**Teoría de la Computación y Verificación de Programas 2020. Examen Fecha 1.**

*Instrucciones:*

1. *El examen es INDIVIDUAL.*
2. *Al recibir el examen, enseguida enviar un mail a* [*rrosenfeld@practia.global*](mailto:rrosenfeld@practia.global) *para notificar que fue recibido.*
3. *Tienen 2 HORAS para responder el máximo de preguntas que puedan, no se detengan mucho en ninguna.*
4. *Tienen que responder preguntas de las 3 UNIDADES..*
5. *Responder sobre este MISMO EMAIL O SOBRE EL WORD ADJUNTO, LO QUE PREFIERAN, SIEMPRE CADA RESPUESTA DEBAJO DE CADA PREGUNTA.*
6. *En el asunto del email de respuesta, poner NOMBRE COMPLETO DEL ALUMNO, DNI Y LEGAJO.*

*¡Mucha suerte!*

**Parte 1. Computabilidad.**

1. ¿Qué es un problema de búsqueda y qué es un problema de decisión? Dar un ejemplo de cada uno.
2. Construir una Máquina de Turing (MT) que acepte el lenguaje de todas las cadenas de cualquier longitud con 0 y 1 pero que tengan sólo 0 o sólo 1. P.ej.: 0, 1, 00, 11, 000, 111, etc.
3. Construir un Autómata Finito (AF) que acepte el lenguaje de todas las cadenas de ceros y unos que empiecen con 1 y terminen con 0. P.ej.: 10, 1010, 111110, etc.
4. Dar la idea general de la prueba de: si L1 ∈ R y L2 ∈ R entonces también L1 ⋂ L2 ∈ R.
5. Dar la idea general de la prueba de: si L1 ∈ R y L2 ∈ R entonces también L1 ⋃ L2 ∈ R.
6. Probar que R  CO-RE.
7. Dar la idea general de la prueba de que todo lenguaje finito es recursivo.
8. Dar la idea general de la prueba de que es recursivo el lenguaje L = {(<Mk>, k) | <Mk> es el código de la MT k-ésima según el orden canónico}.
9. Comentar la idea general para detectar cuándo entra en loop una MT con una sola cinta cuyo cabezal se desplaza en un espacio acotado de K celdas.
10. Dar la idea general de la prueba de que el lenguaje L = {<M> | L(M) tiene por lo menos una cadena de la forma (ab)n con n ≥ 1} es recursivamente numerable.
11. Se prueba que LƩ\* = {<M> | L(M) = Ʃ\*} ∉ RE y que L∅C = {<M> | L(M) ≠ ∅} ∈ RE. Justificar por qué no puede existir una reducción de LƩ\* a L∅C.
12. Dar la idea general de la prueba de que todo lenguaje de RE se reduce a LU.
13. ¿Qué es una MT con oráculo? ¿Cuándo existe una Turing-reducción de un lenguaje L1 a un lenguaje L2, lo que se anota así: L1 ≤T L2?
14. Sea L algún lenguaje de RE, y sea M una MT que acepta L.
    1. Indicar, justificando la respuesta, por qué la siguiente MT con oráculo HP no reconoce L: dado w, invoca al oráculo HP con (<M>, w), y acepta o rechaza según el oráculo responda sí o no, respectivamente.
    2. Corregir la MT con oráculo HP de (a), de manera tal que ahora sí acepte L.
15. Dar la idea general de la prueba de que LU se Turing-reduce a HP.
16. Dar la idea general de la prueba de que si L1 ≤T L2 y L2 ∈ R, entonces L1 ∈ R.
17. Dar la idea general de la prueba de las siguientes implicaciones: A ≤T B ⟶ AC ≤T B y A ≤T B ⟶ A ≤T BC.
18. Aplicando el Teorema de Rice, probar que el siguiente lenguaje L no es recursivo: L = {<M> | las cadenas de L(M) tienen igual cantidad de 1 y de 0}.

