Technische Universität München **Winter term 2016/17**
I7
Prof. J. Esparza / Dr. M. Blondin **11.12.2016**

Automata and Formal Languages — Homework 8

Due 16.12.2016

Exercise 8.1

Consider two processes (process 0 and process 1) being executed through the following generic mutual exclusion algorithm:

```
while true do
    enter(process_id)
    /* critical section                                   */
    leave(process_id)
    for arbitrarily many times do
        /* non critical section                           */
```

(a) Consider the following implementations of **enter** and **leave**:

```
x ← 0

enter(i):
    while x = 1 − i do
        pass

leave(i):
    x ← 1 − i
```

(i) Design a network of automata capturing the executions of the two processes.

(ii) Build the asynchronous product of the network.

(iii) Show that both processes cannot reach their critical sections at the same time.

(iv) If a process wants to enter its critical section, is it always the case that it can eventually enter it? (Hint: reason in terms of infinite executions.)

(b) Consider the following alternative implementations of `enter` and `leave`:

$x_0 \leftarrow \mathit{false}$
$x_1 \leftarrow \mathit{false}$

enter(i):
  $x_i \leftarrow \mathit{true}$
  while x_{1-i} **do**
    pass

leave(i):
  $x_i \leftarrow \mathit{false}$

(i) Design a network of automata capturing the executions of the two processes.

(ii) Can a deadlock occur, i.e. can both processes get stuck trying to enter their critical sections?

Exercise 8.2
Let Σ be a finite alphabet. A language $L \subseteq \Sigma^*$ is *star-free* if it can be expressed by a star-free regular expression, i.e. a regular expression where Kleene star is forbidden, but complementation is allowed. For example, Σ^* is star-free since $\Sigma^* = \overline{\emptyset}$, but $(aa)^*$ is not.

(a) Give star-free regular expressions and FO(Σ) sentences for the following star-free languages:

(i) Σ^+.

(ii) $\Sigma^* A \Sigma^*$ for some $A \subseteq \Sigma$.

(iii) A^* for some $A \subseteq \Sigma$.

(iv) $(ab)^*$.

(v) $\{w \in \Sigma^* : w \text{ does not contain two consecutive } a\}$.

(b) Show that finite and cofinite languages are star-free.

(c) Show that for every sentence $\varphi \in \mathrm{FO}(\Sigma)$, there exists a formula φ^+, with two free variables, such that for every $w \in \Sigma^+$ and $1 \leq i \leq j \leq w$,

$$w \models \varphi^+(i, j) \iff w_i w_{i+1} \cdots w_j \models \varphi \ .$$

(d) Give a polynomial time algorithm that tests whether $\varepsilon \models \varphi$ given some sentence $\varphi \in \mathrm{FO}(\Sigma)$.

(e) Show that every star-free language can be expressed by an FO(Σ) sentence. (Hint: use (c).)

Exercise 8.3
Let $\Sigma = \{a, b\}$.

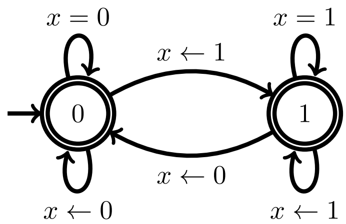
(a) Give an FO(Σ) formula $\varphi_n(x, y)$ of size $O(n)$ such that $\varphi_n(x, y)$ holds $\iff$ $y = x + 2^n$.

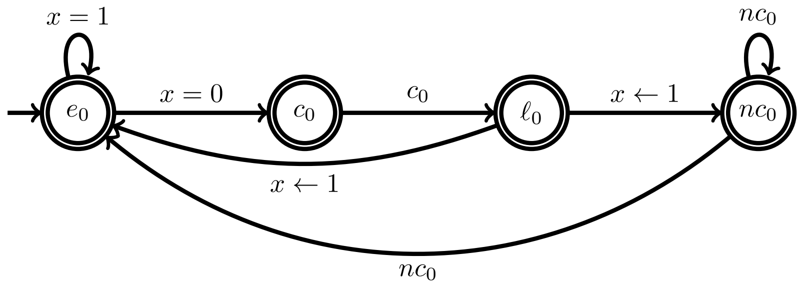
(b) Give an FO(Σ) sentence of size $O(n)$ for $L_n = \{ww : w \in \Sigma^* \text{ and } |w| = 2^n\}$.

