Trabajo Práctico 2
Programación Lógica

Paradigmas de Lenguajes de Programación — 1er cuat. 2014

Fecha de entrega: 12 de junio

1. Introducción

Escrabel^{TM} es un juego de mesa que es una reversión del viejo y conocido *Scrabble*®[1]. A diferencia del original, *Escrabel*^{TM} es monojugador y tiene algunos cambios en las restricciones:

- Se juega con todas las fichas que quedan, en lugar de ir sacando de a siete.
- El mismo premio puede utilizarse dos veces si el casillero se utiliza para dos palabras.
- No hace falta validar que las palabras pertenezcan a un diccionario, aunque se espera que un buen jugador utilice palabras reales.
- Dos palabras pueden tocarse, solaparse o incluso ocupar el mismo espacio; esto último si sólo difieren en letras que fueron reemplazadas por el blanco '*' .

Dado que el juego original trae las fichas `ch`, `ll` y `rr`, se tratará a cada una de ellas como una letra, siendo distintas de `[c,h]`, `[l,l]` y `[r,r]`, respectivamente.

Como en el juego original para el español, la cantidad de fichas por letra se mantiene y sus respectivos puntajes también:

Letra × Cantidad	**Puntaje**
`*` × 2 (blanco)	0
`A` × 12, `E` × 12, `I` × 6, `L` × 4, `N` × 5, `O` × 9, `R` × 5, `S` × 6, `T` × 4, `U` × 5	1
`D` × 5, `G` × 2	2
`B` × 2, `C` × 4, `M` × 2, `P` × 2	3
`F` × 1, `H` × 2, `V` × 1, `Y` × 1	4
`CH` × 1, `Q` × 1	5
`J` × 1, `LL` × 1, `Ñ` × 1, `RR` × 1, `X` × 1	8
`Z` × 1	10

[1] http://es.wikipedia.org/wiki/Scrabble

Finalmente, el objetivo del juego sigue siendo ganar la mayor cantidad de puntos posible. Así, el puntaje se calculará siguiendo las reglas habituales:

- Se suma el puntaje de cada letra multiplicada por el valor correspondiente, en caso de estar ubicada en posición de premio.
- Si alguno de las fichas de la palabra están en posición de premio de palabra, se multiplica el valor calculado anteriormente. Si se cubre más de un premio, se aplican todos ellos.
- Una letra que participa en más de una palabra a la vez, suma en todas ellas.

Estructuras a utilizar

La grilla del tablero se representará utilizando una matriz (lista de filas) **parcialmente instanciada**, donde las variables representan espacios donde aún no se colocó ninguna ficha.

Las letras serán representadas por átomos (`a`, `b`,...). Las fichas blancas (representadas por el átomo `*`), se consideran comodines.

Cada posición de la matriz se identificará con una tupla `(X,Y)`, es decir `(Columna, Fila)`. Asumiremos que `(0,0)` es la posición de arriba a la izquierda.

Adicionalmente, un tablero de un juego estará representado por una tupla:
`(Matriz, PosInicial, ListaDL, ListaDP, ListaTL, ListaTP)`, donde `PosInicial` es la posición donde comenzará la primera palabra que se ubique, y `ListaDP` es la lista de posiciones cuyo premio es "duplica puntos palabra", `ListaTL` es la lista de posiciones cuyo premio es "triplica puntos letra", y así sucesivamente.

2. Guía de trabajo

Se pide diseñar e implementar en Prolog los predicados necesarios para modelar los siguientes ítems:

Tablero de juego

1. Definir una matriz de una dimensión dada (con variables en todas las posiciones), o verificar que una lista de listas sea una matriz de la dimensión dada.
   `matriz(+Filas, +Columnas, ?Matriz)`.

2. Dada una posición de origen, determinar la siguiente posición en la dirección dada (considerar como direcciones los átomos `vertical` y `horizontal`).
   `siguiente(?Direccion, +Origen, ?Destino)`.

