Your name _____

Harvey Mudd College
CS 81 Final Exam
Spring semester, 2005

Part 1 Closed Book

1. This exam is in two parts.
2. For part 1, no reference materials are to be used. Part 1 counts for 25%.
3. You should probably not work on part 1 more than about 1 hour.
4. After submitting your solutions for part 1, you may begin part 2, wherein you may consult a single 2-sided crib sheet. Part 2 counts for 75%.
5. The exam has a 3-hour overall time limit for both parts.
6. It is best to spend your time working on problems that pay off in terms of the most points.

In these problems, Σ is a fixed finite alphabet of m > 1 symbols and Σ* is the set of finite strings of symbols in Σ.

When relevant, we also view Σ* as a set of m-adic numerals, each representing a natural number. (For example, when Σ = {1, 2}, the empty string represents number 0, string 1 represents number 1, string 2 represents number 2, string 11 represents number 3, string 12 represents number 4, etc.)

1. [5 points] Answers in italics

What does it mean for a language to be recursive?

A language is recursive if it is accepted by a Turing machine, i.e. for a given input, the machine always halts and indicates whether or not the input is in the language.

Indicate which of the following languages are recursive. Indicate your reasons:

a. Σ*, the set of all finite strings over Σ.

   ***Recursive**. The machine halts and says "yes" for any input.*

b. The set of all propositional logic formulas with proposition letters in the set Σ (which is assumed not to include the logical connectives).

   ***Recursive**. The machine parses the input string, indicating whether or not it is well-formed. Since the language of propositional logic formulas is context-free, parsing can be done with a pushdown automaton, which has the subset of the capabilities of a Turing machine.*

c. The set of all unsatisfiable propositional formulas with proposition letters in the set Σ (which is assumed not to include the logical connectives).

   ***Recursive**. The machine first checks the input is well-formed. Then it can determine unsatisfiability by trying all possible combinations of truth values for the proposition letters and evaluating each.*

d. The set {<M> | M halts on a blank tape}.
   Here <M> denotes a string effectively encoding a Turing machine M.

   ***Not recursive**. If there were a machine that accepts this language, it could determine whether any machine halts on a blank tape. But the halting problem reduces to that problem.*

e. The set {<M, n> | M halts in n or fewer steps on a blank tape}.
   Here <M, n> denotes a string effectively encoding a Turing machine M and a natural number n.

   ***Recursive**. A machine to accept this language can embed a universal Turing machine that simulates M while counting steps. If M hasn't halted by n steps, it returns the answer "no". Otherwise it returns "yes".*

2. [10 points] Let $L_0, L_1, L_2, \ldots$ be an effective enumeration of all recursively-enumerable subsets of Σ^*.

   a. Consider the language $M = \{i \in \Sigma^* \mid i \in L_i\}$. Does M appear in the enumeration somewhere? Justify.

      Answer:

      Yes, it does. M is recursively-enumerable. There is a turing machine T which, given input i, determines the description of a machine recognizing L_i. The machine for M is like T except that it first writes i on the tape, then behaves like the machine recognizing L_i.

   b. Does $M' = \{i \in \Sigma^* \mid i \notin L_i\}$ appear in the enumeration somewhere? Justify.

      Answer:

      No, it doesn't appear in the enumeration. Suppose it appeared as L_k. Then we would have the contradiction that $k \in L_k$ iff $k \notin L_k$.

3. [10 points] Let $L_0, L_1, L_2, \ldots$ be an effective enumeration of all recursively-enumerable subsets of Σ^*. Define

$$R = \{i \in \Sigma^* \mid L_i \text{ is recursive}\}.$$

Is R recursive? Prove your answer. If you invoke Rice's theorem, provide a proof of the theorem tailored to the question at hand.

Answer:

R is not recursive. Recursiveness is a non-trivial property of recursively-enumerable languages, so by Rice's theorem, we cannot test for it.

To tailor the proof of Rice's theorem, we observe that the empty set is recursive, so that we actually need to work with the complementary set:

$$R' = \{i \in \Sigma^* \mid L_i \text{ is not recursive}\}$$

We show that if R' were recursive, then we could solve the halting problem.

Suppose that there is an algorithm A that, with input i, determines whether or not L_i is not recursive. We use this algorithm to create an algorithm H that solves the halting problem H(<T, x>) = 1 if T halts on x, 0 otherwise.

We need a machine that recognizes a non-recursive (but of course, recursively-enumerable language). The machine M from problem 2 will do.

With input <T, x>, H constructs a machine T' that behaves as follows: With input y, T' sets y aside and simulates T on x. If and when that simulation halts, T' then behaves as M on the original input y.

Hence if T halts on x, then the language recognized by T' is the same as that of M, and is non-recursive. But if T does not halt on x, then the language recognized by T' is $\varnothing$, and thus recursive.

