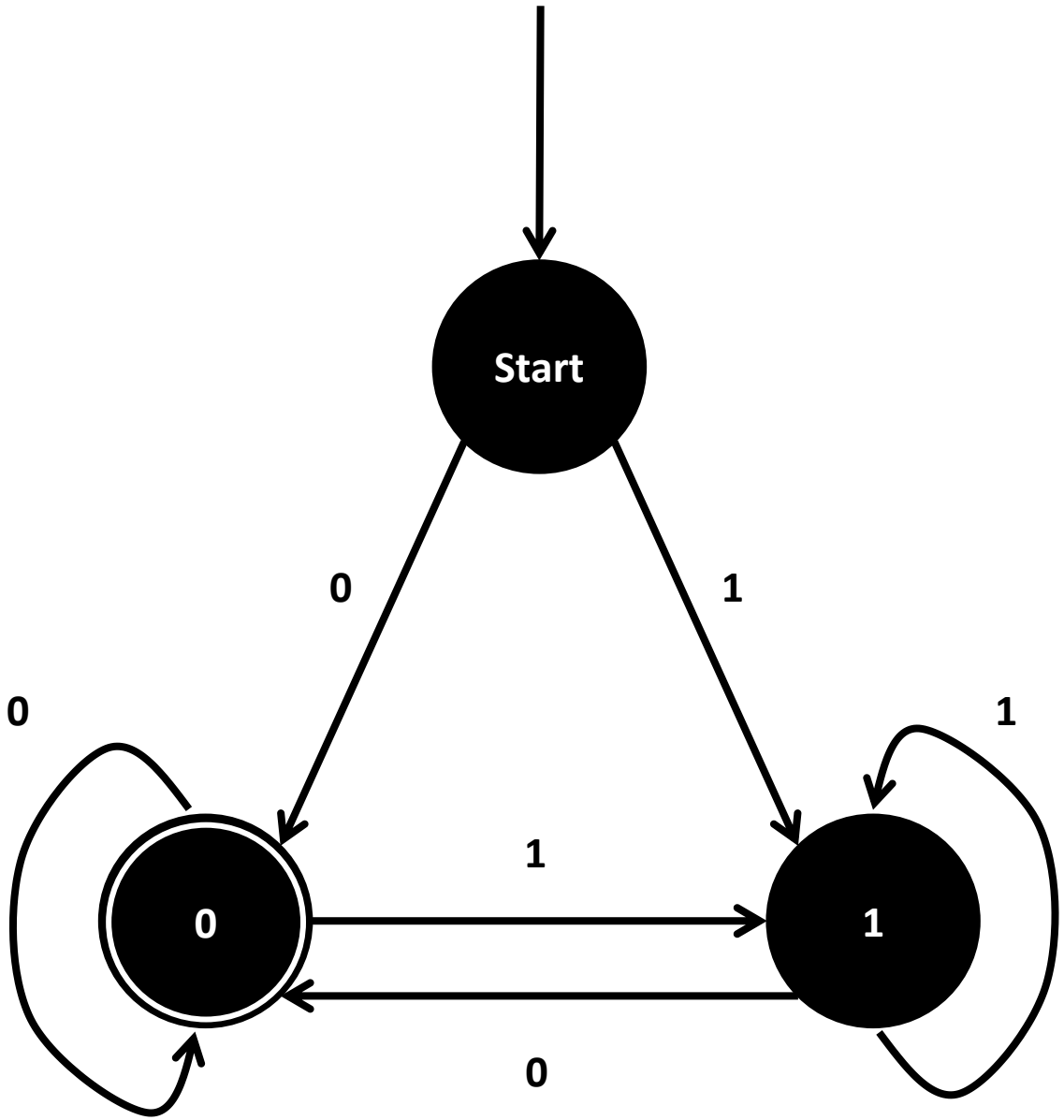
1. A **finite set of states *S***.
2. A **finite set of inputs or actions *A***.
3. A **starting state $s_0 \in S$**.
4. A **transition function f**, or **transition table**, which describe the **transition logic** in the FSM.



