UNIVERSIDAD DE BUENOS AIRES

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Paradigmas de Lenguajes de Programación

Trabajo Práctico - Programación Lógica

4 de noviembre de 2014

Grupo: Resolución Súper Lógico Deteminística

Integrante	LU	Correo electrónico
Aisemberg, Dan	242/12	dea4493@hotmail.com
Almansi, Emilio	674/12	ealmansi@gmail.com
Levy Alfie, Jonás	081/12	jonaslevy5@gmail.com

```
%% Automatas de ejemplo. Si agregan otros, mejor.
ejemplo(1, a(s1, [sf], [(s1, a, sf)])).
ejemplo(2, a(si, [si], [(si, a, si)])).
ejemplo(3, a(si, [si], [])).
ejemplo(4, a(s1, [s2, s3], [(s1, a, s1), (s1, a, s2), (s1, b, s3)])).
ejemplo(5, a(s1, [s2, s3], [(s1, a, s1), (s1, b, s2), (s1, c, s3), (s2, c, s3)])).
ejemplo(6, a(s1, [s3], [(s1, b, s2), (s3, n, s2), (s2, a, s3)])).
ejemplo(7, a(s1, [s2], [(s1, a, s3), (s3, a, s3), (s3, b, s2), (s2, b, s2)])).
ejemplo(8, a(s1, [sf], [(s1, a, s2), (s2, a, s3), (s2, b, s3), (s3, a, s1), (s3, b, s2), (s3
    , b, s4), (s4, f, sf)])). %No deterministico :)
ejemplo(9, a(s1, [s1], [(s1, a, s2), (s2, b, s1)])).
ejemplo(10, a(s1, [s10, s11],
        [(s2, a, s3), (s4, a, s5), (s9, a, s10), (s5, d, s6), (s7, g, s8), (s15, g, s11), (
    s6, i, s7), (s13, l, s14), (s8, m, s9), (s12, o, s13), (s14, o, s15), (s1, p, s2), (s3,
    r, s4), (s2, r, s12), (s10, s, s11)])).

ejemploMalo(1, a(s1, [s2], [(s1, a, s1), (s1, b, s2), (s2, b, s2), (s2, a, s3)])). %s3 es un
     estado sin salida.
ejemploMalo(2, a(s1, [sf], [(s1, a, s1), (sf, b, sf)])). %sf no es alcanzable.
ejemploMalo(3, a(s1, [s2, s3], [(s1, a, s3), (s1, b, s3)])). %s2 no es alcanzable.
ejemploMalo(4, a(s1, [s3], [(s1, a, s3), (s2, b, s3)])). %s2 no es alcanzable.
ejemploMalo(5, a(s1, [s3, s2, s3], [(s1, a, s2), (s2, b, s3)])). %Tiene un estado final
    repetido.
ejemploMalo(6, a(s1, [s3], [(s1, a, s2), (s2, b, s3), (s1, a, s2)])). %Tiene una transicion
    repetida.
ejemploMalo(7, a(s1, [], [(s1, a, s2), (s2, b, s3)])). %No tiene estados finales.

%%Proyectores de automata
inicialDe(a(I, _, _), I).
finalesDe(a(_, F, _), F).
transicionesDe(a(_, _, T), T).

%Auxiliar dada en clase
%desde(+X, -Y).
desde(X, X).
desde(X, Y) :- desde(X, Z), Y is Z + 1.

%% Predicados pedidos.
%% 1) %esDeterministico(+Automata)
%% un automata es deterministico si no tiene transiciones, o si no existen dos
%% transiciones con igual estado de salida S y etiqueta L.
%% G&T: Este predicado NO hace uso de Generate & Test.
esDeterministico(a(_, _, [])).
esDeterministico(a(_, _, [(S, L, _) | Ts])) :- not(member((S, L, _), Ts)),
  esDeterministico(a(_, _, Ts)).

%% 2) estados(+Automata, ?Estados)
%% primero acumulamos todos los estados que aparecen como estado inicial, estado
%% final o en una transicion, sin filtrar repetidos (Ss_rep). despues, ordenamos y quitamos
%% repetidos a la lista obtenida (instanciando Ss_sort). por ultimo, distinguimos dos casos:
%% si el argumento estados es una variable, la instanciamos en Ss_sort.
