Technische Universität München **Winter term 2017/18**
I7
Prof. J. Esparza / Dr. M. Blondin **27.12.2017**

Automata and Formal Languages — Homework 10

Due 09.01.2018

Exercise 10.1
It is late in Munich and you are craving for nuggets. Since the U-bahn is stuck at Sendliger Tor, you have no idea how hungry you will be when reaching the restaurant. Since nuggets are only sold in boxes of 6, 9 and 20, you wonder if it will be possible to buy exactly the amount of nuggets you will be craving for when arriving at the restaurant. You also wonder whether it is always possible to buy the exact amount of nuggets if one is hungry enough. Luckily, you can answer these questions since you are quite knowledgeable about Presburger arithmetic and automata theory.

For every finite set $S \subseteq \mathbb{N}$, let us say that $n \in \mathbb{N}$ is an *S-number* if n can be obtained as a linear combination of elements of S. For example, if $S = \{6, 9, 20\}$, then 67 is an S-number since $67 = 3 \cdot 6 + 1 \cdot 9 + 2 \cdot 20$, but 25 is not. For some sets S, there are only finitely many numbers which are not S-numbers. When this is the case, we say that the largest number which is not an S-number is the *Frobenius number* of S. For example, 7 is the Frobenius number of $\{3, 5\}$, and $S = \{2, 4\}$ has no Frobenius number.

To answer your questions, it suffices to come up with algorithms for Frobenius numbers and to instantiate them with $S = \{6, 9, 20\}$.

(a) Give an algorithm that decides, on input $n \in \mathbb{N}$ and a finite subset $S \subseteq \mathbb{N}$, whether n is an S-number.

(b) Give an algorithm that decides, on input $S \subseteq \mathbb{N}$ (finite), whether S has a Frobenius number. [Hint: [redacted]]

(c) Give an algorithm that computes, on input $S \subseteq \mathbb{N}$ (finite), the Frobenius number of S (assuming it exists).

(d) ★ Show that $S = \{6, 9, 20\}$ has a Frobenius number, and identify this number.

Exercise 10.2
Let $\Sigma = \{a, b\}$. Give an MSO(Σ) sentence for the following languages:

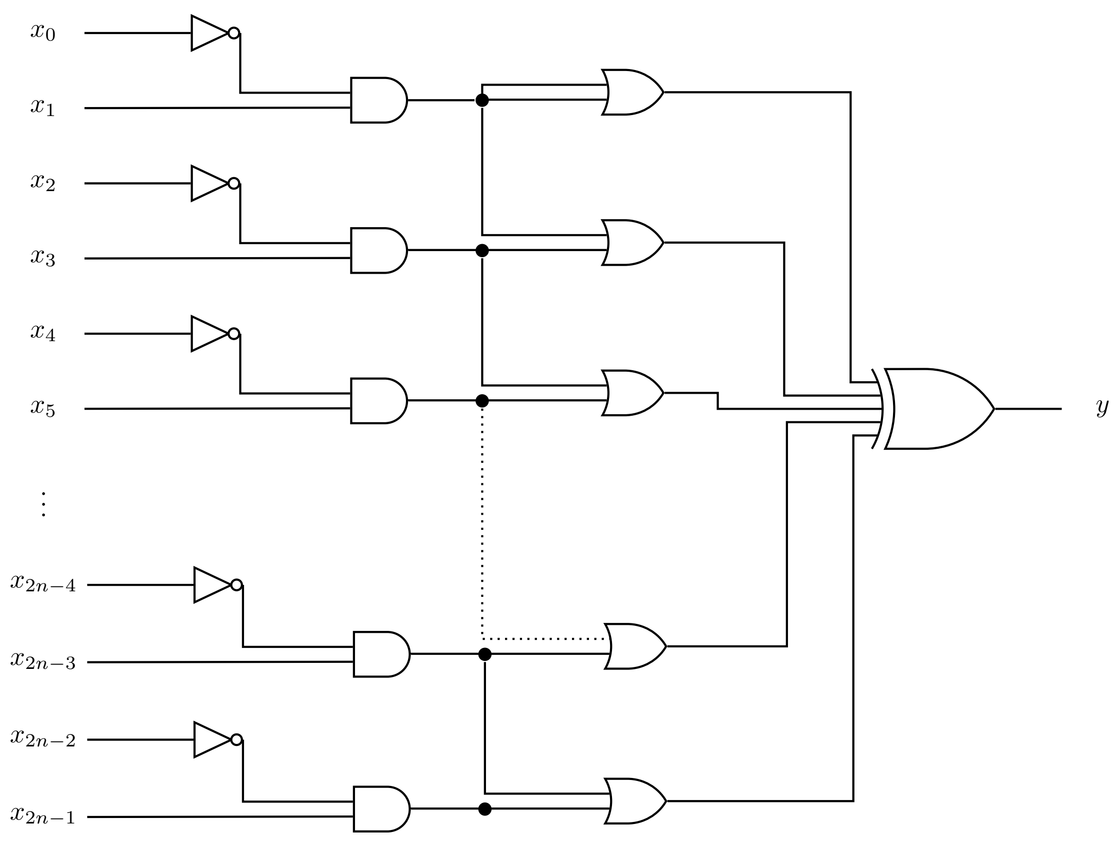
(a) The set of words with an a at every odd position.

