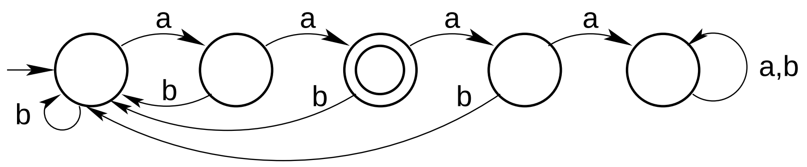
CpSc 421 Final Exam **December 6, 2008**

Do any **five** of problems 1-10.
If you attempt more than five of problems 1-10, please indicate which ones you want graded – otherwise, I'll make an arbitrary choice.

Graded on a scale of 100 points.
You can attempt from 68 to 110 points depending on which problems you choose. If you score over 100, you get to keep the extra credit.

1. (**12 points**)

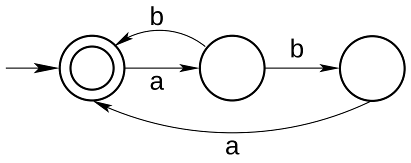
   (a) (**6 points**) Write a regular expression that generates the language that is recognized by the DFA below:



   Solution: $((\epsilon \cup \mathsf{a} \cup \mathsf{aa} \cup \mathsf{aaa})\mathsf{b})^*\mathsf{aa}$

   (b) (**6 points**) Draw the state transition diagram for a DFA or NFA that recognizes the language generated by the regular expression $(ab)^*(aba)^*$.

   Solution:



2. (**12 points**) Let G be the CFG with start variable S, terminals a and b, and rules:

$$\begin{array}{rcl} S & \rightarrow & T \mid U \\ T & \rightarrow & \epsilon \mid \mathsf{a}T\mathsf{bb} \\ U & \rightarrow & \epsilon \mid \mathsf{ba}U\mathsf{a} \end{array}$$

   (a) Which of the strings below are generated by G?

   aabbbb baa bbaaaa aaaabb bababaaaa baabaa

   Solution: aabbbb baa bababaaaa

   (b) Write a grammar that generates the language:

$$\{\mathsf{a}^m\mathsf{b}^n \mid (m = 2n+3) \vee (3m = 5n)\}$$

   Solution:

$$\begin{array}{rcl} S & \rightarrow & TU \mid V \\ T & \rightarrow & \mathsf{aaa} \\ U & \rightarrow & \epsilon \mid \mathsf{aa}U\mathsf{b} \\ V & \rightarrow & \epsilon \mid \mathsf{aaaaa}\,V\mathsf{bbb} \end{array}$$

3. (**12 points**) Let Equiv be a Java method that takes as its input the source code for two Java methods and determines whether or not they return the same values for all possible values of their input parameters. For example, consider the methods:

```
int mult1(int a, int b) {      int mult2(int a, int b) {      int mult3(int a, int b) {
    return(a * b);                 int p = 0;                     return(a & b);
}                                  for(int i = 0; i < a; i++) }
                                       p = p+b;
                                   return(p);
                               }
```

Equiv($\langle$mult1$\rangle$, $\langle$mult2$\rangle$) should return true, but Equiv($\langle$mult1$\rangle$, $\langle$mult3$\rangle$) should return false. Here, $\langle$mult1$\rangle$ denotes the source code for method mult1 and likewise for the other methods.

Show that it is impossible to write an implementation of Equiv that works for all possible Java methods. In particular, show that if a method such as Equiv could be written, then you could use it to solve the halting problem. Assume that all of these methods can use unlimitted amounts of memory.