   Por ejemplo, `siguiente(vertical, (0,0), P)` $\rightsquigarrow$ `P=(0,1)`

Fichas

3. Determinar la lista de fichas que se usaron al momento:
   `fichasUtilizadas(+Matriz,-Fichas)`.
   Por ejemplo, si `M` está instanciada en la matriz del tablero de la Figura 1, entonces `fichasUtilizadas(M, F)` $\rightsquigarrow$ `F=[a,e,p,r,z,*,*]` (no importa el orden).

4. Determinar la lista de fichas disponibles (i.e., fichas existentes aún no ubicadas en la matriz):
`fichasQueQuedan(+Matriz, -Fichas)`.

Letras

5. Determinar qué posición ocupa una letra, admitiendo, para toda letra, las posiciones ocupadas por `*`, pero no por variables:
`buscarLetra(+Letra,+Matriz,?Posicion)`.

6. Ubicar una letra en un posición del tablero, instanciando la posición si en ella hubiera una variable:
`ubicarLetra(+Letra,+Matriz,?Posicion,+FichasDisponibles,-FichasRestantes)`.
Se asume que las fichas disponibles están contenidas en las fichas existentes (i.e., no es necesario verificarlo).
Se sugiere aprovechar el predicado `buscarLetra`, y crear un predicado adicional para unificar con variables.

Palabras

7. Ubicar una palabra en el tablero (instanciando las posiciones necesarias si la palabra no había sido ubicada aún):
`ubicarPalabra(+Palabra,+?Matriz,?Inicial,?Direccion)`.
Las fichas que se coloquen deben ser parte de las fichas disponibles, las que ya estaban en el tablero cuentan como ya utilizadas.

8. Determinar la ubicación de una palabra que ya está en el tablero:
`buscarPalabra(+Palabra,+Matriz,?Posiciones, ?Direccion)`.

Tableros y juegos válidos

9. Permitir validar un tablero:
`tableroValido(+Matriz, +Inicial, +ListaDL, +ListaDP, +ListaTL, +ListaTP)`.
Un tablero de juego válido es aquel en que el tablero es una matriz, donde la casilla inicial está dentro de él, y todos los premios están ubicados dentro del tablero y en distintas posiciones.

10. Verificar que un juego sea válido (u obtener un juego válido), en cuanto a que su tablero sea válido y que las palabras dadas sean realmente ubicables en él, siguiendo las reglas y con las fichas disponibles:
`juegoValido(+?Tablero, +Palabras)`.

    La matriz del tablero puede venir parcialmente instanciada o, incluso, vacía, y se espera que se vaya instanciando a medida que se ubiquen las palabras que no estaban ya en ella.

    Cada palabra debe compartir al menos una posición con alguna palabra ubicada anteriormente o, si es la primera, su posición inicial debe coincidir con la posición inicial del tablero.

Las palabras deben ubicarse en el orden en que aparecen en la lista. Se sugiere utilizar un predicado auxiliar que vaya llevando cuenta de las palabras ya ubicadas.

Se asume que la lista no tiene palabras repetidas (no hace falta verificarlo).

Puntajes

11. Determinar el puntaje de una palabra para un tablero determinado:
`puntajePalabra(+Palabra, +Tablero, -Puntos)`.
Para ello se deberá tener en consideración el puntaje total, letra por letra, y los premios ofrecidos por letra y palabra. Asumir que la palabra ya está ubicada (una sola vez) en el tablero, y que el tablero corresponde a un juego válido.

12. Determinar el puntaje de un juego dados un tablero y un conjunto de palabras:
`puntajeJuego(+?Tablero, +Palabras, -Puntaje)`.
Para esto se recomienda chequear que el juego sea válido, y utilizar predicados auxiliares para acumular el puntaje de cada palabra.

Juego

13. Determinar si un juego es posible:
`juegoPosible(+TableroInicial,+Palabras,-TableroCompleto,-Puntaje)`.
`TableroInicial` es un tablero donde aún no se han ubicado las palabras, y el mismo **no debe instanciarse** más de lo que ya estaba. Para eso, se recomienda realizar una copia del mismo reemplazando las variables existentes por variables frescas, y luego instanciarlo para obtener el tablero completo. Ver predicados para copiar estructuras en el esqueleto del TP.