%% si el argumento estados ya esta instanciado, verificamos que al ordenarlo y quitar
    repetidos se obtenga Ss_sort.
%% G&T: Este predicado NO hace uso de Generate & Test.
estados(a(I, Fs, Ts), Ss) :- estadosConRepetidos(a(I, Fs, Ts), Ss_rep),
  sort(Ss_rep, Ss_sort), (var(Ss), Ss = Ss_sort ; nonvar(Ss), sort(Ss, Ss_sort)).

%% estadosConRepetidos(+Automata, -Estados)
%% G&T: Este predicado NO hace uso de Generate & Test.
estadosConRepetidos(a(I, [], []), [I]).
estadosConRepetidos(a(I, [F | Fs], []), [F | Ss]) :-
  estadosConRepetidos(a(I, Fs, []), Ss).
estadosConRepetidos(a(I, Fs, [(S_d, _, S_h) | Ts]), [S_d, S_h | Ss]) :-
```

```
  estadosConRepetidos(a(I, Fs, Ts), Ss).

%% 3)esCamino(+Automata, ?EstadoInicial, ?EstadoFinal, +Camino)
%% una lista con un unico estado Si siempre es un camino de Si en Si.
%% una lista con al menos dos estados (Si, S2) es un camino de Si a Sf si existe
%% una transicion de Si a S2, y ademas la cola de la lista es un camino entre
%% S2 y Sf.
%% G&T: Este predicado hace uso de Generate & Test en las ultimas 2 clausulas, siendo member
     el generate y esCamino el test.
esCamino(A, Si, Si, [Si]) :- estados(A, Ss), member(Si, Ss).
esCamino(A, Si, Sf, [Si, S2 | X]) :- transicionesDe(A, Ts), member((Si, _, S2), Ts),
  esCamino(A, S2, Sf, [S2 | X]).

%% 4) el predicado anterior es o no reversible con respecto a Camino y por que?
%% El predicado no es reversible con respecto a Camino porque, en caso de no instanciar este
     argumento,
%% uno deberia poder generar todos los caminos posibles en el automata al llamar al
    predicado con los
%% argumentos EstadoInicial y EstadoFinal no instanciados. En ese caso, el predicado del
    punto anterior genera caminos
%% validos en el automata, pero no es capaz de generar todos (pierde soluciones).

%% 5) caminoDeLongitud(+Automata, +N, -Camino, -Etiquetas, ?S1, ?S2)
%% Este predicado funciona de manera recursiva.
%% El caso base es querer generar un camino de longitud 1 entre un estado y si mismo, en
    cuyo caso el camino es ese mismo estado.
%% Este camino existe de existir dicho estado en los estados del automata.
%% El caso general es para caminos de longitud mayor a 1, los cuales se consiguen si existen
     un camino de la longitud pedida menos 1,
%% y una transicion para extender dicho camino.
%% G&T: Este predicado hace uso de Generate & Test en las ultimas 3 clausulas, siendo member
     el generate y las otras dos el test.
caminoDeLongitud(A, 1, [S], [], S, S) :- estados(A, Ss), member(S, Ss).
caminoDeLongitud(A, N, [S1 | Cr], [L | Er], S1, S2) :- N > 1, transicionesDe(A, Ts),
  member((S1, L, S), Ts), Nm1 is N - 1, caminoDeLongitud(A, Nm1, Cr, Er, S, S2).

%% 6) alcanzable(+Automata, +Estado)
%% Un estado es alcanzable si es el estado inicial del automata,
%% o si es alcanzable desde dicho estado inicial.
%% G&T: Este predicado NO hace uso de Generate & Test.
alcanzable(A, S) :- (inicialDe(A, S) ; inicialDe(A, I), alcanzableDesde(A, S, I)), !.

%% alcanzableDesde(+Automata, +Estado, +EstadoDesde)
%% Estado es alcanzable en Automata desde EstadoDesde si existe un camino
%% de longitud entre 2 y la cantidad de estados mas 1 que conecte dichos estados.