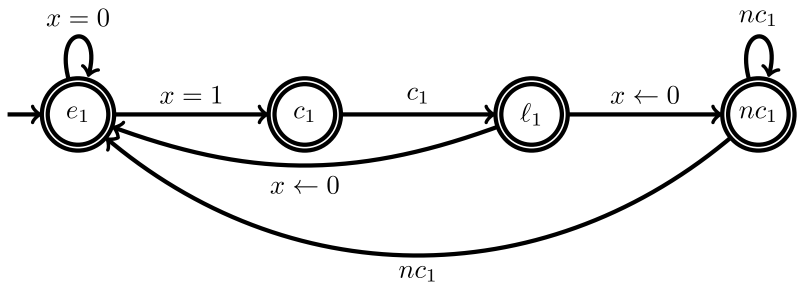
(c) Show that the minimal DFA accepting L_n has at least 2^{2^n} states. (Hint: consider the residuals of L_n.)

Solution 8.1

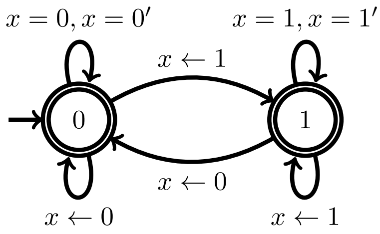
(a) (i)

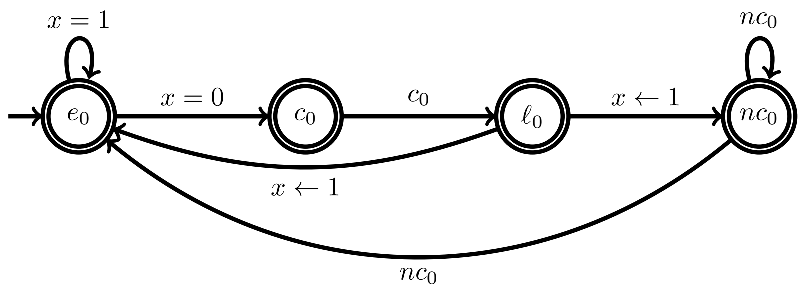


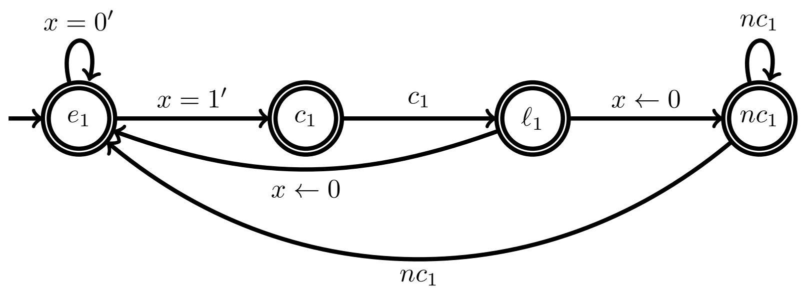




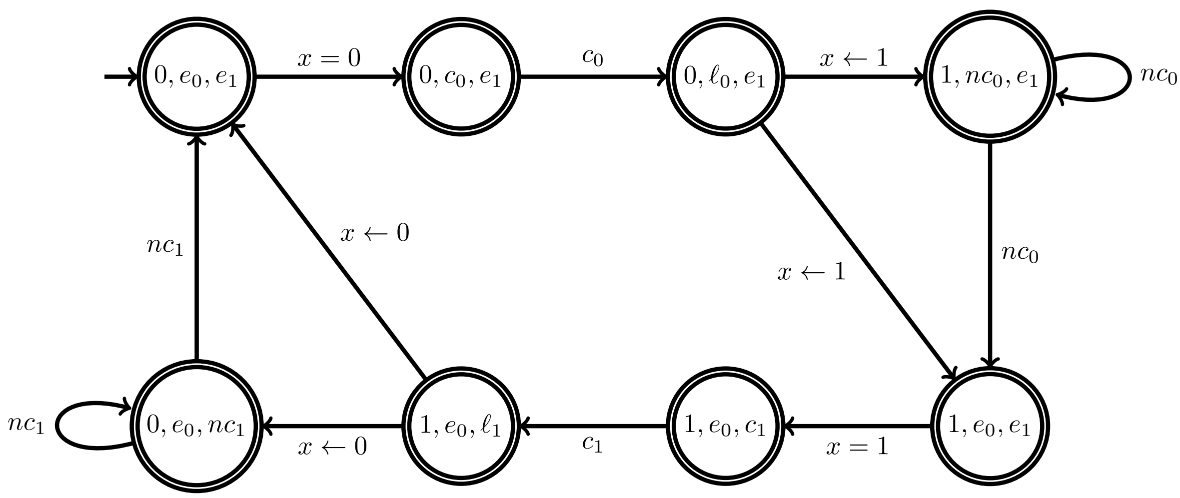
★ As discussed in class, the previous network forces the two processes to read the content of x at the same time. If we want to avoid this, we can add new disjoint actions $x = 0'$ and $x = 1'$ as follows:



