Programació i Algorítmica Avançades

Guia de Laboratori

Curs 2023-24, quadrimestre de primavera

José Luis Balcázar, Jordi Delgado

Departament de Ciències de la Computació, UPC

12/03/2024:

Máquinas de Turing; Introducción a la búsqueda combinatoria

Hoy:

- Programación de máquinas de Turing
- Búsqueda combinatoria exhaustiva
- Problemas para tu trabajo personal posterior

Programación de máquinas de Turing de una cinta

Se aconseja ponerse de acuerdo con los vecinos de mesa para que unos trabajen sobre la página web `http://morphett.info/turing/turing.html` y otros sobre la página alternativa `https://turingmachine.io` de manera que podáis comparar las soluciones. Empezad por revisar dos o tres de los ejemplos que trae la web elegida, entendiendo la sintaxis que se usa en cada una para describir los estados y las transiciones.

En cada uno de los ejercicios que siguen, pensad primero cómo creéis que ha de ser la solución (se autoriza e incluso se anima a trabajar conjuntamente en equipo con los vecinos de mesa) y comprobadla después por separado, cada uno en la web que le corresponda.

1. Construye una máquina de Turing que no recibe entrada y escribe en la cinta el saludo “Hello”.
2. Construye una máquina de Turing que recibe dos palabras separadas por un espacio en blanco y comprueba si son iguales. Empezad pensándolo con un alfabeto de sólo dos letras y ampliadlo después.

3. Construye una máquina de Turing que recibe una palabra y la modifica de manera que quede un espacio en blanco entre cada dos caracteres. Empezad con un alfabeto reducido.

Programación de máquinas de Turing de varias cintas

La web `https://turingmachinesimulator.com` contiene ejemplos de programación de máquinas de Turing de varias cintas. En seguida veréis que eso hace la vida del programador de máquinas de Turing más fácil (pero complica la del programador que ha de fabricar su máquina universal).

1. Entiende algunos de los ejemplos que aparecen preprogramados.
2. Repite en este simulador alguno de los ejercicios hechos en el modelo de una sola cinta.

Búsqueda combinatoria exhaustiva

1. Construye en Python tu propia versión de una función que retorne el *powerset* de un conjunto dado. Pruébala sobre `https://jutge.org/problems/P18957_ca` (de la lista Recursion Revisited) hasta lograr luz verde.
2. Usando esa función, resuelve `https://jutge.org/problems/X94664_en` (de la lista Combinatorial Search Schemes I) hasta lograr luz verde. (En la semana siguiente, empezamos por aquí.)

Problemas para tu trabajo personal posterior

Máquinas de Turing de una cinta: usa los simuladores indicados anteriormente y

1. Construye una máquina de Turing que recibe una palabra y comprueba si se puede partir por la mitad en dos palabras iguales.
2. Construye una máquina de Turing que recibe en su cinta un número natural en binario y termina con el triple de ese número natural escrito en su cinta.

Máquinas de Turing de varias cintas: usa el simulador indicado anteriormente y

1. Repite en este simulador los ejercicios que hayas hecho en clase sobre los otros simuladores.
2. Construye una máquina de Turing para la función *diferencia modificada* entre números naturales, la cual, dados naturales `x` e `y`, calcula `max(x-y, 0)`.

Búsqueda exhaustiva: si en clase no te dio tiempo a completar hasta luz verde los ejercicios de Jutge indicados, hazlo ahora.

- `https://jutge.org/problems/P18957_ca`,
- `https://jutge.org/problems/X94664_en`.

Generadores (opcional): aprende a programar generadores en Python (salvo que ya sepas) y emplea un generador para recorrer el *powerset* en alguno de los ejercicios anteriores. Si tu generador se basa en un recorrido iterativo, busca también un generador recursivo.

19/03/2024:

Búsqueda combinatoria: *set-based backtracking.*

Hoy:

- Búsqueda combinatoria exhaustiva: el ejemplo de la mochila
- *Backtracking:* el ejemplo de la mochila
- Variantes enfocadas a encontrar solamente la primera solución
- La variante de optimización
- El ejemplo de la suma del subconjunto
- Problemas para tu trabajo personal posterior
- Material adicional

Búsqueda combinatoria exhaustiva: el ejemplo de la mochila

Baja a través de la página de PAA (enlace en el Racó) la carpeta cuyo nombre corresponde a la fecha de hoy y descomprímela. En el fichero `inputs_knapsack.txt` encontrarás algunos ejemplos de entradas para el problema decisional de la mochila. Los ficheros `knapsack_dec_all_exh_pws_lab.py` y `knapsack_dec_all_exh_tr_lab.py` contienen programas que resuelven el problema de obtener todas las soluciones mediante búsqueda exhaustiva, tal como los hemos visto en la clase de teoría, siguiendo el enunciado de la transparencia 44 (`Mochila, II`) y siguientes.

