Disclaimer

Aufgaben aus dieser Vorlage stammen aus der Vorlesung *Logik und Logikprogrammierung* und wurden zu Übungszwecken verändert oder anders formuliert! Für die Korrektheit der Lösungen wird keine Gewähr gegeben.

1. Definitionen und Sätze
   (a) Der Korrektheitssatz der Aussagenlogik für den Wahrheitswertebereich B lautet...

   Antwort: Für jede Menge von Formeln Γ und jede Formel φ gilt $\Gamma \vdash \varphi \Rightarrow \Gamma \vdash_B \varphi$.

   (b) Eine Menge von Formeln Γ heißt erfüllbar, wenn...

   Antwort: Sei Γ eine Menge von Formeln. Γ heißt erfüllbar, wenn es eine passende B-Belegung B gibt mit $B(\gamma) = 1_B$ für alle $\gamma \in \Gamma$.

   (c) Der Satz von Cook lautet...

   Antwort: Die Erfüllbarkeit einer endlichen Menge Γ ist NP-vollständig.

   (d) Zwei Formeln α und β heißen äquivalent, wenn...

   Antwort: Zwei Formeln α und β heißen äquivalent ($\alpha \equiv \beta$), wenn für alle passenden B-Belegungen B gilt: $B(\alpha) = B(\beta)$.

   (e) Der Kompaktheitssatz der Aussagenlogik lautet...

   Antwort: Sei Γ eine u.U. unendliche Menge von Formeln. Dann gilt Γ unerfüllbar $\Longleftrightarrow \exists \Gamma' \subseteq \Gamma$ endlich: Γ' unerfüllbar

   (f) Eine Horn Klausel ist eine Formel der Form

   Antwort: Eine Hornklausel hat die Form $(\neg\bot \wedge p_1 \wedge p_2 \wedge ... \wedge p_n) \rightarrow q$ für $n \geq 0$, atomare Formeln $p_1, p_2, ..., p_n$ und q atomare Formel oder $q = \bot$. Eine Hornformel ist eine Konjunktion von Hornklauseln.

2. Wahrheitswertebereiche
   (a) Werte die Formel $\varpi_a = \neg p \wedge \neg\neg p$ im Heytingschen Wahrheitswertebereich $H_{\mathbb{R}}$ aus für die $H_{\mathbb{R}}$-Belegung B mit $B(p) = \mathbb{R}\backslash\{0\}$

   Antwort: $B_{H_{mathbbR}}(\neg p \wedge \neg\neg p) = Inneres(\mathbb{R}/p) \cap p = Inneres(\mathbb{R}\backslash\{\mathbb{R}\backslash\{0\}\}) \cap \mathbb{R}\backslash\{0\} = \{0\} \cap \mathbb{R}\backslash\{0\} = \varnothing$

   (b) Überprüfe ob die Formel $\varphi_B = (\neg p \rightarrow \neg p) \rightarrow p$ eine K_3-Tautologie ist. Ist φ_b eine $B_{\mathbb{R}}$ Tautologie?

   Antwort:

p	$\neg p$	$\phi = (\neg p \rightarrow \neg p)$	$\phi \rightarrow p$
0	1	1	0
$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$
1	0	1	1

   Keine K_3 Tautologie.

   Da keine B Tautologie $\rightarrow$ keine B_R Tautologie

   (c) Überprüfe ob die semantische Folgeung $\{p \rightarrow q, q \rightarrow r\} \Vdash_B r \rightarrow \neg p$ gilt.

   Antwort:

p	q	r	$\neg p$	$\Gamma_1 = p \rightarrow q$	$\Gamma_2 = q \rightarrow r$	$\Phi = r \rightarrow \neg p$	$\Gamma \Vdash \Phi$
0	0	0	1	1	1	1	✓
0	0	1	1	1	1	1	✓
0	1	0	1	1	0	1	
0	1	1	1	1	1	1	✓
1	0	0	0	0	1	1	
1	0	1	0	0	1	0	
1	1	0	0	1	0	1	
1	1	1	0	1	1	0	✗

   Folgerung gilt nicht

3. Erfüllbarkeit

(a) Überprüfe mittels Markierungsalgorithmus, ob die Formel $\varphi_a = (\neg p \vee q) \wedge (t \vee \neg s) \wedge (\neg r \vee s \vee \neg q) \wedge r \wedge (\neg p \vee t) \wedge \neg s \wedge (\neg r \vee p)$ erfüllbar ist.