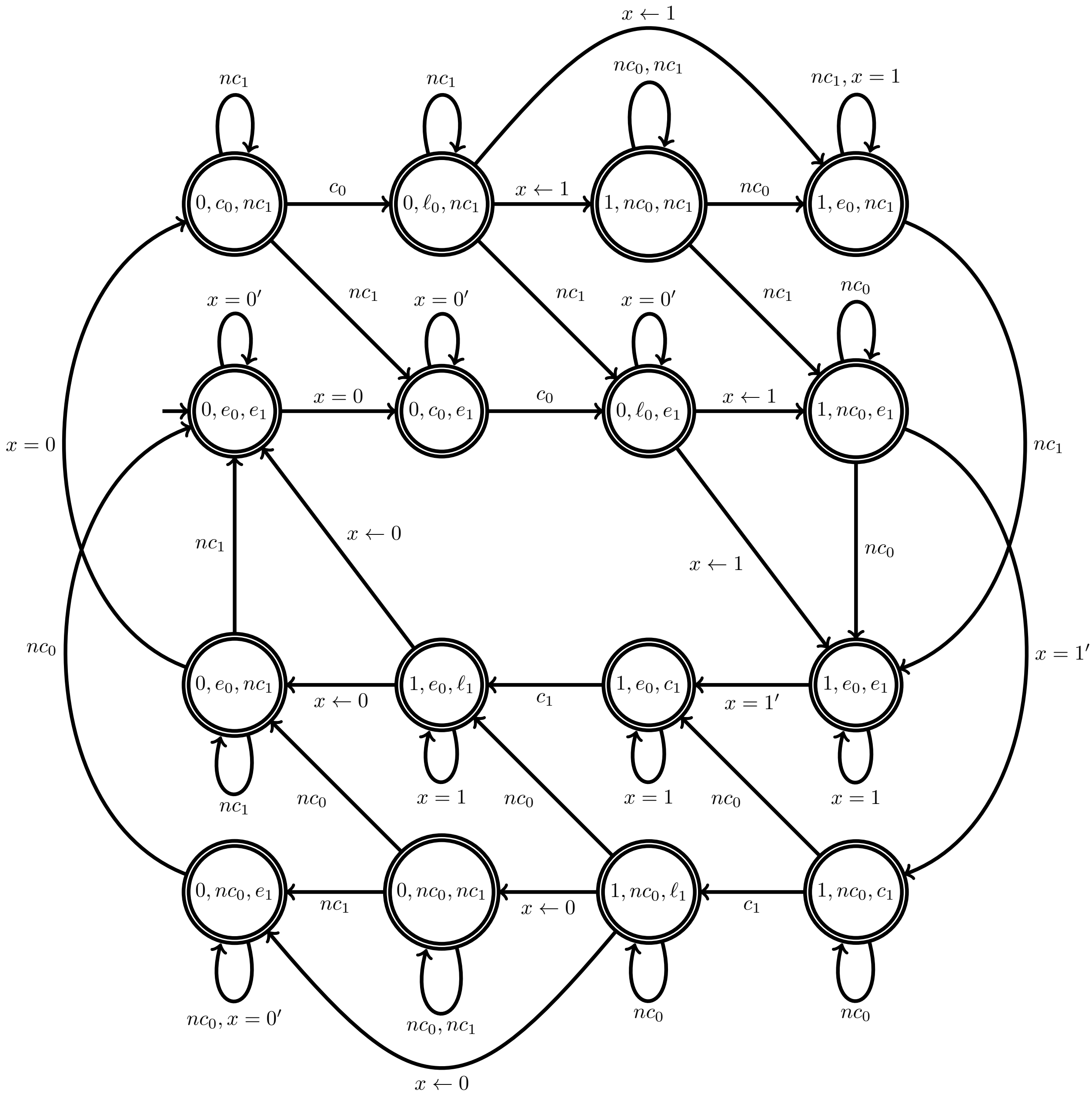




(ii)



