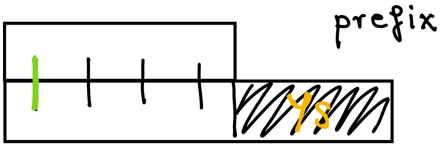
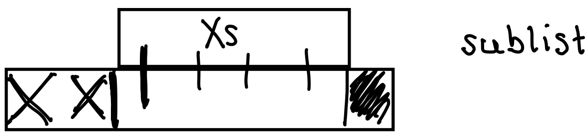
Aufgabe 1 (AGS 13.12)

(a) Programmieren Sie in Prolog⁻ eine binäre Relation `sublist`, die für jedes Paar (l_1, l_2) von Listen über natürlichen Zahlen wahr ist, wenn l_1 eine Teilliste von l_2 ist. Zum Beispiel gilt: `sublist([<2>, <3>],[<1>, <2>, <3>])` wobei `<1>`, `<2>` und `<3>` die übliche Darstellung der natürlichen Zahlen durch `s(0)`, `s(s(0))` und `s(s(s(0)))` abkürzt. *Hinweis:* Nutzen Sie die beiden folgenden Prädikate.

```
1  nat(0).
2  nat(s(X)) :- nat(X).
3
4  listnat([]).
5  listnat([X|XS]) :- nat(X), listnat(XS).
```

$l_2 = [1, 2, 3]$






```
6  prefix( [] , Ys ) :- listnat(Ys).
7  prefix( [X|Xs], [X|Ys] ) :- nat(X), prefix(Xs, Ys).
8
9  sublist( Xs , [Y|Ys] ) :- nat(Y), sublist(Xs, Ys).
10 sublist( Xs , Ys ) :- prefix(Xs, Ys).
```

(b) Bestimmen Sie durch SLD-Resultion für das Goal

```
?- sublist([<4>|XS], [<5>, <4>, <3>]).
```

zwei Belegungen der Variablen `XS`.

1

```
           ?-  sublist([<4>|XS], [<5>,<4>,<3>]).
           ?-  nat(<5>), sublist([<4>|XS], [<4>,<3>]).    %9
           ?-* nat(0), sublist([<4>|XS], [<4>,<3>]).      %2
           ?-  sublist([<4>|XS], [<4>,<3>]).              %1
           ?-  prefix([<4>|XS], [<4>,<3>]).               %10
           ?-  nat(<4>), prefix(XS, [<3>]).               %7
           ?-* nat(0), prefix(XS, [<3>]).                 %2
           ?-  prefix(XS, [<3>]).                         %1
{XS = []}  ?-  listnat([<3>]).                            %6
           ?-  nat(<3>), listnat([]).                     %5
           ?-* nat(0), listnat([]).                       %2
           ?-  listnat([]).                               %1
           ?-  .                                          %4
```

②

```
             ?-  sublist([<4>|XS], [<5>,<4>,<3>]).
             ?-  nat(<5>), sublist([<4>|XS], [<4>,<3>]).        %9
             ?-* nat(0), sublist([<4>|XS], [<4>,<3>]).          %2
             ?-  sublist([<4>|XS], [<4>,<3>]).                  %1
             ?-  prefix([<4>|XS], [<4>,<3>]).                   %10
             ?-  nat(<4>), prefix(XS, [<3>]).                   %7
             ?-* nat(0), prefix(XS, [<3>]).                     %2
             ?-  prefix(XS, [<3>]).                             %1
{XS=[<3>|XS1]} ?-  nat(<3>), prefix(XS1, []).                   %7
             ?-* nat(0), prefix(XS1, []).                       %2
             ?-  prefix(XS1, []).                               %1
{XS1=[]}     ?-  listnat([]).                                   %6
             ?-  .                                              %4
```

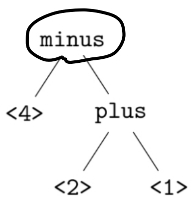
$\Rightarrow$ XS = [<3> | XS1] = [<3> | []] = [<3>]

Aufgabe 2 (AGS 13.13 ⋆)

(a) Ein *binärer Termbaum* ist ein Binärbaum über den zweistelligen Konstruktoren `plus` und `minus` sowie den natürlichen Zahlen (in der selben Form wie in Aufgabe 1) anstelle nullstelliger Konstruktoren. In der Abbildung unten rechts ist ein Beispiel für einen binären Termbaum schematisch dargestellt.