Antwort:

- $\neg\varphi_a = (p \wedge \neg q) \vee (\neg t \wedge s) \vee (r \wedge \neg s \wedge q) \vee \neg r \vee (p \wedge \neg t) \vee s \vee (r \wedge \neg p)$
- Horn Klauseln
  1. $q \to p$
  2. $t \to s$
  3. $s \to r \wedge q$
  4. $r \to \bot$
  5. $t \to p$
  6. $\neg\bot \to s$
  7. $p \to r$
- Markieren
  1. für 6.: s
  2. für 3.: r, q
  3. für 4.+1.: $\bot, p$
  4. für 7.: r
  5. Terme 2 und 5 bleiben übrig → terminiert mit „unerfüllbar"
- $\neg\varphi_a$ unerfüllbar $\Rightarrow$ φ_a erfüllbar

(b) Überprüfe mittels SLD Resolution, ob die Formel $\varphi_b = (r \wedge p) \vee \neg t \vee (p \wedge \neg q) \vee \neg p \vee (\neg r \wedge q \wedge t)$ eine Tautologie ist

Antwort:

- Horn Klauseln
  1. $\neg\bot \to r \wedge p$
  2. $t \to \bot$
  3. $q \to p$
  4. $p \to \bot$
  5. $r \to q \wedge t$
- Markieren
  1. für 2.+3.: $M_0 = \{p, t\}$
  2. für 3.: $M_1 = \{q, t\}$
  3. für 5.: $M_2 = \{r\}$
  4. für 4.: $M_3 = \{r, p\}$
  5. für 1.: $M_4 = \varnothing$
- $M_4 = \varnothing \Rightarrow \{\neg\varphi_b\}$ unerfüllbar → φ Tautologie

4. Monotone Formeln: Eine aussagenlogische Formel φ heißt monoton, falls für alle zu φ passenden B-Belegungen B_1, B_2 mit $B_1(p_i) \leq B_2(p_i)$ für alle $i \in \mathbb{N}$ gilt $B_1(\varphi) \leq B_2(\varphi)$. Beispielsweise sind $p_1 \wedge p_2$ und $\neg\neg p_1$ monoton.

(a) Entscheide, welche der Formeln $\varphi = p_1 \wedge (p_2 \to p_3)$, $\psi = \neg p_1 \to p_2$ monoton sind.

Antwort: Teste für B-Belegung mit Boolschem Wahrheitswertebereich

p_1	p_2	p_3	$p_2 \to p_3$	$\varphi = p_1 \wedge (p_2 \to p_3)$	
0	0	0	1	0	
0	0	1	1	0	
0	1	0	0	1	
0	1	1	1	0	$\Rightarrow$ nicht monoton
1	0	0	1	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	1	1	1	

p_1	p_2	$\neg p_1$	$\psi = \neg p_1 \to p_2$	
0	0	1	0	
0	1	1	1	$\Rightarrow$ monoton
1	0	0	1	
1	1	0	1	

(b) Zeige per vollständiger Induktion über den Formelaufbau, dass aussagenlogische Formeln in denen weder $\neg$ noch $\to$ vorkommen, monoton sind.

Antwort:

5. Definitionen und Sätze: Sei $\sum$ eine Signatur. Verfollständige die folgenden Definitionen und Sätze.

(a) Es gilt $\Delta \vdash \varphi$ für eine $\sum$-Formel φ und eine Menge Δ von $\sum$-Formeln, falls

Antwort: Seien Δ eine Menge von Formeln und φ eine Formel. Dann gilt $\Delta \vdash \varphi \Leftrightarrow \Delta \Vdash_B \varphi$ Insbesondere ist eine Formel genau dann eine B-Tautologie, wenn sie ein Theorem ist.

(b) Der Vollständigkeitssatz der Prädikatenlogik lautet...

Antwort: Sei Γ eine Menge von $\sum$-Formeln und φ eine $\sum$-Formel. Dann gilt $\Gamma \Vdash \varphi \Rightarrow \Gamma \vdash \varphi$. Insbesondere ist jede allgemeingültige Formel ein Theorem.