(b) The set of words with an even number of occurences of a's.

(c) The set of words of odd length with an even number of occurences of a's.

Exercise 10.3

Let $n \in \mathbb{N}_{>0}$. Consider the following circuit C_n:



In case you are not familiar with the above symbols, the first, second, third and fourth layers of gates are respectively NOT, AND, OR and XOR gates. Note that $\text{XOR}(z_1, z_2, \ldots, z_n) = z_1 \oplus z_2 \oplus \cdots \oplus z_n$. For every $n \in \mathbb{N}_{>0}$, let $X_n = \{x \in \{0,1\}^n : C_n \text{ outputs 1 on input } x\}$. Let $X = \bigcup_{n \in \mathbb{N}_{>0}} X_n$. In other words, X is the set of assignments that satisfy some circuit of the infinite family of circuits $\{C_n : n \in \mathbb{N}_{>0}\}$.

(a) Give an MSO sentence ϕ such that $L(\phi) = X$.

(b) ★ Use `MONA` to obtain an automaton accepting X.

To solve (a), you should consider constructing a formula for each layer of gates. For example, the following predicate asserts that *Out* contains precisely the indices of wires set to 1 past the first layer of NOT gates:

$$\begin{aligned}\text{layer1}(\mathit{Out}) = \forall p\ (\text{even}(p) &\rightarrow ((p \in \mathit{Out}) \leftrightarrow Q_0(p))) \land \\ (\text{odd}(p) &\rightarrow ((p \in \mathit{Out}) \leftrightarrow Q_1(p)))\end{aligned}$$

For this question, you may assume that positions begin at 0 instead of 1 in MSO. This is the case in `MONA`.

† Question 10.3 is adapted from an example of Henriksen et al. *Mona: Monadic second-order logic in practice.* Proc. International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 1995.

Solution 10.1

(a) Let $S = \{a_1, a_2, \ldots, a_k\}$. A number $n \in \mathbb{N}$ is an S-number if and only if there exist $x_1, x_2, \ldots, x_k \in \mathbb{N}$ such that $n = a_1x_1 + a_2x_2 + \ldots + a_kx_k$ which is equivalent to $n - a_1x_1 - a_2x_2 - \ldots - a_kx_k = 0$. Therefore, given S, we do the following:

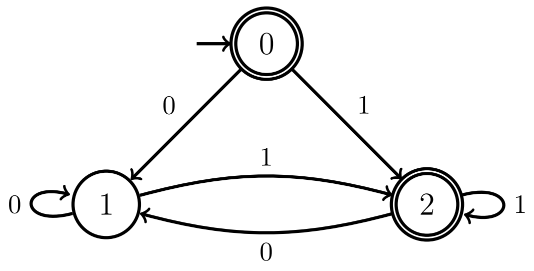
1. Construct a transducer A accepting the solutions of $y - a_1x_1 - a_2x_2 - \ldots - a_kx_k = 0$ (using *EqtoDFA*),
2. Construct an automaton B obtained by projecting A onto y,
3. Test whether lsbf(n) is accepted by B,
4. Return *true* if and only if lsbf(n) is accepted.

★ Note that A is a DFA, but B might be an NFA due to the projection.

(b) Let B the automaton constructed in (a). Note that S has a Frobenius number if and only if $\{n \in \mathbb{N} : \text{lsbf}(n) \notin L(B)\}$ is finite. This suggests to complement B. Since B is an NFA, we must first convert it to a DFA B' and then complement B'. Let C be the resulting DFA.

To test whether S has a Frobenius number, it is now tempting to test whether $L(C)$ is finite. This is however incorrect. Indeed, every natural number has infinitely many *lsbf* encodings, e.g. 2 is encoded by 010^*. Therefore, even if C accepts finitely many numbers, $L(C)$ will be infinite.

To address this issue, we prune $L(C)$ by keeping only the minimal encoding of each number accepted by C. Note that an *lsbf* encoding is minimal if and only if it does not contain any trailing 0. Thus, we can construct a DFA M accepting the set of minimal *lsbf* encodings:



To prune $L(C)$ of the redundant *lsbf* encodings, we construct a new DFA D obtained by intersecting C with M.

It remains to test whether $L(D)$ is finite. By construction, every state of D is reachable from the initial state. However, due to our transformations, it may be the case that some states of D cannot reach a final state. We may remove these states in linear time. This can be done by (implicitly) reversing the arcs of D and then performing a depth-first search from the final states. The states which are not explored by the search are removed from D.

Let D' be the resulting DFA. Testing whether $L(D')$ is finite amounts to testing whether D' contains no cycle. This can be done in linear time using a depth-first search.

The overall algorithm is as follows:

1. Convert B to a DFA B',
2. Obtain a new DFA C by complementing B',
3. Obtain a new DFA D by intersecting C with M,
4. Obtain a new DFA D' by removing every state of D that cannot reach some final state,
5. Test whether D' contains a cycle.
6. Return *true* if and only if D' contains no cycle.

★ Let us show that it is indeed the case that $L(D')$ is finite if and only if D' contains no cycle. Equivalently, we may show that $L(D')$ is infinite if and only if D' contains a cycle. Let $D' = (Q, \{0,1\}, \delta, q_0, F)$.

$\Rightarrow$) Assume $L(D')$ is infinite. By assumption, D' accepts a word w such that $|w| = m$ for some $m > |Q|$. Let $q_0, q_1, \ldots, q_m \in Q$ be such that $q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \cdots \xrightarrow{w_m} q_m$. By the pigeonhole principle, there exist $0 \leq i < j \leq m$ such that $q_i = q_j$. Therefore, D' contains the cycle $q_i \xrightarrow{w_{i+1}} q_{i+1} \xrightarrow{w_{i+2}} \cdots \xrightarrow{w_j} q_i$.

$\Leftarrow$) Assume D' contains a cycle $q \xrightarrow{v} q$ for some $q \in Q$ and $v \in \{0,1\}^+$. By construction of D', state q is reachable from q_0, and q can reach some final state $q_f \in F$. Therefore, there exist $u, w \in \{0,1\}^*$ such that $q_0 \xrightarrow{u} q \xrightarrow{v} q \xrightarrow{w} q_f$. Since $q \xrightarrow{v} q$ can be iterated arbitrarily many times, every word of uv^*w is accepted by D', which implies that $L(D')$ is infinite. □

(c) Assume S has a Frobenius number. Let D' be the DFA obtained in (b). The Frobenius number of S is the largest natural number n accepted by D'. By assumption, $L(D')$ is finite. Thus, we could find n by using a brute force approach where we go through all words accepted by D'. It is however possible to find n much more efficiently.

