PRUEBA DE EVALUACIÓN CONTINUA NÚMERO 1

SOLUCIONES

ENUNCIADOS DE EXAMEN:

1. Cuestionario. Este cuestionario se compone de diez preguntas de tipo test de respuesta V/F. Cada respuesta correcta aporta 0.25 puntos, cada respuesta incorrecta resta 0.1 puntos, y las repuestas incorrectas no puntúan. La puntuación máxima del test es de [2.5 puntos]. Marque con un aspa (×) la respuesta que considere oportuna así.

CUESTIONARIO:

a) Todos los modelos de cálculo teórico (de computación) son equivalentes: □ V ☒ F (basta dar unos sencillos contraejemplos: while-loop no es equivalente a L, su universo de funciones es más reducido; si a L le quitamos la instrucción de bifurcación el modelo resultante no es equivalente a L, etc…)

b) Existen funciones URM-computables que no son L-computables: □ V ☒ F (URM y L son lenguajes con capacidad de computación Universal (Davis, 1994 y Cutland, 1980-1994); esto es, calculan las mismas funciones. Es fácil imaginar, aunque no tan fácil escribir, teoremas de transformación entre ambos).

c) La While-Loop computabilidad amplía la clase de funciones que puede computar L: □ V ☒ F (no es así: muy al contrario, el modelo while-loop calcula un subconjunto del conjunto de funciones que puede calcular L)

d) Las trayectorias en el espacio de configuraciones del modelo While-Loop siempre son finitas: ☒ V □ F (sobre esto, no hay mucho que decir…)

e) Considere una función de k variables $f(x_1, \dots, x_k)$, que es totalmente computable bajo L y bajo URM; entonces, las trayectorias (computaciones) en los espacios de configuraciones de ambos modelos de la función son iguales, siempre que la función se compute en ambos modelos para los mismos valores de $(x_1, \dots, x_k)$: □ V ☒ F (dado que según el modelo la función se calculará con instancias específicas del mismo, con diferente número de instrucciones y, en su caso, control iterativo diferente, podemos decir que las trayectorias serán finitas bajo ambos modelos pero en general, ni mucho menos iguales)

f) Para cualquier función $f(x_1, \dots, x_k)$, si f es URM-computable, $f(x_1, \dots, x_k) + 1$ también lo será: ☒ V □ F (bastará tomar al programa que calcula a la función original y añadirle la instrucción S(1))

g) En el modelo L, la computabilidad parcial de una función $\vartheta(x)$ tal que $\vartheta(k) = \uparrow$ si k es par, estando definida en el resto del dominio, es algo que es algo intrínseco a la función $\vartheta(x)$: ☒ V □ F (es algo propio de la función; si una función no está definida en un punto, el programa que la calcule dará igual; cualquiera de ellos deberá hacer lo mismo que la función, y no dar output en ese punto… o no estaremos calculando a la función)

h) Toda función $p(x) = \sum_{i=0}^{n} a_i x^i$ con coeficientes $a_i \in \mathbb{N}$ es siempre totalmente L-computable: ☒ V □ F (sobre esto no hay mucho que decir…pues suma, producto y potencia son funciones L-computables)

i) La función $\gcd(x_1, x_2)$ es URM-computable: ☒ V ☐ F (pues se trataría de dar un URM-programa, que es relativamente fácil –o no- de imaginar… pero sí, es urm-computable; veáse Cutland 1994 si de desean más detalles).

j) La función $f(x) = x \; mod \; 5$ no es L-computable: ☐ V ☒ F

(sí que lo es; véase el ejercicio para $x \; mod \; 3$ ya planteado, y se tiene con un par de cambios…)

2. Sea $\rho(x)$ una función URM-computable en sentido total. Demuestre que para cada $n \geq 1$, la función $\rho_n(x) = \rho(x) + n$ es también URM-computable en sentido total. [1 punto].