(c) Der Satz von Löwenheim-Skolem lautet...

Antwort: Sei Γ erfüllbare und höchstens abzählbar unendliche Menge von $\sum$-Formeln. Dann existiert ein höchstens abzählbar unendliches Modell von Γ.

(d) Die (elementare) Theorie einer $\sum$-Struktur A ist

Antwort: Eine $\sum$-Struktur ist ein Tupel $A = (U_A, (f^A)_{f\in\Omega}, (R^A)_{R\in Rel})$, wobei

- U_A eine nichtleere Menge, das Universum,
- $R^A \supseteq U_A^{ar(R)}$ eine Relation der Stelligkeit $ar(R)$ für $R \in Rel$ und
- $f^A : U_A^{ar(f)} \to U_A$ eine Funktion der Stelligkeit $ar(f)$ für $f \in \Omega$ ist.

6. Natürliches Schließen

(a) Gebe die Regeln $(\forall - I)$, $(\exists - E)$ und (GfG) inklusive Bedingung an

Antwort:
$\forall - I : \frac{\varphi}{\forall x \varphi}$ Bedingung: x nicht frei in Hypothesen
$\forall - E : \frac{\forall x \varphi}{\varphi[x:=t]}$ Bedingung: über keine Variable aus t wird in φ quantifiziert
$\exists - I : \frac{\varphi[x:=t]}{\exists x \varphi}$ Bedingung: über keine Variable in t wird in φ quantifiziert
$\exists - E : \frac{\exists x \varphi \quad \sigma}{\sigma}$ Bedingung: x weder frei in Hypothesen noch in σ
$(GfG) : \frac{\varphi[x:=s] \quad s=t}{\varphi[x:=t]}$ Bedingung: über keine Variable aus s oder t wird in φ quantifiziert

(b) Zeige, dass $\forall x \exists y (f(x) = y)$ ein Theorem ist, indem du eine entsprechende Deduktion angibst

Antwort:

(c) Zeige, dass $\exists x \forall y (f(x) = y)$ nicht allgemeingültig ist

Antwort:

(d) Zeige, dass die Formel aus c) erfüllbar ist

Antwort:

7. Prädikatenlogische Definierbarkeit: Betrachte im folgenden Graphen als $\sum$-Struktur, wobei $\sum$ eine Signatur mit einem zweistelligen Relationssymbol E ist.

(a) Betrachte den (kommt noch) Graphen und die $\sum$-Formel $\varphi_a = \forall x \exists y \exists z (((E(x,y) \wedge E(y,z)) \vee (E(y,x) \wedge E(z,x))) \wedge y \neq z)$. Gebe eine Kante an, sodass G mit dieser zusätzlichen Kante als $\sum_a$-Struktur ein Modell der Formel φ_a ist. Begründe deine Antwort.

Antwort:

(b) Betrachte die folgenden (kommen noch) Graphen G_1 und G_2. Gebe einen $\sum$-Satz φ_b an, so dass $G_1 \Vdash \varphi_b$ und $G_2 \nVdash \varphi_b$ gilt.

Antwort:

(c) Gebe einen $\sum$-Satz φ_c an, so dass für alle $\sum$-Strukturen A genau dann $A \Vdash \varphi_c$ gilt, wenn E^A eine Äquivalenzrelation ist (d.h. reflexiv, symmetrisch und transitiv).

Antwort:

8. Normalformeln und Unifikatoren

(a) Betrachte die Formel $\varphi = \forall x(\exists y(R(x,y) \wedge \neg\exists x(R(y,x))))$. Gebe eine Formel ψ_1 in Pränexform an, die äquivalent zu φ ist und eine Formel ψ_2 in Skolemform, die erfüllbarkeitsäquivalent zu φ ist.

