# CSL 705: Theory of Computation

II semester 2011-12

Wed 02 May 2012 18:00-20:00 Major WS-213 Max Marks 60

1. *Please answer in the space provided on the question paper. The other sheets are only for rough work and will not be collected.*
2. *You may use any paper-based material including your class notes and any other text books.*
3. *You are not allowed to share reference material or rough pages during the exam.*
4. *You are not allowed to bring into the exam hall any electronic gadgets such as computers, mobile phones or calculators.*
5. *Please keep your identity card with you. You may be asked for it at any time for verification.*

1. **(10 marks).** Prove that the set of regular expressions over a non-empty finite alphabet $\Sigma$ is a context-free language.

*Solution.* Let $\Sigma = \{a_i \mid 1 \leq i \leq n\}$, $\boldsymbol{\Sigma} = \{\boldsymbol{a}_i \mid 1 \leq i \leq n\}$ Let $\Gamma = \boldsymbol{\Sigma} \cup \{\boldsymbol{\varepsilon}, \boldsymbol{\emptyset}, \boldsymbol{.}, \boldsymbol{*}, \boldsymbol{+}, \boldsymbol{(}, \boldsymbol{)}\}$. Then the context-free grammar $R = \langle V, \Gamma, P, S \rangle$ where $P$ the set of productions is given by the following rules.

$$\begin{array}{rcl} S & \rightarrow & \boldsymbol{\emptyset} \mid \boldsymbol{\varepsilon} \mid \boldsymbol{a}_1 \mid \cdots \mid \boldsymbol{a}_n \mid R \\ R & \rightarrow & R\boldsymbol{.}S \mid R\boldsymbol{*} \mid \boldsymbol{(}S\boldsymbol{)} \mid S \boldsymbol{+} R \end{array}$$

generates the regular expressions.

2. **(10 marks)** Prove that the set of all Turing machines is a recursive set.

*Solution.* A set $N \subseteq \mathbb{N}$ is recursive iff membership in $N$ is decidable. Let $N = \{\lceil T \rceil \mid \text{T is a Turing machine}\}$, where $\lceil T \rceil$ is the code of a Turing machine $T$. Hence for any $z \in \mathbb{N}$ the predicate "*$z$ is the code of a Turing machine*" needs to be decidable. Hence it suffices to show that the predicate istm is decidable. However, we can prove the stronger claim that it is primitive recursive.

**Claim 0.1** *The predicate* istm *is primitive recursive.*

*Proof:* Consider any $z \in \mathbb{N}$. If $z$ indeed is the code of a Turing machine then $\lceil \delta \rceil = d = \mathsf{proj}_1^{(3)}(z)$, $k = \mathsf{kpar}(z) = \mathsf{proj}_2^{(3)}(z)$ and $m = \mathsf{mpar}(z) = \mathsf{proj}_3^{(3)}(z)$ yield respectively the codes of the transition function $\delta$, the number $k$ of tape-symbols and the number $m$ of states of the Turing machine respectively. From the notes we know that these functions $\mathsf{proj}_i^{(3)}$ for $i \in \{1, 2, 3\}$ are all primitive recursive. We also know from the notes that the functions hd and tl are primitive recursive. If $d = 0$ then the transition table of the Turing machine is the empty list and the predicate $\mathsf{istm}(z)$ is false. Otherwise let $t_1 = \mathsf{hd}(d)$. it is easy to check whether $t_1$ encodes a valid transition using the predicate istrans defined as

$$\mathsf{istrans}(t, k, m) = \mathsf{isstate}(\mathsf{src}(t)) \wedge \mathsf{istapesym}(\mathsf{rd}(t)) \wedge \mathsf{isstate}(\mathsf{tgt}(t)) \wedge \mathsf{istapesym}(\mathsf{wr}(t)) \wedge (\mathsf{dir}(t) \leq 1)$$

where each function including $\wedge$ and $\leq$ is known to be primitive recursive. Then

$$\mathsf{istm}(z) = (z > 0) \wedge \mathsf{istranslist}(d, k, m)$$

where

$$\mathsf{istranslist}(d, k, m) = \mathsf{istrans}(\mathsf{hd}(d)) \wedge \mathsf{istranslist}(\mathsf{tl}(d), k, m)$$

**Note.** The above is a simplified definition of the predicate istm which does not take into account various other (not so important) factors which could also be defined primitive-recursively. Some of these are

(a) Every transition has to be distinct from other transitions.

(b) There is no transition from the final state.

(c) There is no non-determinism in the machine.

3. **(1+4+5 = 10 marks).** We may code finite sets of natural numbers by the function $fs$ which for any finite set $T = \{a_1, \ldots, a_m\}$, is defined as $fs(T) = \prod_{a_i \in T} p_{a_i+1}$ where $p_n$ is the $n$-th prime (with $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, etc.).

   (a) What is $fs(\emptyset)$?

   (b) Prove that $fs$ is primitive recursive.