1. Entiende el contenido del programa `knapsack_dec_all_exh_pws_lab.py` y pruébalo con algunas llamadas para entender bien su funcionamiento.
2. Entiende el contenido del programa `knapsack_dec_all_exh_tr_lab.py` y pruébalo con algunas llamadas para entender bien su funcionamiento.
3. Completa ambos con un programa principal que realice la lectura de datos y la escritura de resultados de manera que obtengas luz verde en X94664 (todas las referencias al Jutge hoy están en la lista `Combinatorial Search I`).

Backtracking: el ejemplo de la mochila

Modifica el contenido del programa `knapsack_dec_all_exh_tr_lab.py` (en la versión que acabas de completar para obtener luz verde, por supuesto) siguiendo las indicaciones que vimos en clase para evitar exploraciones innecesarias (transparencia 56, `Mochila, VII`, y las anteriores y siguientes). Comprueba que lo haces correctamente mediante una nueva luz verde en el mismo problema X94664.

Variantes enfocadas a encontrar solamente la primera solución

1. Modifica tu extensión al programa `knapsack_dec_all_exh_pws_lab.py` para que retorne únicamente una solución, la primera que encuentre.
2. Modifica tu extensión al programa `knapsack_dec_all_exh_tr_lab.py` para que retorne únicamente una solución, la primera que encuentre.
3. Modifica tu programa de *backtracking* del apartado anterior para que retorne únicamente una solución, la primera que encuentre.

La variante de optimización

No hay particular dificultad en usar los casos dados en el fichero `inputs_knapsack.txt` para la variante de optimización: basta omitir el primero de los valores, que corresponde al valor mínimo deseado, dando solamente el peso máximo, el número de objetos, y los pares de valor y peso por objeto. Estos ejemplos están tomados de la web `https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01`, la cual contiene unos cuantos más, adicionalmente, por si los precisas (pero están en un formato un poquito diferente).

1. Completa el programa `knapsack_dec_opt_exh_pws_lab.py`, que optimiza por búsqueda exhaustiva (transparencia 65, `Mochila, XIV`, salvo que han cambiado levemente algunos identificadores), obteniendo luz verde en X59240.
2. Resuelve el mismo poblema por *backtracking* (transparencia 66, `Mochila, XV`).

El ejemplo de la suma del subconjunto

Subset sum es un problema muy similar; de hecho, ocasionalmente, se encuentran referencias a *Subset sum* bajo el nombre *Knapsack*. Al no tener que diferenciar los pesos de los valores, es algo más simple. Resuelve el problema P40685 hasta obtener luz verde. *Ten mucha precaución*: el enunciado permite valores negativos, así que tendrás que inventar algo para poder podar alguna parte de la exploración.

Problemas para tu trabajo personal posterior

1. Completa las variantes del problema de la mochila que transmiten como parámetro la solución parcial candidata en curso, tanto en la versión que hace copias en

las llamadas como en la que mantiene una única solución parcial candidata que se va modificando con `append()` y `pop()`, a partir de los programas que se han explicado en la clase de teoría (transparencias 59 y 60).

Material adicional

```
def knapsack(weights, values, current_item, max_w):
    if current_item == -1:
        return ([],0,0)
    else:
        "current_item >= 0"
        best0, bestw0, bestv0 = knapsack(weights, values, current_item - 1, max_w)
        if weights[current_item] <= max_w:
            best1, bestw1, bestv1 = knapsack(
                weights, values, current_item - 1, max_w - weights[current_item])
            if bestv1 + values[current_item] > bestv0:
                best1.append(current_item)
                return (best1, bestw1 + weights[current_item],
                               bestv1 + values[current_item])
    return best0, bestw0, bestv0
```

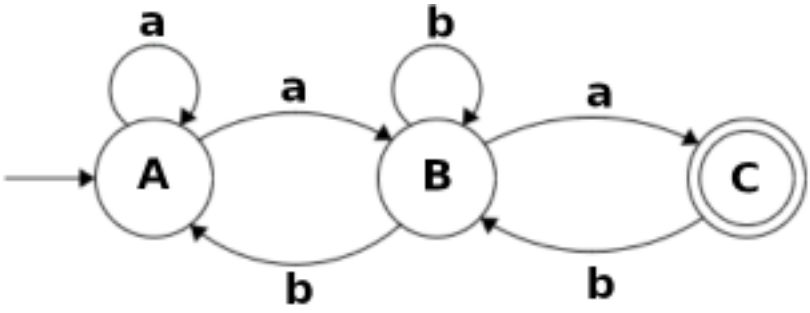
02/04/2024:

Preparación del examen parcial

Entra en examen:

- lenguajes regulares:
  - autómatas finitos,
  - expresiones regulares,
  - gramáticas regulares...
- lenguajes incontextuales:
  - gramáticas incontextuales,
  - autómatas a pila...
- búsqueda combinatoria: *backtracking*,
- búsqueda combinatoria: *greedy*.

