Trabajo Práctico 2 - Programación Lógica
Teléfonos celulares

Fecha de entrega: Jueves 8 de noviembre, hasta las 21 hs.

Los celulares actuales tienen incorporado un dicionario de palabras y un algoritmo que va restringiendo las posibles palabras que puedan ser resultado de la presión de una serie de teclas del aparato.

El objetivo del TP es representar el comportamiento predictivo de los mensajes de texto de los celulares.

Para ello se definen dos predicados que brindan la información necesaria del teléfono celular:

1. Se define el mapeo entre las teclas de celular y los caracteres de la siguiente manera:

```
teclado([        (1, [1]), (2, [2,a,b,c]),   (3, [3,d,e,f]),
           (4, [4,g,h,i]), (5, [5,j,k,l]),   (6, [6,m,o,n]),
         (7, [7,p,q,r,s]), (8, [8,t,u,v]), (9, [9,w,x,y,z]),
                 (0, [0]),       (*, [-]),         (#, [#])
        ])
```

   es decir con una lista de tuplas (D, Xs) donde D es el dígito presionado y Xs es la lista de caracteres que puede representar dicho dígito. Los caracteres se representan por medio de un átomo o de un número entero. Notar que el carácter " " está representado por el átomo "-"

2. El diccionario de palabras se representa de la siguiente manera:

```
diccionario([ [1,a], [l,a], [c,a,s,a], [a], [d,e], [r,e,j,a], [t,i,e,n,e],
              [c,a,s,a,m,i,e,n,t,o], [d,e,l], [a,n,t,e,s]
            ])
```

   es decir una lista de lista de caracteres que representan las palabras conocidas. Puede asumirse que ninguna palabra del diccionario contiene el átomo "-".

1. Predicados pedidos

Se pide definir los siguientes predicados, respetando la instanciación pedida y de tal manera que no se devuelvan soluciones repetidas:

1. `teclasNecesarias(+Palabra, -ListaDigitos)` donde `Palabra` es una lista de caracteres y tiene éxito si la `ListaDigitos` es la lista con los dígitos que deben presionarse para obtenerla. Ejemplo:

```
?- teclasNecesarias([c,a,s,a], Ds).

Ds = [2, 2, 7, 2] ;

No
```

2. `palabraPosible(+ListaDigitos, -Palabra)` donde `ListaDigitos` es una lista de teclas presionadas y tiene éxito si `Palabra` es una palabra del diccionario, y con las teclas presionadas se obtiene esa palabra o un prefijo de la misma.

   Ejemplo:

```
?- palabraPosible([2], P).

P = [a] ;

P = [a, n, t, e, s] ;

P = [c, a, s, a] ;

P = [c, a, s, a, m, i, e, n, t, o] ;

No
```

3. `todasLasPalabrasPosibles(+ListaDigitos, -Palabras)` donde `ListaDigitos` es una lista de teclas presionadas y tiene éxito si `Palabras` es una lista de palabras del diccionario tal que las teclas presionadas generan una lista de caracteres que puede ser prefijo de la mismas.

   Nota: tener en cuenta que la solución debe ser vista como un conjunto. O sea, soluciones con las mismas palabras pero en distinto orden deben ser consideradas como iguales (no deben devolverse repetidos). Ejemplo:

```
?- todasLasPalabrasPosibles([2], Ps).

Ps = [[a], [a, n, t, e, s], [c, a, s, a], [c, a, s, a, m, i, e, n, t, o]] ;

No

?- todasLasPalabrasPosibles([2], [[c, a, s, a],[a], [a, n, t, e, s],
 [c, a, s, a, m, i, e, n, t, o]]).

Yes
```