Programmieren Sie in Prolog⁻ eine binäre Relation `eval`, die genau die Paare (`T`, `X`) enthält, sodass `T` ein binärer Termbaum und `X` das natürlichzahlige Ergebnis der Auswertung dieses Terms ist. Die Subtraktion soll nur definiert sein, wenn der Minuend größer oder gleich dem Subtrahenden ist. Beispielweise soll der Term aus der rechts gezeigten Abbildung das Ergebnis `<1>` haben.

Hinweis: Nutzen Sie dafür die Prädikate `nat` aus Aufgabe 1 und `sum`.




```
sum(0, Y, Y) :- nat(Y).
sum(s(X), Y, s(S)) :- sum(X, Y, S).
```

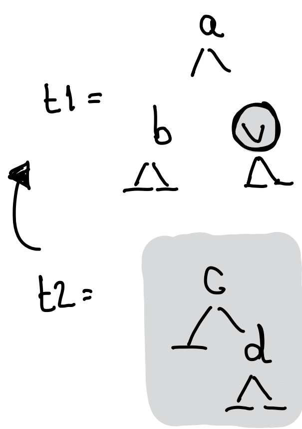
sum(x, y, z) ⇝ x + y = z   minus(4, plus(2,1))
eval(t, x) ⇝ evaluiere(t) = x   = 4 - (2+1) = 1

```
eval( X            , X ).
eval( plus(L,R),   X )  :- eval(L,LE), eval(R,RE), sum(LE,RE,X).
eval( minus(L,R),  X )  :- eval(L,LE), eval(R,RE), sum(RE,X,LE).
```

LE - RE = X
$\Leftrightarrow$ LE = RE + X

(b) Gegeben seien die zwei Terme

```
<t1> = tree(a, tree(b, nil, nil), tree(v, nil, nil))
und <t2> = tree(c, nil, tree(d, nil, nil))
```



und das folgende Prolog⁻-Programm:

```
1 istree(nil).
2 istree(tree(_, L, R)) :- istree(L), istree(R).
3
4 insert(nil, _, nil).
5 insert(tree(v, _, _), T,                T) :- istree(T).
6 insert(tree(X, L, R), T, tree(X, LT, RT)) :- insert(L, T, LT
     ), insert(R, T, RT).
```

Bestimmen Sie durch SLD-Refutation für das Goal `?- insert(<t1>, <t2>, X).` eine Belegung der Variablen `X`.

Hinweis: Sie dürfen die oben genannten Bäume weiterhin mit `<t1>` und `<t2>` abkürzen. Mehrere Resolutionsschritte unter Anwendung der selben Zeile können Sie mit `?-*` zusammenfassen.

?- insert(<t1>, <t2>, X).

{ X = tree(a, LT1, RT1) } ?- insert(tree(b, nil, nil), <t2>, LT1), insert(tree(v, nil, nil), <t2>, RT1). %6

{ LT1 = tree(b, LT2, RT2) } ?- insert(nil, <t2>, LT2), insert(nil, <t2>, RT2), insert(tree(v, nil, nil), <t2>, RT2). %6

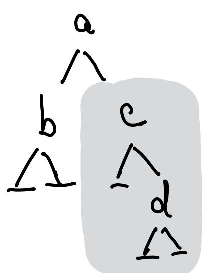
{ LT2 = nil, RT2 = nil } ?-* insert(tree(v, nil, nil), <t2>, RT2). %4

{ RT2 = <t2> } ?- istree(<t2>) %5

?-* istree(nil), istree(nil), istree(nil) %2

?-* . %1

X = tree(a, LT1, RT1)
= tree(a, tree(b, LT2, RT2), <t2>)
= tree(a, tree(b, nil, nil), <t2>)



<t2>