- Caso base: para $n = 1$, hay que probar que $\rho_1(x) = \rho(x) + 1$ es Urm-computable. Se nos indica que $\rho(x)$ es Urm-computable, y existe por tanto un Urm-programa que la calcula. Sea P dicho programa. Construimos un nuevo programa: $Q \equiv P \; S(1)$. Este programa sitúa en R_1 el natural $\rho(x)$, como resultado de ejecutar las instrucciones de P, y a continuación incrementar en uno dicho registro, que ahora vale $\rho(x) + 1$. Pero entonces $\alpha_Q^{(1)}(x) = \; \rho(x) + 1 = \rho_1(x)$, como queríamos, y hemos encontrado un URM- programa que calcula a $\rho_1(x)$,
- Hipótesis de inducción: suponemos que la función $\rho_n(x) = \rho(x) + n$ es Urm-computable.
- Fase inductiva: probamos que $\rho_{n+1}(x) = \rho(x) + n + 1$ es Urm-computable. Pero podemos poner que $\rho_{n+1}(x) = \rho_n(x) + 1$. Por H.I. sabemos que $\rho_n(x)$ es Urm-computable, y disponemos de un Urm-programa T que la calcula. El programa $R \equiv T \; S(1)$ incrementa en uno el registro R_1 que contiene un valor igual $\rho(x) + n$ tras ejecutar T, y por tanto su nuevo valor es $(\rho(x) + n) + 1$. Pero entonces, $\alpha_T^{(1)}(x) = \; (\rho(x) + n) + 1 = \rho_{n+1}(x)$, como queríamos.

3. Demuestre que la función $f(x) = x/2$ si x es par, $f(x) = \uparrow$ en otro caso, es parcialmente URM-computable. Proponga computaciones para $f(2)$ y $f(1)$. La computación para $f(1)$ debe mostrar claramente que en este punto la función no está definida. [1 punto].

Hay muchas propuestas posibles de URM-programas que calculan a la función propuesta. Aquí va uno, aunque obviamente, **cualquier otro funcionalmente válido ha sido admitido**:

```
J(1,2,6)
S(3)
S(2)
S(2)
J(1,1,1)
T(3,1)
```

No pongo el cálculo de las computaciones (al fín y al cabo lo podéis trazar con el simulador) y simplemente diré dos cosas que deberían bastar:

- Si el input es un número par x el programa debe situar en R_1 como output $x/2$, y la computación debe terminar en una configuración final coherente con el modelo.

- Si el input es un número impar x la "computación" no terminará, y se obtendrá un comportamiento de lazo indefinido.

4. Considere los conjuntos de funciones siguientes: [1.5 puntos]

$COMP - L = \{f: f\ es\ L - parcialmente\ computable\}$

$COMP - Urm = \{f: f\ es\ URM - parcialmente\ computable\}$

$COMP - Wl = \{f: f\ es\ While - Loop\ computable\}$

- Establezca justificando el por qué, las relaciones de inclusión entre todos ellos. La ausencia de la justificación solicitada invalidará la respuesta.

Puesto que $COMP - L$ y $COMP - URM$ son equivalentes y calculan las mismas funciones (esto, en el Tema 1, ha sido establecido) podemos poner que $COMP - L = COMP - URM$, de donde trivialmente, se deriva que $COMP - L \subseteq COMP - Urm$ y que $COMP - L \supseteq COMP - Urm$. Por otra parte, sabemos que $COMP - Wl$ únicamente calcula funciones totales, de donde $COMP - Wl \subseteq COMP - L$ y $COMP - Wl \subseteq COMP - Urm$. Los recíprocos no son ciertos. Hay infinidad de funciones parcialmente computables (con L y con URM) que no son while-loop computables, pero basta con una para demostrarlo: escójase la que se quiera; luego $COMP - L \nsubseteq COMP - Wl$ y $COMP - Urm \nsubseteq COMP - Wl$.

- ¿Cuál es entonces el resultado de intersectar los tres conjuntos propuestos?

La intersección de los tres conjuntos proporciona los elementos comunes a todos ellos, en este caso: $COMP - L \cap COMP - URM \cap\ COMP - Wl = COMP - Wl$.