Antwort:

- $\forall x(\exists y(R(x,y) \wedge \neg\exists x(R(y,x))))$
- $\forall x(\exists x_2 \exists y(R(x,y) \wedge \neg R(y,x_2)))$
- $\forall x(\exists x_2 \exists y(R(x,y) \wedge \neg R(y,x_2)))$
- $\forall x \exists x_2 \exists y(R(x,y \wedge \neg R(y,x_2)))$ (Pränexform)
- $\forall x(R(x,y) \wedge \neg R(g(x),h(x)))[x_2 := h(x)][y := g(x)]$
- $\forall x(R(x,y) \wedge \neg R(g(x),h(x)))$ (Skolemform)

(b) Sei $\sum$ eine Signatur mit zweistelligem Relationssymbol R, zweistelligem Funktionssymbol f, einstelligem Funktionssymbol g und Konstanten a, b. Ermittle mit dem Unifikationsalgorithmus, ob die atomare Formel unifizierbar ist und gebe einen allgemeinsten Unifikator an, falls dieser existiert.

$$(R(x, f(y, g(a))), R(a, f(g(x), y)))$$

Antwort:

$\varphi_1\sigma$	$\varphi_2\sigma$	σ
$R(x, f(y, g(a)))$	$R(a, f(g(x), y))$	id
$R(a, f(y, g(a)))$	$R(a, f(g(x), y))$	$id[x := a]$
$R(a, f(g(x), g(a)))$	$R(a, f(g(x), g(x)))$	$id[x := a][y := g(x)]$

Terminiert nicht unifizierbar

(c) Sei $\sum$ eine Signatur mit zweistelligem Relationssymbol R, zweistelligem Funktionssymbol f, einstelligem Funktionssymbol g und Konstanten a, b. Ermittle mit dem Unifikationsalgorithmus, ob die atomare Formel unifizierbar ist und gebe einen allgemeinsten Unifikator an, falls dieser existiert.

$$(R(f(g, x), y), R(f(y, z), z))$$

Antwort:

$\varphi_1\sigma$	$\varphi_2\sigma$	σ
$R(f(g, x), y)$	$R(f(y, z), z))$	id
$R(f(y, x), y)$	$R(f(y, z), z))$	$id[y := y]$

Terminiert nicht unifizierbar

9. Gegeben sei folgende Wissensbasis:

- über(rot, orange).
- über(orange, gelb).
- über(gelb, grün).
- über(grün, blau).
- über(blau, violett).
- top(X): über(_, X), !, fail.
- top(_).

- oben(X):-über(X,_),top(X).

Wie antwortet ein Prolog System mit dieser Wissensbasis auf die folgenden Fragen:

(a) ?-top(grün).

Antwort: {gelb}, true

(b) ?-top(X).

Antwort: {rot, orange, gelb, grün, blau }, false

(c) ?-top(rot).

Antwort: {}, false

(d) ?-oben(grün).

Antwort: false

(e) ?-oben(X).

Antwort: {rot}, true

(f) ?-oben(rot).

Antwort: true

10. Man implementiere folgende Prädikate in Form von Prolog Klauseln

(a) Das Prädikat $parition(L, E, Kl, Gr)$ soll eine gegebene Liste ganzer Zahlen L in zwei Teillisten partitionieren.

- die Liste Kl aller Elemente aus L, welche kleiner oder gleich E sind und
- die Liste Gr aller Elemente aus L, welche größer als E sind

Beispiel: ?-parition([1,2,3,4,5,6], 3, Kl, Gr).

- $Kl = [1, 2, 3]$
- $Gr = [4, 5, 6]$

Antwort:

```
% delete Funktion
delete(_, [], []).
delete(X, [X|Xs], Xs).
delete(X, [Y|Ys], [Y|Zs]) :-
  delete(X, Ys, Zs).

% append Funktion
append([], Xs, Xs).
append([X|Xs], Ys, [X|Zs]) :-
  append(Xs, Ys, Zs).

% partition Funktion
partition([], E, Kl, Gr).
partition([K|R], E, Kl, Gr):-
  K<E,
  append(K, Kl, Kl),
  partition(R, E, Kl, Gr).
partition([K|R], E, Kl, Gr):-
  K>E,
  append(K, Gr, Gr),
  partition(R, E, Kl, Gr).
```

(b) Das Prädikat $merge(L1, L2, L)$ soll zwei sortierte Listen mit ganzen Zahlen $L1$ und $L2$ zu einer sortierten Liste L verschmelzen.