Problema del examen parcial del curso pasado Aplica el lema de Arden para encontrar una expresión regular que describa el lenguaje formal aceptado por el autómata finito siguiente:



Algorítmica

1. Si aún no lo tienes resuelto, resuelve hasta obtener luz verde el problema X33098 (problema del examen parcial del curso pasado).
2. Si ya tienes ese problema resuelto, pero no has intentado aún P81009, inténtalo ahora hasta lograr: (a) AC en los juegos de prueba públicos y (b) o bien AC o bien "time limit exceeded" (más probablemente será lo segundo) en los privados. En general, para obtener luz verde, probablemente sea necesario aplicar un esquema algorítmico que estudiaremos después del examen parcial.
3. Si ya estás en ese punto con ambos problemas, resuelve hasta obtener luz verde el problema P81629.
4. El problema P92008 es muy similar; sin embargo, es más estricto en términos de eficiencia y existen soluciones que pasan P81629 y, sin embargo, en éste dan "time limit exceeded". Prueba a basarte en tu solución a P81629 y, de ser necesario, encuentra cómo acelerarlo hasta obtener luz verde en P92008.
5. Si hay enunciados indicados en los días anteriores que aún no has resuelto, resuélvelos.
6. Completa, si aún no lo has hecho, las listas Combinatorial Search I y Combinatorial Search II, excepto X52116.

Material adicional

El código indicado a continuación obtiene luz verde en X33098. Coincide con el desarrollado en clase el martes 2 de abril en el grupo de las 12h, y es muy similar al indicado en la clase de las 8h del mismo día, pero algo más simple y claro de entender.

```
def opt_b(clayleft, prices, currsize):
    if currsize == 0 or clayleft == 0:
        return list(), 0.0
    else:
        "try not making bowl of currsize, then making it"
        s0, b0 = opt_b(clayleft, prices, currsize - 1)
        if currsize <= clayleft:
            s1, b1 = opt_b(clayleft - currsize, prices, currsize)
            if b1 + prices[currsize] > b0:
                s1.append(currsize)
                s0, b0 = s1, b1 + prices[currsize]
        return s0, b0

from pytokr import pytokr
item = pytokr()

clay = int(item())
prices = [0.0] # placeholder for length 0 so that lengths and indexes coincide
for _ in range(clay):
    prices.append(float(item()))

s, b = opt_b(clay, prices, clay)
print(f"{b:.2f}") # o bien: print(f"{b:.2f}", s)
```

16/04/2024:

Programación Dinámica, I

"The gold ones are Galleons", he [Hagrid] explained. "Seventeen silver Sickles to a Galleon and twenty-nine Knuts to a Sickle, it's easy enough."

J. K. Rowling: *Harry Potter and the Philosopher's Stone*

- *Giving change*
- Aún otra vez "La Mochila"
- Problemas adicionales para clase o para tu trabajo personal posterior

"Giving change"

a/ Repasa y entiende los programas vistos en clase. Si mantenemos (por ejemplo, en un `dict`) la tabla bidimensional, facilitará la comprensión el recorrer las denominaciones de monedas por sus índices:

```
def minchange(goal, denoms):
  denoms = [0] + list(denoms) # placeholder
  dptable = {}
  for quantity in range(goal + 1):
    "init table for no coins"
    dptable[0, quantity] = float("inf")
  for coin in range(len(denoms)):
    dptable[coin, 0] = 0
  for quantity in range(1, goal + 1):
    '''
    dptable[coin, quantity]:
    how many coins up to that one needed to add up to quantity
    '''
    for coin in range(1, len(denoms)):
      "try using it"
      if (denoms[coin] <= quantity and
        1 + dptable[coin, quantity - denoms[coin]]
          < dptable[coin - 1, quantity]):
        dptable[coin, quantity] = 1 + dptable[coin, quantity - denoms[coin]]
      else:
        dptable[coin, quantity] = dptable[coin - 1, quantity]
  return ...
```

b/ Complétalo siguiendo las indicaciones como para poderlo probar en P81009, pero no lo intentes aún: de momento, asegúrate de cómo funciona en tu propia máquina.