Solution: In class we defined a function, $accept(M, w, n)$ which returns true if TM M accepts string w after at most n steps. This function is Turing computable and could be implemented as a Java method. Let accept be such a Java method. Likewise, we can have a Java method halt($\langle M\rangle$, $\langle w\rangle$, n) which returns true iff the Turing machine described by $\langle M\rangle$ halts after at most n steps when run with input $\langle w\rangle$. Given a particular TM, M, and a particular input string w for M, I'll define

```
boolean MacceptsW(java.math.BigInteger n) {
    return(halt(⟨M⟩, ⟨w⟩, n););
}
```

I used a BigInteger to allow calls to halt with n larger than the largest Java int (or long). I'll accept solutions that don't point out this detail. Now, I'll define

```
boolean alwaysFalse(java.math.BigInteger n) {
    return(false);
}
```

If MacceptsW is equivalent to alwaysFalse, then M does not halt when run with input w. Thus I could use the Equiv method to decide the (non-)halting problem. This shows that such a method cannot be written.

Note: Equiv is the Java equivalent of language E from homework 9, question 5, but that was a "hard" question and this is an easy one. In particular, you only have to show that you could use Equiv to solve the halting problem and not the "harder" problems considered in the homework version.

4. (**12 points**)

   (a) (**2 points**) Give a one or two sentence definition of what it means for a language to be in NP.

   Solution: A language is in NP if it can be decided by a non-deterministic Turing machine that runs in polynomial time.

   (b) (**2 points**) Give a one or two sentence definition of what it means for a language to be NP hard.

   Solution: A language, B, is NP hard if any language in NP is polynomial time reducible to B.

   (c) (**2 points**) Give a one or two sentence definition of what it means for a language to be NP complete.

   Solution: A language is NP complete if it is both in NP and NP hard. NP complete problems are in some sense the "hardest" problems in NP; for example, the appear to be harder than languages that can be decided in polynomial time by a deterministic (i.e. real) computer although such polynomial time languages are also in NP.

   (d) (**6 points**) State which of the languages below are in NP, which are NP hard, and which are NP complete:

   i. Strings consisting of an equal number of a's and b's.
   ii. Strings that describe a Turing machine that halts when run with the empty string as its input.
   iii. Strings that describe a satisfiable Boolean formula.
   iv. Strings that are the binary encoding of prime number.
   v. Strings that describe a graph, $G = (V, E)$, and an integer k, such that there is a set $U \subseteq V$ with $|U| = k$ such that for every edge $(v_1, v_2) \in E$ either $v_1 \in U$ or $v_2 \in U$ or both.
   vi. The empty language.

   Solution:

   In NP: Every language mentioned above except ii.

   NP hard: Languages ii, iii, and v are NP hard.
   For one point of extra credit, point out that languages i, and iv are NP hard iff $P = NP$.

   NP complete: Languages iii, and v are NP complete.
   As above, langauges i and iv are NP complete iff $P = NP$.

5. (**20 points**) A context-free grammar is *right linear* iff every rule is of the form $A \rightarrow xB$ or $A \rightarrow x$ where A and B are variables and x is a string of terminals. Prove that if G is a right-linear context-free grammar, then $L(G)$ is regular.

   Solution 1: Let $G = (V, \Sigma, R, S)$ be a right-linear CFG. We can assume without loss of generality that all rules of G are of the form $A \rightarrow cB$, $A \rightarrow Z$, or $Z \rightarrow \epsilon$ with $A, B \in V$ and $c \in \Sigma$ and where Z is a new variable for which there is exactly one rule: $Z \rightarrow \epsilon$. To show this, we have to handle three types of rules allowed in right-linear grammars that aren't in this simplified version:

   $A \rightarrow B$ with $x = \epsilon$): Delete this rule, and for each rule of the form $B \rightarrow xC$ add a rule $A \rightarrow xC$, and for each rule of the form $B \rightarrow x$, add a rule of the form $A \rightarrow x$.

   $A \rightarrow xB$ with $|x| > 1$: Delete this rule. Let $x = c_1 \cdots c_k$ with $c_1, \ldots, c_k \in \Sigma$. Add new variables AB_1, $\ldots AB_{k-1}$ and rules $A \rightarrow c_1 AB_1$, $AB_1 \rightarrow c_2 AB_2$, $\ldots AB_{k-2} \rightarrow c_{k-1} AB_{k-1}$, and $AB_{k-1} \rightarrow c_k B$.