4. `oracionPosible(+ListaDigitos, -Oracion)` donde `ListaDigitos` es una lista de teclas presionadas que puede incluir el * (que se mapea al espacio), y tiene éxito si se formó una oración correcta. Una oración es correcta si cada una de las secuencias de teclas entre los * puede formar una palabra del diccionario (es decir una palabra como en los items anteriores). Ejemplo:

```
?- oracionPosible([2,*,3], O).

O = [a, -, d, e] ;

O = [a, -, d, e, l] ;

O = [a, n, t, e, s, -, d, e] ;

O = [a, n, t, e, s, -, d, e, l] ;

O = [c, a, s, a, -, d, e] ;

O = [c, a, s, a, -, d, e, l] ;

O = [c, a, s, a, m, i, e, n, t, o, -, d, e] ;

O = [c, a, s, a, m, i, e, n, t, o, -, d, e, l] ;

No
```

1.1. Juego adicional (opcional)

`meJuegoPor(-Xs)`

Este predicado es verdadero si `Xs` es la lista de literales por la que se juega el grupo. La lista debe tener tamaño 5. Cada literal debe ser una palabra (de al menos dos letras) del idioma español. Los literales deben ser todos distintos y alguno de ellos debe ser necesariamente `edificio`.

El grupo que se juegue por la lista de literales que, puesta entre comillas, arroje más resultados en Google (en español) será el grupo ganador de un premio sorpresa.

Aclaración: La catedra se reserva el derecho a descalificar grupos que considere no estén jugando limpio (no vale que el predicado sea verdadero para más de un valor de `Xs` ¡y tampoco vale crear millones de páginas web con la frase que uno elige ni nada similar!).

Por ejemplo (arroja 1 resultado en Google):

```
?- meJuegoPor(Xs).
Xs = [este,es,un,edificio,lindo]
No
```

2. 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. Se pedirá un pequeño informe donde se explique cada predicado definido, aclarando cómo se relaciona, en caso de hacerlo, con los demás predicados (por ejemplo, indicando los predicados que hacen referencia a ellos) y cómo

interviene en la solución del problema. También se debe explicitar cuáles de los argumentos de los predicados auxiliares deben estar instanciados usando `+` y `-`.

Adicionalmente, es deseable que la solución propuesta pueda lidiar con problemas de cierto tamaño. En este caso el diccionario es pequeño, pero si la técnica de *generate and test* se implementa de manera ingenua será difícil llegar a resolver aun problemas de tamaño reducido. Ver el apéndice A.

Para cumplir con este objetivo se recomienda que **antes** de comenzar a implementar diseñen de qué manera van a aplicar la técnica de *generate and test*. Esta decisión debe estar debidamente documentada en el informe. Es decir, deben indicar qué decisiones tomaron para hacer eficiente el proceso de resolución.

3. 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. Asimismo, se debe enviar un mail conteniendo el código fuente Prolog a la dirección `plp-docentes@dc.uba.ar`. Dicho mail debe cumplir con el siguiente formato:

- El subject debe ser "[PLP;TP-PL]" seguido inmediatamente del nombre del grupo.
- **Solamente** el código Prolog comentado (**no** debe incluirse el informe) debe acompañar el mail en forma de **attachment**. El mismo debe poder ejecutarse en SWI-Prolog (indicar claramente cómo se debe ejecutar).

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

A. 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 éstos, 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.

B. Algunos *predicados* definidos en SWI-Prolog

Esta sección contiene algunos predicados ya definidos en la actual implementación de `SWI-Prolog` que pueden ser de utilidad para el desarrollo de trabajo práctico. La descripción de los mismos la pueden hallar en la ayuda de `SWI-Prolog` (invocada con el predicado `help`).

- `member(-Elem, -List)`
- `append(-List1, -List2, -List3)`
- `sublist(+Pred, +List1, -List2)`
- `subset(+Subset, -Set)`
- `maplist(+Pred, -List)`
- `maplist(+Pred, -List1, -List2)`
- `maplist(+Pred, -List1, -List2, -List3)`
- `setof(+Template, +Goal, -Set)`
- `not(+Goal)`