c/ Cuando trabajemos con la tabla unidimensional, no es preciso usar los índices de las denominaciones y podemos iterar directamente sobre ellas:

```
def minchange(goal, denoms):
    dptable = [ float("inf") ] * (goal + 1)
    dptable[0] = 0
    for quant in range(1, goal+1):
        '''
        dptable[quant]: how many coins to add up to it
        '''
        for coin in denoms:
            "try using it"
            if coin <= quant:
                dptable[quant] = min(dptable[quant], 1 + dptable[quant - coin])

    return ...
```

Complétalo y compara con la versión anterior.

d/ Cuando creas que los entiendes suficientemente, modifícalos para poder obtener luz verde en P81009; sin embargo, has de tener en cuenta que, en ese problema, **no** tienes garantía de que esté la denominación 1 y, por tanto, es posible que, en algunos casos, no haya solución. Es muy poco probable que logres luz verde con la variante que mantiene la tabla bidimensional, que probablemente dará "time limit exceeded" en los juegos de prueba privados. Con la alternativa de tabla unidimensional es posible que lo logres.

e/ Añade (ahora ya al margen del Jutge) una función que permita reconstruir la solución que usa el mínimo de monedas, a base de comparar los contenidos de las celdas, tal como hemos visto en clase:

```
from collections import Counter

def trace(gctab, denoms, q):
    "returns solution in dict r, uses non-generalizable method"
    r = Counter()
    d = ... # use it to traverse denominations from large to small
    while q:
        "hack - we have just two cases and, in one, t[d,q] == t[d-1,q]"
        if ...:
            "low end, only smallest coins remain to be used"
            r[denoms[d]] += q
            break
        elif gctab[d, q] == gctab[d-1, q]:
            "coins of denoms[d] units were not employed"
            d -= 1
        else:
            "at least one coin of denoms[d] units was employed"
            r[denoms[d]] += 1
            q -= denoms[d]
    return r
```

Completa el programa y asegúrate de que funciona adecuadamente.

f/ La manera *generalizable* de poder reconstruir la solución en una aplicación de programación dinámica es mediante el mantenimiento de una segunda tabla. En este caso, queremos poder reconstruir las soluciones así:

```
def trace(best, goal):
    coins = []
    while goal:
        use = best[goal]
        coins.append(use)
        goal -= use
    return coins
```

Has de añadir al programa principal la gestión de la tabla secundaria (que aquí llamamos `best`): cada vez que cambia la tabla principal, en la segunda tabla anotamos el por qué. La solución óptima viene, entonces, en forma de lista con repeticiones en lugar de un Counter. Completa el programa, asegúrate de que funciona adecuadamente, por ejemplo, comparando con el anterior y, finalmente, intenta reemplazar la lista por un Counter.

Aún otra vez "La Mochila"

Inspirándote en las propuestas vistas para *Giving Change*, resuelve la variante 0/1 de Knapsack por Programación Dinámica: cada objeto puede elegirse para ponerlo en la mochila o bien dejarse fuera. Puedes usar X59240 (que ya conoces) para probar tus soluciones.

Problemas adicionales para clase o para tu trabajo personal posterior

- *Echando una mano a la alfarera*, versión DP, X52116.
- *Cortes de la varilla* ("Rod cutting"), X71353, que posiblemente ya has resuelto por *backtracking* pero ahora toca hacerlo por programación dinámica.
- *Canonical coin systems (2)*, X88410 (la versión X24976 es la del examen y supone que se buscan soluciones óptimas por *backtracking*; en cambio, X88410 requerirá hacerlo por Programación Dinámica y muy probablemente de error por *time limit exceeded* si se sube una solución basada en *backtracking*.

23/04/2024:

Programación Dinámica, II

- Supersequences
- Aún otra vez caminos mínimos
- Ejemplos adicionales

Supersequences

Completa el ejemplo de encontrar la supersecuencia más corta de dos *string* dados. Para ello, has de reflexionar con sumo cuidado para identificar qué parte de la tabla bidimensional hay que inicializar antes de lanzar el doble bucle que la llena y, sobre todo, entender cuál es la inicialización adecuada. Puedes probar tus soluciones en X40399 de la lista Combinatorial Search III.

Por supuesto, has de partir de la ecuación de Bellman y el esbozo de algoritmo vistos en clase y documentados en las transparencias ("Supersecuencias, III" y "IV").

All-pairs shortest paths

Dado un grafo (implementado de alguna de las maneras que tengas de Programació i Algoritmes 2), construye y programa el algoritmo de Floyd a partir de la correspondiente ecuación de Bellman vista en clase (Transparencia "Caminos mínimos, IV").

Ejemplos adicionales

Ya listábamos la semana pasada estos ejemplos a resolver; atácalos si aún no lo has hecho:

- *Echando una mano a la alfarera*, versión DP, X52116 en la lista Combinatorial Search III.
- *Cortes de la varilla* ("Rod cutting"), X71353 en la lista Combinatorial Search I. Posiblemente ya lo has resuelto por *backtracking* (y si no lo has hecho, hazlo pronto) pero ahora toca resolverlo por programación dinámica.