   $A \rightarrow x$: Let $x = c_1 \cdots c_k$ with $c_1, \ldots, c_k \in \Sigma$. Add new variables AB_1, $\ldots AB_{k-11}$ and rules $A \rightarrow c_1 A_1$, $A_1 \rightarrow c_2 A_2$, $\ldots A_{k-2} \rightarrow c_{k-1} A_{k-1}$, and $A_{k-1} \rightarrow c_k Z$. If $A \rightarrow \epsilon$, we simply add the rule $A \rightarrow Z$.

   It is straightforward to show that each of these transformations preserves the language of the grammar.

   Define an NFA, $N = (V, \Sigma, \Delta, S, \{Z\})$ be a NFA where there is a transition from state A to state B on symbol c iff $A \rightarrow cB$ is a rule in R. If $A \rightarrow Z$, then there is an edge with label ϵ from state A to state Z.

   I'll now show that $L(N) = L(G)$. Because the languages of NFAs are regular, this shows that the languages generated by right-linear grammars are regular as well.

Let $w \in L(G)$: Then there is some derivation of w using the rules of G. We have

$$S \Rightarrow c_1V_1 \Rightarrow c_1c_2V_2 \Rightarrow \dots x_iV_i \Rightarrow \dots x_{i+1}V_{i+1} \Rightarrow \dots wZ \Rightarrow \dots w$$

A simple induction argument shows that we $x_iV_i \Rightarrow x_{i+1}V_{i+1}$ iff there is some $c \in \Sigma$ such that $x_{i+1} = x_ic$ and $V_i \rightarrow cV_{i+1} \in R$. Another simple induction argument shows that $V_i \in \Delta(S, x_i)$ from which we conclude $Z \in \Delta(S, w)$ which means that $w \in L(N)$ as required.

Let $w \in L(N)$: Let $w = c_1 \cdots c_k$ then there is some sequence of states $q_0 \dots q_k$ with $q_0 = S$, $q_k = Z$ and for all $i \in 0 \dots k-1$, $q_{i+1} \in \Delta(q_i, c_{i+1})$. A simple induction argument then shows that for all $i \in 1 \ddots k-1$, $S \overset{*}{\Rightarrow} c_1 \cdots c_iV_{i+1}$ and therefore $S \overset{*}{\Rightarrow} wZ \Rightarrow w$. Thus $w \in L(G)$ as required.

Solution 2: Here's a shorter solution using a GNFA as described in Sipser chapter 1.3.

Let $G = (V, \Sigma, R, S)$ be a right linear grammar. Define a GNFA (as described in Sipser, chapter 1.3) with states $V \cup \{S', Z\}$ where S' and Z are new symbols. For $A, B \in V$, there the edge from A to B is labeled with the regular expression

$$\bigcup_{x \mid (A \rightarrow xB) \in R'} x$$

The edge from A to B has a label of $\emptyset$ if there are no such rules. Likewise, the edge from A to Z is labeld with the regular expression

$$\bigcup_{x \mid (A \rightarrow x) \in R'} x$$

There is an edge from S' to S with a label of ϵ, and edges from S' to all other states labeled $\emptyset$. Observe that there are no incoming edges to S' and no outgoing edges from Z. Thus, F is a GNFA. Therefore, $L(F)$ is regular.

I'll now show that $L(F) = L(G)$. Let $s \in L(F)$. Then, we can find states $q_1, \dots, q_k$ such that $q_1 = S'$, $q_k = Z$, and strings $w_1, \dots w_k$ such that $s = w_1 \cdots w_k$, and for each $1 \leq i < k$, the edge from q_i to q_{i+1} is labeled with a regular expression that matches w_i. By the construction of F, we can show that for each $1 \leq i \leq k$, $S \overset{*}{\Rightarrow} w_1 \cdots w_iq_{i+1}$ and therefore $S \overset{*}{\Rightarrow} w_1 \cdots w_kZ \Rightarrow s$. Thus, $s \in L(G)$. A similar argument shows that if $s \in L(G)$, then we can use a derivation of s to find an accepting run of F and therefore $s \in L(F)$. Therefore, $L(F) = L(G)$, and $L(G)$ is regular as required.