So B gives the description of T' to A which indicates whether or not the language of T' is recursive. The outcome of this indication tells whether or not T halts on x. So B solves the halting problem.

Since the halting problem is not solvable, our assumption that A exists is wrong: There is no algorithm that will tell whether an arbitrary Turing machine accepts a non-recursive language.

Your name _____

Harvey Mudd College
CS 81 Final Exam
Spring semester, 2005

Part 2

1. For this part, you may use a single 2-sided crib sheet, but no other reference material.
2. Please turn in your crib sheet along with the exam. It will be returned to you.
3. Part 2 counts for 75%.
4. The exam has a 3-hour overall time limit for both parts.
5. It is best to spend your time working on problems that pay off in terms of the most points.

4. [5 points]

Prove the following sequent

$a \vee b,\ \ a \vee \neg b \vdash a$

using each of:

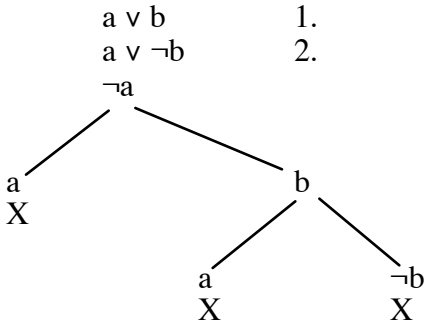
a. natural deduction
b. the tree method
c. resolution

Answer:

a. Natural deduction:

1.	$a \vee b$	Premise
2.	$a \vee \neg b$	Premise
3.	a	Assumption
4.	b	Assumption
5.	a	Assumption
6.	$\neg b$	Assumption
7.	$\perp$	$\neg$e 4, 6
8.	a	$\perp$e 7
9.	a	ve 2, 5-5, 6-8
10.	a	ve 1, 3-3, 4-9

b. Tree method



c. Resolution

1.	$a \vee b$	
2.	$a \vee \neg b$	
3.	$\neg a$	
4.	b	1, 3
5.	$\neg b$	2, 3
6.	$\perp$	4, 5

(More space for number 4 if you need it.)

5. [30 points]

Prove the following sequent

$$\forall x\ (p(x) \vee q(x)) \vdash (\forall x\ p(x)) \vee (\exists x\ q(x))$$

using each of:

a. natural deduction
b. the tree method
c. resolution

Also, give an English-language proof based on one of the above proofs.

You may assume, without deriving them, any of the deMorgan rules, including ones for quantifiers. However, state clearly what you are assuming.

Answer:

a. natural deduction

1.	$\forall x\ (p(x) \vee q(x))$	Premise
2.	$(\forall x\ p(x)) \vee \neg(\forall x\ p(x))$	LEM
3.	$(\forall x\ p(x))$	Assumption
4.	$(\forall x\ p(x)) \vee (\exists x\ q(x))$	$\vee$i 3
5.	$\neg(\forall x\ p(x))$	Assumption
6.	$(\exists x\ \neg p(x))$	deMorgan rule below
7.	$x_0\ \neg p(x_0))$	Assumption
8.	$p(x_0) \vee q(x_0))$	$\forall$e 1
9.	$p(x_0)$	Assumption
10.	$\perp$	$\neg$e 7, 9
11.	$q(x_0)$	$\perp$e 10
12.	$q(x_0))$	Assumption
13.	$q(x_0)$	$\vee$e 8, 9-11, 12-12
14.	$\exists x\ q(x)$	$\exists$i 13
15.	$\exists x\ q(x)$	$\exists$e 6, 7-14
16.	$(\forall x\ p(x)) \vee (\exists x\ q(x))$	$\vee$i 15
17.	$(\forall x\ p(x)) \vee (\exists x\ q(x))$	$\vee$e 2, 3-4, 5-16

Using the derived deMorgan rule: $\dfrac{\neg(\forall x\ p(x))}{(\exists x\ \neg p(x))}$

(More space for number 5 if you need it)

b. Tree method

$\forall x\ (p(x) \vee q(x))$	6.
$\neg((\forall x\ p(x)) \vee (\exists x\ q(x)))$	1.
$\neg(\forall x\ p(x)$	2.
$\neg(\exists x\ q(x))$	3.
$(\exists x\ \neg p(x))$	4.
$(\forall x\ \neg q(x))$	5.
$\neg p(x_0)$	7.
$\neg q(x_0)$	8.
$p(x_0) \vee q(x_0)$	9.

c. Resolution

1.	$p(x) \vee q(x)$	
2.	$\neg p(x_0)$	
3.	$\neg q(x)$	
4.	$p(x)$	1, 3
5.	$\perp$	2, 4

English (from the tree method):

Suppose that $\forall x\ (p(x) \vee q(x))$ but *not* $((\forall x\ p(x)) \vee (\exists x\ q(x)))$, to draw a contradiction. By deMorgan's law, not(A or B) is equivalent to (not A) and (not B), i.e. not $(\forall x\ p(x))$ *and* not $(\exists x\ q(x))$. Moreover, by deMorgan's law for quantifiers, the first of these is equivalent to $(\exists x\ \neg p(x))$, while the second is equivalent to $(\forall x\ \neg q(x))$.

