Problems Solved:

26	27	28	29	30

Name:

Matrikel-Nr.:

Problem 26. Consider the following term rewriting system:

$$\begin{array}{ll} \text{(Rule 1)} & p(x, s(y)) \to p(s(s(x)), y) \\ \text{(Rule 2)} & p(x, 0) \to s(x) \end{array}$$

1. Show that

$$p(0, s(p(0, s(0)))) \xrightarrow{*} s(s(s(s(s(s(s(s(s(0)))))))))$$

by a suitable reduction sequence. For each reduction step, underline the subterm that you reduce, and indicate the reduction rule and the matching substitution σ used explicitly.

2. Prove or disprove (an informal argument suffices)

$$p(0, p(0, p(0, p(0, s(0))))) \xrightarrow{*} \underbrace{s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(s(0)))))))))))))))}_{16\times s}.$$

Solution of Problem 26:

1.

$$\begin{aligned} p(0, s(\underline{p(0, s(0))})) &\overset{(1)}{\to} p(0, s(\underline{p(s(s(0)), 0)})) \\ &\overset{(1)}{\to} \underline{p(0, s(s(s(s(0)))))} \\ &\overset{(1)}{\to} \underline{p(s(s(0)), s(s(s(0))))} \\ &\overset{(1)}{\to} \underline{p(s(s(s(s(0)))), s(s(0)))} \\ &\overset{(1)}{\to} \underline{p(s(s(s(s(s(s(0)))))), s(0))} \\ &\overset{(1)}{\to} \underline{p(s(s(s(s(s(s(s(s(0)))))))), 0)} \\ &\overset{(2)}{\to} s(s(s(s(s(s(s(s(s(0)))))))))) \end{aligned}$$

2. Clearly, any s in the second argument of p leads to 2 s in the first argument. Consider the outermost p. After resolving the inner p by Rule 1 and Rule 2, there will be an even number of s in the first argument of the outermost p and a 0 in the second argument. Applying Rule 2 leads to an odd number of s in the result. That number is, therefore not equal to 16. Informally, p is the function $(x, y) \mapsto x + 2y + 1$. If $x = 0$, then the result cannot be 16.

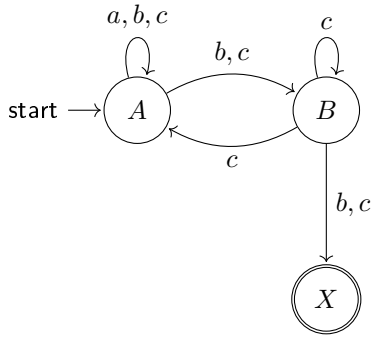
Problem 27. Construct a DFSM recognizing $L(G)$ where $G = (\{A, B\}, \{a, b, c\}, P, A)$ with the production rules P given by

$$A \to aA|bA|cA|bB|cB,$$
$$B \to cA|cB|b|c.$$

Hint: Start by a constructing a NFSM N. Then turn N into a DFSM D such that $L(G) = L(N) = L(D)$.
"Construct" means to explain how you turn the grammar into a DFSM. Simply writing down a DFSM D with the required property, does not count as a solution unless you *prove* that $L(G) = L(D)$.

Solution of Problem 27:

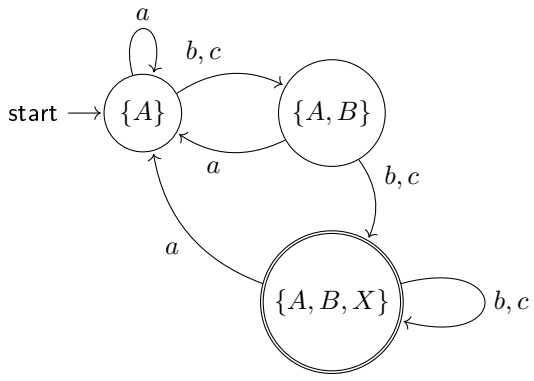
Since this grammar has a particular form, we first turn it into a NFSM N that accepts the same language. The terminal symbols will form the alphabet of N. Every non-terminal symbol of the grammar becomes a state of N. Additionally, we use a state X for the case that there is no non-terminal symbol on the right of a rule. X is also the only accepting state. Let the NDFSM $N = (Q, \Sigma, \nu, A, F)$ where $Q = \{A, B, X\}$, $\Sigma = \{a, b\}$, $F = \{X\}$. The transition function $\delta : Q \times \Sigma \to Q$ can be constructed from looking at the production rules. Every rule of the form $U \to \sigma V$ becomes a transition from state U to state V labelled with the letter σ, i.e., the graph of ν is as follows.



This NFSM N can easily be translated into a DFSM

$$D = (\{\{A\}, \{A, B\}, \{A, B, C\}\}, \{a, b, c\}, \delta, \{A\}, \{\{A, B, X\}\})$$

by the general powerset construction where δ is given as follows.