Elements of an FSM with outputs

5. And we add **a finite set of possible outputs Y**,
6. And an **output function g**, which decides on an output $y \in Y$ to produce given any action $a \in A$ taken in any given state $s \in S$.

$$g: S \times A \rightarrow Y$$
$$g(s, a) = y$$



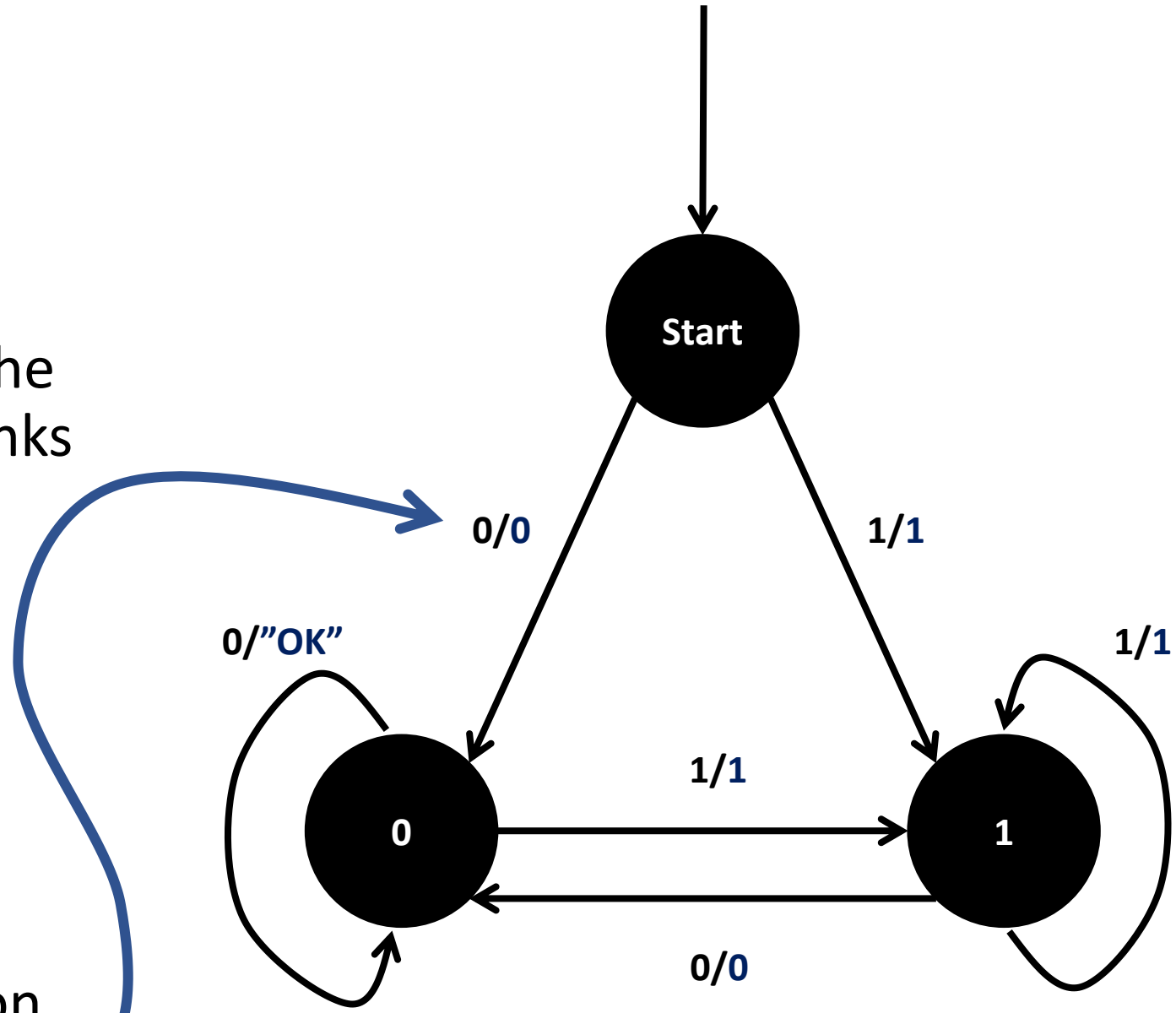
Elements of an FSM with outputs

Outputs are then added using the "a/y" notation on each of the links of the FSM.