From $(\exists x\ \neg p(x))$, let x_0 be such that $\neg p(x_0)$. From $(\forall x\ \neg q(x))$, we have $\neg q(x_0))$. So there is a value x_0, such that $\neg p(x_0) \wedge \neg q(x_0)$. By deMorgan's law, this is equivalent to $\neg(p(x_0) \vee q(x_0))$, which stands in direct contradiction to $\forall x\ (p(x) \vee q(x))$.

6. [10 points]

Suppose that $L \subseteq N$ is the range of a total recursive function T having the property

$$\forall i \in N \; \forall j \in N \; ((i < j) \rightarrow T(i) < T(j))$$

Show by an informal argument that the characteristic function of L is recursive. (Recall that the characteristic function of L is the function ch_L, where $ch_L(x) = 1$ if $x \in L$ and 0 otherwise.)

Answer:

Suppose that L and T are as stated. We want to show that ch_L is recursive.

Here is an algorithm for computing ch_L: With input x, enumerate values T(0), T(1), T(2), … until an i is found such that $T(i) \geq x$. If $T(i) = x$, then return 1. Otherwise return 0.

In other words, using the μ operator, $ch_L(x) = eq(x, T(\mu i\, [not(sub(T(i), x)) = 0)])$.

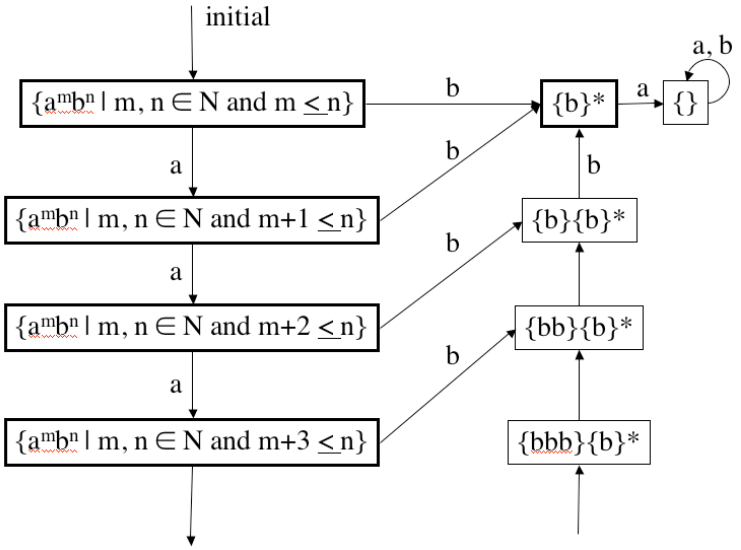
However, due to the property of T stated, for a given x, we will eventually find a $T(i) \geq x$, so the algorithm is guaranteed to terminate.

7. [15 points]

Concisely describe the abstract states (also known as Myhill-Nerode equivalence classes) of the following languages:

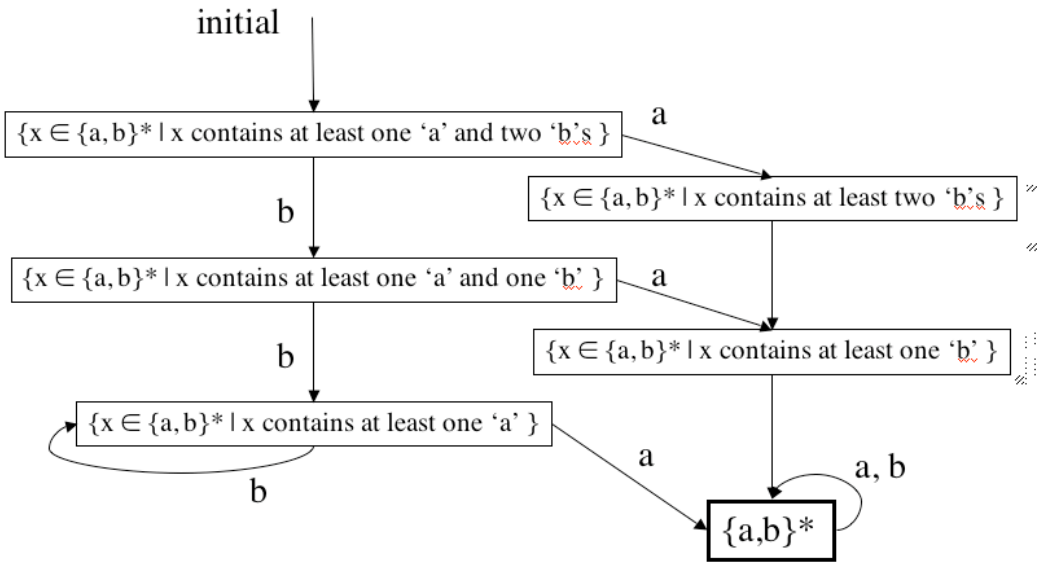
a. $\{a^m b^n \mid m, n \in N \text{ and } m \leq n\}$

