Automata and Formal Languages — Exercise Sheet 7

Exercise 7.1
Let $\mathrm{val} : \{0,1\}^* \to \mathbb{N}$ be the function that associates to every word $w \in \{0,1\}^*$ the number $\mathrm{val}(w)$ represented by w in the *least significant bit first* encoding.

(a) Give a transducer that doubles numbers, i.e. a transducer accepting

$$L_1 = \{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \mathrm{val}(y) = 2 \cdot \mathrm{val}(x)\}.$$

(b) Give an algorithm that takes $k \in \mathbb{N}$ as input, and that produces a transducer A_k accepting

$$L_k = \left\{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \mathrm{val}(y) = 2^k \cdot \mathrm{val}(x)\right\}.$$

Hint: use (a) and consider operations seen in class.

(c) Give a transducer for the addition of two numbers, i.e. a transducer accepting

$$\{[x,y,z] \in (\{0,1\} \times \{0,1\} \times \{0,1\})^* \mid \mathrm{val}(z) = \mathrm{val}(x) + \mathrm{val}(y)\}.$$

(d) Show that the following language has infinitely many residuals, and hence that it is not regular:

$$\left\{[x,y] \in (\{0,1\} \times \{0,1\})^* \mid \mathrm{val}(y) = \mathrm{val}(x)^2\right\}.$$

Exercise 7.2
As we have seen, the application of the **Post** and **Pre** operations to transducers requires to compute the padding closure in order to guarantee that the resulting automaton accepts either all or none of the encodings of an object. The padding closure has been defined for encodings where padding occurs *on the right*, i.e., w belongs to the padding closure of an NFA A iff $w\#^k \in L(A)$ for some $k \in \mathbb{N}$. However, in some natural encodings, like the *most-significant-bit-first* encoding of natural numbers, padding occurs *on the left*. Give an algorithm for computing the padding closure of an NFA when padding occurs on the left, i.e. where we consider $\#^k w$.

Exercise 7.3
Let $L_1 = \{baa, aaa, bab\}$ and $L_2 = \{baa, aab\}$.

(a) Give an algorithm for the following operation:

INPUT:	A fixed-length language $L \subseteq \Sigma^k$ described explicitly as a set of words.
OUTPUT:	State q of the master automaton over Σ such that $L(q) = L$.

(b) Use the previous algorithm to build the states of the master automaton for L_1 and L_2.

(c) Compute the state of the master automaton representing $L_1 \cup L_2$.

(d) Identify the kernels $\langle L_1 \rangle$, $\langle L_2 \rangle$, and $\langle L_1 \cup L_2 \rangle$.

Exercise 7.4

We define the *language* of a Boolean formula φ over variables $x_1, \ldots, x_n$ as:

$$L(\varphi) = \{a_1 a_2 \cdots a_n \in \{0,1\}^n : \text{ the assignment } x_1 \mapsto a_1, \ldots, x_n \mapsto a_n \text{ satisfies } \varphi\}.$$

(a) Give a polynomial-time algorithm that takes as input a DFA A recognizing a language of length n, and returns a Boolean formula φ such that $L(\varphi) = L(A)$.

(b) Give an exponential-time algorithm that takes a Boolean formula φ as input, and returns a DFA A recognizing $L(\varphi)$.

Solution 7.1

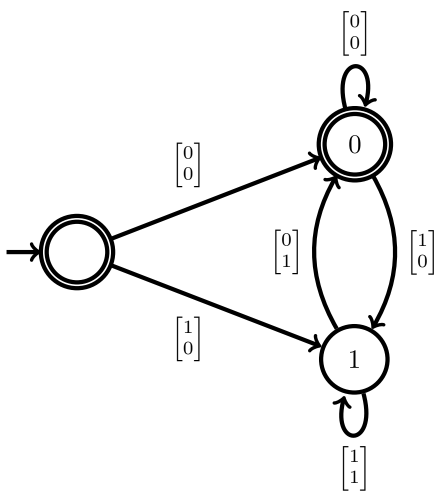
(a) Let $[x_1 x_2 \cdots x_n, y_1 y_2 \cdots y_n] \in (\{0,1\} \times \{0,1\})^n$ where $n \geq 2$. Multiplying a binary number by two shifts its bits and adds a zero. For example, the word

$$\begin{bmatrix}10110\\01011\end{bmatrix}$$

belongs to the language since it encodes $[13, 26]$. Thus, we have $\text{val}(y) = 2 \cdot \text{val}(x)$ if and only if $y_1 = 0$, $x_n = 0$, and $y_i = x_{i-1}$ for every $1 < i \leq n$. From this observation, we construct a transducer that

