BLATT 4

DANIEL SCHMIDT & PAMELA FLEISCHMANN

Aufgabe 1. Sei $V = (\mathcal{P}(\mathbb{N}), \subseteq)$ und $R \subseteq \mathbb{N}$. Dann gilt
a. $\tau(M) = \{x - 1 | \, x \in M \backslash \{1\}\}$ ist monoton und die leere Menge ist der einzige Fixpunkt.
b. $\tau(M) = M \backslash R$ ist monoton und genau die $F \in \mathcal{P}(\mathbb{N})$ mit $F \cap R = \emptyset$ sind die Fixpunkte.
c. $\tau(M) = R \backslash M$ ist nicht monoton und hat keine Fixpunkte.
d. $\tau(M) = \{1\} \cup \{2x | \, x \in M\}$ ist monoton und $F = \{1\} \cup \{n \in \mathbb{N} | \, n \equiv_2 0\}$ ist der einzige Fixpunkt.

Beweis. ad a. Seien $M_1, M_2 \in \mathcal{P}(\mathbb{N})$ mit $M_1 \subseteq M_2$. Seit weiter $m \in \tau(M_1)$. Dann existiert ein $n \in M_1 \backslash \{1\}$ mit $m = n - 1$. Wegen $M_1 \subseteq M_2$ gilt $n \in M_2 \backslash \{1\}$. Damit gilt $m \in \tau(M_2)$ und τ ist monoton.
Für alle $x \in \emptyset$ gilt offensichtlich $x - 1 \in \emptyset$ und somit ist die leere Menge ein Fixpunkt von τ.
Annahme: $\exists M \in \mathcal{P} \backslash \{\emptyset\} : \tau(M) = M$
Wegen $M \neq \emptyset$ gilt $|M| \geq 1$. Ist $M = \{1\}$, so ist $\tau(M) = \emptyset \neq M$. ↯
Somit existiert in M mindestens ein $m \neq 1$. Wähle dieses maximal. Wegen $M = \tau(M)$ gilt $m \in \tau(M)$ und somit existiert ein $n \in M \backslash \{1\}$ mit $m = n - 1$. Damit gilt $m + 1 = n \in M$, was ein Widerspruch zur Maximalität von m ist. Damit ist $\emptyset$ der einzige Fixpunkt von τ.

ad b. Seien $M_1, M_2 \in \mathcal{P}(\mathbb{N})$ mit $M_1 \subseteq M_2$. Seit weiter $m \in \tau(M_1)$. Dann gilt $x \in M_1 \backslash R$, also $x \in M_1$ und $x \notin R$. Wegen $M_1 \subseteq M_2$ gilt $x \in M_2$ und es folgt $x \in M_2 \backslash R = \tau(M_2)$. Damit ist τ monoton.
Für $F \in \mathcal{P}(\mathbb{N})$ mit $F \cap R = \emptyset$ gilt offensichtlich $\tau(M) = F \backslash R = F$.
Annahme: $\exists M \in \mathcal{P}(\mathbb{N}) : \, M \cap R \neq \emptyset \wedge \tau_(M) = M$
Dann gilt $\tau(M) = M \backslash R \subset M$. ↯ Damit sind die zu R disjunkten Teilmengen die einzigen Fixpunkte von τ.

ad c. Annahme: τ ist monoton.
Seien $M_1, M_2 \in \mathcal{P}(\mathbb{N})$ mit $M_1 \subset M_2$ und $\tau(M_1) \subseteq \tau(M_2)$. Sei $x \in R \cap M_2 \backslash M_1$. Dann gilt $x \notin M_1$ und $x \in R$ und somit $x \in R \backslash M_1 = \tau(M_1)$. Damit gilt $x \in \tau(M_2) = R \backslash M_2$, also auch $x \notin M_2$. ↯.
Annahme: $\exists M \in \mathcal{P}(\mathbb{N})$ mit $\tau(M) = M$
Dann gilt für jedes $m \in M$ aber $m \in \tau(M) = R \backslash M$ also $m \notin M$. ↯
ad d. Seien $M_1, M_2 \in \mathcal{P}(\mathbb{N})$ mit $M_1 \subseteq M_2$. Seit weiter $m \in \tau(M_1)$. Ist $m = 1$, so ist m offensichtlich Element von $\tau(M_2)$. Sei also $m = 2n$ für ein passendes $n \in M_1$. Wegen $M_1 \subseteq M_2$ gilt $n \in M_2$ und somit $m \in \tau(M_2)$.

