```
*****************************************************************************
************                        Interfaz                         ************
*****************************************************************************

Se explica con: Restricción
Géneros: restricción

*****************************************************************************
************                     Operaciones                         ************
*****************************************************************************

Var(in s : string) → res : restricción
Pre ≡ {true}
Post ≡ {res =obs ⟨s⟩}
Complejidad: O(1)
Descripcion: Crea una nueva restricción

And(in r1 : restricción, in r2 : restricción) → res : restricción
Pre ≡ {true}
Post ≡ {res =obs r1 AND r2}
Complejidad: O(1)
Descripcion: Crea una nueva restricción que tiene que cumplir con r1 y r2

Or(in r1 : restricción, in r2 : restricción) → res : restricción
Pre ≡ {true}
Post ≡ {res =obs r1 OR r2}
Complejidad: O(1)
Descripcion: Crea una nueva restricción que tiene que cumplir con r1 o r2

Not(in r : restricción) → res : restricción
Pre ≡ {true}
Post ≡ {res =obs NOT r}
Complejidad: O(1)
Descripcion: Crea una nueva restricción que no tiene que cumplir con r

Verifica?(in tags : conjRapidoString, in r : restricción) → res : bool
Pre ≡ {true}
Post ≡ {res =obs verifica?(tags, r)}
Complejidad: O(#r)
Descripcion: Evalua si la restricción r evalua a verdadero asumiendo que las
variables que figuran en tags son verdaderas.


Copiar(in r: restricción) → res : restricción
Pre ≡ {true}
Post ≡ {res =obs r}
Complejidad: O(R)
Descripcion: Devuelve una copia de la restricción.

*****************************************************************************
************                    Representación                       ************
*****************************************************************************

restricción se representa con estr
    donde estr es tupla(
                    tipo  : Enumerado(VAR, AND, OR, NOT),
                    op1   : puntero(estr),
                    op2   : puntero(estr),
                    valor : string)

Rep: ^(estr) → boolean
Rep(e) ≡ true ⟺
    No hay ciclos en el arbol ∧ι
    Ningún nodo de más abajo en el arbol tiene punteros a e ∧ι
    (e.tipo =obs VAR ⟺ (e.op1 =obs NULL ∧ e.op2 =obs NULL)) ∧
    (e.tipo =obs NOT ⟺ (¬(e.op1 =obs NULL) ∧ e.op2 =obs NULL)) ∧
    ((e.tipo =obs AND ∨ e.tipo =obs OR) ⟺ (¬(e.op1 =obs NULL) ∧ ¬(e.op2 =obs NULL))) ∧
    (¬(e.op1 =obs NULL) ⟹ι Rep(*e.op1)) ∧
    (¬(e.op2 =obs NULL) ⟹ι Rep(*e.op2)) ∧ (Long(e.valor) <= 64)
```

```
Abs: ^(estr) → Restricción                                                    {Rep(e)}
Abs(e) ≡
    if e.tipo =obs VAR then
        ⟨e.valor⟩
    else if e.tipo =obs AND then
        Abs(*e.op1) AND Abs(*e.op2)
    else if e.tipo =obs OR then
        Abs(*e.op1) OR Abs(*e.op2)
    else
        NOT Abs(*e.op1)
    fi fi fi

#•: ^(estr) → nat
#(e) ≡
    if e.tipo =obs VAR then
        1
    else if e.tipo =obs AND then
        1 + #(*e.op1) + #(*e.op2)
    else if e.tipo =obs OR then
        1 + #(*e.op1) + #(*e.op2)
    else
        1 + #(*e.op1)
    fi fi fi

*****************************************************************************
************                      Algoritmos                       ************
*****************************************************************************

iVar(in s : string) → res : estr
    res ← ⟨tipo: VAR, op1: NULL, op2: NULL, valor: s⟩
end function

iAnd(in r1 : estr, in r2 : estr) → res : estr
    res ← ⟨tipo: AND, op1: &r1, op2: &r2, valor: ""⟩
end function

iOr(in r1 : estr, in r2 : estr) → res : estr
    res ← ⟨tipo: OR, op1: &r1, op2: &r2, valor: ""⟩
end function

iNot(in r : estr) → res : estr
    res ← ⟨tipo: NOT, op1: &r, op2: NULL, valor: ""⟩
end function

iVerifica?(in tags : conj(string), in r : estr) → res : bool
    case
        [] tipo == AND
            res ← Verifica?(tags, *r.op1) ∧ Verifica?(tags, *r.op2)
        [] tipo == OR
            res ← Verifica?(tags, *r.op1) ∨ Verifica?(tags, *r.op2)
        [] tipo == NOT
            res ← ¬Verifica?(tags, *r.op1)
        [] tipo == VAR
            res ← Pertenece?(tags, r.valor)
    end case
end function

La complejidad de iVerifica? es O(#r), siendo #r la cantidad de restricciones
anidadas. Esto se debe a que para verificar una restriccion vale para un
conj(string), si esta no es un Var, debe verificar que valga para sus restricciones
anidadas en op1 y/o op2 (depende si la restriccion es de tipo NOT o no),
recursivamente, hasta llegar a un Var. Luego, cada restriccion devuelve el
resultado anidado, por lo que se pasaria por cada nodo 2 veces, ergo, O(2R), y
por algebra de ordenes, esto es equivalente a O(R)
```

```
iCopiar(in r: estr) → res: estr
    case
        [] tipo == AND
            res ← ⟨tipo: AND, op1: &Copiar(*r.op1), op2: &Copiar(*r.op2), valor: ""⟩
        [] tipo == OR
            res ← ⟨tipo: OR, op1: &Copiar(*r.op1), op2: &Copiar(*r.op2), valor: ""⟩
        [] tipo == NOT
            res ← ⟨tipo: Not, op1: &Copiar(*r.op1), op2: NULL, valor: ""⟩
        [] tipo == VAR
            res ← ⟨tipo: VAR, op1: NULL, op2: NULL, valor: Copiar(s)⟩
    end case
end function
```

La complejidad de este algoritmo es O(R) porque se tienen que copiar todos los nodos.
Como el valor esta acotado a 64 caracteres por enunciado, O(Copiar(s)) ≡ O(1)