**COT 6410 Spring2014 Final Exam Sample E1 Key**

**1**. Let set **A** be recursive, **B** be re non-recursive and **C** be non-re. Choosing from among **(REC)** **recursive**, **(RE)** **re non-recursive**, **(NR)** **non-re**, categorize the set **D** in each of a) through d) by listing **all** possible categories. No justification is required.

**a.) D = ~C RE, NR**

**b.) D ⊆ (A∪C) REC, RE, NR**

**c.) D = ~B NR**

**d.) D = B − A REC, RE**

**2**. Choosing from among **(D)** **decidable**, **(U)** **undecidable**, **(?)** **unknown**, categorize each of the following decision problems. No proofs are required.

|  |  |  |  |
| --- | --- | --- | --- |
| **Problem / Language Class** | **Regular** | **Context Free** | **Context Sensitive** |
| L = Σ\* ? | ***D*** | ***U*** | ***U*** |
| **L = φ ?** | ***D*** | ***D*** | ***U*** |
| **L = L2 ?** | ***D*** | ***U*** | ***U*** |
| **x ∈ L2, for arbitrary x ?** | ***D*** | ***D*** | ***D*** |

**3.** Use **PCP** to show the undecidability of the problem to determine if the intersection of two context free languages is non-empty. That is, show how to create two grammars **GA** and **GB** based on some instance **P = <<x1,x2,…,xn>, <y1,y2,…,yn>>** of **PCP**, such that **L(GA) ∩ L(GB) ≠ φ** iff **P** has a solution. Assume that **P** is over the alphabet **Σ**. You should discuss what languages your grammars produce and why this is relevant, but no formal proof is required.

**GA = ( { A } , Σ ∪ { [ i ] | 1≤i≤n } , A , PA } GB = ( { B } , Σ ∪ { [ i ] | 1≤i≤n } , B , PB }**

**PA : A → xi A [ i ] | xi [ i ] PB : A → yi B [ i ] | yi [ i ]**

**L(GA) = { xi1 xi2 … xip [ip] … [i2] [i1] | p ≥ 1, 1 ≤ it ≤ n, 1 ≤ t ≤ p }**

**L(GB) = { yj1 yj2 … yjq [jq] … [j2] [j1] | q ≥ 1, 1 ≤ ju ≤ n, 1 ≤ u ≤ q }**

**L(GA) ∩ L(GB) = { w[kr] … [k2] [k1] | r ≥ 1, 1 ≤ kv ≤ n, 1 ≤ v ≤ r }, where**

**w = xk1 xk2 … xkr = yk1 yk2 … ykr**

**If L(GA) ∩ L(GB) ≠ φ then such a w exists and thus k1 , k2 , … , kr is a solution to this instance of PCP. This shows that a decision procedure for the non-emptiness of the intersection of CFLs implies a decision procedure for PCP, which we have already shown is undecidable. Hence, the non-emptiness of the intersection of CFLs is undecidable. Q.E.D.**

**4.** Consider the set of indices **CONSTANT = { f | ∃K ∀y [ ϕf(y) = K ] }**. Use Rice’s Theorem to show that **CONSTANT** is not recursive. Hint: There are two properties that must be demonstrated.