Ist $x \in F$, so gilt entweder $x = 1$ oder $x = 2\ell$ für ein $\ell \in \mathbb{N}$. Damit gilt $x \in \tau_(F)$. Ist $x \in \tau(F)$, so ist x entweder 1 oder es existiert ein $y \in F$ mit $x = 2y$. Im ersten Fall ist x offensichtlich in $\tau(F)$ und im zweiten Fall ist x offensichtlich gerade und somit auch in $\tau(F)$.
Annahme: $\exists M \in \mathcal{P}(\mathbb{N})\backslash\{F\} : \tau(M) = M$
Wegen $1 \in \tau(M) = M$ ist M nicht leer. Wähle $m \in M$ maximal. m kann o.B.d.A. als echt größer 1 angenommen werden, da mit $1 \in M$ schon $2 \in \tau(M) = M$ gilt. Gäbe es ein ℓ mit $m = 2\ell + 1$, so wäre $\ell \geq 1$ und somit $2\ell + 1 \in \tau(M)$. Damit gälte $2\ell + 1 = 2n$ für ein $n \in M$. ↯ Somit kann m als gerade vorausgesetzt werden. Damit enthält M außer der 1 nur gerade Zahlen. Wegen $m \in M$ gilt aber $m < 2m \in \tau(M) = M$, was ein Widerspruch zur Maximalität von m ist. □

Aufgabe 2. Sei P ein Datalog-Programm. Dann gilt
1) Mit je zwei Herbrand-Modellen I_1 und I_2 von P ist auch $I_1 \cap I_2$ ein Herbrand-Modell von P.
2) Es existiert ein P und es existieren Herbrand-Modelle I_1 und I_2 von P, so dass $I_1 \cup I_2$ kein Herbrand-Modell von P ist.
3) Die Herbrand-Basis HB ist ein Herbrand-Modell von P.
4) Es existiert ein P, so dass $\emptyset$ ein Herbrand-Modell von P ist.
5) Seien Konst_P die Menge der in P vorkommenden Konstantensymbole und I ein beliebiges Herbrand-Modell von P. Dann ist auch $I = \{p(k_1, \dots, k_j) \in I |\, p \in \mathrm{Pred}, k_1, \dots, k_j \in \mathrm{Konst}_P\}$ ein Herbrand-Modell von P.

Beweis. ad 1) Seien I_1 und I_2 Herbrand-Modelle von P und $d \in P$. Ist d ein Grundatom, so gilt $d \in I_1, I_2$ und somit $d \in I_1 \cap I_2$. Sei d nun $q(\dots) : -p_1(\dots), \dots, p_m(\dots)$ und ϱ eine Belegung von d. Ist ein $||p_i(\dots)||_\varrho$ $(i \in [m])$ weder in I_1 noch in I_2, so gilt auch $||p_i(\dots)||_\varrho \notin I_1 \cap I_2$ und wegen der falschen Prämisse ist die Regel gültig und es gilt $\models_{I_1 \cap I_2, \varrho} d$. Sind für alle $i \in [m]$ $p_i(\dots) \in I_1, I_2$, so gilt die Behauptung offensichtlich. Seien nun $||p_{i_1}||_\varrho, \dots, ||p_{i_k}||_\varrho \in I_1$ und $||p_{j_1}||_\varrho, \dots, ||p_{j_\ell}||_\varrho \in I_2$ mit $i_1, \dots, i_k, j_1, \dots, j_\ell \in [m]$ und $\{i_1, \dots, i_k\} \cap \{j_1 \dots, j_k\} \neq [m], \emptyset$. Dann existiert ein $c \in [m]$, so dass $||p_c(\dots)||_\varrho \notin I_1 \cap I_2$ gilt. Damit ist die Prämisse falsch und die Regel gültig unter $I_1 \cap I_2$.