14. Obtener (cada) juego óptimo —es decir, de puntaje máximo— a partir de un tablero original y una lista de palabras a ubicar en el orden dado:
`juegoOptimo(+TableroInicial,+Palabras,-TableroCompleto,-Puntaje)`.
Así, se deberá definir disposición de las palabras en el tablero y el puntaje obtenido. Normalmente habrá más de una solución óptima.

3. Algunos Juegos de Ejemplo

Los ejemplos provistos funcionarán de acuerdo a las siguientes definiciones de tableros:

```
tablero1(t(M, (7,7), DLS, DPS, TLS, TPS)) :- matriz(15, 15, M),
        TPS=[(0,0), (0,7), (0,14), (7,0), (7,14), (14,0), (14,7), (14,14)],
        DPS=[(1,1),(2,2),(3,3),(4,4),(7,7),(10,10),(11,11),(12,12),(13,13),(13,1),(12,2),
        (11,3),(10,4),(4,10),(3,11),(2,12),(1,13)],
        TLS=[(1,5),(1,9),(5,1),(5,5),(5,9),(5,13),(9,1),(9,5),(9,9),(9,13),(13,5),(13,9)],
        DL1=[(0,3),(0,11),(3,0),(3,14),(11,0),(11,14),(14,3),(14,11),(6,6),(8,8),(8,6),(6,8)],
        DL2=[(2,6),(2,8),(3,7),(6,2),(8,2),(7,3),(12,6),(12,8),(11,7),(6,12),(8,12),(7,11)],
        append(DL1,DL2,DLS).

tablero2(t(M,(0,0),[(1,2),(2,1)],[(0,0)],[(1,1)],[(2,2)])) :- matriz(3, 3, M).

tablero4(
    t(M,(2,2),[(1,1),(1,3),(3,1),(3,3)],[(0,0),(2,2),(4,4)],[(0,2),(2,0),(2,4),(4,2)],[(0,4),(4,0)]))
                                                        :- matriz(5, 5, M).
```

Test 1

Existen dos (2) disposiciones óptimas (soluciones) de las fichas en el `tablero1` para generar las palabras 'paz', 'pez' y 'zar' (en ese orden), que permiten obtener 66 puntos. Una de las soluciones se muestra en la Figura 1.

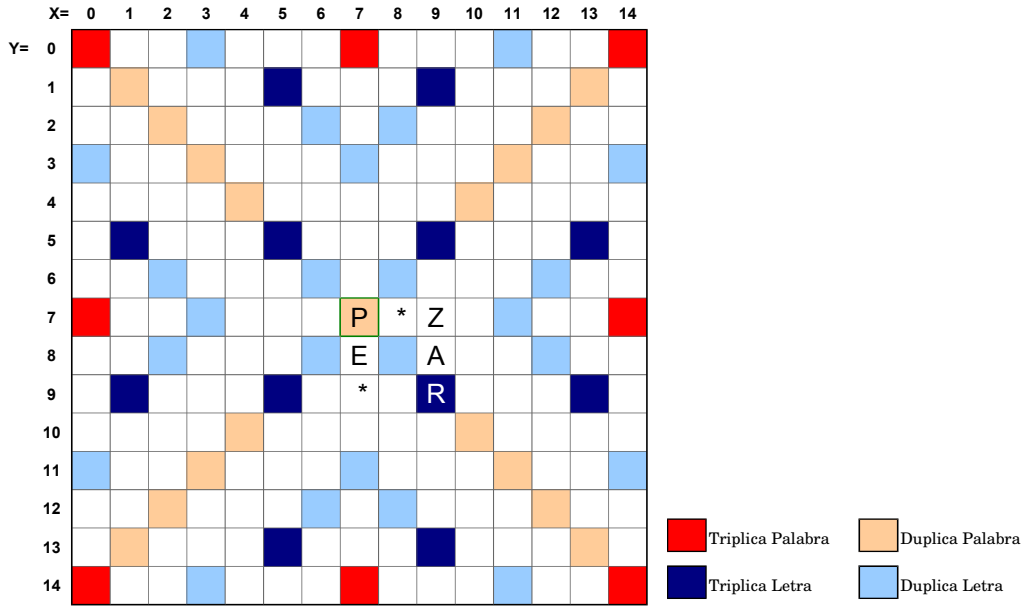

Figura 1: Tablero óptimo para caso de prueba 1

Test 2

Existen dos (2) soluciones óptimas en el `tablero1` si se generan las palabras 'paz' y 'pez', tal como se muestra en la Figura 2 Así, se obtienen 52 puntos.

