Programmierung

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).
```

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

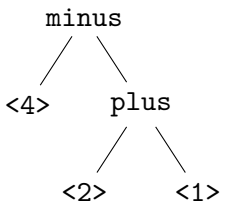
```
?- sublist([<4>|XS], [<5>, <4>, <3>]).
```

zwei Belegungen der Variablen `XS`.

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`.




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

(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.

Zusatzaufgabe 1 (AGs 13.17 $\star$)

(a) Programmieren Sie in Prolog$^-$ eine zweistellige Relation `last2`, die alle Paare `(Xs, Ys)` von Listen über natürlichen Zahlen enthält, sodass folgendes gilt: `Ys` ist eine zweielementige Liste, die aus den letzten beiden Elemente von `Xs` in der Reihenfolge ihres Vorkommens in `Xs` besteht. Beispielsweise soll `last2([7, 1, 3], [1, 3])` gelten.

Hinweis: Nutzen Sie die Relationen `nat` und `natlist` aus Aufgabe 1.

(b) Gegeben seien die zweistelligen Relationen `lt` und `leq`, die alle Paare `(X, Y)` von natürlichen Zahlen enthalten, sodass `X` $<$ `Y` bzw. `X` $\leq$ `Y` gilt:

```
lt(0, s(X)) :- nat(X).          leq(0, X) :- nat(X).
lt(s(X), s(Y)) :- lt(X, Y).     leq(s(X), s(Y)) :- leq(X, Y).
```

Programmieren Sie in Prolog$^-$ die dreistellige Relation `insert`, die alle Tupel `(Xs, Y, Zs)` enthält, wobei `Xs` eine Liste über natürlichen Zahlen, `Y` eine natürliche Zahl und `Zs` die Liste ist, die entsteht, wenn `Y` so weit rechts wie möglich in `Xs` eingefügt wird, dass alle seine Vorgänger in `Zs` kleiner als `Y` sind. Bspw. sollen `insert([1, 4, 7], 6, [1, 4, 6, 7])` und `insert([2, 1], 2, [2, 2, 1])` gelten.

(c) Gegeben seien (zusätzlich zu den Relationen aus Aufgabe 1) die Relationen `sum` und `fib`.

```
6  sum(0, X, X) :- nat(X).
7  sum(s(X), Y, s(Z)) :- sum(X, Y, Z).
8
9  fib(0, <1>).
10 fib(<1>, <1>).
11 fib(s(s(N)), Z) :- fib(N, X), fib(s(N), Y), sum(X, Y, Z).
```

Bestimmen Sie für das Goal `?- fib(N, <2>).` eine Belegung der Prolog-Variablen `N` mittels SLD-Refutation. Notieren Sie in jedem Schritt die verwendete Programmzeile. Mehrere Resolutionsschritte unter Anwendung derselben Programmzeile können Sie mit `?-*` zusammenfassen.

Hinweis: Nutzen Sie die Prolog-Evaluationsstrategie, d.h. wenden Sie in jedem Schritt die oberste Programmzeile an, die mit dem am weitesten links stehenden Literal des Goals unifizierbar ist.

Zusatzaufgabe 2 (AGS 13.15 ⋆)

Ein *Binärbaum* in Prolog⁻ ist entweder von der Form `nil` oder von der Form `tree(X, L, R)`, wobei `X` eine natürliche Zahl, `L` ein Binärbaum und `R` ein Binärbaum sind. Die *Menge der Schlüsselwerte* eines Binärbaums `T` ist leer, falls `T` von der Form `nil` ist, und, falls `T` von der Form `tree(X, L, R)` ist, dann umfasst sie genau `X` sowie alle Schlüsselwerte von `L` und `R`.
Natürliche Zahlen stellen wir in Prolog⁻ als Terme über dem einstelligen Funktionssymbol `s` und dem nullstelligen Funktionssymbol `0` dar:

```
1  nat(0).
2  nat(s(X)) :- nat(X).
```

Dabei kürzen wir wie in der Vorlesung den Term für die natürliche Zahl `n` mit `<n>` ab, z.B. `s(s(s(0)))` = `<3>`.

(a) Ein Binärbaum `T` hat die *Suchbaumeigenschaft*, wenn `T` von der Form `nil` ist oder wenn `T` von der Form `tree(X, L, R)` ist und `X` größer ist als jeder Schlüsselwert von `L`, `X` kleiner ist als jeder Schlüsselwert von `R` und sowohl `L` als auch `R` die Suchbaumeigenschaft haben.

Programmieren Sie in Prolog⁻ eine einstellige Relation `search_tree`, die genau die Binärbäume enthält, für die die Suchbaumeigenschaft gilt.

Hinweise:

- Verwenden Sie die zweistellige Relation `lt` (*echt kleiner als*):

```
3  lt(0, s(X)) :- nat(X).
4  lt(s(X), s(Y)) :- lt(X, Y).
```

- Programmieren Sie die zweistelligen Hilfsrelationen `all_less` und `all_greater`, die alle Paare `(T, N)` von Binärbäumen `T` und natürlichen Zahlen `N` enthalten, sodass alle Schlüsselwerte in `T` echt kleiner (bzw. echt größer) als `N` sind.

(b) Zur Bestimmung der Höhe eines Binärbaums sei folgendes Prolog⁻-Programm gegeben:

```
5  max(X, X, X) :- nat(X).
6  max(X, Y, X) :- lt(Y, X).
7  max(X, Y, Y) :- lt(X, Y).
8  height(nil, 0).
9  height(tree(T, L, R), s(H)) :- height(L, HL), height(R, HR),
       max(HL, HR, H).
```

Bestimmen Sie für das Goal `?- height(tree(<7>, nil, nil), X).` eine Belegung der Prolog-Variablen `X` mittels SLD-Refutation. Notieren Sie in jedem Schritt die verwendete Programmzeile. Mehrere Resolutionsschritte unter Anwendung derselben Programmzeile können Sie mit `?-*` zusammenfassen.

Geben Sie abschließend die auf diese Art ermittelte Belegung der Variablen `X` an.