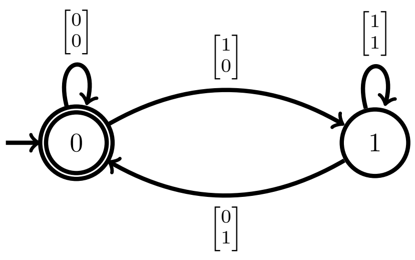
- tests whether the first bit of y is 0,
- tests whether y is consistent with x, by keeping the last bit of x in memory,
- accepts $[x, y]$ if the last bit of x is 0.

Note that words $[\varepsilon, \varepsilon]$ and $[0, 0]$ both encode the numerical values $[0, 0]$. Therefore, they should also be accepted since $2 \cdot 0 = 0$. We obtain the following transducer:



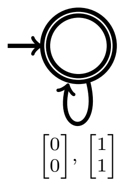
$\begin{bmatrix}0\\0\end{bmatrix}$
0
$\begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
1
$\begin{bmatrix}1\\1\end{bmatrix}$

★ The initial state can be merged with state 0 as they have the same outgoing transitions.



$\begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
0
1
$\begin{bmatrix}0\\1\end{bmatrix}$

(b) We construct A_0 as the following transducer accepting $\{[x, y] \in (\{0,1\} \times \{0,1\})^* : y = x\}$:



$\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}$

Let A_1 be the transducer obtained in (a). For every $k > 1$, we define $A_k = \mathit{Join}(A_{k-1}, A_1)$. A simple inductions show that $L(A_k) = L_k$ for every $k \in \mathbb{N}$.

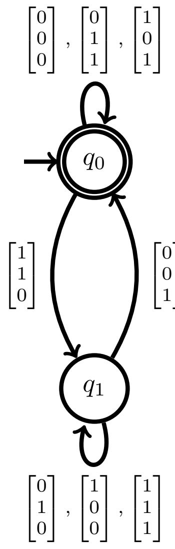
(c) We construct a transducer that computes the addition by keeping the current carry bit. Consider some tuple $[x, y, z] \in \{0,1\}^3$ and a carry bit r. Adding x, y and r leads to the bit

$$z = (x + y + r) \bmod 2. \tag{1}$$

Moreover, it yields a new carry bit r' such that $r' = 1$ if $x+y+r > 1$ and $r' = 0$ otherwise. The folllowing table identifies the new carry bit r' of the tuples that satisfy (1):

	$\begin{bmatrix}0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\1\end{bmatrix}$	$\begin{bmatrix}1\\0\\0\end{bmatrix}$	$\begin{bmatrix}1\\0\\1\end{bmatrix}$	$\begin{bmatrix}1\\1\\0\end{bmatrix}$	$\begin{bmatrix}1\\1\\1\end{bmatrix}$
$r = 0$	0	x	x	0	x	0	1	x
$r = 1$	x	0	1	x	1	x	x	1

We construct our transducer from the above table:



$\begin{bmatrix}0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\1\end{bmatrix}, \begin{bmatrix}1\\0\\1\end{bmatrix}$
q_0
$\begin{bmatrix}1\\1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\\1\end{bmatrix}$
q_1
$\begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\1\\1\end{bmatrix}$

(d) For every $n \in \mathbb{N}_{>0}$, let

$$u_n = \begin{bmatrix}0^n1\\0^n0\end{bmatrix} \text{ and } v_n = \begin{bmatrix}0^{n-1}0\\0^{n-1}1\end{bmatrix}.$$

Let $i, j \in \mathbb{N}_{>0}$ be such that $i \neq j$. We claim that $L^{u_i} \neq L^{u_j}$. We have

$$u_iv_i = \begin{bmatrix}0^i10^i\\0^{2i}1\end{bmatrix} \text{ and } u_jv_i = \begin{bmatrix}0^j10^i\\0^{i+j}1\end{bmatrix}.$$

