Your name _____

Harvey Mudd College
**CS 81 Final Exam**
Fall semester, 2006

**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 45 minutes.
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%. There is also a 20% extra credit problem on part 2.
5. Once you start part 2, it is not permitted to return to part 1.
6. The exam has a 3-hour overall time limit for both parts combined.
7. It is best to spend your time working on problems that pay off in terms of the most points.

1. [5 points]

Consider the following famlies of languages that we have studied:

- regular
- context-free
- recursive
- recursively-enumerable

In what important way is the family "recursive" different from all of the others?

**Answer:**

Of the families given, *recursive* is the only one that does not have a purely-syntactic machine-based characterization. Instead, recursive requires an emergent property of the machine not halting. We have proved that there is no way to determine this based on examination of the machine's encoding.

Note: It is not true that recursive is the only family closed under complementation. The regular languages also have this property.

2. [10 points]

For each language row below, check the *left-most* column entry for which the language is in that family. Use the row numbers should you wish to attach a note clarifying your answer to any item. (For the rows involving formulas, assume that the variables and proposition symbols in the formula are represented using a finite alphabet, e.g. proposition symbols are p, p', p'', p''', … so that a finite alphabet can be used overall.)

| | | Regular | Context-Free | Recursive | Recursively-Enumerable | None of the preceding |
|---|---|---|---|---|---|---|
| 1 | The set of all regular expressions. | | ✓ | | | |
| 2 | The set of all Turing machine encodings, using an encoding such as the one in our text. | I will accept this too. | | ✓ | | |
| 3 | The set of all Turing machine encodings of Turing machines that always halt when started with their own description as input. | | | | ✓ | |
| 4 | The set of all surnames in the most recent Los Angeles telephone directory. | ✓ | | | | |
| 5 | The set of all predicate logic formulas. | | ✓ | | | |
| 6 | The set of all universally-valid predicate logic formulas. | | | | ✓ | |
| 7 | The set of all satisfiable predicate logic formulas. | | | | | ✓ |
| 8 | The set of all propositional logic formulas. | | ✓ | | | |
| 9 | The set of all satisfiable propositional logic formulas. | | | ✓ | | |
| 10 | The empty set. | ✓ | | | | |

Notes:

2: The encoding in the text book is not regular because it requires listing all of the states and tape symbols up front. But this seems unnecessary and a regular encoding is possible. In any case, the encoding is certainly recursive. We have to be able to tell whether something is a valid encoding or not.

6: The set of all theorems is recursively-enumerable. So the set of all valid formulas is, by the completeness theorem.

7: A formula is satisfiable iff its negation is not valid. Although we can enumerate the valid formulas, we cannot enumerate the invalid ones, otherwise the set of formulas would be recursive. But we have argued that this is not the case.

3. [5 points]

I.M. Brilliant has devised a new Turing-machine encoding that he thinks will "help solve a variant of the halting problem". Instead of using $M_0, M_1, M_2, \ldots$ as the enumeration of all Turing machines using the encoding discussed in the text, I.M. suggests encoding the Turing machines as $N_0, N_1, N_2, \ldots$ in the following way:

a. As sets, the two sets of machines are equal $\{M_0, M_1, M_2, \ldots\}=\{N_0, N_1, N_2, \ldots\}$, so both cover all of the machines.

b. The even-numbered $N_0, N_2, N_4, \ldots$ correspond to machines that halt on every input.

c. The odd-numbered machines in $N_1, N_3, N_5, \ldots$ correspond to machines that fail to halt on at least one input.

Comment on I.M.'s approach, specifically on its general utility and a possible algorithm for transforming the new numbering into the old numbering.

**Answer:**

I.M.'s approach is not helpful, because there is no way to convert between his encoding/numbering of machines and the standard one. If there were a way, then the language AlwaysHalts would be recursive, by doing the conversion and checking the numbers. But we have shown that AlwaysHalts is not recursive.

4. [5 points]

Goldbach's conjecture is that every even integer > 2 is the sum of two primes (not necessarily distinct). Consider the language over {0, 1}:

L = {x | x = 1 if Goldbach's conjecture is true, 0 otherwise}

Show that L is recursive.