Para evitar *spoilers*, no se incluyen aquí aún soluciones, pero aparecerán en un futuro próximo.

30/04/2024:

Pares de Cantor y funciones recursivas parciales

- Pares y tuplas entre los números naturales
- Construcción de funciones recursivas parciales y de sus números
- Minimización, cuantificación acotada y distinción de casos

Pares y tuplas entre los números naturales

A través de la `página usual de PAA`, descarga el `zip` con el intérprete de PReFScript y descomprímelo en la carpeta adecuada como para poder llamar a Python e importar el módulo `cantorpairs`. Aconsejo importarlo como: `import cantorpairs as cp`.

1. La función `cp.dp(x, y)` implementa la formación de pares que hemos visto en clase (el *dotted pair*). Consigue printar una matriz `n x m` con los valores de la función de *dotted pair* de manera que sea visible la estructura diagonal, y explica esa estructura diagonal analizando la fórmula que tienes en las transparencias.
2. Las funciones inversas son `cp.pr_L(z)` (componente izquierda) y `cp.pr_R(z)` (componente derecha). Comprueba y entiende su funcionamiento.
3. Explora las funciones `cp.tup_i()` que codifica en un único natural una tupla (o cualquier iterable) de naturales recibida como único argumento, y `cp.tup_e()` que

hace lo mismo recibiendo los elementos de la tupla como argumentos. Combínalas con `cp.pr_L(z)` y `cp.pr_R(z)` para asegurar que comprendes su funcionamiento.

4. La función `cp.s_tup(t, k)` descarta los primeros `k` elementos de la tupla `t` y retorna el sufijo formado por el resto. Explórala y entiende su funcionamiento.
5. La función `cp.pr(t, k)` retorna el elemento en la posición `k` de la tupla `t`. Explórala y entiende su funcionamiento.
6. Revisa y entiende cómo están programadas todas las funciones del módulo `cantorpairs`.

Construcción de funciones recursivas parciales y de sus números

Ahora consigue poder importar la clase `PReFScript()` del módulo `prefscript.py` que también tienes en el mismo `zip`. El fichero `uses_prefscript.py` del mismo lugar muestra cómo se puede usar. **No abras aún** el fichero `.prfs` que los acompaña.

Inicialmente, crea un objeto de esta clase desde el intérprete de Python:

```
my_fs = PReFScript("Store Goedel numbers")
```

Pronto, en cuando tengas claro cómo funcionan los números de Gödel, preferirás poder prescindir de ellos: simplemente, omitirás ese string y crearás los objetos con `PReFScript()`.

Un objeto de esa clase nos permite mantener una colección de funciones recursivas parciales, definidas unas en términos de otras. Las básicas siempre vienen ya disponibles desde la creación del objeto.

El método `list()` de esos objetos te da la lista de funciones disponibles. Se puede obtener más información de cada función con `list(name)`. Existen tres maneras de añadir nuevas funciones a la lista mediante los siguientes métodos disponibles para objetos de clase `PReFScript()`:

- El método `dialog()` que "te lleva de la mano" preguntándote lo que va necesitando.
- El método `define()` del que puedes ver ejemplos en `uses_prefscript.py`.
- El método `load()` que permite cargar funciones definidas en un fichero aparte con una sintaxis que explicaremos luego.

La adición de nuevas funciones sigue los criterios con los que hemos definido las funciones recursivas parciales. Es decir, tal como se deduce de la definición en las transparencias "Funciones Recursivas Parciales II" y "III", una función es recursiva parcial si y sólo si es posible construirla en uno de estos objetos.

1. Construye, a partir de las funciones básicas y mediante composición y formación de pares, la función que asocia a `x` el valor `<x.1>` (la podríamos llamar `piggyback_1`).

2. Construye, a partir de las funciones básicas y mediante composición y formación de pares, la función *anterior* (modificado): `x - 1` para `x > 0` y `0` para `x = 0`. (Obviamente, vale usar la función del ejercicio anterior.) Usa `list()` para ir viendo cómo los respectivos números de Gödel van cuadrando con los especificados en las transparencias.

Los objetos de clase `PReFScript()` ofrecen adicionalmente el método `to_python(name)` que devuelve una función Python ejecutable a partir del string dado como nombre a cualquiera de las funciones definidas hasta ese momento. Comprueba su funcionamiento con las funciones que llevamos definidas, así como con las que vayamos definiendo a continuación. Necesitarás ayudarte de `cp.dp()`, `cp.pr_L(z)` y `cp.pr_R(z)` en múltiples ocasiones.