**First, show CONSTANT is non-trivial.**

**Z(x) = 0, which can be implemented as the TM R, is in CONSTANT**

**S(x) = x+1, which can be implemented by the TM C11R, is not in CONSTANT**

**Thus, CONSTANT is non-trivial**

**Second, let f, g be two arbitrary computable functions with the same I/O behavior.**

**That is, ∀x, if f(x) is defined, then f(x) = g(x); otherwise both diverge, i.e., f(x)↑ and g(x)↑**

**Now, f ∈ CONSTANT**

**⇔ ∃K ∀x [ f(x) = K ] by definition of CONSTANT**

**⇔ ∀x [ g(x) = C ] where C is the instance of K above, since ∀x [ f(x) = g(x) ]**

**⇔ ∃K ∀x [ g(x) = K ] from above**

**⇔ g ∈ CONSTANT by definition of CONSTANT**

**Since CONSTANT meets both conditions of Rice’s Theorem, it is undecidable. Q.E.D.**

**5.** Show that **CONSTANT ≡m TOT**, where **TOT = { f | ∀y ϕf(y)↓ }**.

**CONSTANT ≤m TOT**

**Let f be an arbitrary effective procedure.**

**Define gf by**

**gf (0) = f(0)**

**gf (y+1) = f(y+1) + μ z [f(y+1) = f(y) ]**

**Now, if f ∈ CONSTANT then ∀y [ f(y)↓ and [ f(y+1) = f(y) ] ].**

**Under this circumstance, μ z [f(y+1) = f(y) ] is 0 for all y and gf (y) = f(y) for all y.   
Clearly, then gf ∈ TOT**

**If, however, f ∉ CONSTANT then ∃y [f(y+1) ≠ f(y) ] and thus, ∃y gf (y)↑.**

**Choose the least y meeting this condition.**

**If f(y)↑ then gf (y)↑ since f(y) is in gf (y)’s definition (the 1st term).**

**If f(y)↓ but [f(y+1) ≠ f(y)] then gf (y)↑ since μ z [ f(y+1) = f(y) ]↑ (the 2nd term).**

**Clearly, then gf ∉ TOT**

**Combining these, f ∈ CONSTANT ⇔ gf ∈ TOT and thus CONSTANT ≤m TOT**

**TOT ≤m CONSTANT**

**Let f be an an arbitrary effective procedure.**

**Define gf by**

**gf (y) = f(y) – f(y)**

**Now, if f ∈ TOT then ∀y [ f(y)↓ ] and thus ∀y gf (y) = 0 . Clearly, then gf ∈ CONSTANT**

**If, however, f ∉ TOT then ∃y [f(y)↑ ] and thus, ∃y [gf (y)↑]. Clearly , then gf ∉ CONSTANT**

**Combining these, f ∈ TOT ⇔ gf ∈ CONSTANT and thus TOT ≤m CONSTANT**

**Hence, CONSTANT ≡m TOT. Q.E.D.**

**6.** Why does Rice’s Theorem have nothing to say about the following? Explain by showing some condition of Rice’s Theorem that is not met by the stated property.

**AT-LEAST-LINEAR = { f | ∀y ϕf(y) converges in no fewer than y steps }**.

**We can deny the 2nd condition of Rice’s Theorem since**

**Z, where Z(x) = 0, implemented by the TM R converges in one step no matter what x is and hence is not in AT-LEAST-LINEAR**

**Z’, defined by the TM *R L* R, is in AT-LEAST-LINEAR**

**However, ∀x [ Z(x) = Z’(x) ], so they have the same I/O behavior and yet one is in and the other is out of AT-LEAST-LINEAR, denying the 2nd condition of Rice’s Theorem**

**7.** The trace language of a computational device like a Turing Machine is a language of the form  
**Trace = { C1#C2# … Cn# | Ci ⇒ Ci+1, 1 ≤ i < n }  
Trace** is Context Sensitive, non-Context Free. Actually, a trace language typically has every other configuration word reversed, but the concept is the same. Oddly, the complement of such a trace is Context Free. Explain what makes its complement a **CFL**. In other words, describe the characteristics of this complement and why these characteristics are amenable to a **CFG** description.

**The complement of a trace needs to include strings that either do not look like a trace (that’s easy) or look like one, but have one or more errors. By one or more errors, we just mean that there is a pair Cj#Cj+1# where it is not the case that Cj ⇒ Cj+1. A PDA can guess which configuration starts this pair, push that configuration into its stack and check that the next one is in error (of course, this generally means one element of the pair is reversed). Such checking is within the capabilities of a PDA.**

**8.** We demonstrated a proof that the context sensitive languages are not closed under homomorphism, To start, we assumed **G = (N, Σ, S, P)** is an arbitrary Phrase Structured Grammar, with **N** its set of non-terminals, **Σ** its terminal alphabet, **S** its starting non-terminal and **P** its productions (rules). Since **G** is a PSG, it can have length increasing, length preserving and length decreasing rules. We wished to convert **G** to a CSG, **G’ = (N’, Σ’, S’, P’)** where there are no rules that are length decreasing (since a CSG cannot have these). We developed a way to pad the length decreasing rules from **G** and then a homomorphism that gets rid of these padding characters. Define **G’** and the homomorphism **h** that we discussed in class and then briefly discuss why this new grammar and homomorphism combine so **h(L(G’)) = L(G)**, thereby showing that all re sets are the homomorphic images of CSLs.

**Define N’ = N ∪ {S’, D}, where D and S’ are new symbols;**

**Σ’ = Σ ∪ {$}, where $ is a new symbol;**

**P’ contains**

**S’ → S$ is in P’**

**If α → β is in P and |α| ≤ |β|, then α → β is in P’**

**If α → β is in P and |α| > |β|, then α → βDk is in P’, where k = |α| - |β**

**Dx → xD is in P’, for all x ∈ N ∪ Σ**

**D$ → $$ is in P’**

**It is clear that these rules are all length increasing or length preserving and hence G’ is a CSG.   
L(G’) = {w$j | w ∈ L(G) and j is some integer >0 }**

**Define the homomorphism h by**

**h(a) = a for all a ∈ Σ**

**h($) = λ (the string of length 0)**

**h(L(G’)) = { w | w∈ L(G) } = L(G)**

**This completes our constructive justification.**