Answer: The abstract states (there is an infinite set) are identified by the node labels in this diagram. (The diagram itself is not required):



b. $\{x \in \{a, b\}^* \mid x \text{ contains at least one 'a' and two 'b's }\}$

Answer: The abstract states identified by the node labels in this diagram. (The diagram itself is not required):



Which, if any, of the above languages are regular?
Answer: Only the second language is regular.

8. [10 points]

Construct an algorithm that will determine, for any two regular expressions R and S, whether or not L(R) ⊆ L(S).

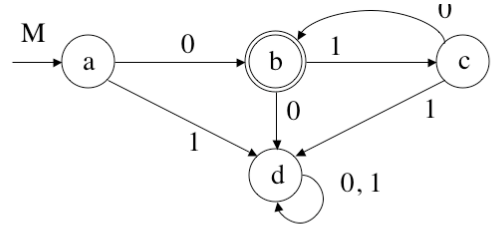
Answer:

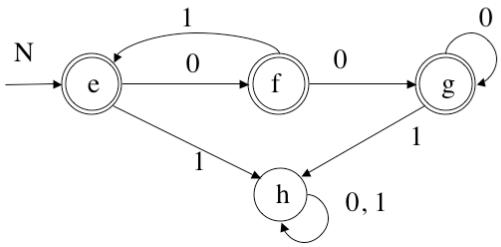
Construct DFA's M and N for the regular expressions R and S respectively.

Then construct the product machine for those two DFA's. The states of the product machine are of the form (q, q') where q is a state of M and q' is a state of N.

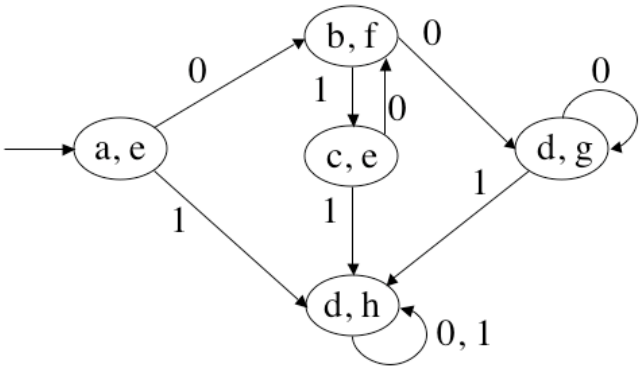
If for every pair (q, q') where q is an accepting q' is also an accepting state, then L(R) ⊆ L(S). Otherwise not.

Example: Suppose we want to know whether L(0(10)*) ⊆ L((01)*0*). The DFA's for these two languages are shown as M and N.





The product machine is shown below. We can see that for an accepting state from M, namely (b, f), f is an accepting state for N. Thus the inclusion holds.



9. [5 points]

Construct an algorithm that will determine, for any context-free grammar G, whether or not $L(G) = \varnothing$.

Answer:

Let S be the start symbol of the grammar. Then $L(G) = \{ x \in \Sigma^* \mid S \Rightarrow^* x\}$, where Σ is the terminal alphabet. It suffices to determine the set $R = \{A \in N \mid \exists\, x \in \Sigma^* \; A \Rightarrow^* x\}$, where N is the auxiliary alphabet. Then $L(G) = \varnothing$ iff whether $S \notin R$.

The set R can be determined as follows:

```
R := ∅
T := { A ∈ N | ∃ x ∈ Σ* A → x}              // Note →,  not ⇒*
while( not T ⊆ R )
        {
        R := R ∪ T
        T := { A ∈ N | ∃ x ∈ (R ∪ Σ)* A → x}
        }
```

Examples:

1. $S \rightarrow A$ $A \rightarrow SBa$ $B \rightarrow b$ with start symbol S

   $R = \varnothing$, $R = \{B\}$ which does not contain S, therefore $L = \varnothing$.

   We can see that any derived string will always have either an S or an A in it.

2. $S \rightarrow A$ $A \rightarrow Ba$ $B \rightarrow b$ with start symbol S

   $R = \varnothing$, $R = \{B\}$, $R = \{A, B\}$, $R = \{S, A, B\}$ which does contain S, therefore $L \neq \varnothing$.

   $Ba \in L$.