Test 3

Existen 2 soluciones óptimas en el `tablero4` si se generan las palabras 'pan', 'pez' y 'agua', tal como se muestra en la Figura 3. Así, se obtienen 88 puntos.

Test 6

Existen 2 soluciones óptimas en el `tablero2` si se generan las palabras 'pan', 'pez' y 'agua', tal como se muestra en la Figura 4. En ellas se obtienen 91 puntos.

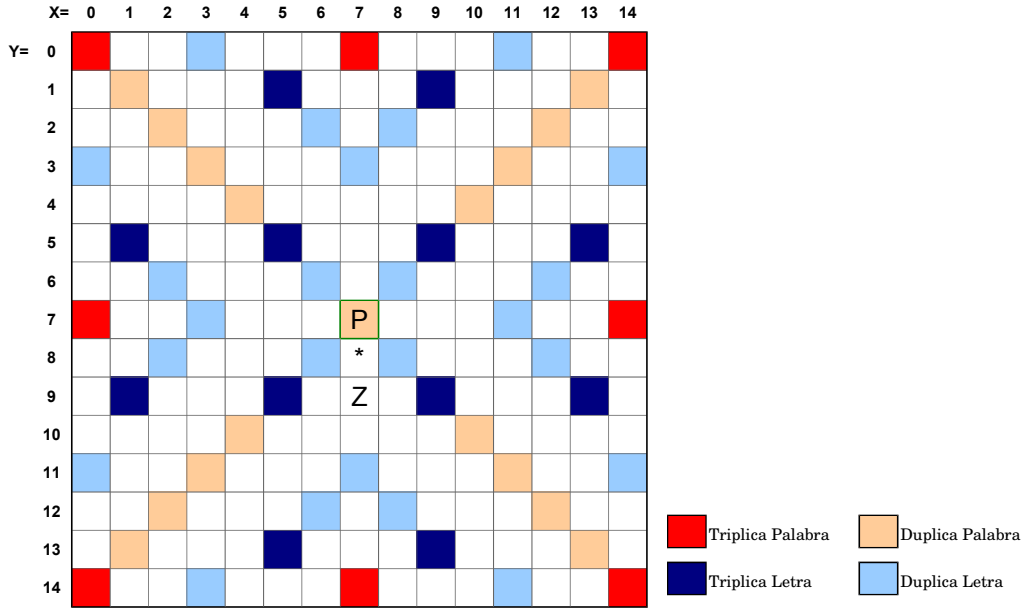

Figura 2: Uno de los tableros óptimos para caso de prueba 2

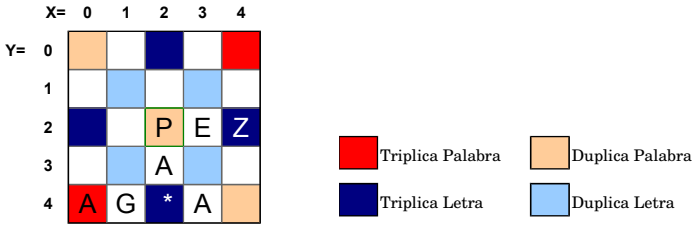

Figura 3: Uno de los tableros óptimos para caso de prueba 3

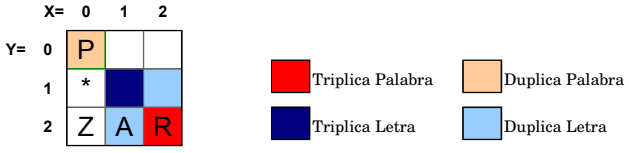

Figura 4: Uno de los tableros óptimos para caso de prueba 6

4. Condiciones de aprobación

El principal objetivo de este trabajo es evaluar el correcto uso del lenguaje PROLOG de forma declarativa para resolver el problema planteado. El código debe estar *adecuadamente* comentado (es decir, los comentarios que escriban deben facilitar la lectura de los predicados). También se debe explicitar cuáles de los argumentos de los predicados auxiliares deben estar instanciados usando `+`, `-` y `?`.