Therefore, u_iv_i encodes $[2^i, 2^{2i}]$, and u_jv_i encodes $[2^j, 2^{i+j}]$. We observe that u_iv_i belongs to the language since $2^{2i} = (2^i)^2$. However, u_jv_i does not belong to the language since $2^{i+j} \neq 2^{2j} = (2^j)^2$. Thus $v_i \in L^{u_i}$ and $v_i \notin L^{u_j}$. This shows that $L^{u_i} \neq L^{u_j}$ for any $i, j \in \mathbb{N}_{>0}$ such that $i \neq j$. There are an infinite number of such pairs, and thus an infinite number of residuals. □

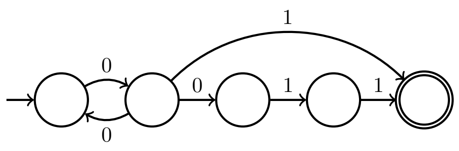
Solution 7.2

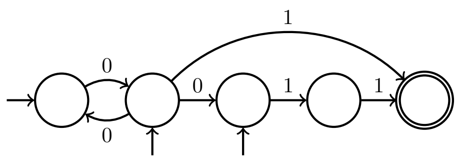
Instead of enlarging the set of final states as done by *PadClosure*, we symmetrically enlarge the set of initial states Q_0 to the the set

$$Q_0' = \{q : q_0 \xrightarrow{0^n} q \text{ for some } q_0 \in Q_0, n \in \mathbb{N}\}.$$

This modification yields the following algorithm:

For example, the NFA depicted below on the left recognizes the set of numbers $\{1, 3\}$ under MSBF encodings $(\# = 0)$. Its padding closure, whichs recognizes the same set, is depicted on the right:





```
Input: NFA $A = (\Sigma \times \Sigma, Q, \delta, Q_0, F)$
Output: new set $Q'_0$ of initial states
PadClosure'($A, \#$):
    $W \leftarrow Q_0$; $Q'_0 \leftarrow \emptyset$;
    while $W \neq \emptyset$ do
        pick $q$ from $W$
        add $q$ to $Q'_0$
        forall $(q, \#, q') \in \delta$ do
            if $q' \notin Q'_0$ then
                add $q'$ to $W$
    return $Q'_0$
```

In the lecture notes, the padding closure of an NFA A is defined as an NFA A' that accepts a word w if and only if the first NFA accepts $w\#^n$ for some $n \geq 0$ and padding symbol #. With this definition, if A accepts $a\#$ but not $a\#\#$, then A' will accept a and $a\#$ but it also will not accept $a\#\#$. However, it makes sense to want our padding closure to accept *all* encodings of a word. To do this, we replace the last line by the following

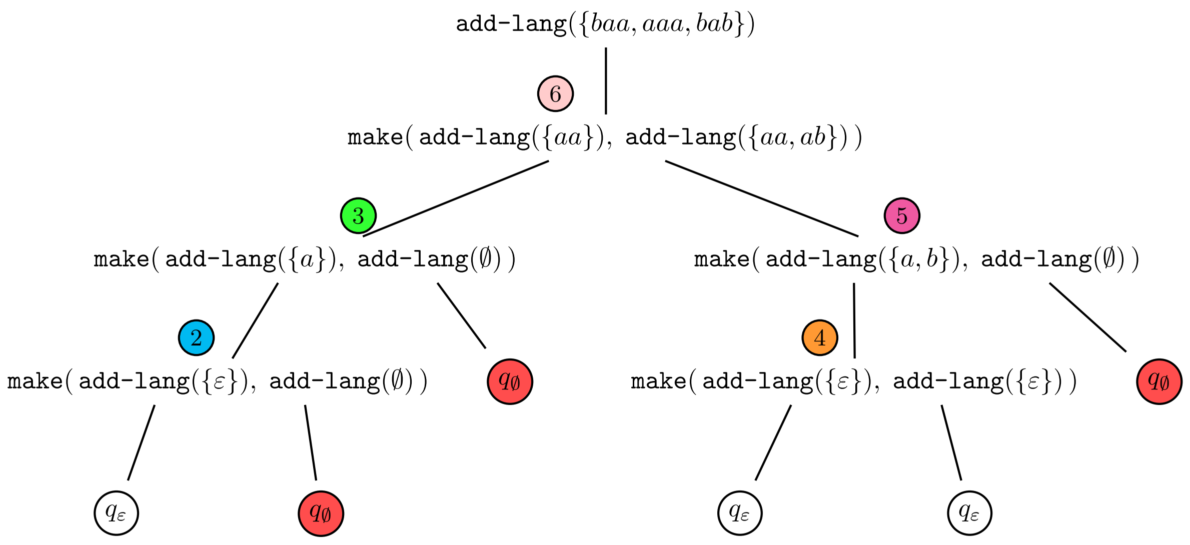
```
forall $q \in Q'_0$ do
    add $(q, \#, q)$ to $\delta'$
return $Q'_0, \delta'$
```