★ For the second solution where asynchronous reading is allowed, we obtain the following automaton:



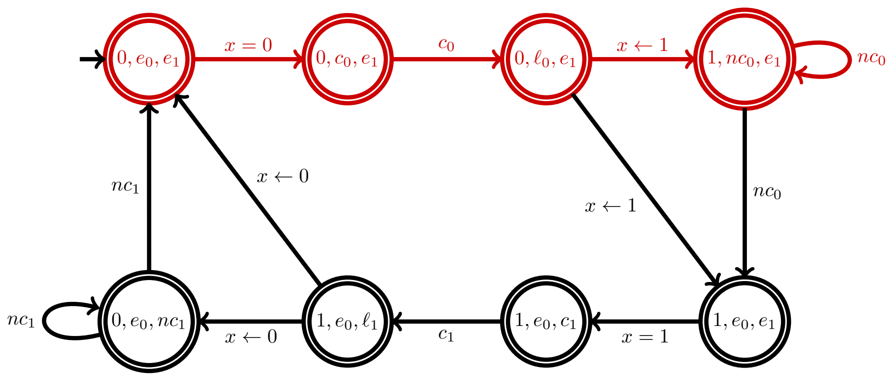
(iii) Both processes can reach their critical section at the same time if, and only if, the asynchronous product contains a state of the form (x, c_0, c_1). Since it contains none, this behaviour cannot occur.

★ It also cannot occur in our second modeling.

(iv) No. Consider the following infinite run:

$$(0, e_0, e_1) \xrightarrow{x=0} (0, c_0, e_1) \xrightarrow{c_0} (0, \ell_0, e_1) \xrightarrow{x \leftarrow 1} (1, nc_0, e_1) \xrightarrow{nc_0} (1, nc_0, e_1) \xrightarrow{nc_0} \cdots$$

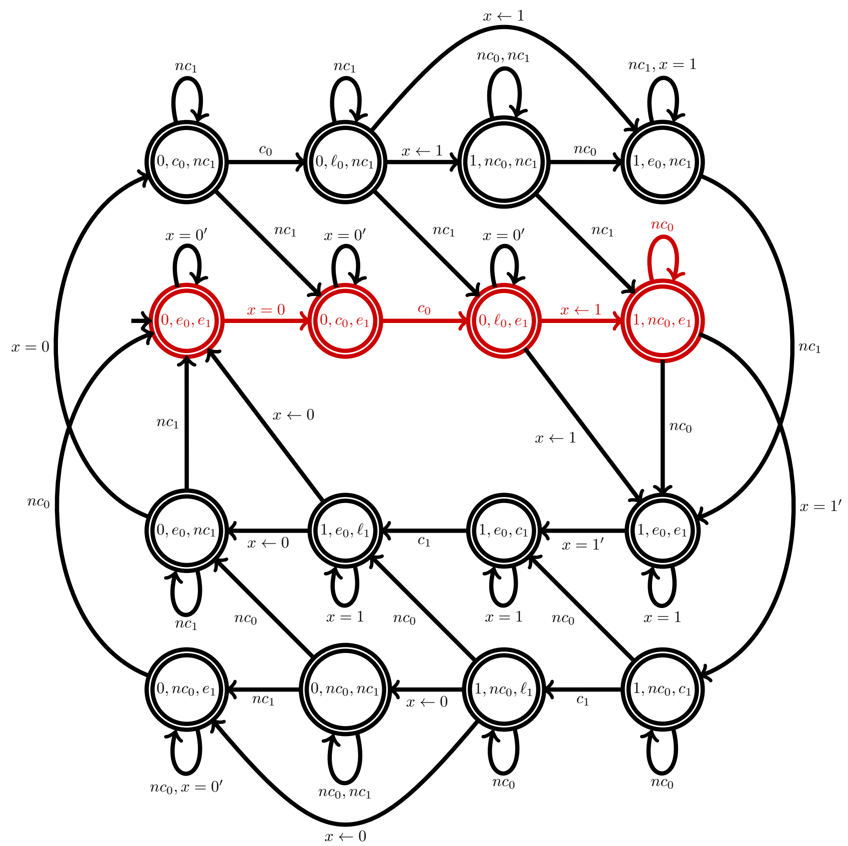
illustrated in red:



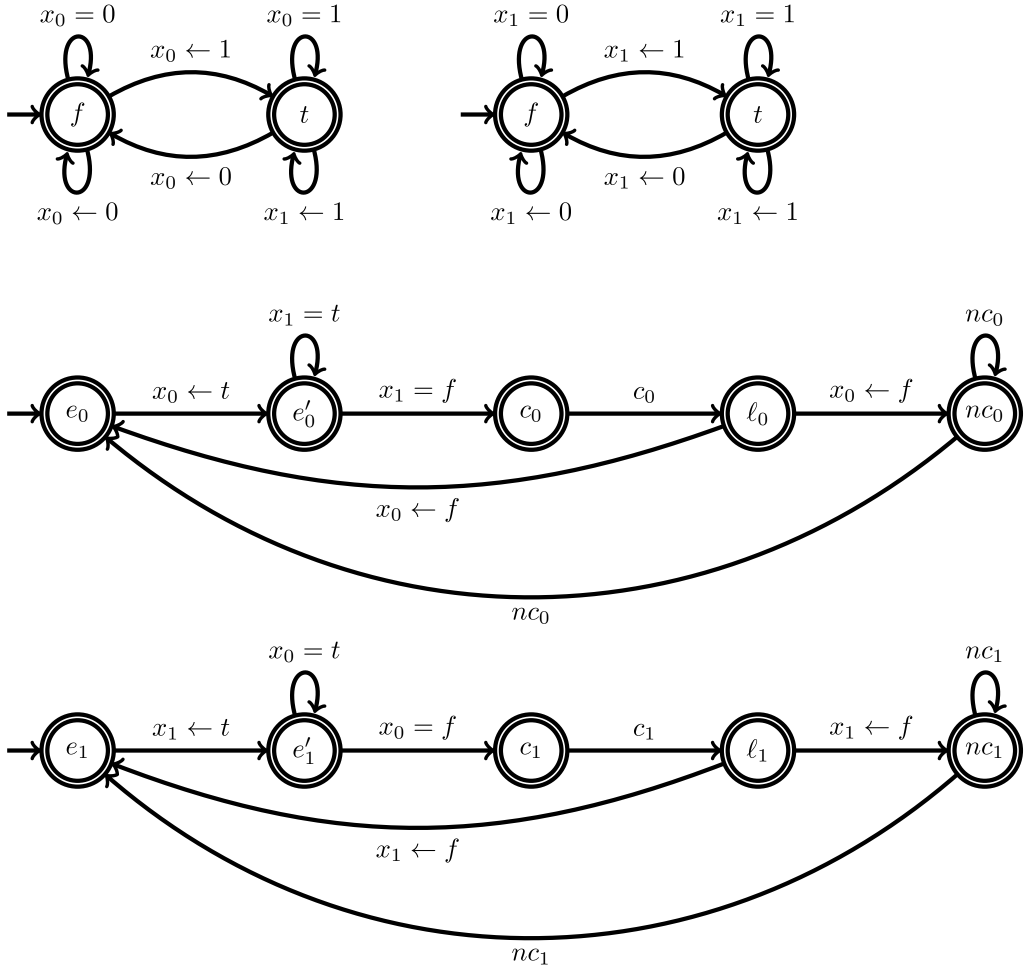
The second process remains in e_1 throughout this infinite run, so it never enters its critical section. Since we have restricted x to be read at the same time, a process can stay in its non critical section as long as it wants while the other one cannot do anything.

★ In our second modeling, this infinite run still occurs as illustrated below.