La entrega debe incluir casos de prueba en los que se ejecute al menos una vez cada predicado definido.

Adicionalmente, es deseable que la solución propuesta pueda lidiar con problemas de cierto tamaño. En este caso la cantidad de hechos y reglas es pequeña, pero si la técnica de *generate and test* se implementa de manera ingenua será difícil llegar a resolver aun problemas de tamaño reducido. Ver la sección 5.

En el caso de los predicados que utilicen la técnica de *generate and test*, deberán indicarlo en los comentarios.

5. Algunas consideraciones sobre *generate and test*

Al generar y testear se toma un candidato y se verifica si sirve o no como solución. Para eso hay que decidir cuál es el universo de posibles soluciones. Una mala decisión de este universo puede impactar muy fuertemente en la eficiencia.

Por ejemplo, si se quieren calcular los factores primos de un número, una forma es probar número por número, en este caso el universo de posibles soluciones son los naturales. Otra forma sería tomar como universo de soluciones posibles los números primos. En ambos casos el algoritmo de verificación es el mismo: tomar un candidato y ver si divide o no al número en cuestión.

Si el universo de soluciones posibles son todos los naturales, se tardará mucho en encontrar las soluciones y además, al no estar acotado el conjunto de partida, no se terminará nunca. Si en cambio contamos con una forma eficiente de enumerar los números primos y probar con estos, entonces encontraremos las soluciones de manera mucho más rápida.

En muchos otros casos se pueden generar las soluciones de manera directa, y el predicado *test* sería uno que es verdadero siempre. Esto es deseable siempre que la generación sea un procedimiento sencillo y relativamente eficiente.

Además de la elección del universo de candidatos es importante tener en cuenta que, por como funciona el algoritmo detrás de PROLOG, **cuanto antes** podemos descartar una solución tanto mejor. Es decir que si al ir construyendo un candidato a solución nos damos cuenta por el camino que no va a servir podemos evitarnos una o varias ramas de ejecución potencialmente muy largas.

En algunos problemas se puede partir el proceso de generación y testeo en dos o más etapas que se aplican intercaladas. Por ejemplo:

```
esSolucion(X) :- generarParteUno(X1), testearParteUno(X1),
                 generarParteDos(X1,X), testearParteDos(X).
```

Donde `generarParteDos(+X1,-X2)` parte de una solución parcial ya testeada `X1` y la aumenta para conseguir un candidato a solución `X2` que a su vez deberá ser testeado, ya asumiendo que una parte es correcta.

6. Pautas de Entrega

Se debe entregar el código impreso con la implementación de los predicados pedidos. Cada predicado debe contar con un comentario donde se explique su funcionamiento. Cada predicado asociado a los ejercicios debe contar con ejemplos que muestren que exhibe la funcionalidad solicitada. Además, se debe enviar un e-mail conteniendo el código fuente en Prolog a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PL]` seguido inmediatamente del nombre del grupo.
- El código Prolog debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto (puede adjuntarse un `.zip` o `.tar.gz`).

El código debe poder ser ejecutado en SWI-Prolog. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté adecuadamente comentado. Los objetivos a evaluar en la implementación de los predicados son:

- corrección,
- declaratividad,
- reutilización de predicados previamente definidos
- Uso de unificación, backtracking, generate and test y reversibilidad de los predicados que correspondan.
- Salvo donde se indique lo contrario, los predicados no deben instanciar soluciones repetidas. Vale aclarar que no es necesario filtrar las soluciones repetidas si la repetición proviene de las características de la entrada.

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

7. Referencias y sugerencias

Como referencia se recomienda la bibliografía incluída en el sitio de la materia (ver sección *Bibliografía* → *Programación Lógica*).

Se recomienda que, siempre que sea posible, utilicen los predicados ISO y los de SWI-Prolog ya disponibles. Recomendamos especialmente examinar los predicados y metapredicados que figuran en la sección *Otros* de la página de la materia. Pueden hallar la descripción de los mismos en la ayuda de `SWI-Prolog` (a la que acceden con el predicado `help`). También se puede acceder a la documentación online de SWI-Prolog.