%% Esto es asi porque definimos que un estado no es alcanzable desde si mismo,
%% a menos que forme parte de un ciclo, en cuyo caso el ciclo contendria a todos los estados
     como maximo.
%% G&T: Este predicado hace uso de Generate & Test en las ultimas 2 clausulas (sin contar el
     cut), siendo between el generate y caminoDeLongitud el test.
alcanzableDesde(A, S, Sd) :- estados(A, Ss), length(Ss, N), Np1 is N + 1,
  between(2, Np1, M), caminoDeLongitud(A, M, _, _, Sd, S), !.

%% 7) automataValido(+Automata)
%% Un automata es valido si
%% todos los estados tienen transiciones salientes (a menos que sea final en cuyo caso puede
     no tener),
%% todos los estados son alcanzables desde el estado inicial,
%% tiene al menos un estado final,
%% no hay estados repetidos,
%% y no hay transiciones repetidas.
%% Escribimos un predicado auxiliar para cada uno de estos items, asi un automata es valido
    si cumple con todos los predicados simultaneamente.
```

```
%% G&T: Este predicado NO hace uso de Generate & Test.
automataValido(A) :-
  transicionesSalientes(A),
  estadosAlcanzables(A),
  alMenosUnFinal(A),
  finalesSinRepetidos(A),
  transicionesSinRepetidos(A).

%% transicionesSalientes(+Automata)
%% Esto se cumple si para cada estado, o bien pertenece a los estados finales,
%% o hay alguna transicion que salga de dicho estado.
%% G&T: Este predicado NO hace uso de Generate & Test.
transicionesSalientes(A) :- finalesDe(A, Fs), transicionesDe(A, Ts), estados(A, Ss),
  forall(member(S, Ss), ( member(S, Fs) ; member((S, _, _), Ts) ) ).

%% estadosAlcanzables(+Automata)
%% que sean todos los estados del automata alcanzables segun el predicado alcanzable.
%% G&T: Este predicado NO hace uso de Generate & Test.
estadosAlcanzables(A) :- estados(A, Ss),
  forall(member(S, Ss), ( alcanzable(A, S) ) ).

%% alMenosUnFinal(+Automata)
%% si la longitud de estados finales es mayor a 0.
%% G&T: Este predicado NO hace uso de Generate & Test.
alMenosUnFinal(A) :- finalesDe(A, Fs), length(Fs, Fs_len), Fs_len > 0.

%% finalesSinRepetidos(+Automata)
%% si la longitud de estados, y el conjunto que se obtiene de ellos,
%% es la misma.
%% G&T: Este predicado NO hace uso de Generate & Test.
finalesSinRepetidos(A) :- finalesDe(A, Fs), list_to_set(Fs, Fs_set),
  length(Fs, N), length(Fs_set, N).

%% transicionesSinRepetidos(+Automatas)
%% si la longitud de transiciones, y el conjunto que se obtiene de ellas,
%% es la misma.
%% G&T: Este predicado NO hace uso de Generate & Test.
transicionesSinRepetidos(A) :- transicionesDe(A, Ts), list_to_set(Ts, Ts_set),
  length(Ts, N), length(Ts_set, N).

%%--- NOTA: De aca en adelante se asume que los automatas son validos.
%% 8) hayCiclo(+Automata)
%% Hay algun ciclo en un automata si existe un estado que sea alcanzable desde si mismo,
    segun el predicado alcanzableDesde.
%% G&T: Este predicado NO hace uso de Generate & Test.
hayCiclo(A) :- alcanzableDesde(A, S, S).

%% 9) reconoce(+Automata, ?Palabra)
%% Un automata reconoce una palabra si la palabra es una lista de etiquetas
%% que se consumen en un camino desde el estado inicial hasta uno final.
%% Hay dos casos para esto;
%% Si el automata no posee ciclos, entonces hay una cantidad finita de caminos entre el
    estado inicial y uno final
%% los cuales a lo sumo pasan por todos los estados.