3. Explica por qué cualquier función constante está entre las funciones recursivas parciales.

4. Explica por qué está entre las funciones recursivas parciales, para cada natural `i`, la función que asocia a `x` el valor `<x.i>`; idem para `<i.x>`.

5. ¿Cómo se construyen las dos proyecciones π^L y π^R a partir de las funciones básicas? ¿Y la función que a `x` le asocia `<x.x>`? ¿Y la que a `<x.y>` le asocia `<y.x>`?

Podemos operar con los naturales 0 y 1 como si fuesen valores booleanos. Recordemos que las funciones características son funciones totales que sólo toman los valores 0 y 1. Se pueden ver, por tanto, como predicados, o funciones booleanas.

6. Construye las operaciones de negación, conjunción y disyunción.

7. Construye la función signo: `sg(x) = 0 if x == 0 else 1` (es útil para una de las posibles maneras de construir la operación de disyunción: ¿cómo?).

8. Construye las funciones que, dado `<x.y>`, comparan `x` con `y` según criterios de orden o de igualdad, devolviendo 0 o 1 como booleanos.

Empezamos a tener en cada momento varias funciones en juego y si, por lo que sea, hemos de salir del intérprete de Python y volver a empezar, la situación puede ser un poco frustrante. Los objetos de clase `PReFScript()` aportan el método `load(filename)` que lee definiciones de un fichero con el nombre indicado. Podemos usar como ejemplo el fichero `script_bool.prfs`. En la llamada a `load()`, hay que dar el nombre completo del fichero, no se supone `.prfs` por defecto. La extensión es acrónimo de "partial recursive functions script". Usa este mecanismo en adelante a tu propia discreción.

Minimización, cuantificación acotada y distinción de casos

De los tres mecanismos de definición de nuevas funciones, hemos practicado con `comp` y con `pair`. El tercero, `mu` (por la letra griega μ que suele usarse tradicionalmente para la minimización) nos permite *buscar* un número natural que cumpla una cierta condición.

1. Construye la función que lleva de `x` a `x+1` de una nueva manera: buscando el primer valor `z` que sea mayor que `x`.

2. A través de tantos pasos como te sean necesarios, construye la función que, dados `x` e `y`, retorna el cociente de dividir `x` entre `y`.

A partir de algún punto (posiblemente cercano a éste) encontrarás innecesario mantener calculados los números de Gödel de las funciones que construyes. Puedes ahorrar su cálculo inicializando el objeto con una llamada a `PReFScript()` sin parámetros. Adicionalmente, éste es buen punto para revelar que el método `list()` admite un segundo parámetro, que puede recibir los valores `w_code = 1` o `w_code = 2` (por `with code`), que indican cómo se ha construido esa función en términos de función recursiva parcial y/o en términos de Python.

3. Sea P un predicado recursivo. Consideramos:

   $$P'(\langle x.t\rangle) = 1 \iff \exists y(y \leq t \wedge P(\langle x.y\rangle) = 1),$$

   $$P''(\langle x.t\rangle) = 1 \iff \forall y(y \leq t \Rightarrow P(\langle x.y\rangle) = 1).$$

Decimos que estos predicados se obtienen por cuantificación existencial acotada y por cuantificación universal acotada, respectivamente. El esquema de minimización nos permite argumentar que también son recursivos. ¿Cómo se logra?

4. Sea `A` un conjunto de naturales, y sean `f` y `g` funciones totales. Combinándolas con las básicas, y con la función característica de `A`, mediante composición y formación de pares, queremos obtener la función `h` donde `h(x)` vale:

   `f(x) si x` $\in$ `A`

   `g(x) si x` $\notin$ `A`

¿Cómo se logra?

07/05/2024:

Profundización en las funciones recursivas parciales

- Repaso: composición y formación de pares
- Repaso: minimización, cuantificación acotada y distinción de casos
- Recursión primitiva
- El predicado T de Kleene

Repaso: composición y formación de pares

Revisa tus soluciones a los ejercicios con PReFScript de la semana anterior. Completa los que te falten; inventa algunos ejercicios similares por tu propia cuenta y resuélvelos.

Repaso: minimización, cuantificación acotada y distinción de casos

Revisa esos temas de la semana anterior o bien, si no te dio tiempo a trabajarlos, hazlo ahora.

Recursión primitiva

Todas las aplicaciones prácticas de programación recursiva caen en el ámbito de la llamada "recursión primitiva". Desde la perspectiva teórica, existen funciones calculables que no se pueden formalizar mediante recursión primitiva, como la llamada función de Ackerman. Pero, de hecho, únicamente funciones que corresponden a los casos más simples de recursión primitiva tienen significado práctico: lo estudiaremos en el tema de Teoría de la Complejidad.