6. (**20 points**) Let $A = \{\langle M \rangle \mid L(M) = (L(M))^{\mathcal{R}}\}$ where $\langle M \rangle$ is a string describing Turing machine M, and $(L(M))^{\mathcal{R}}$ is the language consisting of all strings whose reversals are in $L(M)$.

Determine whether or not A is Turing decidable, and give a short proof.

Solution 1: This is a property of the language of the Turing machine M. It is a non-trivial property: for example, a TM that accepts the empty language is in A but a TM that only accepts `01` is not in A. Thus, Rice's theorem applies (See HW9, Q4), and A is not Turing decidable.

Solution 2: OK, you're not required to know Rice's theorem. Here's a reduction from A_{TM}. Given M and w, construct a new TM, M' that on input s does the following:

```
If $s = 01$, accept.
Else if $s \neq 10$, reject.
Else
    Run $M$ on input $w$.
    If $M$ accepts $w$, then accept.
    Else if $M$ rejects $w$, then reject.
    Else $M$ loops and we never get here.
```

With this construction, M' recognizes $\{01, 10\}$ if M accepts w, and M' recognizes $\{01\}$ otherwise. Thus, $M' \in A$ iff M accepts w. I've reduced A_{TM} to A; therefore, A is undecidable.

7. (**20 points**) Let $\mathit{spaceBound}(M, w, n)$ be true iff Turing machine M accesses at most n different tape locations when run with input w. Let

$$\begin{aligned} B_1 &= \{M\#w \mid \mathit{spaceBound}(M, w, 2^{|w|})\} \\ B_2 &= \{M \mid \forall w.\ \mathit{spaceBound}(M, w, 2^{|w|})\} \end{aligned}$$

One of these langauages is decidable and one is not. Determine which is which and give a short proof for each answer.

Solution:

B_1 is decidable: Let Γ be the tape alphabet of M. Simulate M for $2^{|w|}|\Gamma|^{2^{|w|}}$ steps. If M ever steps outside of the first $2^{|w|}$ tape squares, then reject. If M halts without exceeding this space bound, then accept. Otherwise, M is still running. Note that there are $2^{|w|}|\Gamma|^{2^{|w|}}$ possible configurations with that use at most $2^{|w|}$ tape squares. Therefore, M is looping in the first $2^{|w|}$ tape squares and we can accept.

B_2 is not decidable: Given a TM M and an input string w for M, build a new TM, M' that on input x does the following:

```
If $x$ is a not valid computaitonal history for $M$ accepting $w$,
    then $M'$ rejects.
Else /* $x$ is a valid computational history for $M$ accepting $w$ */
    $M'$ writes an infinite string of 0's on its tape.
```

A TM can check a computational history using $|x| + 1$ space (the $+1$ is because it reads the blank follwoing x. Thus, $M' \in A$ iff M does not accept w. I've reduced $\overline{A_{TM}}$ to A; $\overline{A_{TM}}$ is undecidable, therefore A is undecidable as well.

Note: You **can't** use Rice's theorem for this problem because it is **not** a question about the language that M accepts, it is a question about how much space M uses.

8. (**20 points**) Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs with nodes V_i and edges E_i (with $i \in \{1, 2\}$). The SUBGRAPH ISOMORPHISM problem is to determine whether or not G_2 is isomorphic to a subgraph of G_1. Show that SUBGRAPH ISOMORPHISM is NP-complete.

