Lösningsförslag för Tenta i Programmeringsparadigm 2016-03-15 08.00-11.00

Funktionell Programmering

1 (a) Värdet 2 knyts till `x`. En anonym funktion med ett argument (motsvarar `y`) returneras, funktionskroppen motsvaras av `sub y 2`.

(b) `generell_annat funk x y = funk y x`

(c) `bytArgumentOrdning`

(d) `annat1 :: (t1 -> t2 -> t) -> t2 -> t1 -> t`

2 (a)
```
distance (Manhattan (Vector2D x1 y1) (Vector2D x2 y2)) =
   abs(x1 - x2 ) + abs (y1 - y2)
```

(b) `distance (Manhattan (Vector2D 1 2) (Vector2D 3 4))`

3 (a) adderaPositivaHeltal 3 4
adderaPositivaHeltal 2 5
adderaPositivaHeltal 1 6
adderaPositivaHeltal 0 7
7

(b) Funktionen är svansrekursiv eftersom inga beräkningar läggs på stacken (argumenten `tal1` och `tal2` uppdateras istället).

(c) För att göra upprepningar, i det imperativa kan man använda loopar t.ex. for- och while-loopar.

(d)
```
adderaPositivaHeltalLst 0 tal2 lst = lst ++ [(0, tal2)]
adderaPositivaHeltalLst tal1 tal2 lst =
    adderaPositivaHeltalLst (tal1 - 1) (tal2 + 1) (lst ++ [(tal1, tal2)])
```

Logikprogrammering

4 1. Mål: `member(X, [1, 2]).`
   2. Skapar instans av första regeln: `member(X1, [X1|A1]).`
      Lyckas med unifieringen: `X=1, X1=1, A1=[2].`
      Går vidare till nästa mål.
   3. Mål: `\+ member(1, Y).`
      3.1. Mål: `member(1, Y).`
      3.2. Skapar instans av första regeln: `member(X2, [X2|A2]).`
      Lyckas med unifieringen: `X2=1, Y=[1|A2].`

      Negationen misslyckas, backtrackar till 2.
   2. Skapar instans av andra regeln: `member(X3, [A3|T1]) :- member(X3, T1).`
      Unifierar: `X=X3, A3=1, T1=[2].`
      2.1. Mål: `member(X3, [2]).`
      2.2. Skapar instans av första regeln: `member(X4, [X4|A3]).`
      Lyckas med unifieringen: `X3=2, X4=2, A3=[].`

      Går vidare till nästa mål.
   3. Mål: `\+ member(2, Y).`
      3.1. Mål: `member(2, Y).`
      3.2. Skapar instans av första regeln: `member(X5, [X5|A4]).`
      Lyckas med unifieringen: `X5=2, Y=[2|A4].`

      Negationen misslyckas, backtrackar igen till 2.
   2. Inga fler regler att använda, målet misslyckas.

   Svar: frågan misslyckas.

5 (a) Under logisk läsning av ett Prolog-program menas läsningen av reglerna som en logisk formel. Ordningen mellan klausuler inom en regel och ordningen mellan olika regler är alltså oväsentlig för logiska läsningen.

   Under procedurell läsning menas däremot en läsning som tar hänsyn till kontrollflödet, och därmed spelar ordningen mellan klausuler inom en regel och ordningen mellan olika regler en stor roll.

(b) En förfader till X är antingen en förälder till X eller en förfader till en förälder till X. Utifrån detta kan vi bygga två program med samma logiska läsning:

```
forfader(X, Y) :- foralder(X, Y).
forfader(X, Y) :- foralder(X, Z), forfader(Z, Y).
```

och

```
forfader(X, Y) :- forfader(Z, Y), foralder(X, Z).
forfader(X, Y) :- foralder(X, Y).
```

Fast första programmet fungerar korrekt vid frågor som

```
?- forfader(kia, X).
```

medan det andra programmet aldrig terminerar.

6 (a) Här är två lösningsförslag. (Observera att motivering hur lösningen fungerar inte inkluderas här, trots att det ingick i uppgiften.)