Es probable que los ejercicios de hoy resulten demasiado farragosos si se intentan desarrollar completamente en PReFScript. Valora ese mecanismo en su faceta formativa y deja de emplearlo tan pronto como detectes que le dedicas demasiado tiempo sin aprender nada.

1. Partiendo de la formulación recursiva estándar del factorial, explica por qué es una función recursiva total. *Pista:* las construcciones de composición se vuelven insuficientes; el proceso consiste en buscar, mediante una minimización, un número que codifique la totalidad de los resultados intermedios. La corrección de esos resultados intermedios se expresa por cuantificación universal acotada.
2. Partiendo de la formulación recursiva estándar de la función de Fibonacci, explica por qué es una función recursiva total.
3. Identifica de qué manera puedes enunciar una afirmación que indique la manera de encontrar funciones recursivas totales a partir de formulaciones recursivas.

El predicado T de Kleene

Este predicado (función total que retorna 0 o 1 en su rol de "cierto" o "falso") formaliza la propiedad que podemos describir, de manera intuitiva, como sigue: "dados i, x, y y t, la funci'on de índice i, sobre el argumento x, evalúa a y, y ello se puede justificar usando una secuencia de valores intermedios globalmente acotada por t".

Podemos asimilar el valor t a una cota sobre el uso de recursos disponibles para calcular $\phi_i(x)$: si no nos basta con, imaginemos, tiempo t para completar el cálculo, el predicado se hace 0. De hecho, si partiéramos de máquinas de Turing, el rol del parámetro t sería exactamente "tiempo" o número de pasos dados.

1. ¿Cómo se puede definir T? Buscamos una definición que nos permita argumentar que es recursivo. *Pista:* Lo organizamos en forma de recursión primitiva, "desmontando" el índice i en sus constituyentes, que nos indican cómo se calcula la función de índice i a partir de otras más sencillas.
2. Explica lo que significa la afirmación siguiente:

$$\forall t, t'(t < t') : T(i, x, y, t) = 1 \Rightarrow T(i, x, y, t') = 1.$$

3. Conecta T con la función universal mediante las propiedades siguientes:

- Para todo t, si $T(i, x, y, t) = 1$ entonces $\phi_i(x) \downarrow$ y además $\phi_i(x) = y$.
- Para todo i y x, $\phi_i(x) \downarrow$ si y sólo si $\exists t \, \exists y \, (T(i, x, y, t) = 1)$.

¿Cómo combinamos todos estos hechos para asegurar que la función universal es recursiva parcial?

14/05/2024:

Más funciones recursivas parciales

Los ejercicios de hoy son a realizar con papel y bolígrafo (o equivalente digital tipo tablet), no sobre PReFScript (a lo más, puedes resolver en PReFScript el primero, para ver qué números concretos son p y q con nuestra codificación de las funciones recursivas parciales, si tienes esa curiosidad).

1. Revisa tu construcción (o realízala si no la tienes) como funciones recursivas parciales de las funciones $f(y) = \langle 0.y \rangle$ y $g(\langle x.y \rangle) = \langle x + 1.y \rangle$. Sea p el número de Gödel de f y q el de g, de manera que $\phi_p = f$ y $\phi_q = g$. Usamos estos valores de p y q en los ejercicios siguientes.
2. Justifica que es recursiva parcial la función $h(0) = p$, $h(n + 1) = \langle 1.\langle q.h(n) \rangle \rangle$.

3. ¿Qué puedes decir, ahora, de $\phi_{h(x)}(y)$?
4. Justifica que es recursiva parcial la función $s(\langle i.x\rangle) = \langle 1.\langle i.h(x)\rangle$. Observamos que es total.
5. ¿Qué puedes decir, ahora, de $\phi_{s(\langle i.x\rangle)}(y)$? (Una vez hayas encontrado una respuesta interesante, cotéjalo con la noción de parametrización en las transparencias.)
6. Justifica que hay una función recursiva parcial s tal que $\phi_{s(x)}(y) = \phi_x(x)$ para todo y. *Pista:* Combina lo anterior con la función universal.
7. Justifica que hay una función recursiva parcial ϕ_e tal que $\phi_e(\langle x.y\rangle) = \phi_{\phi_x(x)}(y)$ para todo x e y; observamos que $\phi_e(\langle x.y\rangle)$ queda indefinida si $\phi_x(x)$ está indefinida, y que si $\phi_x(x) = k$ sí está definida, entonces $\phi_e(\langle x.y\rangle) = \phi_k(y)$ que, a su vez, también puede estar indefinida.
8. Obtén de las funciones anteriores una función recursiva total s tal que $\phi_{s(x)} = \phi_{\phi_x(x)}$ (igualdad en tanto que funciones, o sea, están definidas en los mismos y y con el mismo resultado).
9. Dada una función f recursiva total, cualquiera, consideramos la composición $\phi_m = f \circ s$ y el índice $n = s(m)$. ¿Qué relación podemos encontrar entre ϕ_n y $\phi_{f(n)}$?
10. Justifica que existe un índice ℓ tal que la función ϕ_ℓ cumple $\phi_\ell(\ell) = 1$ y $\phi_\ell(y) = 0$ si $y \neq \ell$. (Es "el programa que se reconoce a sí mismo".)