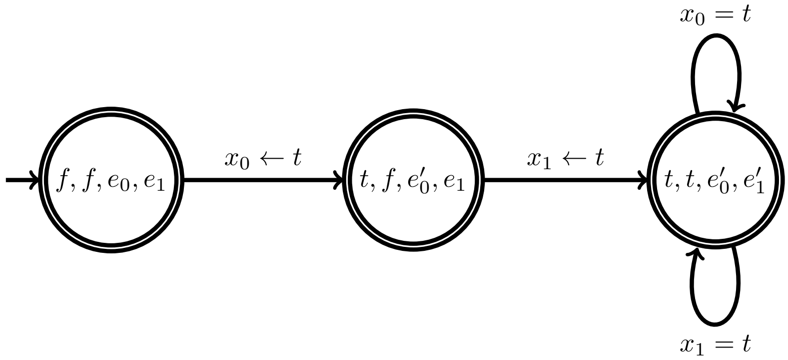
However, here the second process is not stuck since it could take transition $(1, nc_0, e_1) \xrightarrow{x=1'} (1, nc_0, c_1)$ to reach its critical section. Therefore, the red infinite run only occurs if the process scheduler can let a process i run forever even though process $1-i$ could make progress.



(b) (i)


$x_0 = 0$
$x_0 = 1$
$x_0 \leftarrow 1$
f
t
$x_0 \leftarrow 0$
$x_0 \leftarrow 0$
$x_1 \leftarrow 1$
$x_1 = 0$
$x_1 = 1$
$x_1 \leftarrow 1$
f
t
$x_1 \leftarrow 0$
$x_1 \leftarrow 0$
$x_1 \leftarrow 1$
$x_1 = t$
nc_0
e_0
$x_0 \leftarrow t$
e_0'
$x_1 = f$
c_0
c_0
ℓ_0
$x_0 \leftarrow f$
nc_0
$x_0 \leftarrow f$
nc_0
$x_0 = t$
nc_1
e_1
$x_1 \leftarrow t$
e_1'
$x_0 = f$
c_1
c_1
ℓ_1
$x_1 \leftarrow f$
nc_1
$x_1 \leftarrow f$
nc_1

(ii) Yes, consider this fragment of the asynchronous product of the network:



When (t, t, e'_0, e'_1) is reached, both processes are still trying to enter their critical section, and it is impossible to move to a new state.

Solution 8.2

(a) (i) $\overline{\emptyset} \cdot \Sigma$ and $\exists x \ \text{first}(x)$.

(ii) $\overline{\emptyset} \cdot A \cdot \overline{\emptyset}$ and $\exists x \ \bigvee_{a \in A} Q_a(x)$.

(iii) $\overline{\Sigma^* \overline{A} \Sigma^*}$ and $\forall x \ \bigwedge_{a \in A} Q_a(x)$.

(iv) $\overline{b\Sigma^* + \Sigma^* a + \Sigma^* aa \Sigma^* + \Sigma^* bb \Sigma^*}$ and

$$(\neg \exists x \ \text{first}(x)) \vee [(\exists x \ \text{first}(x) \wedge Q_a(x)) \wedge (\exists x \ \text{last}(x) \wedge Q_b(x)) \wedge \\ (\forall x, y \ (Q_a(x) \wedge y = x + 1) \to Q_b(y)) \wedge (\forall x, y \ (Q_b(x) \wedge y = x + 1) \to Q_a(y))] \ .$$

(v) $\overline{\Sigma^* aa \Sigma^*}$ and $\forall x, y \ (Q_a(x) \wedge y = x + 1) \to \neg Q_a(y)$.

(b) Every finite language $L = \{w_1, w_2, \ldots, w_m\}$ can be expressed as $w_1 + w_2 + \cdots + w_m$. For every cofinite language L, there exists a finite language A such that $L = \overline{A}$. Since star-free regular expressions allow for complementation, cofinite languages are also star-free. □

(c) We build φ^+ using the following inductive rules:

$$\begin{aligned}
(x < y)^+(i, j) &= x < y \\
Q_a(x)^+(i, j) &= Q_a(x) \\
(\neg \psi)^+(i, j) &= \neg \psi^+(i, j) \\
(\psi_1 \vee \psi_2)^+(i, j) &= \psi_1^+(i, j) \vee \psi_2^+(i, j) \\
(\exists x \ \psi)^+(i, j) &= \exists x \ (i \leq x \wedge x \leq j) \wedge \psi^+(i, j) \ .
\end{aligned}$$

