013. Dadas dos listas, determinar si una es prefijo de la otra

Escriba una función que toma dos listas como entrada y devuelve un booleano. Si la primer lista es prefijo de la segunda lista, devuelve verdadero; sino devuelve falso.

Veamos algunos ejemplos:

(prefijo? ‘(a b c) ‘(a b c)) -> #t una lista es prefijo de sí misma
(prefijo? ‘(a b c) ‘(a b c d)) -> #t
(prefijo? ‘(a b c) ‘(a a b c d)) -> #f no es prefijo en el caso de sublista
(prefijo? ‘() ‘(a b c)) -> #t la lista vacía es prefijo de cualquier lista
(prefijo? ‘(a b) ‘(a c d)) -> #f
(prefijo? ‘(a b) ‘(b c d)) -> #f
(prefijo? ‘(a b c) ‘(a b)) -> #f

Analicemos los argumentos en el caso que sí es prefijo:

(prefijo? ‘(a b) ‘(a b c)) -> #t
a / (b) --- a / (b c)
b / () --- b / (c)
() --- (c)
acierta porque luego de comparar por igual elemento a elemento, se llega a vaciar la lista del prefijo. No importa que queden elementos en el segundo argumento.

Y el caso que no es prefijo:

(prefijo? ‘(a b) ‘(a c d)) -> #f
a / (b) --- a / (c d)
b / () --- c / (d)
falla porque b no es igual a c

Otra que falla:

(prefijo? ‘(a b c) ‘(a b)) -> #f
a / (b c) --- a / (b)
b / (c) --- b / ()
falla, porque se vació la segunda lista y aún faltan casar elementos del prefijo (la lista es más corta que el prefijo)

Entonces:

- cuando la primera lista se vacía, el programa debe devolver verdadero.
- Cuando las cabezas son diferentes, debe devolver falso.
- Cuando la primer lista no está vacía y la segunda sí, se devuelve falso.
- En otro caso se debe eliminar las cabezas (que son iguales) y se continúa recursivamente.

Ahora podemos considerar el código que implemente estas condiciones:

```
(define prefijo?
  (lambda (pref lst)
    (cond [
           ((null? pref) #t)
           ((not (equal? (car pref) (car lst))) #f
           ((and (not (null? pref)) (null? lst)) #f)
           (else (prefijo? (cdr pref) (cdr lst)))])))
```

La verdad es que se ve medio feo. Las condiciones que devuelven explícitamente **`#t`** o **`#f`** muestran que la expresión booleana se puede escribir mejor. (Pista: el enano imperativo nos sofoca con sus explicaciones… pues imperativas)

Podemos preguntarnos como si fuera una fórmula booleana que debemos escribir: ¿cuándo es cierto que **`pref`** es prefijo de **`lst`**?

Cuando el **pref** es nulo o cuando siendo no nulos el **pref** ni **lst**, las cabezas son iguales y es prefijo la cola del **pref** de la cola de **lst**. Por cierto, si **pref** es no nulo y **lst** es nulo, eso es da falso como resultado. Uff! El castellano no parece apropiado para expresarlo. Pongamos un poco de lógica proposicional para aclarar:

nulo_pref: pref es nula
nulo_lst: lst es nula
cabezas_iguales: el car de pref es igual al car de lst
prefijos_colas: prefijo? aplicado a los cdr de ambos argumentos

es prefijo si y sólo si: (*nulo_pref* or (¬*nulo_lst* and *cabezas_iguales* and *prefijos_colas*))

(tengo que asegurar que **lst** no es nula para poder tomar la cabeza)
Entonces quedaría:

```
(define prefijo?
  (lambda (pref lst)
    (or (null? pref)
        (and (not (null? lst))
             (equal? (car pref) (car lst))
             (prefijo? (cdr pref) (cdr lst)) ))))
```

Note la descripción declarativa acerca de la relación entre el par de valores de entrada, que entrega como valor de salida. No es una serie de “si hacés esto, retornar eso”.

Por otro lado si queremos convencernos de que está correcto, tenemos la argumentación lógica. Tenemos código sobre el cual podemos ***razonar lógicamente***. Pretty cool!