Observe that D' is acyclic. Therefore, we may compute a topological ordering $q_0, q_1, \ldots, q_m$ of Q. For every $0 \leq i \leq m$, let $\ell_i = \text{argmax}_{w \in L_i} \text{value}(w)$ where $L_i = \{w \in \{0,1\}^* : q_0 \xrightarrow{w} q_i\}$. Due to the topological ordering, each ℓ_i can be computed as follows:

$$\ell_i = \begin{cases} \varepsilon & \text{if } i = 0, \\ \text{argmax}_{w \in W} \text{value}(w) \text{ where } W = \{\ell_j \cdot a : 0 \leq j < i, a \in \{0,1\}, \delta(q_j, a) = q_i\} & \text{if } i > 0. \end{cases}$$

Once each ℓ_i is computed, we can easily derive n since $n = \max\{\text{value}(\ell_i) : q_i \in F\}$.

★ To test whether $\text{value}(u) \geq \text{value}(v)$, it is not necessary to convert u and v to their numerical values. Instead, the test can be carried by testing whether u is greater or equal to v under the colexicographic ordering, i.e. $u^R \succeq_{\text{lex}} v^R$.

(d) By constructing automaton D' for $S = \{6, 9, 20\}$, we observe that D' has no cycle. Therefore, S has a Frobenius number. By executing the procedure described in (c), we obtain 43 as the Frobenius number of S. See `frobenius.py` for a Python implementation.

Solution 10.2

(a) We first define a formula that asserts that a set contains the odd positions:

$$\text{odd}(P) = \forall p\colon (p \in P \leftrightarrow (\text{first}(p) \lor \exists q\colon (p = q + 2 \land q \in P))).$$

The sentence for the given language is:

$$\exists O\colon (\text{odd}(O) \land (\forall p\colon p \in O \rightarrow Q_a(p)).$$

(b)

$$\begin{aligned} \exists E\ \forall p, q\colon (&\text{last}(p) &&\rightarrow p \in E) \land \\ (&\text{first}(p) &&\rightarrow (p \in E \leftrightarrow \neg Q_a(p))) \land \\ (&p = q + 1 &&\rightarrow (p \in E \leftrightarrow ((q \in E) \oplus Q_a(p)))) \end{aligned}$$

(c) Let φ be the formula of (b) and let $\psi = \exists P, p\colon \text{odd}(P) \land \text{last}(p) \land p \in P$. The sentence for the given language is $\psi \land \varphi$.

Solution 10.3

(a) As suggested, we construct a formula for each layer of gates. Before doing so, we construct formulas to test whether a position is even or odd:

$$\begin{aligned} \text{even}(p) &= \exists E\colon [p \in E \land \forall q\ (q \in E \leftrightarrow (\text{first}(q) \lor (\exists r\colon r \in E \land q = r + 2))], \\ \text{odd}(p) &= \neg\text{even}(p). \end{aligned}$$

We will also need the two following abbreviations:

$$\begin{aligned} (p = 1) &= \exists q\colon \text{first}(q) \land p = q + 1, \\ (p > 1) &= \exists q, r\colon \text{first}(q) \land r = q + 1 \land p > r. \end{aligned}$$

The formula for the first layer was already given:

$$\begin{aligned}\text{layer1}(Out) = \forall p\colon (\text{even}(p) &\to (p \in Out \leftrightarrow Q_0(p))) \land \\ (\text{odd}(p) &\to (p \in Out \leftrightarrow Q_1(p))).\end{aligned}$$