(d)

Input: sentence $\varphi \in \mathrm{FO}(\Sigma)$.
Output: $\varepsilon \models \varphi$?
`has-empty(`φ`):`
  if $\varphi = \neg\psi$ **then**
    return $\neg$`has-empty(`ψ`)`
  else if $\varphi = \psi_1 \vee \psi_2$ **then**
    return `has-empty(`ψ_1`)` $\vee$ `has-empty(`ψ_2`)`
  else if $\varphi = \exists\ \psi$ **then**
    return *false*

(e)

Input: star-free regular expression r.
Output: sentence $\varphi \in \mathrm{FO}(\Sigma)$ s.t. $L(\varphi) = L(r)$.
`formula(`r`):`
if $r = \varepsilon$ **then**
  return $\forall x\ \mathrm{first}(x)$
else if $r = a$ for some $a \in \Sigma$ **then**
  return $(\exists x \text{ true}) \wedge (\forall x\ \mathrm{first}(x) \wedge Q_a(x))$
else if $r = \overline{s}$ **then**
  return $\neg$`formula(s)`
else if $r = s_1 + s_2$ **then**
  return `formula(`s_1`)` $\vee$ `formula(`s_2`)`
else if $r = s_1 \cdot s_2$ **then**
  return $(\neg \exists x\ \mathrm{first}(x) \wedge (\varepsilon \in L(s_1)) \wedge (\varepsilon \in L(s_2))) \vee$
  (`formula(`s_1`)` $\wedge (\varepsilon \in L(s_2))) \vee$
  $((\varepsilon \in L(s_1)) \wedge$ `formula(`s_2`)`$) \vee$
  $(\exists x, y, y', z\ \mathrm{first}(x) \wedge y' = y + 1 \wedge \mathrm{last}(z) \wedge \texttt{formula(}s_1\texttt{)}^+(x, y) \wedge \texttt{formula(}s_2\texttt{)}^+(y', z))$

Solution 8.3

(a) To simplify the notation, let us write "$y = x + 2^n$" for "$\varphi_n(x, y)$". We can define φ_n inductively as follows:

$$y = x + 2^n := \exists t\ (t = x + 2^{n-1}) \wedge (y = t + 2^{n-1}))\ .$$

However, this yields a formula of exponential size. The formula can be made linear by rewriting it in the following way:

$$\begin{aligned} y = x + 2^n &:= \exists t\ \forall x', y'\ ((x' = x \wedge y' = t) \rightarrow (y' = x' + 2^{n-1})) \wedge ((x' = t \wedge y' = y) \rightarrow (y' = x' + 2^{n-1})) \\ &= \exists t\ \forall x', y'\ (\neg(x' = x \wedge y' = t) \vee (y' = x' + 2^{n-1})) \wedge (\neg(x' = t \wedge y' = y) \vee (y' = x' + 2^{n-1})) \\ &= \exists t\ \forall x', y'\ ((\neg(x' = x \wedge y' = t) \wedge (\neg(x' = t \wedge y' = y)) \vee (y' = x' + 2^{n-1}) \\ &= \exists t\ \forall x', y'\ ((x' = x \wedge y' = t) \vee (x' = t \wedge y' = y)) \rightarrow (y' = x' + 2^{n-1}) \end{aligned}$$

(b)

$$\varphi = \overbrace{[\exists x, y, y', z\ \mathrm{first}(x) \wedge (y = x + 2^n) \wedge (y = y' + 1) \wedge (z = y' + 2^n) \wedge \mathrm{last}(z)]}^{\text{word has length } 2^n + 2^n} \wedge$$
$$\underbrace{[\forall x, y \bigwedge_{\sigma \in \{a,b\}} (Q_\sigma(x) \wedge y = x + 2^n) \rightarrow Q_\sigma(y)]}_{\text{word is of the form } ww}\ .$$

(c) Let $u, v \in \{a, b\}^*$ such that $|u| = |v| = 2^n$ and $u \neq v$. We have $uu \in L_n$ and $uv \notin L_n$. Therefore, all words of length 2^n belong to distinct residuals. There are 2^{2^n} such words, hence L_n has at least 2^{2^n} residuals. □