In the FSM on the right, the output set Y is defined as

$$Y = \{0, 1, OK\}$$

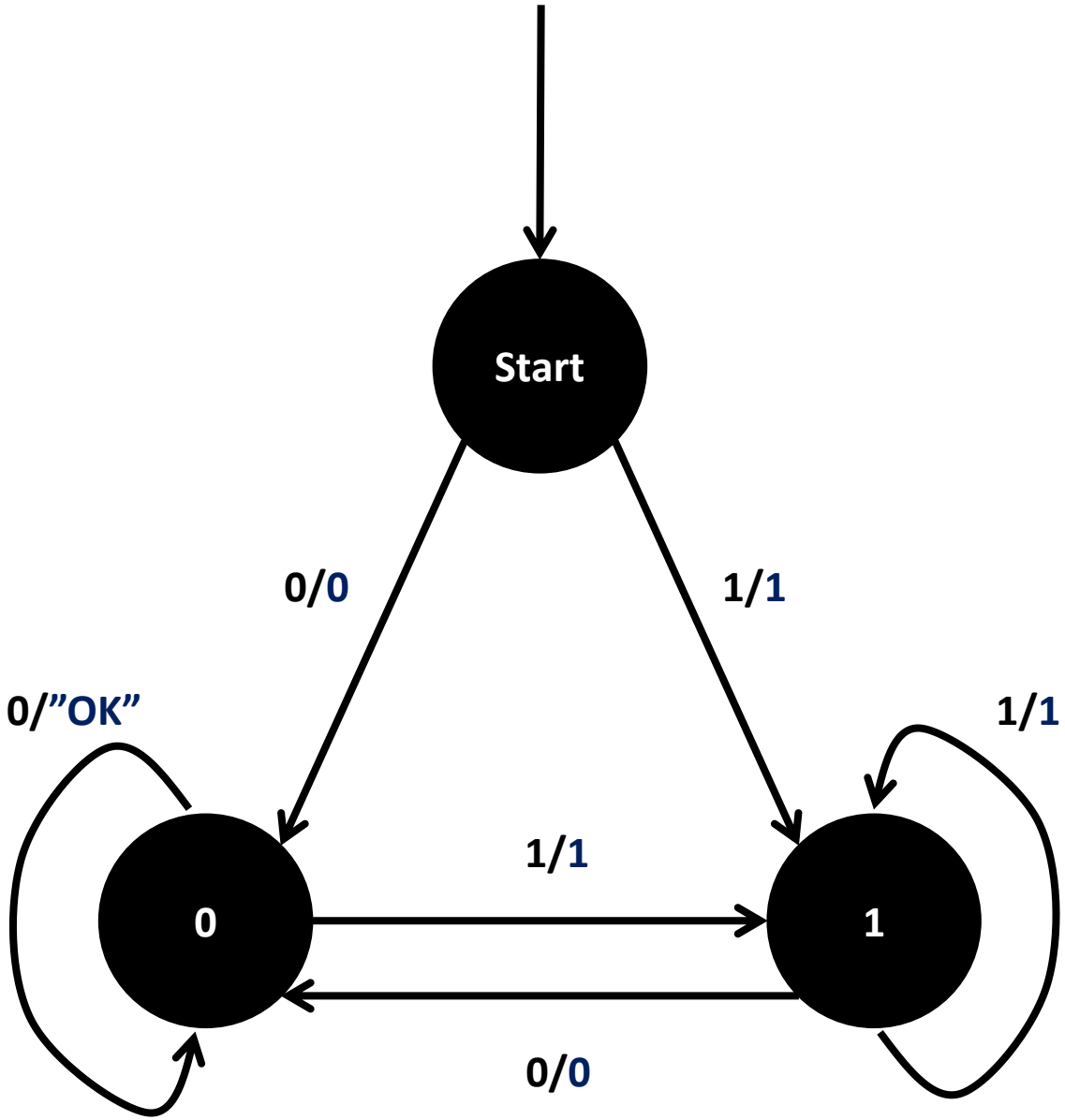
When on start node, using action 0 produces an output 0.



Elements of an FSM with outputs

Could also define outputs in the form of a **table of values** to be produced if a given action *a*, is taken in a state *s*.

Similar to the **transition table** from earlier, which gave us the new state *s'* if a given action *a*, is taken in a state *s*.