5. Considere el siguiente L-programa, denotado por P, que utiliza las macros de suma y resta parcial. Se pide: [1 punto]

```
Z <- X1-X2
Y <- Z+X3
```

a) Escriba, generalizando el modelo semántico de L para tratar a las macros indicadas como instrucciones de base, las computaciones para los siguientes valores de entrada:

- `X1=1, X2=2, X3=1.`
- `X1=2, X2=2, X3=1.`
- `X1=2, X2=1, X3=2.`

Esto es elemental, y no las doy íntegras por escrito; diré que la primera implica una llamada a la macro de resta parcial, que no retorna, quedando el programa sin output y la función indefinida, y la segunda y tercera vienen dadas por el cálculo obvio, con configuración terminal con "contador de programa" igual a 3.

b) Defina ahora $\Psi_P^{(3)}(x_1, x_2, x_3) = \begin{cases} \uparrow & si\ x_1 < x_2 \\ (x_1 - x_2) + x_3 & en\ otro\ caso \end{cases}$

6. Considere el modelo de computación $PROC$, que posee una estructura de memoria formada por una colección de registros principales $R_1, R_2, \ldots$ junto con tres registros accesorios para cálculos K_1, K_2, K_3. Los n valores iniciales de trabajo se sitúan en los registros $R_1, \ldots, R_n$, y los registros restantes (incluidos los accesorios) inicialmente contienen el valor cero. El output del modelo se sitúa en el registro R_1. Todos los registros toman valores en $\mathbb{N}$. El juego de instrucciones del modelo es: [3 puntos].

- $LOAD(R_m, K_i)$: carga el contenido del registro m-ésimo en el registro accesorio i-ésimo.
- $STORE(K_i, R_m)$: carga el contenido del registro accesorio i-ésimo en el registro m-ésimo.
- ADD: suma los contenidos de los registros K_1, K_2 y almacena el resultado en K_3.
- SUB: resta los contenidos de los registros K_1, K_2 y almacena el resultado en K_3. Si la resta no es un número natural, K_3 se carga con cero.
- $Z(R)$: pone a cero el registro R, que puede ser bien principal, bien accesorio.
- $JUMP(\mathrm{K}_1, K_2, n)$: compara los contenidos de los registros K_1, K_2; si son iguales se bifurca a la n-ésima instrucción, si esta existe; si no, el programa termina; si son distintos, se continúa por la siguiente instrucción.

Se pide:

a) Desarrollar un modelo semántico para $PROC$, incluyendo definiciones de: estado, configuración, configuración sucesora, computación, funciones parcial y totalmente $PROC$-computables.
b) Demostrar que la función $\nabla(x) = 3x$ es $PROC$-computable.
c) Calcular la computación para $\nabla(3)$.
d) Demostrar que la función

$$\delta(\mathrm{x}_1, x_2) = \begin{cases} \uparrow \; si \; \mathrm{x}_1 = x_2 \\ \\ \mathrm{x}_1 + x_2 \; en \; otro \; caso \end{cases}$$

es parcialmente $PROC$-computable.

a) No doy las definiciones del modelo íntegras (es más de lo mismo de siempre), y los únicos puntos dignos de mención sobre esto son:
    - Recordar, para SUB, cargar K3 a cero cuando la resta de K1 y K2 no es un número natural.
    - Recordar, para $JUMP(\mathrm{K}_1, K_2, n)$ que si se bifurca a una instrucción n que no existe, debería cargarse el contador con $long(P) + 1$, para representar la terminación del programa
b) El $PROC$-programa más evidente para calcular a la función $\nabla(x) = 3x$ puede ser algo como lo que sigue, aunque **cualquier equivalente funcionalmente equivalente ha sido, lógicamente, admitido**:

```
LOAD(R1,K1)
LOAD(R1,K2)
ADD
STORE(K3,R2)
LOAD(R2,K2)
ADD
STORE(K3,R1)
```

c) No pongo la computación íntegra, pero debería ser finita, dejar en R_1 un valor igual 9, y acabar en un valor de contador de programa coherente con lo establecido en la semántica, normalmente igual a $long(P) + 1$.

d) Nuevamente, doy el $PROC$-programa más simple para calcular a la función δ, aunque de nuevo, **cualquier equivalente funcional se ha considerado válido**:

```
LOAD(R1,K1)
LOAD(R1,K2)
JUMP(K1,K2,1)
ADD
STORE(K3,R1)
```

(continúe aquí)

(continúe aquí)