Möjlig lösning 1:

```
delEl(X, empty, empty).

delEl(X, leaf(X), empty).
delEl(X, leaf(D), leaf(D)).

delEl(X, branch(TL, TR), empty) :- delEl(X, TL, empty), delEl(X, TR, empty).
delEl(X, branch(TL, TR), TRX) :- delEl(X, TL, empty), delEl(X, TR, TRX).
delEl(X, branch(TL, TR), TLX) :- delEl(X, TR, empty), delEl(X, TL, TLX).
delEl(X, branch(TL, TR), branch(TLX, TRX)) :- delEl(X, TL, TLX), delEl(X, TR, TRX).
```

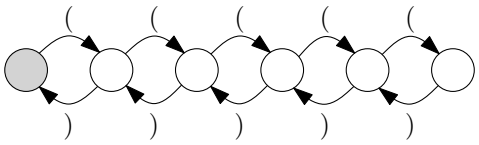
Möjlig lösning 2:

```
elems(empty, [], X).
elems(leaf(X), [], X) :- !.
elems(leaf(D), [D], X).
elems(branch(N1, N2), L, X) :-
  elems(N1, L1, X),
  elems(N2, L2, X),
  append(L1, L2, L3).

maketree([], empty).
maketree([H], leaf(H)).
maketree([H1|[H2|T]], branch(leaf(H1), R)) :- maketree([H2|T], R).

delEl(X, T, TX) :- elems(T, L, X), maketree(L, TX).
```

(b)
```
T = lbranch(lbranch(leaf(1))).
```

Formella språk och syntaxanalys

7 En möjlig lösning (tillståndet markerat i grått är starttillstånd, och även det enda accepterande tillståndet).



8 (a) Möjlig lösning:

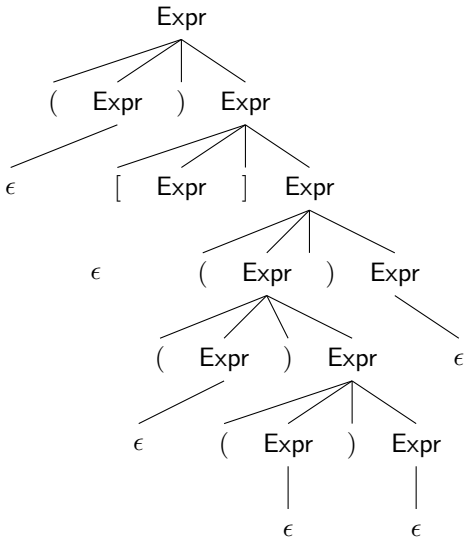
$$\mathsf{Expr} \rightarrow \mathsf{(}\ \mathsf{Expr}\ \mathsf{)}\ \mathsf{Expr} \mid \mathsf{[}\ \mathsf{Expr}\ \mathsf{]}\ \mathsf{Expr} \mid \mathsf{\{}\ \mathsf{Expr}\ \mathsf{\}}\ \mathsf{Expr} \mid \epsilon$$

Startsymbol (och enda icke-slutsymbol) är Expr.

(b) För lösningen ovan: ja, grammatiken är LL(1). För att parsa ett Expr behöver vi kunna avgöra vilken av de fyra produktionsreglerna som ska appliceras genom att bara titta på nästa tecken, vilket enkelt kan göras (om nästa tecken är '(', '[', eller '{' så är det motsvarande av de tre första reglerna som ska användas, annars den sista regeln).

(Notera att detta svar beror på lösningen ovan, andra lösningar kanske inte ger en LL(1)-grammatik.)

(c) Sträng: `()[](()())`
Syntaxträd i grammatiken ovan:



(d) Låt L_1 vara språket vi är intresserade av i uppgiften, balanserade parentesuttryck över de tre typerna av paranteser.
Låt L_2 vara språket med alla strängar över tecknen '(' och ')', dvs alla strängar bestående av parenteser, oavsett om de är balanserade eller inte.
Vi har då att språket $L_1 \cap L_2$ består av alla balanserade parentesuttryck över vanliga parenteser, '(' och ')'. Detta är inte reguljärt. Eftersom snittet av två reguljära språk är reguljärt betyder detta att antingen L_1 eller L_2 inte är reguljärt. Men det är enkelt att se att L_2 är reguljärt. Alltså är inte L_1 reguljärt.

9 (a) **Falskt**.
Ett konkret motexempel: låt L vara språket som består av alla balanserade parentesuttryck, vilket inte är reguljärt. Alla strängar i L har jämn längd. Med andra ord är språket "alla strängar i L av udda längd" tomt. Det tomma språket är reguljärt.

(b) **Sant**.
Antag att vi har en grammatik för L med startsymbol Start. Vi kan skriva om denna till en grammatik för L^* genom att införa en ny icke-slutsymbol NewStart med följande produktionsregler, och låta NewStart bli den nya startsymbolen:

$$\mathsf{NewStart} \rightarrow \epsilon \mid \mathsf{Start}\ \mathsf{NewStart}$$

(c) **Sant**.
De kontextfria språken är en större språkklass än de reguljära.