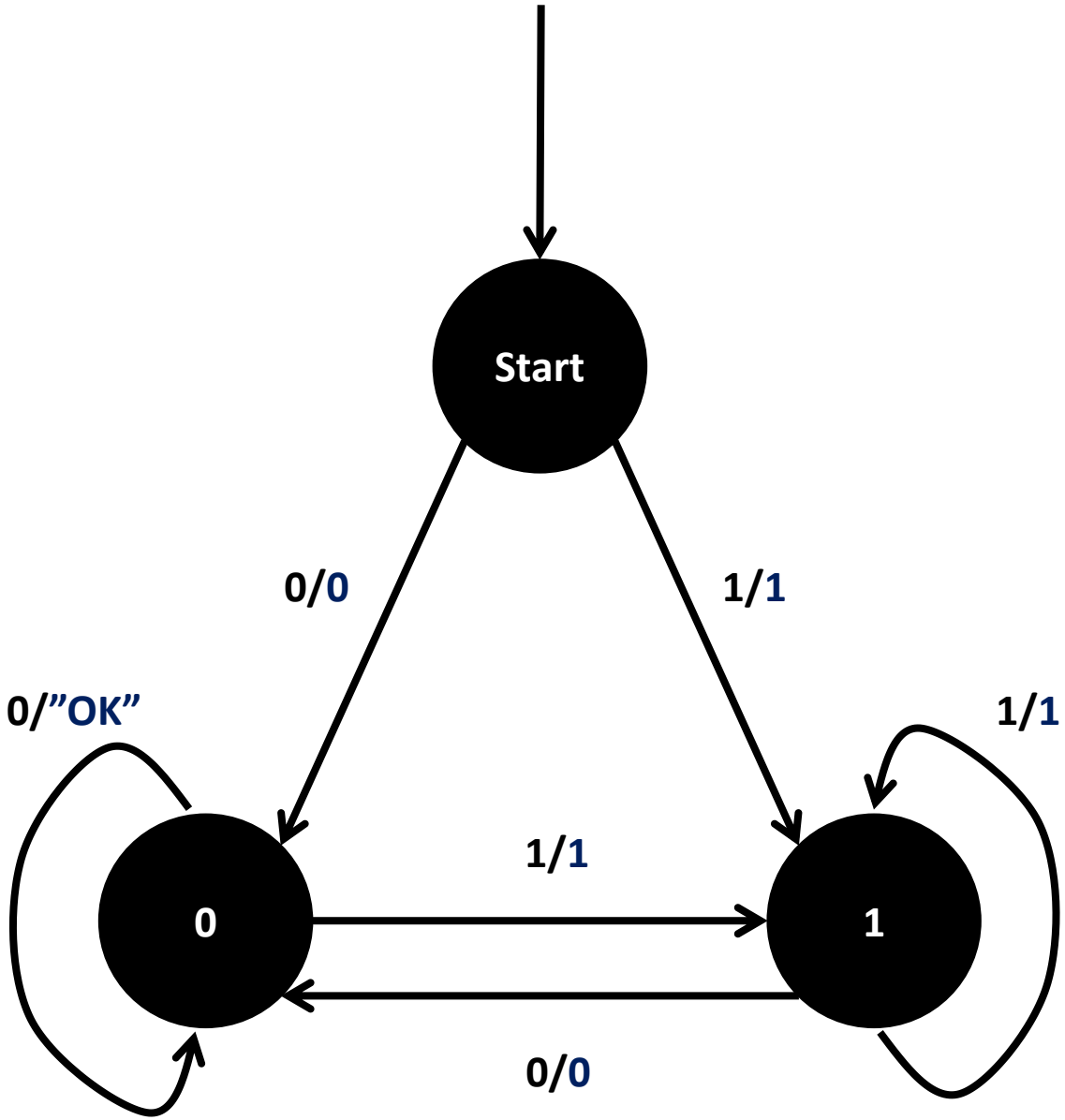
Practice 3

In the FSM on the right, the output set Y is defined as $Y = \{0, 1, OK\}$, and we use the "a/y" notation.

Question: Let us assume that the FSM stops when an output "OK" is seen or the string *x* runs out of characters.

The FSM considers acceptable inputs *x* any input that produces "OK" as an output at some point.