%% Si hay ciclos entonces la cantidad de palabras pasa a ser infinita, porque se puede pasar
     una cantidad arbitraria de veces por un ciclo
%% y luego llegar a un estado final. Por tanto las palabras son arbitrariamente grandes
    tambien.
%% G&T: Este predicado hace uso de Generate & Test en las ultimas 2 clausulas en ambas
    reglas, siendo member el generate y caminoDeLongitud el test.

%% Sin ciclo (finitos)
reconoce(A, P) :- not(hayCiclo(A)), estados(A, Ss), length(Ss, N),
```

```
  between(1, N, M), inicialDe(A, I), finalesDe(A, Fs), member(F, Fs),
  caminoDeLongitud(A, M, _, P, I, F).

%% Con ciclo (infinitos)
reconoce(A, P) :- hayCiclo(A), length(P, N), Np1 is N + 1,
  inicialDe(A, I), finalesDe(A, Fs), member(F, Fs),
  caminoDeLongitud(A, Np1, _, P, I, F).

%% 10) PalabraMasCorta(+Automata, ?Palabra)
%% Para conseguir las palabras mas cortas, primero conseguimos la longitud que tienen,
%% y luego armamos los caminos de dicha longitud mas 1, que empiecen en el estado inicial y
    terminen en alguno final.
%% G&T: Este predicado hace uso de Generate & Test en las ultimas 2 clausulas, siendo member
     el generate y caminoDeLongitud el test.
palabraMasCorta(A, P) :-
  longitudPalabraMasCorta(A, N), Np1 is N + 1, inicialDe(A, I), finalesDe(A, Fs),
  member(F, Fs), caminoDeLongitud(A, Np1, _, P, I, F).

%% longitudPalabraMasCorta(+Automata, -Longitud)
%% Como el predicado reconoce va armando las palabras con caminoDeLongitud, mientras va
    aumentando dicho longitud,
%% entonces va armando las palabras de manera creciente en longitud. De manera que la primer
     palabra que genere
%% sera una de longitud minima. Usamos cut (!) para quedarnos unicamente con este primer
    resultado,
%% de modo que el predicado nos da la longitud que queremos.
%% G&T: Este predicado NO hace uso de Generate & Test.
longitudPalabraMasCorta(A, N) :-
  reconoce(A, PC), length(PC, N), !.

%%-----------------
%%----- Tests -----
%%-----------------

%% Tests nuestros
tests_predicados :-
  print('Testeando todos los predicados\n'),
  tests_esDeterministico,
  tests_estados,
  tests_esCamino,
  tests_caminoDeLongitud,
  tests_alcanzable,
  tests_automataValido,
  tests_hayCiclo,
  tests_reconoce,
  tests_palabraMasCorta,
  print('Todos Ok!\n').

tests_esDeterministico :-
  print('testeando: esDeterministico\n'),
  ejemplo(1, A1), esDeterministico(A1), print('Test 1 Ok\n'),
  ejemplo(2, A2), esDeterministico(A2), print('Test 2 Ok\n'),
  ejemplo(3, A3), esDeterministico(A3), print('Test 3 Ok\n'),
  ejemplo(4, A4), not(esDeterministico(A4)), print('Test 4 Ok\n'),
  ejemplo(5, A5), esDeterministico(A5), print('Test 5 Ok\n'),
  ejemplo(6, A6), esDeterministico(A6), print('Test 6 Ok\n'),
  ejemplo(7, A7), esDeterministico(A7), print('Test 7 Ok\n'),
  ejemplo(8, A8), not(esDeterministico(A8)), print('Test 8 Ok\n'),
  ejemplo(9, A9), esDeterministico(A9), print('Test 9 Ok\n'),
  ejemplo(10, A10), esDeterministico(A10), print('Test 10 Ok\n'),
  ejemploMalo(1, AM1), esDeterministico(AM1), print('Test 10 Ok\n'),
  ejemploMalo(2, AM2), esDeterministico(AM2), print('Test 11 Ok\n'),
  ejemploMalo(3, AM3), esDeterministico(AM3), print('Test 12 Ok\n'),
```

```
  ejemploMalo(4, AM4), esDeterministico(AM4), print('Test 13 Ok\n'),
  ejemploMalo(5, AM5), esDeterministico(AM5), print('Test 14 Ok\n'),
  ejemploMalo(6, AM6), not(esDeterministico(AM6)), print('Test 15 Ok\n'),
  ejemploMalo(7, AM7), esDeterministico(AM7), print('Test 16 Ok\n'),
  print('esDeterministico Ok\n\n').