Starting from the second layer, there are only n wires instead of $2n$. We make the choice of using the odd positions to represent these wires, i.e. we consider the wires to be labeled by $1, 3, \ldots, 2n-1$. Thus, for every odd number p, the value of wire p after the AND gate is the conjunction of wires $p-1$ and p prior to the AND gate:

$$\text{layer2}(In, Out) = \forall p, q\colon (\text{odd}(p) \land p = q + 1) \to (p \in Out \leftrightarrow (q \in In \land p \in In)).$$

On the third layer, there are two cases to consider. For wire 1, the output value is simply the input value. For wire $p > 1$, the output value is the disjunction of wires $p-2$ and p prior to the OR gate:

$$\begin{aligned}\text{layer3}(In, Out) = \forall p, q\colon \text{odd}(p) \to [(p = 1 &\to (p \in Out \leftrightarrow p \in In) \land \\ ((p > 1 \land p = q + 2) &\to (p \in Out \leftrightarrow (q \in In \lor p \in In)))].\end{aligned}$$

The fourth layer is the trickiest layer. Observe that the XOR gate outputs 1 if and only if it takes an odd number of 1's as input. Therefore, we define the set

$$O = \{p : p \text{ is a wire and there is an odd number of wires set to 1 among wires } \{1, 3, \ldots, p\}\}$$

and we enforce the last wire to belong to O:

$$\begin{aligned}\text{layer4}(In) = \exists O, r\colon & (\text{last}(r) \land r \in O) \ \land \\ & \forall p, q\colon (\text{odd}(p) \to (p = 1 \to (p \in O \leftrightarrow p \in In)) \land \\ & \qquad ((p > 1 \land p = q + 2) \to (p \in O \leftrightarrow ((q \in O) \oplus (p \in In))))).\end{aligned}$$

Finally, we ensure that the number of input gates is even, and that the circuit outputs 1:

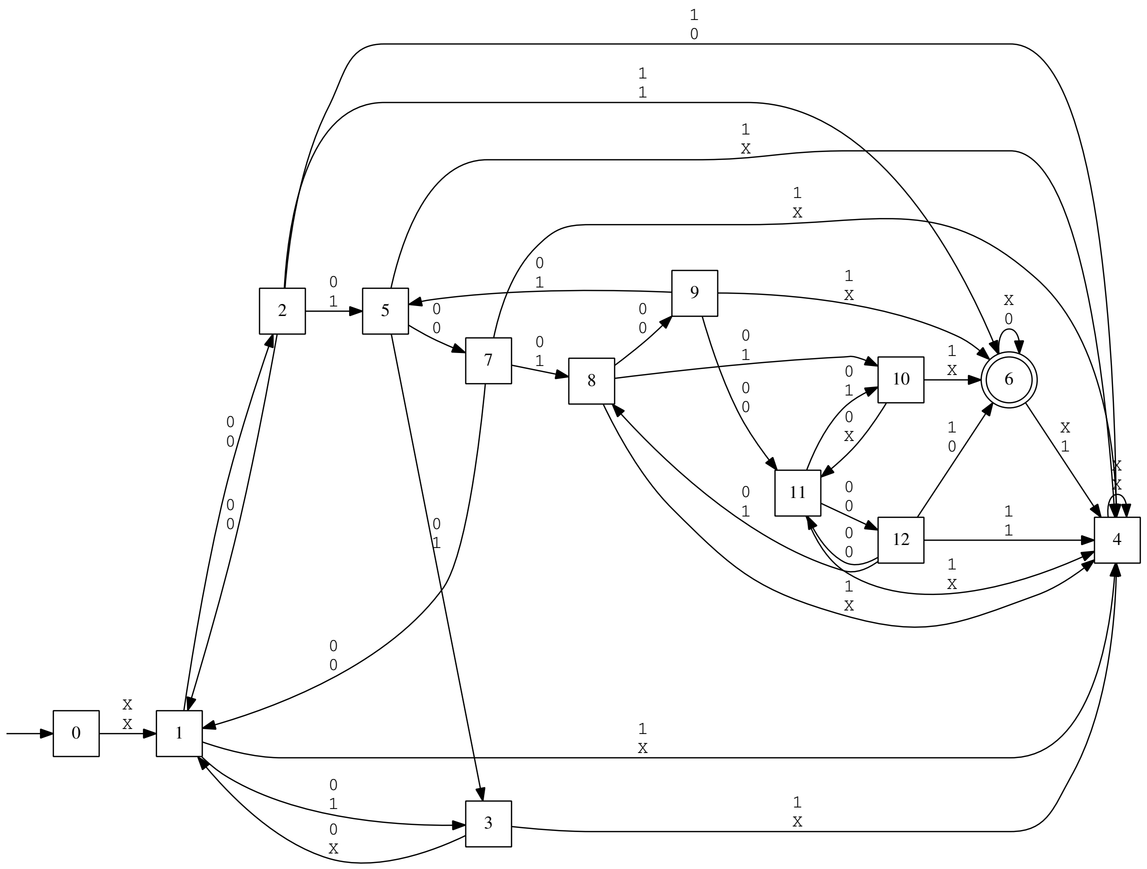
$$\exists p, O_1, O_2, O_3\colon \text{last}(p) \land \text{odd}(p) \land \text{layer1}(O_1) \land \text{layer2}(O_1, O_2) \land \text{layer3}(O_2, O_3) \land \text{layer4}(O_3).$$