Which input strings x are then considered acceptable?

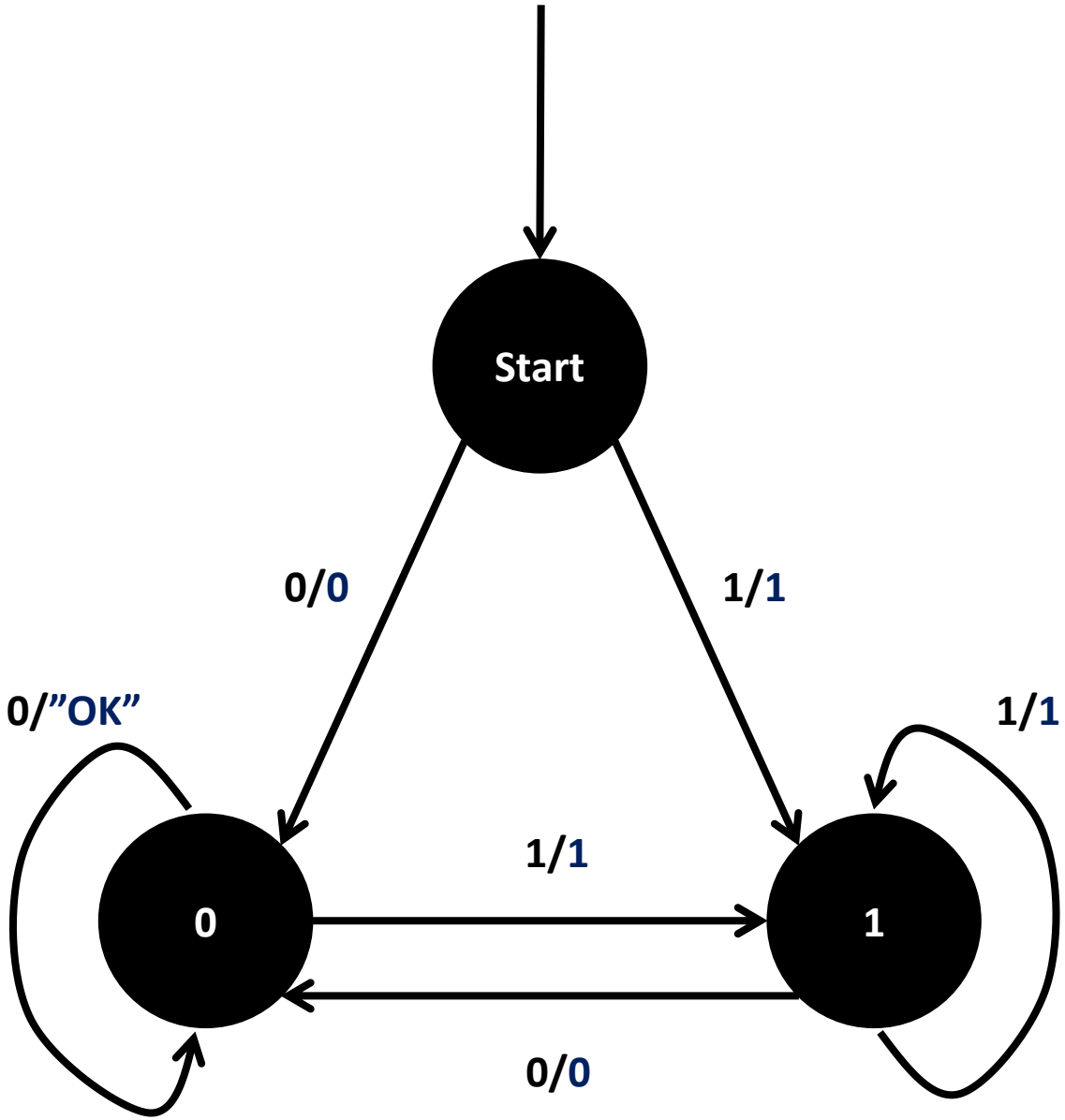


Practice 3

In the FSM on the right, the output set Y is defined as $Y = \{0, 1, OK\}$, and we use the "a/y" notation.

Answer: The "OK" output is produced if and only if there is an action $a = 0$ being taken while in state $s = 0$.