tests_estados :-
  print('testeando: estados\n'),
  ejemplo(1, A1_a), estados(A1_a, Ss1_a), Ss1_a = [s1, sf], print('Test 1 (a) Ok\n'),
  ejemplo(1, A1_b), estados(A1_b, [s1, sf, s1, sf]), print('Test 1 (b) Ok\n'),
  ejemplo(2, A2_a), estados(A2_a, Ss2_a), Ss2_a = [si], print('Test 2 (a) Ok\n'),
  ejemplo(2, A2_b), estados(A2_b, [si, si]), print('Test 2 (b) Ok\n'),
  ejemplo(3, A3_a), estados(A3_a, Ss3_a), Ss3_a = [si], print('Test 3 (a) Ok\n'),
  ejemplo(3, A3_b), estados(A3_b, [si, si]), print('Test 3 (b) Ok\n'),
  ejemplo(4, A4_a), estados(A4_a, Ss4_a), Ss4_a = [s1, s2, s3], print('Test 4 (a) Ok\n'),
  ejemplo(4, A4_b), estados(A4_b, [s1, s2, s3, s1, s2, s3]), print('Test 4 (b) Ok\n'),
  ejemplo(5, A5_a), estados(A5_a, Ss5_a), Ss5_a = [s1, s2, s3], print('Test 5 (a) Ok\n'),
  ejemplo(5, A5_b), estados(A5_b, [s1, s2, s3, s1, s2, s3]), print('Test 5 (b) Ok\n'),
  ejemplo(6, A6_a), estados(A6_a, Ss6_a), Ss6_a = [s1, s2, s3], print('Test 6 (a) Ok\n'),
  ejemplo(6, A6_b), estados(A6_b, [s1, s2, s3, s1, s2, s3]), print('Test 6 (b) Ok\n'),
  ejemplo(7, A7_a), estados(A7_a, Ss7_a), Ss7_a = [s1, s2, s3], print('Test 7 (a) Ok\n'),
  ejemplo(7, A7_b), estados(A7_b, [s1, s2, s3, s1, s2, s3]), print('Test 7 (b) Ok\n'),
  ejemplo(8, A8_a), estados(A8_a, Ss8_a), Ss8_a = [s1, s2, s3, s4, sf], print('Test 8 (a) Ok
    \n'),
  ejemplo(8, A8_b), estados(A8_b, [s1, s2, s3, s4, sf, s1, s2, s3, s4, sf]), print('Test 8 (
    b) Ok\n'),
  ejemplo(9, A9_a), estados(A9_a, Ss9_a), Ss9_a = [s1, s2], print('Test 9 (a) Ok\n'),
  ejemplo(9, A9_b), estados(A9_b, [s1, s2, s1, s2]), print('Test 9 (b) Ok\n'),
  ejemplo(10, A10_a), estados(A10_a, Ss10_a), Ss10_a = [s1, s10, s11, s12, s13, s14, s15, s2
    , s3, s4, s5, s6, s7, s8, s9], print('Test 10 (a) Ok\n'),
  ejemplo(10, A10_b), estados(A10_b, [s1, s10, s11, s12, s13, s14, s15, s2, s3, s4, s5, s6,
    s7, s8, s9, s1, s10, s11, s12, s13, s14, s15, s2, s3, s4, s5, s6, s7, s8, s9]), print('
    Test 10 (b) Ok\n'),