Problem 28. Let $Q(x) = \{ y \in \mathbb{N} \mid x \leq y^2 \} \subseteq \mathbb{N}$ and $f : \mathbb{N} \to \mathbb{N}$ be the (partial) function

$$f(x) = \begin{cases} \min Q(x) & \text{if } Q(x) \neq \emptyset, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

1. Is f LOOP-computable?
2. Is f a primitive recursive function?
3. Is f a WHILE-computable function?
4. Is f a μ-recursive function?

In each case justify your answer. If it is *yes*, give a corresponding program and/or an explicit definition as a (primitive/μ-) recursive function.
Remark: When defining f, you are allowed to use the Definition 29 and 30 from the lecture notes and the primitive recursive functions (respectively loop programs computing these functions)

$$m : \mathbb{N}^2 \to \mathbb{N}, \quad (x, y) \mapsto x \cdot y$$

$u : \mathbb{N}^2 \to \mathbb{N}$,

$$u(x, y) = \begin{cases} 0 & \text{if } x = y, \\ 1 & \text{if } x \neq y. \end{cases}$$

and $IF : \mathbb{N}^3 \to \mathbb{N}$,

$$IF(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise.} \end{cases}$$

Other functions or rules are forbidden.

Solution of Problem 28:

Of course, the answer is *yes* in all four cases. If you can define a function by mu-recursion, it may be nevertheless also primitive recursive.

1. Clearly $Q(x)$ is non-empty for any $x \in \mathbb{N}$, i.e., f is a total function. It is LOOP-computable, because the following program computes f.
   (input is in x_1, output is in x_0)

```
x0 := 0;            // result
x2 := 0;            // start with y=0
LOOP x1 DO
    x3 := 0;
    LOOP x2 DO LOOP x2 DO x3:=x3+1; END; END; // x3=y^2
    // if x3 < x1 then x0 := x2 + 1; END;
    x4 := x1;
    LOOP x3 DO x4 := x4 - 1; END; // x4 := x1 - x3 -- x-y^2
    LOOP x4 DO x0 := x2 + 1; // if 0 < x4 then x0 := x2 + 1
    x2 := x2 + 1;
END;
```

2. We can compute the first values for $(x, f(x))$ for $n = 0, 1, \ldots 10$. That results in

   $$(0,0), (1,1), (2,2)(3,2), (4,2), (5,3), (6,3), (7,3), (8,3), (9,3), (10,4)$$

   Note that

   $$f(x) = f(x-1) \text{ if } f(x-1)^2 \neq (x-1) \tag{1}$$

   and

   $$f(x) = f(x-1) + 1 \text{ if } f(x-1)^2 = (x-1). \tag{2}$$

   Unfortunately, the condition is not of the form $y = 0$, so we must move it "somehow" into the first argument of a two-argument function h.
   Note that for primitive recursive functions, the recursion scheme is the only way to provide case distinctions. We start with an ansatz for a yet unknown function g.

   $$f(x) = \begin{cases} 0 & \text{if } x = 0, \\ g(x-1, f(x-1)) & \text{otherwise.} \end{cases}$$

   If we want to use the conditions (1) and (2) from above, we must compare the square of the second argument of $g(a, b)$ with its first argument. Note that a and b correspond to $x-1$ and $f(x-1)$, respectively. We can use the function u via $u(a, b^2)$ in order to get a result 0 (not equal) or 1 (equal). Since in the primitive recursion scheme, one can "branch" upon a zero test of the first argument, we introduce another function h such that $g(a, b) = h(u(a, b^2), b)$. Note that this equation is not formally a correct application of the composition scheme from Definition 29 of the lecture notes. We make it formal later. Let's first deal with a definition of the (yet unknown) funktion h. If the first argument of h is zero, then we are in the case $f(x-1)^2 \neq (x-1)$, thus we can simply return the second argument of h. Otherwise, we just apply the recursion scheme with a (yet unknown) function i.

   $$h(y, x) = \begin{cases} \mathrm{proj}_1^1(x) & \text{if } y = 0, \\ i(y-1, h(y-1, x), x) & \text{otherwise.} \end{cases}$$

   Note that, the recursion scheme wants a function with one argument in case $y = 0$, so formally we have to write $\mathrm{proj}_1^1(x)$ instead of simply writing x. According condition (2), the function i should return the successor of its last argument, i.e.,

   $$i(x, y, z) = s(\mathrm{proj}_3^3(x, y, z))$$

   Now we define g in a formal way. For that we need some helper functions p and q. Whereas q stands for squaring its second argument, $p(a, b)$ is

needed to form $u(a, b^2)$ in a formal manner by applying the composition scheme.

$$\begin{aligned} g(a,b) &= h(p(a,b), \mathrm{proj}_2^2(a,b)) \\ p(a',b') &= u(\mathrm{proj}_1^2(a',b'), q(a',b')) \\ q(a'',b'') &= m(\mathrm{proj}_2^2(a'',b''), \mathrm{proj}_2^2(a'',b'')) \qquad (b'')^2 \end{aligned}$$