**Parte 2. Complejidad Computacional.**

1. ¿Cuándo un lenguaje pertenece a la clase TIME (T(n))? ¿Cuándo un lenguaje pertenece a la clase SPACE(S(n))?
2. Dar la idea general de la prueba de que L = {(<M>, w, 1k) | M es una MT determinística con una sola cinta que acepta w en a lo sumo k pasos} ∈ P.
3. Dar la idea general de la prueba de que el problema SAT está en NP.
4. El lenguaje de números M = {(a1,…,an, b) | existe un subconjunto de los ai que suman b} representa el problema de la mochila. Dar la idea general de la prueba de que M está en NP.
5. Explicar el concepto “certificado suscinto”. Dar un ejemplo.
6. Sea L1  P y sea L2 el lenguaje de todos los códigos de MT. Dar la idea general de la prueba de que L1 αP L2.
7. Probar que para todo lenguaje L se cumple L αP L.
8. Dados dos lenguajes L1 y L2, se sabe que existe una reducción polinomial de L1 a L2 y una reducción polinomial de L2 a L1. Probar que si L1 es un lenguaje NP-completo entonces lo es también L2.
9. Asumiendo P ≠ NP, sea un lenguaje L ∈ NP. Indicar en cada caso si lo enunciado se cumple o no, justificando la respuesta:
   1. Existe una reducción polinomial de L a SAT.
   2. No existe una reducción polinomial de SAT a L.
   3. Si LC ∈ NP, entonces L está en P.
10. Explicar cuándo se sospecha que un lenguaje está en la clase NPI. Dar un ejemplo.
11. ¿Cuál es la clase espacial que se considera como clase de problemas de resolución eficiente en el marco de la complejidad espacial? Justificar.
12. Sea M una MT con una sola cinta que trabaja en tiempo T(n). ¿Cuánto mide a lo sumo una configuración (contenido de la cinta) de una computación de M?
13. ¿Por qué asumiendo P ≠ NP y sabiendo que SAT es NP-completo, el problema de búsqueda FSAT no puede tener resolución temporal polinomial?
14. ¿Cuándo un problema A (de decisión o de búsqueda) es Cook-reducible a un problema B (de decisión o de búsqueda), lo que se expresa así: A ≤C B?
15. Explicar por qué si A ≤C B, B es tan o más difícil que A en el sentido de la complejidad temporal.
16. Definir conceptualmente qué es una aproximación polinomial, en el marco de los problemas de optimización (búsqueda de máximo o de mínimo).
17. Dar la idea general de la prueba de que P  RP.
18. ¿Por qué las máquinas cuánticas ponen en duda la Tesis Fuerte de Church-Turing?

**Parte 3. Verificación de Programas.**

1. ¿Qué es un estado? Explicar además el concepto de variante de un estado σ, expresado así: σ[x|n].
2. Justificar en cada caso por qué son verdaderos los enunciados siguientes:
   1. Que exista algún σ tal que σ | p y val(π (S, σ)) | q, no implica que | {p} S {q}.
   2. Que exista algún σ tal que σ | p y val(π (S, σ)) |≠ q, no implica que | {p} S {q}.
   3. Que exista algún σ tal que σ | p y val(π (S, σ)) |≠ q, no implica que | <p> S <q>.
3. Comente brevemente la diferencia entre la definición operacional y la definición denotacional (o funcional, o matemática) de la semántica de un lenguaje de programación.
4. Probar, usando la definición semántica del lenguaje de programación PLW, que la instrucción if B then S1 else S2 fi es equivalente (hace lo mismo) a la instrucción if not B then S2 else S1 fi.
5. ¿Qué es una especificación?
6. Especificar un programa que, definido sobre los números enteros, tenga como inputs dos números y calcule el mayor de los dos.
7. Diferenciar prueba a posteriori de desarrollo sistemático de programas. ¿Cuál es la idea del segundo concepto, basado en lo que hemos estudiado de verificación de programas?
8. Probar que se cumple {x = 50} x := 100 {x = 100}:
   1. Semánticamente, es decir empleando la definición semántica de correctitud parcial.
   2. Sintácticamente, es decir empleando el método H de verificación de correctitud parcial.
9. Justificar por qué no se cumple <true> while k ≠ 0 do skip do <k = 0>.
10. Indicar si se cumplen o no, justificando las respuestas, las siguientes ternas:
    1. <true> S <true>, cualquiera sea el programa S.
    2. <p> x := 0 <p  x = 0>, cualquiera sea la aserción p.
11. Probar con el método H, usando el invariante true, la terna {true} while true do skip od {false}.
12. Definir los conceptos “sensatez” y “completitud”, relacionados con los métodos de verificación de programas.
13. Probar la sensatez de la regla:   {p  q} S {r} , {p  ¬ q} S {r}

{p} S {r}

1. Definir el concepto de “composicionalidad” de un método de verificación de programas.
2. Definir los conceptos “invariante” y “variante (o función cota)” asociados a una instrucción while.
3. ¿Cómo se relacionan los órdenes parciales bien fundados con las pruebas de terminación de los programas?
4. ¿Por qué se dice que la completitud del método H es “relativa”?