  ejemploMalo(1, AM1_a), estados(AM1_a, SsM1_a), SsM1_a = [s1, s2, s3], print('Test 11 (a)
    Ok\n'),
  ejemploMalo(1, AM1_b), estados(AM1_b, [s1, s2, s3, s1, s2, s3]), print('Test 11 (b) Ok\n')
    ,
  ejemploMalo(2, AM2_a), estados(AM2_a, SsM2_a), SsM2_a = [s1, sf], print('Test 12 (a) Ok\n'
    ),
  ejemploMalo(2, AM2_b), estados(AM2_b, [s1, sf, s1, sf]), print('Test 12 (b) Ok\n'),
  ejemploMalo(3, AM3_a), estados(AM3_a, SsM3_a), SsM3_a = [s1, s2, s3], print('Test 13 (a)
    Ok\n'),
  ejemploMalo(3, AM3_b), estados(AM3_b, [s1, s2, s3, s1, s2, s3]), print('Test 13 (b) Ok\n')
    ,
  ejemploMalo(4, AM4_a), estados(AM4_a, SsM4_a), SsM4_a = [s1, s2, s3], print('Test 14 (a)
    Ok\n'),
  ejemploMalo(4, AM4_b), estados(AM4_b, [s1, s2, s3, s1, s2, s3]), print('Test 14 (b) Ok\n')
    ,
  ejemploMalo(5, AM5_a), estados(AM5_a, SsM5_a), SsM5_a = [s1, s2, s3], print('Test 15 (a)
    Ok\n'),
  ejemploMalo(5, AM5_b), estados(AM5_b, [s1, s2, s3, s1, s2, s3]), print('Test 15 (b) Ok\n')
    ,
  ejemploMalo(6, AM6_a), estados(AM6_a, SsM6_a), SsM6_a = [s1, s2, s3], print('Test 16 (a)
    Ok\n'),
  ejemploMalo(6, AM6_b), estados(AM6_b, [s1, s2, s3, s1, s2, s3]), print('Test 16 (b) Ok\n')
    ,
  ejemploMalo(7, AM7_a), estados(AM7_a, SsM7_a), SsM7_a = [s1, s2, s3], print('Test 17 (a)
    Ok\n'),
  ejemploMalo(7, AM7_b), estados(AM7_b, [s1, s2, s3, s1, s2, s3]), print('Test 17 (b) Ok\n')
    ,
  print('estados Ok\n\n').

tests_esCamino :-
```

```
  print('testeando: esCamino\n'),
  ejemplo(5, A1), esCamino(A1, s1, s2, [s1,s2]), !, print('Test 1 Ok\n'),
  ejemplo(5, A2), esCamino(A2, _, _, [s1,s1,s2,s3]), !, print('Test 2 Ok\n'),
  ejemplo(5, A3), esCamino(A3, s1, s1, [s1]), !, print('Test 3 Ok\n'),
  ejemplo(5, A4), esCamino(A4, s1, s1, [s1,s1,s1,s1,s1,s1]), !, print('Test 4 Ok\n'),
  print('esCamino Ok\n\n').

tests_caminoDeLongitud :-
  print('testeando: caminoDeLongitud\n'),
  ejemplo(5, A1), caminoDeLongitud(A1, 3, _, _, _, _), !, print('Test 1 Ok\n'),
  print('caminoDeLongitud Ok\n\n').