28/05/2024:

Lambda-càlcul i problemes d'examen

Lambda-càlcul

Fem els exercicis fent servir la calculadora:

`https://jacksongl.github.io/files/demo/lambda`

1/ Exercici slide 17:

```
SUCC (SUCC 0) =
```

```
((lambda a b c. b (a b c)) ((lambda a b c. b (a b c)) (lambda s z. z)))
```

Troba la forma normal (aplica alfa i beta reduccions fins que no puguis reduir-ho més).

2/ Exercici slide 18 (I) Definim la suma com `x SUCC y`, és a dir,

```
SUMA = (lambda x y. (x (lambda a b c. (b (a b c))) y))
```

i com `2 = (lambda s z. s (s z))` i `3 = (lambda s z. s (s (s z)))`, tot plegat queda (escrivim `((SUMA 2) 3)`):

```
(((lambda x y. (x (lambda a b c. (b (a b c))) y))
   (lambda s z. s (s z))) (lambda s z. s (s (s z))))
```

Troba la forma normal (aplica alfa i beta reduccions fins que no puguis reduir-ho més).

3/ Exercici slide 18 (II) També podem definir la suma com:

```
SUMA' = (lambda p q x y. (p x (q x y))).
```

Així, 2 + 3 serà

```
((SUMA' 2) 3):
   (((lambda p q x y. (p x (q x y)))
      (lambda s z. s (s (s z)))) (lambda s z. s (s z)))
```

Troba la forma normal (aplica alfa i beta reduccions fins que no puguis reduir-ho més).

4/ Exercici slide 18 (III) Podem definir la suma d'una tercera manera:

```
SUMA'' = (lambda p q. (p (lambda w y x. y (w y x)) q))
```

Això vol dir que `2 + 3` s'escriuria així:

```
(((lambda p q. (p (lambda w y x. y (w y x)) q))
   (lambda s z. s (s z))) (lambda s z. s (s (s z))))
```

Troba la forma normal (aplica alfa i beta reduccions fins que no puguis reduir-ho més).

5/ Exercici slide 18 (i IV) Defineix la funció PROD (producte) i escriu i redueix l'expressió 2 * 3.

6/ Exercici slide 20 Escriure el combinador Y en Python. Hi ha cap problema? Per què?

Problemes d'examen

Aquests problemes sobre λ-càlcul, sobre teoria de la calculabilitat i sobre complexitat van ser demanats a exàmens anteriorment.

1. La funció binària **nor** entre predicats és certa exactament quan tots dos arguments son falsos. Argumenta que aquesta funció és recursiva parcial (i, de fet, total).

2. Argumenta que la funció que a cada número natural associa la aproximació entera per defecte de la seva arrel cúbica és recursiva parcial. De la teva argumentació, ¿es dedueix que és total?

3. Considerem el conjunt $A = \{i \mid \phi_i(42)\downarrow\}$, és a dir, els índexos i tals que el valor 42 és al domini de la funció ϕ_i. Argumenta que és indecidible.

4. Sabent que, en el λ-càlcul, *True* es representa per $\lambda\texttt{xy.x}$ i *False* per $\lambda\texttt{xy.y}$, argumenta en detall que $\lambda\texttt{ab.ab(}\lambda\texttt{xy.y)}$ és la implementació de la funció AND.

5. Amb els mateixos λ-termes per *True* i *False*, quina funció implementa el λ-terme $\lambda\texttt{v.(v}\,\lambda\texttt{xy.y}\,\lambda\texttt{xy.x)}$? Argumenta la teva resposta.

6. Recordem el problema *Subset Sum*: donats sumands $s_i \in \mathbb{N}$ i objectiu $t \in \mathbb{N}$, podem trobar un subconjunt I tal que $\sum_{i\in I} s_i = t$? Considerem ara el problema *Partition*: donats sumands $s_i \in \mathbb{N}$, podem trobar un subconjunt I tal que $\sum_{i\in I} x_i = \sum_{i\notin I} x_i$? Argumenta, el més convincentment que puguis, que *Subset Sum* $\leq_m^p$ *Partition*. Qué en pots deduir immediatament sobre *Partition*? Argumenta que la reducció en el sentit contrari també existeix, però, *sense construir-la explícitament*.