**Answer:**

L is a finite set. Every finite set is recursive, since there is an always-halting Turing machine that accepts it. We just don't know the value of x yet.

Your name _____

Harvey Mudd College
**CS 81 Final Exam**
Fall semester, 2006

**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. You are not permitted to return to part 1 after starting part 2.
5. The exam has a 3-hour overall time limit for both parts combined.
6. It is best to spend your time working on problems that pay off in terms of the most points.

5. [10 points]

The sequent below states that the composition of two 1-1 functions is itself 1-1. Prove it using natural deduction.

$\forall x \forall y\ (f(x) = f(y) \rightarrow x = y)$ (premise)

$\forall x \forall y\ (g(x) = g(y) \rightarrow x = y)$ (premise)

$\vdash \forall x \forall y\ (f(g(x)) = f(g(y)) \rightarrow x = y)$ (conclusion)

**Answer**:

| | | |
|---|---|---|
| 1. | $\forall x \forall y\ f(x) = f(y) \rightarrow x = y$ | Premise |
| 2. | $\forall x \forall y\ g(x) = g(y) \rightarrow x = y$ | Premise |
| 3. | $f(g(x_0)) = f(g(y_0))$ | Assumption |
| 4. | $\forall y\ \ f(g(x_0)) = f(g(y)) \rightarrow g(x_0) = g(y)$ | $\forall$E, 1 |
| 5. | $f(g(x_0)) = f(g(y_0)) \rightarrow g(x_0) = g(y_0)$ | $\forall$E, 4. |
| 6. | $g(x_0) = g(y_0)$ | $\rightarrow$E, 3, 5 |
| 7. | $\forall y\ g(x_0) = g(y) \rightarrow x_0 = y$ | $\forall$E, 2 |
| 8. | $g(x_0) = g(y_0) \rightarrow x_0 = y_0$ | $\forall$E, 7 |
| 9. | $x_0 = y_0$ | $\rightarrow$E, 6, 8 |
| 10. | $f(g(x_0)) = f(g(y_0)) \rightarrow x_0 = y_0$ | $\rightarrow$I, 3-9 |
| 11. | $\forall y\ f(g(x_0)) = f(g(y)) \rightarrow x_0 = y$ | $\forall$I, 10 |
| 12. | $\forall x \forall y\ f(g(x)) = f(g(y)) \rightarrow x = y$ | $\forall$I, 11 |

6. [15 points]

Prove using natural deduction:

$\vdash \exists x\,(\varphi \wedge \psi) \rightarrow ((\exists x\varphi) \wedge (\exists x\psi))$

**Answer**:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\varphi\wedge\psi]_1}{\varphi}\wedge E}{\exists x\varphi}\exists I \qquad \dfrac{[\varphi\wedge\psi]_1}{\psi}\wedge E}{\dfrac{[\exists x\,(\varphi\wedge\psi)]_2 \qquad (\exists x\varphi)\wedge(\exists x\psi)}{(\exists x\varphi)\wedge(\exists x\psi)}\exists E_1}\exists I}{\exists x\,(\varphi\wedge\psi)\rightarrow((\exists x\varphi)\wedge(\exists x\psi))}\wedge I \rightarrow I_2$$

7. [15 points]

Using the tree method, establish whether or not the following formula is valid.

$(\exists x\ (P(x) \vee Q(x))) \leftrightarrow ((\exists x\ P(x)) \vee (\exists x\ Q(x)))$

**Answer:**

We negate the formula and check satisfiability:

$\neg((\exists x\ (P(x) \vee Q(x))) \leftrightarrow ((\exists x\ P(x)) \vee (\exists x\ Q(x))))$

$(\exists x\ (P(x) \vee Q(x))$ ✓
$\neg((\exists x\ P(x)) \vee (\exists x\ Q(x)))$ ✓
$\neg((\exists x\ P(x))$ ✓
$\neg(\exists x\ Q(x))$ ✓
$\forall x\ \neg P(x)$
$\forall x\ \neg Q(x)$
$P(x_0) \vee Q(x_0)$

| | |
|---|---|
| $P(x_0)$ | $Q(x_0)$ |
| $\neg P(x_0)$ | $\neg Q(x_0)$ |
| $\perp$ | $\perp$ |

