York University
CSE 2001 Fall 2017 – Assignment 3 of 4
Instructor: Jeff Edmonds

1. You are to give me a context free grammar to generate the language of all tuples of tuples and characters $\{a, b, c\}$. For example, $\langle a, a, \langle b, c, \langle b\rangle\rangle, a, \langle\rangle\rangle$. Note that the terminal symbols are the characters 'a', 'b', 'c', '$\langle$', '$\rangle$', and ','. Note the tuples can be of arbitrary lengths. Hint, use the following nonterminal symbols:

   - T to represent a tuple. (The start symbol).
   - L to represent a list of tuples and characters $\{a, b, c\}$. For example, "$a, a, \langle b, c, \langle b\rangle\rangle, a, \langle\rangle$".
   - I to represent one item, namely either one tuple or one character from $\{a, b, c\}$.

   Be sure that the brackets are formed in matching pairs and that the commas are formed to appear singly between items.

   Demonstrate your grammar by giving a parsing of the string $\langle a, \langle\rangle, b\rangle$

   - Answer:
     A tuple is a list with $\langle\ \rangle$ brackets around it.
     $T \Rightarrow \langle L\rangle$
     A list is a sequence of items separated by commas. Because a list can be of an arbitrary length and a grammar rule must be of some constant length, we much describe the concept "list" recursively. A list of length one consists of a single item. A longer "list" consists of a first item, followed by a comma, followed by a shorter "list".
     $L \Rightarrow I, L \mid I$
     A list could also be an empty list. However, adding the rule $L \Rightarrow \epsilon$ would allow "a," to be a list. One solution is to define a list as $L \Rightarrow L' \mid \epsilon$
     where L' is a list with at least one item.
     Here a quicker solution is to restrict "lists" to having at least one item and to include the rule $T \Rightarrow \langle\rangle$
     An item is either a tuple or a character.
     $I \Rightarrow T \mid a \mid b \mid c$

   Answer:

```
        T
<       L    >
<I,     L    >
<I,     I, L>
<I,     I, I>
<a,     T, b>
<a,    <>, b>
```

2. Parsing: If possible, write pseudo code for parsing the following grammar.

$$\begin{aligned} S &\Rightarrow A\ a\ S \\ &\Rightarrow A\ b \\ &\Rightarrow A \end{aligned}$$

   A parsing can be presented as a little picture of the parse tree or as a tuple as done in the assignment.

   - Answer:

     algorithm $GetS\,(s, i)$

     $\langle$***pre−cond***$\rangle$**:** s is a string of tokens and i is an index that indicates a starting point within s.

$\langle \textbf{\textit{post-cond}} \rangle$**:** The output consists of a parsing p of the longest substring $s[i], s[i+1], \ldots, s[j-1]$ of s that starts at index i and is a valid S. The output also includes the index j of the token that comes immediately after the parsed S .

```
begin
    ⟨p_A, j_A⟩ = GetA(s, i)
    if( s[j_A] = 'a' )
        ⟨p_S, j_S⟩ = GetS(s, j_A + 1)
        return( ⟨p_A a p_S⟩ , j_S)
    elseif( s[j_A] = 'b' )
        return( ⟨p_A b⟩ , j_A + 1)
    else
        return( ⟨p_A⟩ , j_A)
end algorithm
```

3. Consider alphabets $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{p, q, r, s, t\}$.

   Σ_1^* consists of all finite strings over Σ_1. Similarly Σ_2^*. We want to determine whether or not Σ_1^* and Σ_2^* have the same *size*. One way of proving that they do is to set up a bijection between them. This can be done, but it is tricky.

   Clearly $|\Sigma_1^*| \leq |\Sigma_2^*|$. Hence, what remains is to determine whether or not $|\Sigma_1^*| \geq |\Sigma_2^*|$. This is true if and only if there is a mapping (encoding) $f : \Sigma_2^* \rightarrow \Sigma_1^*$ such that each string in Σ_2^* is mapped to a unique string in Σ_1^*. (It might not be a bijection because some strings in Σ_1^* might not get mapped to.) In other words, can you use strings over Σ_1 to *name* all strings over Σ_2.

   If you think that such mapping exists, explain why and give pseudo code for computing f. If you think that no such mappings f exists, carefully explain why. Recall that Jeff says that a set is countably-infinite in size if and only if each element in the set has a unique finite description.

   - Answer: The sets Σ_1^* and Σ_2^* are the same size. They are both countably infinite. Using Jeff's definition, this is because each string in each set has a finite description.

     Pseudo code for computing f would go as follows. A string in Σ_2^* is a string of the characters p, q, r, s and t. Encode each p with the two letters aa, encode q with ab, r with ac, s with ba, and t with bb. Concatenating all these codes together gives a unique string in Σ_1^*.

4. The Halting Problem is Undecidable

   (a) Use first order logic to state that problem P is computable. Might the TM mentioned in this sentence fail to halt on some input?

   - Answer: $\exists M \; \forall I \; P(I) = M(I)$. For P to be computable/decidable, this M must on each input halt and give the correct answer.

   (b) Suppose I give you as an oracle a Universal Turing Machine. With this extra help, does this change with whether you can solve the Halting problem?

   - Answer: No help. We do have a TM for a Universal Turing Machine. And the Halting problem is not computable.

   (c) Suppose you think it undignified to feed a TM M a description "M" of itself. Instead, of making M's nemesis be $I_M =$ "M", lets instead define $I_M = F(M)$ where $F(M)$ is the description of what the TM M fears the most. For example, $F(M_{Sherlock\ Homes}) =$ "$Moriarty$" and $F(M_{Super\ Man}) =$ "$Kryptonite$".

      i. Suppose $F(M)$ is distinct for each TM M, i.e. $\forall M, M',\ M \neq M' \Rightarrow F(M) \neq F(M')$. Using this new nemesis input, give the proof that there is a problem P_{hard} that is uncomputable. This is done by giving the first order logic statement and then playing the game.
      (Six quick sentences, i.e. I removed all the chat from the posted proof.)
      If you have memorized the proof in the slides and you put it here unchanged you will get 60%.

- Answer: Proving the first order logic statement: $\exists P_{hard}\ \forall M\ \exists I_M\ M(I_M) \neq P_{hard}(I_M)$
  Define problem P_{hard} so that $P_{hard}(F(M))$ is anything different than $M(F(M))$.
  Let M be an arbitrary TM.
  Define input I_M to be M's nemesis $F(M)$.
  We win because $M(I_M) \neq P_{hard}(I_M)$.
  This completes the proof that there is an uncomputable computation problem.

ii. (Bonus Question so no marks for a blank):
Suppose $F(M)$ is not distinct for each TM M, i.e. $\exists M, M',\ M \neq M'$ and $F(M) = F(M')$. Suppose we want P_{hard} to be a language, i.e. its output is in $\{Yes, No\}$. What does wrong in your previous proof?

- Answer: Suppose M and M' are such that $F(M) = F(M') = I$. We both define $P_{hard}(I)$ to be anything different than $M(I)$ and anything different than $M'(I)$. But what if $M(I) = No$, $M'(I) = Yes$, and $P_{hard}(I)$ must be in $\{Yes, No\}$. Then we have a problem.