(b) The formulas obtained in (a) can be expressed as follows in MONA:

```
var1 last_var;
var2 Bits;
defaultwhere1(p) = p <= last_var;
defaultwhere2(P) = P sub {0,...,last_var};

# Utility predicates
pred Q0(var1 p) = (p notin Bits);
pred Q1(var1 p) = (p in Bits);

pred even(var1 p) =
     ex2 E: (p in E) &
            (all1 q: (q in E) <=> ((q = 0) | (ex1 r: (r in E) & (q = r + 2))));
pred odd(var1 p) = ~even(p);
pred xor(var0 p, var0 q) = ((~p) <=> q);

# Circuit layers
pred layer1(var2 Out) =
     all1 p:
          ((even(p) => ((p in Out) <=> Q0(p))) &
           (odd(p)  => ((p in Out) <=> Q1(p))));

pred layer2(var2 In, var2 Out) =
     all1 p:
          odd(p) => ((p in Out) <=> ((p-1 in In) & (p in In)));
```

```
pred layer3(var2 In, var2 Out) =
     all1 p:
          odd(p) =>
                 (((p = 1) => ((p in Out) <=> (p in In))) &
                  ((p > 1) => ((p in Out) <=> (p-2 in In | p in In))));

pred layer4(var2 In) =
     ex2 O: (last_var in O) &
         (all1 p: odd(p) =>
               (((p = 1) => ((p in O) <=> (p in In))) &
                ((p > 1) => ((p in O) <=> xor(p-2 in O, p in In)))));

# Assertions
assert(odd(last_var));
assert(ex2 O1, O2, O3:
           layer1(O1) & layer2(O1, O2) & layer3(O2, O3) & layer4(O3));
```