Antwort:

```
merge([], L2, L2).
merge(L1, [], L1).
merge([K1|R2], [K2|R2], L):-
  K1>K2,
  append(K2, L, L),
  merge([K1|R1], R2, L).
merge([K1|R2], [K2|R2], L):-
  K2=<K1,
  append(K1, L, L),
  merge(R1, [K2|R2], L).
```

(c) Das Prädikat *listmerge*(*ListenListe*, *L*) bekommt eine Liste sortierter Listen *ListenListe* und soll sie zu einer sortierten Liste L verschmelzen. Das in Aufgabe b) definierte Prädikat *merge* kann dabei verwendet werden.

Antwort:

```
listmerge([], L).
listmerge([K|R], L):-
  merge(K, L, L2),
  listmerge(R, L2).
```

(d) Das Prädikat *am_groesten*(L, Max) soll das größte Element Max einer Zahlenliste L ermitteln. Falls L leer ist, soll „nein" geantwortet werden.

Antwort:

```
am_groesten([], Max):-
  fail.
am_groesten([K|R], Max):-
  K>Max,
  am_groesten(R, K).
am_groesten([K|R], Max):-
  K=<Max,
  am_groesten(R, Max).
```

(e) Das Prädikat *am_kuerzesten*(*ListenListe*, *L*) soll aus einer Liste von *ListenListe* die kürzeste Liste L ermitteln. Dies soll möglichst effizient geschehen:

- Gestalte die Prozedur rechtsrekursiv
- Sehe davon ab, Listenlängen explizit zu ermitteln. Ermittle diese mit einem Hilfsprädikat *kuerzer_als*($L1, L2$)
- Höre mit der Suche auf, sobald eine leere Liste gefunden wurde. Kürzer geht nicht

Falls *ListenListe* leer ist, soll „nein" geantwortet werden.

Antwort:

```
am_kuerzesten([], L).
am_kuerzesten([K,R], L):-
  kuerzer_als(K,L),
  am_kuerzesten(R, K).
am_kuerzesten([K,R], L):-
  !kuerzer_als(K,L),
  am_kuerzesten(R, L).
```

11. Ein binärer Suchbaum mit natürlichen Zahlen in den Knoten sei in Prolog wie folgt als strukturierter Term repräsentiert:

- leerer Baum: nil
- nichtlerer Baum: baum(Wurzel, LinkerUnterbaum, RechterUnterbaum)

Beispiel Baum mit Wurzel 6, Wurzel 4 im linken Unterbaum, Wurzel 7 im rechten Unterbaum und 2,4 und 9 als Blätter: $baum(6, baum(4, baum(2, nil, nil), baum(5, nil, nil)), baum(7, nil, baum(0, nil, nil)))$.
Man implementiere folgende Prädikate in Prolog

(a) Das Prädikat *enthalten*(*Baum*, *Zahl*) bekommt einen binären Suchbaum *Baum* sowie eine Zahl *Zahl* und soll entscheiden, ob diese Zahl in Baum enthalten ist und die Antwort „ja" oder „nein" liefern.

Antwort:

```
baum(nil).
baum(Wurzel, Links, Rechts):-
  baum(Links),
  baum(Rechts).

ist_knoten(X, baum(X, Links, Rechts)).
ist_knoten(X, baum(Y, Links, Rechts)) :-
  ist_knoten(X, Links).
ist_knoten(X, baum(Y, Links, Rechts)) :-
  ist_knoten(X, Rechts).

enhalten(baum(X, Links, Rechts), X).
enhalten(baum(Y, Links, Rechts), X):-
  enthalten(Links, X).
enthalten(baum(Y, Links, Rechts), X):-
  enthalten(Rechts, X).
```

(b) Das Prädikat $flatten(Baum, Liste)$ soll aus einem gegebenen Suchbaum *Baum* die Liste *Liste* aller der im Baum enthaltenen Zahlen in aufsteigender sortierter Reihenfolge liefern.

Antwort:

```
flatten([], []).
flatten(baum(X, Links, Rechts), L):-
  insert(X, L, L2),
  flatten(Links, L2),
  flatten(Rechts, L2).
flatten([], L):-
  insertionSort(L, []).

insert(X, [], [X]).
insert(X, [X1|L1], [X, X1|L1]):-
  X=<X1,
  !.
insert(X, [X1|L1], [X1|L]):-
  insert(X, L1, L).

insertionSort([], []):-
  !.
insertionSort([X|L], S):-
  insertionSort(L, S1),
  insert(X, S1, S).
```