Hints:

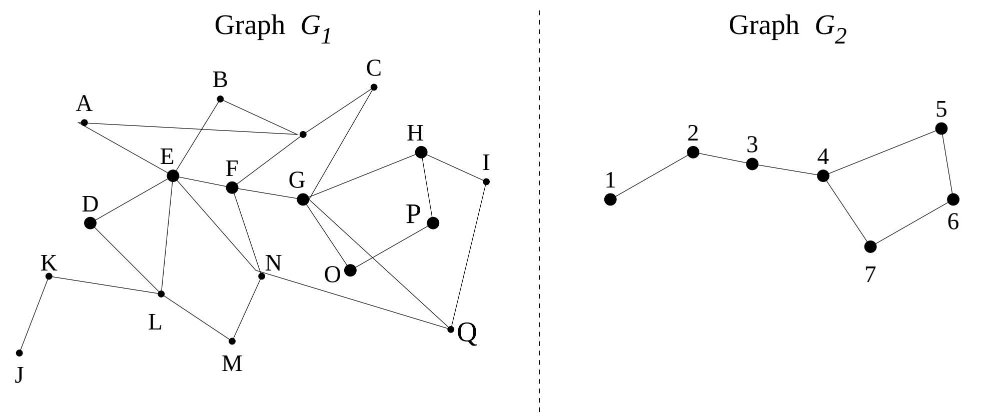
- Consider a reduction from CLIQUE.
- "Graph G is a *subgraph* of graph H if the nodes of G are a subset of the nodes of H, and the edges of G are the edges of H on the corresponding nodes." (Sipser, page 11).
- Graphs G and H are *isomorphic* if the nodes of G can be reordered so that it is identical to H. (see Sipser, exercise 7.11).
- See figure 1.

Solution:

GRAPH ISOMORPHISM is in NP: A list of vertices from G and how they pair with vertices from H suffices as a certificate. A deterministic TM can check in polynomial time that each vertex from H has a corresponding vertex from G and the equivalence of this subgraph of G with H.

GRAPH ISOMORPHISM is in NP-hard: Let (G, k) be an instance of CLIQUE. If k is greater than the number of nodes of G, then reject immediately. Otherwise, let H be the complete graph with k nodes. G has a clique of size k iff H is a isomorphic to a subgraph of G. Thus, this is a polynomial time reduction from clique to subgraph isomorphism which proves that subgraph isomorphism is NP hard.

Note: I first checked that k is at most the number of nodes in G. Otherwise, if one specified a clique size that was exponentially larger than the number of nodes in G, then the rest of the reduction would not be polynomial time. I won't take off points for solutions that miss this detail, but I'll give a point for extra credit if you noted it.



Graph G_2 is isomorphic to a subgraph of G_1 by the following correspondence of vertices: (D, E, F, G H, O, P) $\longleftrightarrow$ (1, 2, 3, 4, 5, 7, 6)

Figure 1: A subgraph isomorphism example.

GRAPH ISOMORPHISM is in NP-complete: I've shown above that subgraph isomorphism is in NP and that it is NP hard. Therefore, subgraph isomorphism is NP complete.

9. (**25 points**) Let D be a DFA, G be a CFG, and M be a TM. Define

$$\begin{array}{rcl} A(M,D) & = & \{y \mid \exists x \in L(M).\ xy \in L(D)\} \\ B & = & \{\langle M, D, y\rangle \mid y \in A(M,D)\} \\ C(M,G) & = & \{y \mid \exists x \in L(M).\ xy \in L(G)\} \end{array}$$

(a) (**9 points**) Prove that $A(M,D)$ is regular.