   (c) Prove that the predicate $isfs : \mathbb{N} \to \{0, 1\}$ which determines whether a given number $n$ encodes a finite set of natural numbers, is also primitive recursive.

*Solution.*

(a) $fs(\emptyset) = 1$.

(b) We assume that every finite set has no repetitions of elements (i.e. it is "*square-free*") and is an unordered sequence.

$$fs\{a_1, \ldots, a_n\} = \prod_{a_i \in T} p_{a_i+1} = \prod_{1 \le i \le n} prime(a_i + 1)$$

$fs$ would be primitive recursive provided for the function $prime$ can be defined primitive recursively. For the sake of primitive recursiveness we define $prime(0) = 1$ and for each $m > 0$ we define $prime(m + 1) = prime(\sigma(m)) = \mu p < m! + 1[p > prime(m) \wedge \forall i \le m[i \le 1 \vee i \not| p]]$.

Notice that there is guaranteed to be a prime between $prime(m)$ and $m! + 1$ for each $m > 0$ (in fact there is guaranteed to be a prime between $prime(m)$ and $(\prod_{1 \le i \le m} prime(i)) + 1$ by Euclid's proof of the infinitude of primes).

From the lecture notes and the tutorials we know that bounded quantification, bounded minimization, case analysis, factorial and boolean operations are all primitive recursive. Hence it is easy to see that $prime$ is a primitive recursive function.

(c) As mentioned earlier we simply need to define a primitive recursive predicate which is true iff and only if a number is "*square-free*".

$$isfs(n) = n > 0 \wedge issquarefree(n)$$

where

$$issquarefree(n) = \forall i < n[\neg(1 < i) \vee ((i \times i) \not| n)]$$

Here again note that all the operations used are primitive recursive.

4. **(10 marks).** For unary computable functions on $\mathbb{N}$, prove that the problem of function equality is undecidable.

*Solution.* Let $f, g : \mathbb{N} \rightharpoonup \mathbb{N}$ be any two Turing-computable functions on $\mathbb{N}$ implemented by Turing machines $T_f$ and $T_g$ respectively. They are equal iff for each $x \in \mathbb{N}$ either both $f(x)$ and $g(x)$ are undefined or $f(x) = g(x) = y \in \mathbb{N}$ for some $y$.

We know that diff (which computes the absolute value of the difference of two numbers) is a primitive recursive function (see the tutorial sheet on primitive-recursion). Let $h = \mathsf{diff} \circ (f, g)$.

If function equality were decidable by a predicate feq we could use the solution to the problem of deciding whether $h(x) = 0$ for all $x \in \mathbb{N}$ for any unary computable function $h$. Let $T_{feq}$ be the Turing machine which decides function equality for arbitrary pairs of unary computable functions. For any unary computable function $h$ implemented by a Turing machine $T_h$ we take the code $\lceil T_h \rceil$ and take $\mathsf{fst}(\lceil T_h \rceil) = m$ and $\mathsf{snd}(\lceil T_h \rceil) = n$ and evaluate the primitive recursive predicate $\mathsf{istm}(m) \wedge \mathsf{istm}(n)$. If this predicate is true we run $T_{feq}$ on the Turing machines obtained from $\lfloor m \rfloor$ and $\lfloor n \rfloor$ and take the difference of the values obtained. Since $T_{feq}$ is guaranteed to give a result one way or other we may now use this mechanism to decide whether every Turing machine halts on every input.

But we know that there is no such Turing machine, since the halting problem is undecidable (actually semi-decidable). That is, there is no mechanism to guarantee for every unary computable function and every argument whether the machine halts. Hence the assumption that function equality is decidable must be false.

5. **(10 marks).** Prove that a function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ is computable <u>if and only if</u> $dom(f) = \{x \in \mathbb{N} \mid f(x)$ is defined $\}$ is a recursively enumerable set.

<u>*Solution.*</u> ($\Rightarrow$) If $f$ is (Turing)-computable there exists a Turing machine $T_f$ which can compute the value of $f(x)$ for each $x \in \mathbb{N}$ for which it is defined and does not halt otherwise. Consider the predicate $h_f : \mathbb{N} \longrightarrow \mathbb{2} = \{0, 1\}$ such that $h_f(x) = \begin{cases} 1 & \textit{if } T_f \textit{ halts on } x \\ 0 & \textit{otherwise} \end{cases}$. Clearly $dom(f)$ is the characteristic set defined by this predicate and $dom(f)$ is the set defined by $h_f^+(x) = \begin{cases} 1 & \textit{if } T_f \textit{ halts on } x \\ \textit{undefined} & \textit{otherwise} \end{cases}$ which is a partial recursive function which is Turing computable and $dom(f)$ is the set accepted by the Turing machine which computes $h_f^+$ and is hence recursively enumerable.

($\Leftarrow$) The converse is unfortunately not true.