3. The LOOP-program from above is also a WHILE-programm.
4. A primitive recursive function is μ-recursive.

Problem 29. According to Definition 32 of the lecture notes, there are no natural numbers in Lambda calculus. However, natural numbers can be encoded (known as Church encoding) as "Church numerals" (see below), i.e., as functions $\mathbf{n}$ that map any function f to its n-fold application $f^n = f \circ \ldots \circ f$. Note that we denote such a "natural number" representation via boldface symbols in order to emphasize that these are lambda terms. In other words, we define Church numerals as follows. By letting "application" bind stronger than "abstraction", we avoid writing parentheses where appropriate.

$$\begin{aligned} \mathbf{0} &= \lambda f.\lambda x.x \\ \mathbf{1} &= \lambda f.\lambda x.fx \\ \mathbf{2} &= \lambda f.\lambda x.f(fx) \\ \mathbf{3} &= \lambda f.\lambda x.f(f(fx)) \\ \mathbf{4} &= \lambda f.\lambda x.f(f(f(fx))) \\ &\vdots \\ \mathbf{n} &= \lambda f.\lambda x.\underbrace{f(\cdots(f}_{n\text{-fold}}x)\cdots) \end{aligned}$$

1. Define a lambda term add that represents addition of "Church numerals".
2. Show the intermediate steps of a reduction from ((add **2**) **1**) to **3**.

Hint: a bit of literature research may help.

Solution of Problem 29:

Except for the derivation, the solution is more or less given here. `https://en.wikipedia.org/wiki/Church_encoding` In other words, this task actually asks the students find (maybe in books or on the Internet) a definition of add and how to apply the normal form algorithm in Lambda calculus. Additionally, the student learns how natural numbers can be embedded into lambda calculus.

1. Clearly, if $\mathbf{n}$ is a natural number, then $\mathbf{n}f$ stands for the lambda term of n-fold application of f and $(\mathbf{n}f)x$ for the lambda term of n-fold application of f to x. Similarly for $(\mathbf{m}f)y$. If we now replace y in the

second term by $(\mathbf{n}f)x$, the result would be $(m+n)$-fold application of f to x. We only have to abstract in order to arrive at our additon function.

$$\text{add} = \lambda m.\lambda n.\lambda f.\lambda x.((mf)((nf)x)) \tag{3}$$

2. Here is the derivation to add **2** and **1**.

$$\begin{aligned}((\text{add } \mathbf{2})\ \mathbf{1}) &= ((\lambda m.\lambda n.\lambda f.\lambda x.((mf)((nf)x)))\ \mathbf{2})\ \mathbf{1}\\ &\to (\lambda n.\lambda f.\lambda x.((\mathbf{2}f)((nf)x)))\ \mathbf{1}\\ &\to \lambda f.\lambda x.(\mathbf{2}f)((\mathbf{1}f)x)\\ &= \lambda f.\lambda x.(\mathbf{2}f)(((\lambda g.\lambda y.gy)f)x)\\ &\to \lambda f.\lambda x.(\mathbf{2}f)((\lambda y.fy)x)\\ &\to \lambda f.\lambda x.((\lambda g.\lambda y.g(gy))f)(fx)\\ &\to \lambda f.\lambda x.(\lambda y.f(fy))(fx)\\ &\to \lambda f.\lambda x.f(f(fx))\\ &= \mathbf{3}\end{aligned}$$

Problem 30. Let $\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$ and

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}.$$

Furthermore let for a natural number $n > 0$ the word w_n be given as the first n digits after the comma in the decimal expansion of e and $L = \{w_n \,|\, n \in \mathbb{N} \setminus \{0\}\}$.

(a) Is there a grammar G with Σ as the set of terminal symbols such that $L(G) = L$?

(b) Is L a context-free language?

Prove your answers!
Hint: Please note that e can be computed with arbitrary precision by evaluating a sufficiently large number of summands in the summation term.
For the proof of the second part of the task you have to look into additional literature. In particular, look for the Pumping Lemma for context-free languages.

Solution of Problem 30:

(a) According to a theorem from the lecture, every recursively enumerable language can be described by a grammar. According to the hint, L is recursively enumerable, a TM that generates L can easily be constructed.

(b) The proof is indirect. Suppose L is context-free. Then according to the Pumping Lemma there is a number n such that for every word $z \in L$, with $|z| > n$ there is a splitting $z = uvwxy$ with

(i) $|vwx| \leq n$,
(ii) $|vx| > 0$,
(iii) $\forall i \in \mathbb{N} : uv^i wx^i y \in L$.

Since in L there are no 2 words of the same length and all words of L correspond to the digits of e, it means that the decimal expansion of e is periodic from a certain point on. A periodic decimal expansion, however, means that e is a rational number, in contrast to the fact that e is irrational. Therefore, the assumption is wrong and L is not context-free.