$\neg(\exists x\ (P(x) \vee Q(x)))$ ✓
$((\exists x\ P(x)) \vee (\exists x\ Q(x))$ ✓
$\forall x\ \neg(P(x) \vee Q(x))$
$\exists x\ P(x)$ ✓
$P(x_1)$
$\neg(P(x_1) \vee Q(x_1))$
$\neg P(x_1)$
$\neg Q(x_1)$
$\perp$

$\exists x\ Q(x)$ ✓
$Q(x_2)$
$\neg(P(x_2) \vee Q(x_2))$
$\neg P(x_1)$
$\neg Q(x_1)$
$\perp$

All paths close, so the original formula was valid.

8. [20 points]

"Polish" notation is a notation for terms that doesn't require parentheses. It relies on each function symbol having a distinct arity. For simplicity, we consider here a term language with just two symbols, a 2-ary function symbol, **f**, and a constant symbol, **a**. In this case, the terms in Polish notation are given by:

a. The constant symbol **a** is a term.
b. If *X* and *Y* are terms, then so is **f***XY*.
c. The only terms are those derived by the above rules.

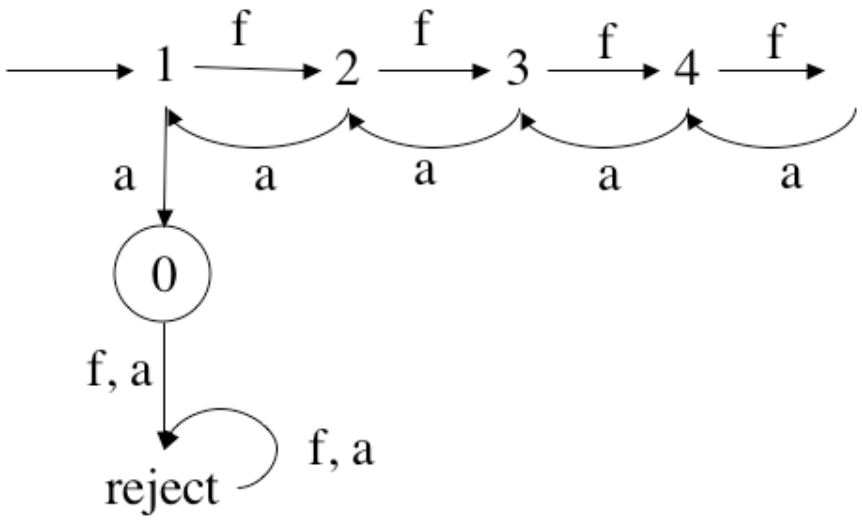
For example, here are five terms: **a**, **faa**, **fafaa**, **ffaafaa**, **ffaafafaa**.

Let L be the language of all terms as above.

a. [2/20] Give a context-free grammar for L.
**Answer**:
S is the start-symbol
S → a
S → f S S

b. [8/20] Diagram the abstract states of L and the transitions between them.
**Answer:**
Call the deficit of a string the quantity $\#_f(x) - \#_a(x) + 1$. Evidently, for any prefix of a derived string, the deficit must be non-negative. Also, the deficit of a derived terminal string must be exactly 0. Thus if the prefix of a string has a negative string, the string must be rejected.

Below is the transition diagram for abstract states. The numbers on the states refer to the deficit. Adding an f to the end of an incomplete string increases the deficit by a net of 1, since the term ultimately represented requires one more argument than it did before. Similarly the deficit can be decreased by adding an a to the end of the string.



c. [4/20] Show that L is not regular.

**Answer**:
The deficit can be made 0 by adding exactly that number of a's to the end of the string. Thus no two distinct states are equivalent. Since there is a state for each natural number, there is an infinite number of abstract states and thus the language is not regular.

d. [6/20]Give the transition rules for a pushdown automaton that accepts L.

**Answer**:
These rules essentially implement the abstract state transition diagram. There will be a number of f's on the stack equal to the value of the deficit. There are three control states: {initial, ok, accept, reject}. The pda accepts by control-state.