tests_alcanzable :-
  print('testeando: alcanzable\n'),
  ejemplo(1, A1), forall(member(S, [s1, sf]), alcanzable(A1, S)), print('Test 1 Ok\n'),
  ejemplo(2, A2), forall(member(S, [si]), alcanzable(A2, S)), print('Test 2 Ok\n'),
  ejemplo(3, A3), forall(member(S, [si]), alcanzable(A3, S)), print('Test 3 Ok\n'),
  ejemplo(4, A4), forall(member(S, [s1, s2, s3]), alcanzable(A4, S)), print('Test 4 Ok\n'),
  ejemplo(5, A5), forall(member(S, [s1, s2, s3]), alcanzable(A5, S)), print('Test 5 Ok\n'),
  ejemplo(6, A6), forall(member(S, [s1, s2, s3]), alcanzable(A6, S)), print('Test 6 Ok\n'),
  ejemplo(7, A7), forall(member(S, [s1, s2, s3]), alcanzable(A7, S)), print('Test 7 Ok\n'),
  ejemplo(8, A8), forall(member(S, [s1, s2, s3, s4, sf]), alcanzable(A8, S)), print('Test 8
    Ok\n'),
  ejemplo(9, A9), forall(member(S, [s1, s2]), alcanzable(A9, S)), print('Test 9 Ok\n'),
  ejemplo(10, A10), forall(member(S, [s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13
    , s14, s15]), alcanzable(A10, S)), print('Test 10 Ok\n'),
  ejemploMalo(1, AM1), forall(member(S, [s1, s2, s3]), alcanzable(AM1, S)), print('Test 11
    Ok\n'),
  ejemploMalo(2, AM2), not(alcanzable(AM2, sf)), forall(member(S, [s1]), alcanzable(AM2, S))
    , print('Test 12 Ok\n'),
  ejemploMalo(3, AM3), not(alcanzable(AM3, s2)), forall(member(S, [s1, s3]), alcanzable(AM3,
    S)), print('Test 13 Ok\n'),
  ejemploMalo(4, AM4), not(alcanzable(AM4, s2)), forall(member(S, [s1, s3]), alcanzable(AM4,
    S)), print('Test 14 Ok\n'),
  ejemploMalo(5, AM5), forall(member(S, [s1, s2, s3]), alcanzable(AM5, S)), print('Test 15
    Ok\n'),
  ejemploMalo(6, AM6), forall(member(S, [s1, s2, s3]), alcanzable(AM6, S)), print('Test 16
    Ok\n'),
  ejemploMalo(7, AM7), forall(member(S, [s1, s2, s3]), alcanzable(AM7, S)), print('Test 17
    Ok\n'),
  print('alcanzable Ok\n\n').

tests_automataValido :-
  print('testeando: automataValido\n'),
  ejemplo(1, A1), automataValido(A1), print('Test 1 Ok\n'),
  ejemplo(2, A2), automataValido(A2), print('Test 2 Ok\n'),
  ejemplo(3, A3), automataValido(A3), print('Test 3 Ok\n'),
  ejemplo(4, A4), automataValido(A4), print('Test 4 Ok\n'),
  ejemplo(5, A5), automataValido(A5), print('Test 5 Ok\n'),
  ejemplo(6, A6), automataValido(A6), print('Test 6 Ok\n'),
  ejemplo(7, A7), automataValido(A7), print('Test 7 Ok\n'),
  ejemplo(8, A8), automataValido(A8), print('Test 8 Ok\n'),
  ejemplo(9, A9), automataValido(A9), print('Test 9 Ok\n'),
  ejemplo(10, A10), automataValido(A10), print('Test 10 Ok\n'),
  ejemploMalo(1, A11), not(automataValido(A11)), print('Test 11 Ok\n'),
  ejemploMalo(2, A12), not(automataValido(A12)), print('Test 12 Ok\n'),
  ejemploMalo(3, A13), not(automataValido(A13)), print('Test 13 Ok\n'),
  ejemploMalo(4, A14), not(automataValido(A14)), print('Test 14 Ok\n'),
  ejemploMalo(5, A15), not(automataValido(A15)), print('Test 15 Ok\n'),
  ejemploMalo(6, A16), not(automataValido(A16)), print('Test 16 Ok\n'),
  ejemploMalo(7, A17), not(automataValido(A17)), print('Test 17 Ok\n'),
  print('automataValido Ok\n\n').
```

```
tests_hayCiclo :-
  print('testeando: hayCiclo\n'),
  ejemplo(1, A1), not(hayCiclo(A1)), print('Test 1 Ok\n'),
  ejemplo(2, A2), hayCiclo(A2), print('Test 2 Ok\n'),
  ejemplo(3, A3), not(hayCiclo(A3)), print('Test 3 Ok\n'),
  ejemplo(4, A4), hayCiclo(A4), print('Test 4 Ok\n'),
  ejemplo(5, A5), hayCiclo(A5), print('Test 5 Ok\n'),
  ejemplo(6, A6), hayCiclo(A6), print('Test 6 Ok\n'),
  ejemplo(7, A7), hayCiclo(A7), print('Test 7 Ok\n'),
  ejemplo(8, A8), hayCiclo(A8), print('Test 8 Ok\n'),
  ejemplo(9, A9), hayCiclo(A9), print('Test 9 Ok\n'),
  ejemplo(10, A10), not(hayCiclo(A10)), print('Test 10 Ok\n'),
  print('hayCiclo Ok\n\n').