6. **(10 marks).** Let $L_1$ and $L_2$ be context-free languages over a nonempty finite alphabet $\Sigma$ accepted by NPDAs $N_1$ and $N_2$ respectively. Construct the NPDAs $N_{1\cup 2}$, $N_{1.2}$ and $N_{1^*}$ which accept the union, concatenation and *-closure of the languages respectively.

*Solution.* We know that it is possible to have NPDAs $N_1 = \langle Q^1, \Sigma, \Gamma^1, \Delta^1, q_0^1, \perp^1, \{q_F^1\}\rangle$ and $N_2 = \langle Q^2, \Sigma, \Gamma^2, \Delta^2, q_0^2, \perp^2, \{q_F^2\}\rangle$ such that $\mathcal{L}_E(N_1) = L_1$ and $\mathcal{L}_E(N_2) = L_2$ such that all the components of $N_1$ are disjoint from the corresponding components of $N_2$. We may then construct

$$N_{1.2} = \langle Q^1 \cup Q^2, \Sigma, \Gamma^{1.2}, \Delta^{1.2}, \perp^{1.2}, \{q_F^2\}\rangle$$

such that $\mathcal{L}_E(N_{1.2}) = L_{1.2}$ where $\Gamma^{1.2} = \Gamma^1 \cup \Gamma^2 \cup \{\perp^{1.2}\}$ and

$$\begin{array}{rcl}\Delta^{1.2} & = & \Delta^1 \cup \Delta^2 \cup \\ & & \{((q_0^1, \varepsilon, \perp^{1.2}), (q_0^1, \perp^1\perp^{1.2})), ((q_F^1, \varepsilon, \perp^{1.2}), (q_0^2, \perp^2\perp^{1.2})), ((q_F^2, \varepsilon, \perp^{1.2}), (q_F^2, \varepsilon)\}\end{array}$$

It then easy to show that $\mathcal{L}_E(N_{1.2}) = L_{1.2}$.

The NPDA for $L_{1^*}$ is given by $N_1 = \langle Q^1, \Sigma, \Gamma^1, \Delta^{1^*}, q_0^1, \perp^{1^*}, \{q_F^1\}\rangle$ where

$$\begin{array}{rcl}\Delta^{1^*} & = & \Delta^1 \cup \\ & & \{((q_0^1, \varepsilon, \perp^{1.2}), (q_0^1, \perp^1\perp^{1.2})), ((q_0^1, \varepsilon, \perp^{1.2}), (q_F^1, \varepsilon))((q_F^1, \varepsilon, \perp^{1.2}), (q_0^1, \perp^1\perp^{1.2})), ((q_F^1, \varepsilon, \perp^{1.2}), (q_F^1, \varepsilon)\}\end{array}$$

$N_{1\cup 2} = \langle Q^1 \times Q^2, \Sigma, \Gamma^1 \times \Gamma^2, \Delta^{1\cup 2}, (\perp^1, \perp^2), (Q^1 \times \{q_F^2\}) \cup (\{q_F^1\} \times Q^2)\rangle$ where

$$\begin{array}{rcl}\Delta^{1\cup 2} & = & \{(((q^1, q^2), a, (\gamma^1, \gamma^2)), ((q^{1'}, q^{2'}), (\gamma^{1'}, \gamma^{2'}))) \mid ((q^1, a, \gamma^1), (q^{1'}, \gamma^{1'})) \in \Delta^1, ((q^2, a, \gamma^2), (q^{2'}, \gamma^{2'})) \in \Delta^2, a \in \Sigma, \\ & & \{(((q^1, q^2), a, (\gamma^1, \gamma^2)), ((q^{1'}, q^2), (\gamma^{1'}, \gamma^2))) \mid ((q^1, a, \gamma^1), (q^{1'}, \gamma^{1'})) \in \Delta^1, (q^2, a, \gamma^2) \notin Dom(\Delta^2), a \in \Sigma, q^1, q^{1'} \\ & & \{(((q^1, q^2), a, (\gamma^1, \gamma^2)), ((q^1, q^{2'}), (\gamma^1, \gamma^2))) \mid ((q^2, a, \gamma^2), (q^{2'}, \gamma^{2'})) \in \Delta^2, (q^1, a, \gamma^1) \notin Dom(\Delta^1), a \in \Sigma, q^2, q^{2'} \\ & & \{(((q_F^1, q^2), \varepsilon, (\varepsilon, \gamma^2)), ((q_F^1, q^2), (\varepsilon, \varepsilon))) \mid q^2 \in Q^2\} \cup \\ & & \{(((q^1, q_F^2), \varepsilon, (\gamma^1.\varepsilon)), ((q^1, q_F^2), (\varepsilon, \varepsilon))) \mid q^1 \in Q^1\}\end{array}$$

where the two automata are combined into a single one where the moves of the individual machines are replicated in the combined automaton. Whenver only machine can make a move it is allowed to do so. If any of the machines reaches a final state and a correpsonding empty stack, the stack of the other machine is also emptied.