Top of stack is at the left.

initial, $\perp$, a $\rightarrow$ accept, $\varepsilon$

initial, $\perp$, f $\rightarrow$ ok, f$\perp$

ok, f, f $\rightarrow$ ok, ff

ok, f, a $\rightarrow$ ok, $\varepsilon$

ok, $\perp$, f $\rightarrow$ ok, ff

ok, $\perp$, a $\rightarrow$ accept, $\varepsilon$

accept, $\perp$, f $\rightarrow$ reject, $\varepsilon$

accept, $\perp$, a $\rightarrow$ reject, $\varepsilon$

9. [15 points]

Consider the set of strings {0, 1}* ordered according to the correspondence with the natural numbers (ε ↔ 0, 0 ↔ 1, 1 ↔ 2, 00 ↔ 3, 01 ↔ 4, etc., where x ↔ n means string x corresponds to number n), so that 01 < 10, etc.

If L is a language, define

$$\min(L) = \begin{cases} \{ x \in L \mid \forall y \in L\ (x \neq y \rightarrow x < y) \} & \text{if } L \neq \varnothing \\ \varnothing & \text{otherwise} \end{cases}$$

Which of the following are true and which are false?

a. If L is recursive, then min(L) is recursive.

b. If L is recursive, then min(L) is recursively-enumerable.

c. If L is recursively-enumerable, then min(L) is recursively-enumerable.

Comment on whether there is an algorithm that will transform an arbitrary Turing machine M, accepting L(M), into one accepting min(L(M)).

**Answer**:

min(L) is always either empty or a single string. Thus *all three items are true*, since all are finite and thus recursive. An effective transformation algorithm is a different matter.

In the case that L is recursisve, there is an algorithm that will transform M to a machine accepting min(L), *provided* that M always halts. We know that there is such an M, although *not every* machine accepting L necessarily always halts. If M always halts, the machine accepting min(L) can be made to halt if L is non-empty by trying M on increasing input strings. If and when M accepts, the value of min(L) is found, and if the input to M' is that value, M' accepts. If L is empty, then no value is ever found that is contained in min(L), so M' never accepts any string, consistent with the definition.

If L is only recursively-enumerable, not recursive, determining min(L) would seem to require testing strings in increasing order for membership in L. But it cannot be determined whether or not the machine accepting L halts on a given string, so it is not likely that we can construct a machine for min(L). This is not a proof that we can't, only the intuition for why we can't.

10. [20 points Extra credit]

You are advised not to attempt this unless you have completed everything else to the best of your ability.

The sequent below states that the composition of two onto functions is itself onto.

Prove it by the resolution method.

$\forall y \exists x\ f(x) = y$ (premise)

$\forall y \exists x\ g(x) = y$ (premise)

$\vdash \forall y \exists x\ f(g(x)) = y$ (conclusion)

Here is an equivalent clause form, to help you get started, where clauses 4-6 represent facts about equality that will likely be needed.

1. E(f(i(y)), y).
2. E(g(j(y)), y).
3. ¬E(f(g(x)), c).
4. ¬E(x, y) ∨ ¬E(y, z) ∨ E(x, z).
5. ¬E(x, y) ∨ E(f(x), f(y)).
6. ¬E(x, y) ∨ E(g(x), g(y)).

Here **E** represents the equality predicate, **i** is a Skolem function representing the inverse of **f**, **j** is a Skolem function representing the inverse of **g**, and **c** is a Skolem constant that arises from negating the conclusion. (There is more that can be said about **E**, but this is enough to enable the proof.)

**Answer**:

Resolve 1 and 4, with binding x4 = f(i(y1)), y4 = y1, giving

7. -E(y, z) | E(f(i(y)), z).

Resolve 1 and 7, with binding y7 = f(i(y1)), z7 = y1, giving

8. E(f(i(f(i(y)))), y).

Resolve 2 and 5, with binding x5 = g(j(y2)), y5 = y2, giving

9. E(f(g(j(y))), f(y)).

Resolve 4 and 9, with binding x4 = f(g(j((y)))), y4 = f(y), giving

10. -E(f(y), z) | E(f(g(j(y))), z)

Resolve 10 and 8, with binding y10 = i(f(i(y))), z10 = y, giving

11. E(f(g(j(i(f(i(y)))))), y).

Resolve 3 and 11, with binding x3 = j(i(f(i(y)))), giving

12. ⊥.