Solution: Let $D = (Q, \Sigma, \delta, q_0, F)$ and let

$$Q' = \{q \in Q \mid \exists x \in L(M).\ \delta(q_0, x) = q\}$$

Note that Q' is finite (even though we may not be able to decide what states are in it!). Now, note that

$$\begin{array}{rcll} A(M,D) & = & \{y \mid \exists x \in L(M).\ xy \in L(D)\}, & \text{def. } A(M,D) \\ & = & \{y \mid \exists x \in L(M).\ \delta(q_0, xy) \in F\}, & \text{def. } L(D) \\ & = & \{y \mid \exists x \in L(M).\ \delta(\delta(q_0, x), y) \in F\}, & \text{prop. of } \delta \\ & = & \{y \mid \exists q \in Q'.\ \delta(q, y) \in F\}, & \text{def. } Q' \\ & = & \bigcup_{q \in Q'} \{y \mid \delta(q,y) \in F\} & \end{array}$$

Now, let $D_q = (Q, \Sigma, \delta, q, F)$. This yields:

$$A(M,D) = \bigcup_{q \in Q'} L(D_Q)$$

Thus, $A(M,D)$ is the finite union of regular languages. Therefore, $A(M,D)$ is regular.

(b) (**8 points**) Prove that B is Turing recognizable but not Turing decidable.

Solution: Given D and y, let $X = \{\langle M\rangle \mid \exists x.\ (x \in L(M)) \wedge (xy \in L(D))\}$. Language X is a property of the language of TM M, and it is non-trivial. For example, I can choose D to be a DFA that recognizes $\{\mathsf{a}\}$ and let $y = \epsilon$. Then $\langle M\rangle \in X$ iff M accepts a. I can build a TM that only accepts

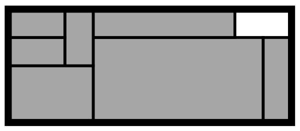
a, and another TM that recognized the empty language. Thus X is not trivial. Rice's theorem applies to show that B is not Turing Decidable.
To show that B is Turing recognizable, I'll show that it is recognized by a NTM, N. N just guesses a string for x. it then runs M on x to verify that M accepts x. Finally, N verifies that $xy \in L(D)$. Because TM's and NTM's recognize the same languages, B is Turing recognizable.

(c) (**8 points**) Prove that $C(M, G)$ is not necessarily context free.

Solution: Let M be a TM that recognizes $\{\mathsf{a}^{m^2} \mid m \in \mathbb{Z}^{\geq 0}\}$, and let G be a CFG that recognizes $\{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{Z}^{\geq 0}\}$. Then

$$C(M, G) \quad = \quad \{b^{m^2} \mid m \in \mathbb{Z}^{\geq 0}\}$$

which is not context free.
To show the last claim, let p be a proposed pumping constant for $C(M, G)$, and let $s = b^{p^2}$. Clearly $s \in C(M, G)$. However, if we pump s we will change its length by an amount between 1 and p which will produce a string of b's whose length is not a perfect square and thus is not in $C(M, G)$. Therefore, $C(M, G)$ does not satisfy the conditions of the pumping lemma and is not context free.



Rectangles (1, 2), (1, 2), (1, 2), (1,3), (1,5), (2,3), and (3,6)
can be packed into rectangle (10,4).

Figure 2: A rectangle packing example.

10. (**25 points**) Let $Q, R_1, R_2, \ldots R_k$ be rectangles. Each rectangle is specified by two integers, one for its width and the other for its height. The RECTANGLE PACKING problem is to determine if it is possible to place rectangles $R_1 \ldots R_k$ in rectangle Q such that none of the R_i rectangles overlap. Each rectangle may be placed at an arbitrary location and with an arbitrary orientation in Q as long as it is contained completely in Q. See figure 2 for an example.

    Show that RECTANGLE PACKING is strongly NP-complete.

    For **20 points**, you can show that RECTANGLE PACKING is NP-complete (without showing the *strongly* part required for a 25 point solution).

    Solution:

    RECTANGLE PACKING is in NP: This one is a little harder than I intended. By "arbitrary location and arbitrary orientation" I meant arbitrary location on the integer grid and either in the original orientation or rotated by 90 degrees. I'll prove that version and accept proofs for that version.
    With this restriction, proving that RECTANGLE PACKING is in NP is straightforward, just guess how to orient each rectangle and where to put its lower left corner.