Solution 7.3

(a)

```
Input: A fixed-length language $L \subseteq \Sigma^k$ described explicitely by a set of words.
Output: State $q$ of the master automaton over $\Sigma$ such that $L(q) = L$.
add-lang($L$):
    if $L = \emptyset$ then
        return $q_\emptyset$
    else if $L = \{\varepsilon\}$ then
        return $q_\varepsilon$
    else
        for $a_i \in \Sigma$ do
            $L^{a_i} \leftarrow \{u \mid a_i u \in L\}$
            $s_i \leftarrow$ add-lang($L^{a_i}$)
        return make($s_1, s_2, ..., s_n$)
```

(b) Executing add-lang(L_1) yields the following computation tree:



add-lang({baa, aaa, bab})
6
make(add-lang({aa}), add-lang({aa, ab}))
3
make(add-lang({a}), add-lang(∅))
5
make(add-lang({a, b}), add-lang(∅))
2
make(add-lang({ε}), add-lang(∅))
q∅
4
make(add-lang({ε}), add-lang({ε}))
q∅
qε
q∅
qε
qε

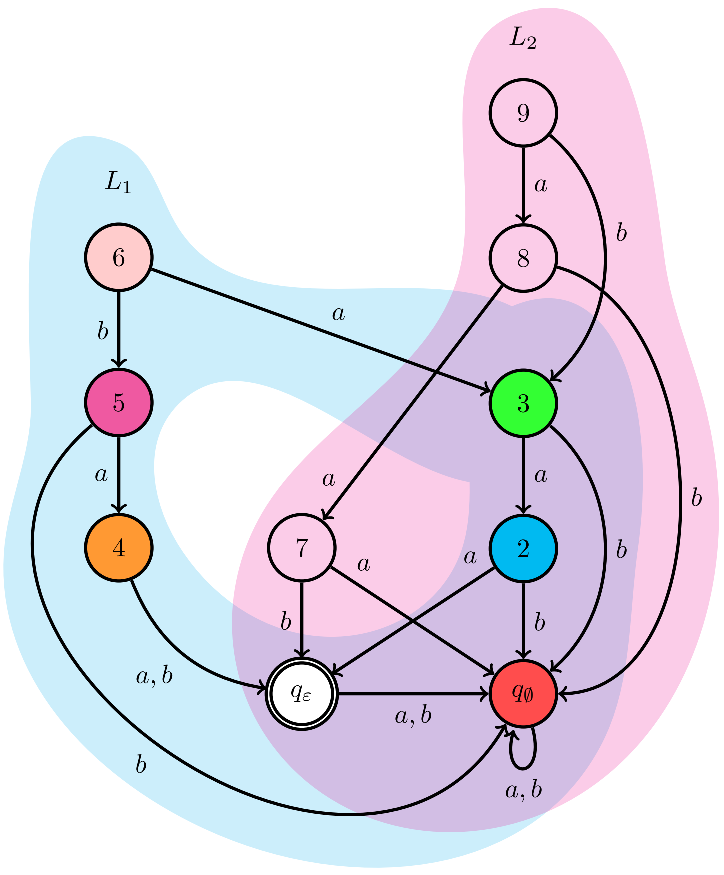
The table obtained after the execution is as follows:

Ident.	a-succ	b-succ
2	q_ε	$q_\emptyset$
3	2	$q_\emptyset$
4	q_ε	q_ε
5	4	$q_\emptyset$
6	3	5

Calling `add-lang`(L_2) adds the following rows to the table and returns 9:

Ident.	a-succ	b-succ
7	$q_\emptyset$	q_ε
8	7	$q_\emptyset$
9	8	3

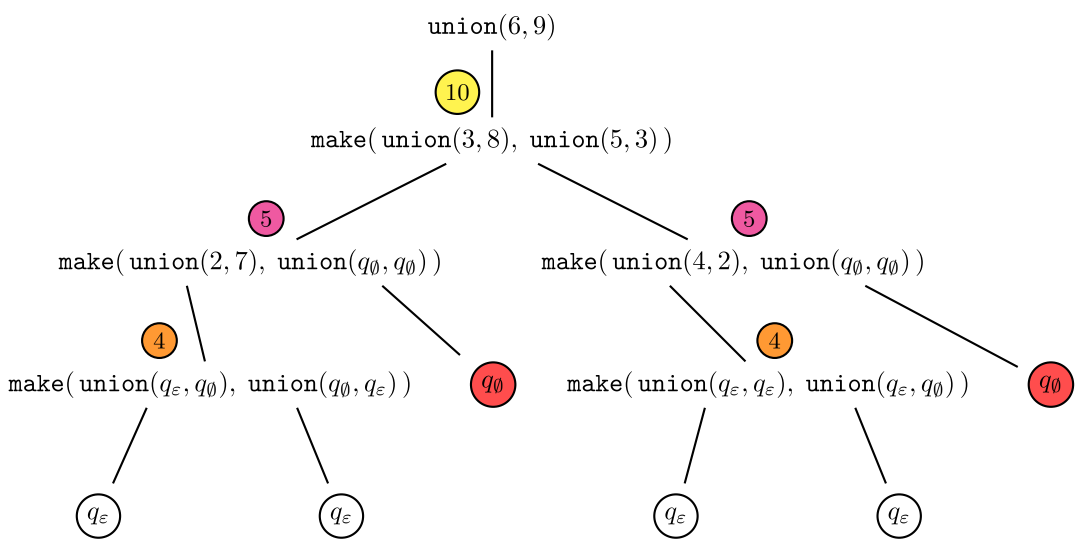
The resulting master automaton fragment is:



(c) Let us first adapt the algorithm for intersection to obtain an algorithm for union:

```
Input: States $p$ and $q$ of same length of the master automaton.
Output: State $r$ of the master automaton such that $L(r) = L(p) \cup L(q)$.
1  union($p, q$) :
2      if $G(p, q)$ is not empty then
3          return $G(p, q)$
4      else if $p = q_\emptyset$ and $q = q_\emptyset$ then
5          return $q_\emptyset$
6      else if $p = q_\varepsilon$ or $q = q_\varepsilon$ then
7          return $q_\varepsilon$
8      else
9          for $a_i \in \Sigma$ do
10             $s_i \leftarrow$ union($p^{a_i}, q^{a_i}$)
11         $G(p, q) \leftarrow$ make($s_1, s_2, \ldots, s_n$)
12         return $G(p, q)$
```

Executing `union`$(6, 9)$ yields the following computation tree:

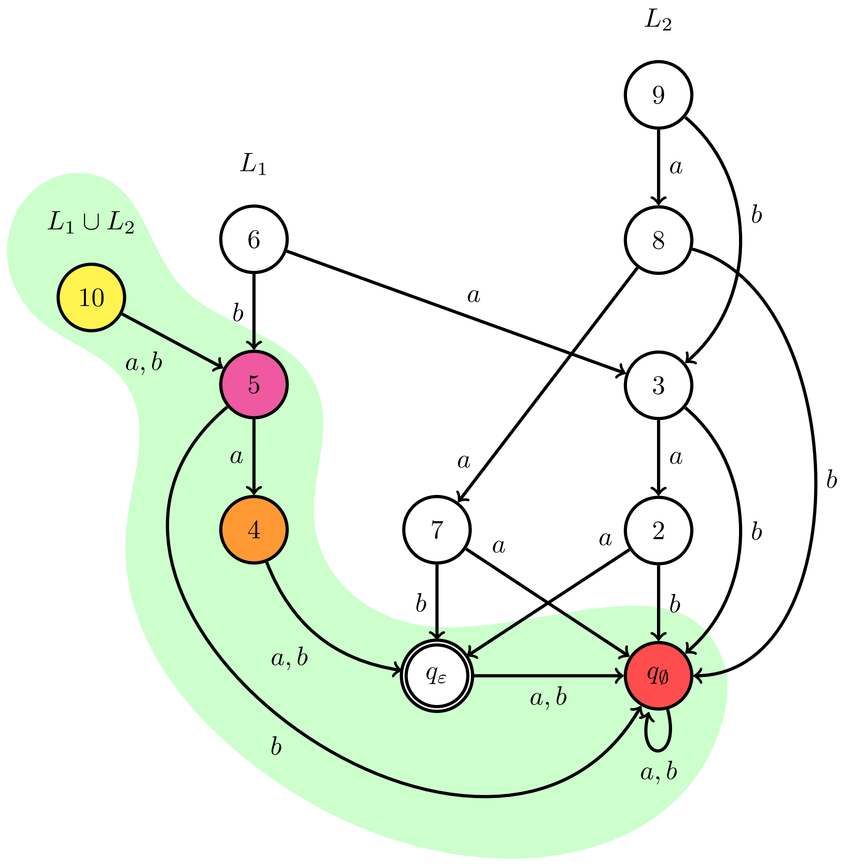


union(6, 9)
10
make(union(3, 8), union(5, 3))
5
make(union(2, 7), union($q_\emptyset$, $q_\emptyset$))
5
make(union(4, 2), union($q_\emptyset$, $q_\emptyset$))
4
make(union(q_ε, $q_\emptyset$), union($q_\emptyset$, q_ε))
$q_\emptyset$
4
make(union(q_ε, q_ε), union(q_ε, $q_\emptyset$))
$q_\emptyset$
q_ε
q_ε
q_ε
q_ε

Calling `union`$(6, 9)$ adds the following row to the table and returns 10:

Ident.	a-succ	b-succ
10	5	5

The new fragment of the master automaton is:



★ Note that `union` could be slightly improved by returning q whenever $p = q$, and by updating $G(q, p)$ at the same time as $G(p, q)$.

(d) The kernels are:

$$\langle L_1 \rangle = L_1,$$
$$\langle L_2 \rangle = L_2,$$
$$\langle L_1 \cup L_2 \rangle = \{aa, ab\}.$$

Solution 7.4

(a) The algorithm takes as input a state of the master automaton and the length of the language it recognizes, and recursively constructs a formula as follows:

```
   Input: state $q$ recognizing a language of length $n$
   Output: formula $\varphi_q$ such that $L(\varphi_q) = L(q)$
 1 DFAtoFormula($q, n$):
 2   if $G(q)$ is not empty then
 3     return $G(q)$
 4   if $q = q_\emptyset$ then
 5     return false
 6   else if $q = q_\epsilon$ then
 7     return true
 8   else
 9     $\varphi_0 \leftarrow DFAtoFormula(q^0, n-1)$
10     $\varphi_1 \leftarrow DFAtoFormula(q^1, n-1)$
11     $\varphi_q \leftarrow (\neg x_n \land \varphi_0) \lor (x_n \land \varphi_1)$
12     $G(q) \leftarrow \varphi_q$
13     return $G(q)$
```

Observe that the parameter n is needed to identify the variable at line 11.

Our algorithm takes as input a table with the state identifiers and successors of all the descendants of q (i.e. the fragment of the master automaton starting in q). This is a polynomial time algorithm because we compute $\varphi_{q'}$ once for every q' descendant of q.

As remarked in the tutorial, this algorithm could be improved by adding an *else* that checks if $q^0 = q^1$ before the last else:

```
1 else if $q^0 = q^1$ then
2   $\varphi \leftarrow DFAtoFormula(q^0, n-1)$
3   $\varphi_q \leftarrow \varphi$
4   $G(q) \leftarrow \varphi_q$
5   return $G(q)$
```

(b) Given a formula φ over variables $x_1, \dots, x_n$, we write $\varphi[x_i/\mathbf{true}]$ and $\varphi[x_i/\mathbf{false}]$ to denote the formulas obtained by replacing all occurrences of x_i in φ by **true** and **false**, respectively. We have that $L(\varphi[x_1/\mathbf{false}]) = L(\varphi)^0$ and $L(\varphi[x_1/\mathbf{true}]) = L(\varphi)^1$. This yields the following algorithm:

```
Input: formula $\varphi$ over variables $x_1, \dots, x_n$, total number of variables $n$, $k$ initially equal to $n$
Output: state $q$ such that $L(\varphi) = L(q)$
FormulatoDFA($\varphi, n, k$):
    if $G(\varphi)$ is not empty then
        return $G(\varphi)$
    if $\varphi = \mathbf{true}$ then
        return $q_\varepsilon$
    else if $\varphi = \mathbf{false}$ then
        return $q_\emptyset$
    else
        $r_0 \leftarrow FormulatoDFA(\varphi[x_{n-k+1}/\mathbf{false}], n, k-1)$
        $r_1 \leftarrow FormulatoDFA(\varphi[x_{n-k+1}/\mathbf{true}], n, k-1)$
        $G(\varphi) \leftarrow make(r_0, r_1)$
        return $G(\varphi)$
```