This is equivalent to having two 0s in succession somewhere in the string *x*. For instance 100, 10100, 1001, 1001111110101, etc...



Practice 4

Consider a vending machine and describe it as a FSM with outputs.

It takes three possible actions.

- “0.5”: insert a 50 cents coin,
- “1”: insert a 1 dollar coin,
- “B”: press the machine button.

It also has four possible outputs:

- “0.5”: give back a 50 cent coin to the user,
- “1”: give back a 1 dollar coin to the user,
- “B”: give a chocolate bar to the user,
- “N”: do nothing.

Practice 4

We would like to define a vending machine that has the following logic.

- Whenever a coin is inserted by the user, the total balance is updated.
- If the user has insert 1.5 dollars in total and presses the button, a bar will be given and the balance will return to 0.
- If the user presses the button but the balance is not yet 1.5 dollars, nothing happens.
- If the user inserts a coin and the new balance exceeds the maximal allowed balance of 1.5 dollars, then the machine will return the last coin the user has inserted.

Question: What could the possible states for this FSM be? Draw a state diagram for this FSM.

More advanced FSMs?

Many variations to the state machines we have describe exist.

For instance,

- **Deterministic Finite State Machines (DFSMs):** A deterministic finite state machine is a mathematical model used to represent a system that has a finite number of states and can transition between those states based on input.

It is basically another name for the FSMs (with no outputs) we have been playing with.

More advanced FSMs?

Many variations to the state machines we have describe exist.

For instance,

- **Mealy Machines:** A Mealy machine is a finite state machine where the output is dependent on both the current state and the input. That is basically another name for our FSM with outputs!
- **Moore Machines:** A Moore machine is a finite state machine where the output is dependent only on the current state. A special case of the Mealy machine.

More advanced FSMs?

Many variations to the state machines we have describe exist.

For instance,

- **Non-Deterministic Finite State Machines (NDFSMs):** In a non-deterministic finite state machine, the machine can transition from one state to multiple states based on the input. This means that there can be multiple possible paths through the machine for a given input.

Clearer with an example.

More advanced FSMs?

As an example of a NDFSM, let us go back to our vending machine.

- The NDFSM for this vending machine could have multiple paths, depending on the coins the user inserts and the order in which they are inserted.

More advanced FSMs?

Many variations to the state machines we have describe exist.

For instance,

- **Stochastic Finite State Machines (SFSMs):** A stochastic finite state machine is a model used to represent a system that has a finite number of states, but where the transitions have randomness aspects included. This means that there will be probabilities of transitioning between states.

More advanced FSMs?

As an example of a SFSM...

- Suppose we have a game where two players take turns.
- On their turn, each player rolls a dice and moves their game piece forward a number of spaces equal to dice roll.
- The game board has 20 spaces numbered 1 through 20, and the player wins if they reach space 20.
- The transitions between the states are then determined by the probabilities of the dice rolls.
- From the Start state, the machine transitions to the next state with equal probability corresponding to rolling a value between 1 and 6.
- This SFSM can be used to simulate the game and determine the probabilities of the first player winning (this player starts and should probably have the advantage over the second player).

Practice 5

We would like to write an FSM with stopping states and no outputs that will take strings *x* consisting of combinations of four characters: Z, A, and M.

Possible combinations include “MAZ”, “AMAZ” and the only acceptable input “ZAMZAM”.

Draw a FSM state diagram, which:

- Has possible States, which you are free to decide,
- Has 3 possible Actions (Z, A, M),
- Has the Start state defined as the starting state,
- Has one stopping state,
- Has the FSM stop in this state, if and only *x* is “ZAMZAM”; otherwise, it stops in another state.

Practice 6

We would like to write an FSM with stopping states and no outputs that will take strings *x* consisting of combinations of four characters: Z, A, and M.