    RECTANGLE PACKING is strongly NP hard: By reduction from 3-PARTITION. An instance of 3-partition consists of integers b and m, and a set, S, of $3m$ elements where for each $s \in S$ there is an associated "weight," $w(s) \in \mathbb{Z}^+$, $\sum_{s \in S} w(s) = mb$, and for each $s \in S$, $b/4 < w(s) < b/2$. The question is: can S be partitioned into m sets such that the sum of the elements in each set is equal to b?
    Given an instance of 3-PARTITION, construct an instance of RECTANGLE PACKING with $3m$ rectangles, R_1 through R_{3m} to be packed into a rectangle Q. The R_i rectangles correspond to the elements of S. In particular, let $S = \{s_1, \ldots s_{3m}\}$ and let rectangle i have width 1 and height $(2m)w(s_i)$. Rectangle Q has width m and height $2mb$. My claim is that this a positive instance of RECTANGLE PACKING iff the original 3-PARTITION problem was a positive instance.
    Consider a solution to RECTANGLE PACKING. Note that each rectangle has a height of at least $2m$ and Q has width of m. Therefore all of the rectangles must be placed in their tall-and-skinny orientation. In that orientation, each rectangle has a width of 1. Therefore, these rectangle form columns. In particular, there are m such columns, and these correspond to the subsets that we need for 3-PARTITION. Because each rectangle has a height, h, with $mb/2 < h < mb$, and the height of Q is $2mb$, each column has three rectangles. If the three rectangles for column i are $R_{i,1}$, $R_{i,2}$ and $R_{i,3}$, place the corresponding elements of S into set s_i. By the construction of the rectangles, these elements of S each have a weight that is the height of the corresponding rectangle divided by $2m$. Thus, the total weight of the elements in the set is $(2mb)/(2m) = b$ as required. Thus, this gives us a solution to the 3-PARTITION problem.
    Likewise, if we have a solution to the 3-PARTITION problem, S_1, S_2, $\ldots S_m$, we can construct a solution to the corresponding RECTANGE PACKING problem by arranging the rectangles in columns as described above, and choosing the rectangles in column i to correspond to the elements of S_i.
    Thus, the RECTANGLE PACKING problem constructed above has a solution iff the original 3-PARTITION PROBLEM has a solution.

RECTANGLE PACKING is strongly NP-complete: We've shown above that RECTANGLE PACKING is in NP and that it is strongly NP-hard. Thus, RECTANGLE PACKING is strongly NP complete as claimed.

11. The xkcd comic (**0 points**), see http://xkcd.com/287/. The obvious choice for the waiter is to bring seven orders of mixed-fruit as the restaurant patron did not specify that the appetizers needed to be distinct. $\$7 \times 2.15 = \$15.05$ as required. Now, if we consider each item on the appetizer menu to be a separate element of S (i.e. no duplicate appetizers), we first note that the sum over all individual appetizers is $\$21.80$, which means we can just as well look for which appetizers are not included; they must total $\$21.80 - \$15.05 = \$6.75$. We can take use the pseudo-polynomial, dynamic programming algorithm. Here are feasible amounts (in ascending order):

amount		order
2.15	=	MF, (i.e. Mixed Fruit)
2.75	=	FF, (i.e. French Fries)
3.35	=	SS, (i.e. Side Salad)
3.55	=	HW, (i.e. Hot Wings)
4.20	=	MS, (i.e. Mozzarella Sticks)
4.90	=	$MF + FF$, (yuck!)
5.50	=	$MF + SS$, (for the health freak)
5.70	=	$MF + HW$
5.80	=	SP, $(i.e. SamplerPlate)$
6.10	=	$FF + SS$
6.30	=	$FF + HW$, (pub fare?)
6.35	=	$MF + MS$
6.90	=	$SS + HW$

We conclude that there is no way for the waiter to deliver appetizers totaling $15.05 without some duplicatoins of appetizers.