tests_reconoce :-
  print('testeando: reconoce\n'),
  ejemplo(10, A1), reconoce(A1, [p, X, r, X, d, i, _, m, X, s]), print('Test 1 Ok\n'),
  ejemplo(9, A2), reconoce(A2, [a, b, a, b, a, b, a, b]), print('Test 2 Ok\n'),
  ejemplo(7, A3), reconoce(A3, [a, a, a, b, b]), print('Test 3 Ok\n'),
  ejemplo(7, A4), not(reconoce(A4, [b])), print('Test 4 Ok\n'),
  print('reconoce Ok\n\n').

tests_palabraMasCorta :-
  print('testeando: palabraMasCorta\n'),
  ejemplo(2, A1), findall(P1, palabraMasCorta(A1, P1), [[]]), print('Test 1 Ok\n'),
  ejemplo(4, A2), findall(P2, palabraMasCorta(A2, P2), Lista2), length(Lista2, 2), sort(
    Lista2, [[a], [b]]), print('Test 2 Ok\n'),
  ejemplo(5, A3), findall(P3, palabraMasCorta(A3, P3), Lista3), length(Lista3, 2), sort(
    Lista3, [[b], [c]]), print('Test 3 Ok\n'),
  ejemplo(6, A4), findall(P4, palabraMasCorta(A4, P4), [[b, a]]), print('Test 4 Ok\n'),
  ejemplo(7, A5), findall(P5, palabraMasCorta(A5, P5), [[a, b]]), print('Test 5 Ok\n'),
  ejemplo(8, A6), findall(P6, palabraMasCorta(A6, P6), Lista6), length(Lista6, 2), sort(
    Lista6, [[a, a, b, f], [a, b, b, f]]), print('Test 6 Ok\n'),
  ejemplo(10, A7), findall(P7, palabraMasCorta(A7, P7), [[p, r, o, l, o, g]]), print('Test
    7 Ok\n'),
  print('palabraMasCorta Ok\n\n').

%% Tests de la catedra
test(1) :- forall(ejemplo(_, A), automataValido(A)).
test(2) :- not((ejemploMalo(_, A), automataValido(A))).
test(3) :- ejemplo(10, A), reconoce(A, [p, X, r, X, d, i, _, m, X, s]).
test(4) :- ejemplo(9, A), reconoce(A, [a, b, a, b, a, b, a, b]).
test(5) :- ejemplo(7, A), reconoce(A, [a, a, a, b, b]).
test(6) :- ejemplo(7, A), not(reconoce(A, [b])).
test(7) :- ejemplo(2, A), findall(P, palabraMasCorta(A, P), [[]]).
test(8) :- ejemplo(4, A), findall(P, palabraMasCorta(A, P), Lista), length(Lista, 2), sort(
    Lista, [[a], [b]]).
test(9) :- ejemplo(5, A), findall(P, palabraMasCorta(A, P), Lista), length(Lista, 2), sort(
    Lista, [[b], [c]]).
test(10) :- ejemplo(6, A), findall(P, palabraMasCorta(A, P), [[b, a]]).
test(11) :- ejemplo(7, A), findall(P, palabraMasCorta(A, P), [[a, b]]).
test(12) :- ejemplo(8, A), findall(P, palabraMasCorta(A, P), Lista), length(Lista, 2), sort
    (Lista, [[a, a, b, f], [a, b, b, f]]).
test(13) :- ejemplo(10, A), findall(P, palabraMasCorta(A, P), [[p, r, o, l, o, g]]).
test(14) :- forall(member(X, [2, 4, 5, 6, 7, 8, 9]), (ejemplo(X, A), hayCiclo(A))).
test(15) :- not((member(X, [1, 3, 10]), ejemplo(X, A), hayCiclo(A))).
tests :- forall(between(1, 15, N), test(N)). %IMPORTANTE: Actualizar la cantidad total de
    tests para contemplar los que agreguen ustedes.
```

Código 1: Implementación y tests