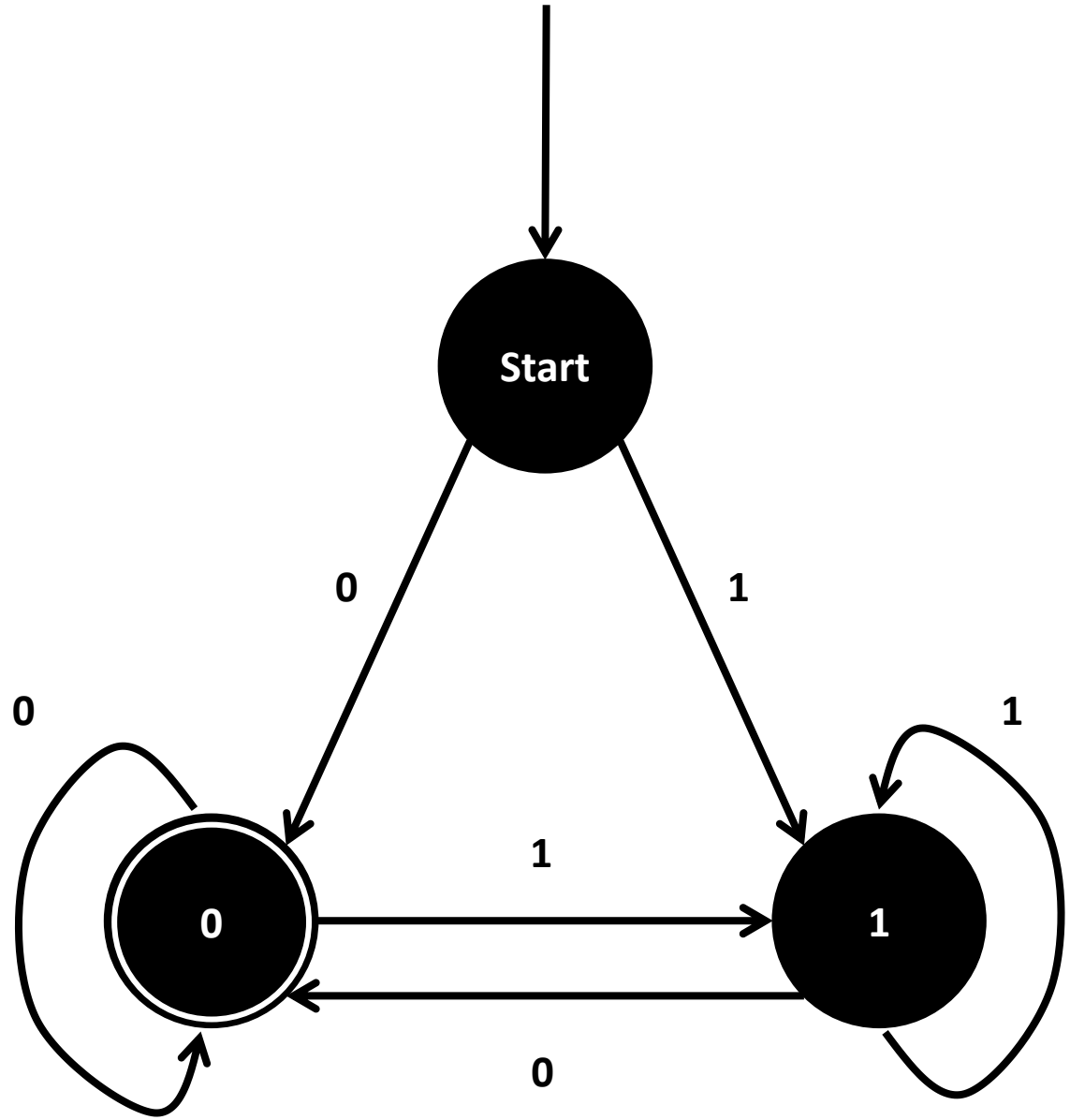
Possible combinations include “MAZ”, “AMAZ” and the only acceptable input “ZAMZAM”.

Draw a FSM state diagram, which:

- Has possible States, which you are free to decide,
- Has 3 possible Actions (Z, A, M),
- Has the Start state defined as the starting state,
- Has one stopping state,
- Has the FSM stop in this state, if and only *x* **contains** the string “ZAM”; otherwise, it stops in another state.

Practice 7

How would you modify this FSM with stopping state, so that it considers as acceptable inputs any string *x* of 0 and 1, that have an even number of zeroes?



Practice 8

Let us go back to the Practice 4 – Vending Machine example.

Question: What if we wanted for the machine to keep track of any amount of money we would have inserted and return the change only after the button has been pressed?

Could this be represented with a simple FSM?