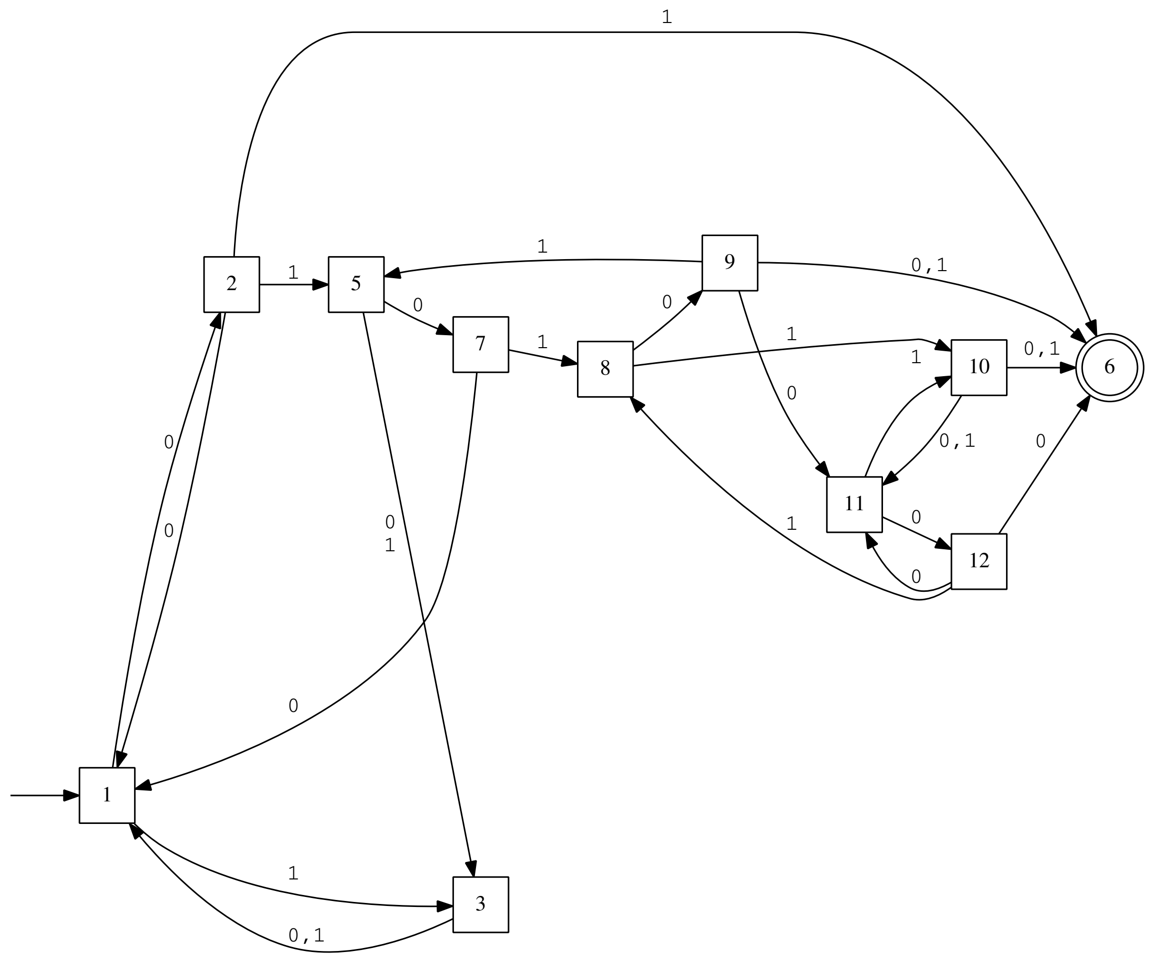
By executing: `mona -gw solution.mona > solution.dot && dot -Tps solution.dot -o solution.ps`, `MONA` yields the following automaton:



The above automaton is not exactly the one we are looking for:

- Symbol `X` stands for "either 0 or 1", so we expand each `X` to the explicit letters.
- We discard state 4 which is a trap state.
- The transition from state 0 to state 1 represents position -1 which is present in `MONA` for technical reasons. Thus, we discard state 0 and make state 1 initial.
- The letters' first component represents the length of the word. Thus, we should stop reading a word upon reading the first 1 in the first component. We must then remove the self-loop on state 6. Finally, we project the letters on their second component.

Overall, we obtain the following automaton:


1
2
1
5
1
9
0,1
0
0
7
1
8
1
1
10
0,1
6
0
0
0
0
1
0,1
0
11
0
1
12
0
0
1
1
3
0,1