ad 2) Betrachte $P = \{p(X) : -q(X), r(X)\}$ sowie $I_1 = \{q(1)\}$ und $I_2 = \{r(1)\}$ für $\mathrm{Konst}_P = \{1\}$. Dann sind I_1 und I_2 Herbrand-Modelle, da in beiden Fällen die Prämissen falsch sind. $I_1 \cup I_2 = \{r(1), q(1)\}$ ist allerdings kein Herbrand-Modell für P, da die Prämisse wahr ist, $q(1)$ aber nicht in $I_1 \cup I_2$ enthalten ist.

ad 3) In HB_P sind alle Grundatome enthalten. Da $\mathrm{cons}(P)$ ein Herbrand-Modell ist, $\mathrm{cons}(P) \subseteq \mathrm{HB}_P$ gilt und HB_P maximal ist, ist HB_P ein Herbrand-Modell von P.

ad 4) Betrachte $P = \{: -p(X)\}$. Da kein Element in der leeren Menge ist, ist die Prämisse falsch und Regel somit gültig.

ad 5) Da I ein Modell ist, sind die Grundatome in I und somit auch in I'. Da für alle Konstanten genau die Klauseln aufgenommen werden, die auch in I sind und I ein Herbrand-Modell ist, überträgt sich die Eigenschaft auf I'.

Aufgabe 3. Geben Sie eine natürlichsprachige Definition folgender Verwandschaftsbeziehungen an, übersetzen Sie diese in reines Datalog und testen Sie Ihre Lösung mit CORAL:
a) elternteil
b) großvater
c) schwester (incl. Halbschwester)
d) onkel
e) nichte
f) vorfahre
g) sind_verwand

ad a)
Die natürlichsprachige Definition wäre "Vater oder Mutter", das Datalog Programm sieht entsprechend so aus:

```
module serie.
export parent(ff,fb,bf).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

end_module.
```

ad b)
Die natürlichsprachige Definition wäre "Vater oder Muter von Vater oder Mutter", das Datalog Programm sieht dementsprechend so aus:

```
module serie.
export parent(ff,fb,bf), grandparent(ff,bf,fb).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).

end_module.
```

ad c)

Die natürlichsprachige Definition wäre "Vater oder Muter von beiden Eingaben sind identisch und die zweite Eingabe (Schwester) ist weiblich", das Datalog Programm sieht so aus:

```
module serie.
export parent(ff,fb,bf), grandparent(ff,bf,fb),
   sister(ff,fb,bf).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sister(X,Y) :- parent(Z, X), parent(Z, Y), weiblich
   (Y).

end_module.
```

ad d)

Die natürlichsprachige Definition wäre "Bruder oder Schwester meines Vaters oder meiner Mutter", das Datalog Programm sieht so aus:

```
module serie.
export parent(ff,fb,bf), grandparent(ff,bf,fb),
   sister(ff,fb,bf), uncle(ff,fb,bf).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sister(X,Y) :- parent(Z, X), parent(Z, Y), weiblich
   (Y).
silbling(X,Y) :- parent(Z,X), parent(Z, Y).
uncle(X,Y) :- parent(Z,Y), silbling(Z,X), maennlich
   (X).

end_module.
```

ad e)

Die natürlichsprachige Definition wäre "Tochter meiner Schwester oder meines Bruders", das Datalog Programm sieht so aus:

```
module serie.
export parent(ff,fb,bf), grandparent(ff,bf,fb),
   sister(ff,fb,bf), uncle(ff,fb,bf), niece(ff,fb,
   bf).

parent(X,Y) :- vater(X,Y).
```

```
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sister(X,Y) :- parent(Z, X), parent(Z, Y), weiblich
   (Y).
silbling(X,Y) :- parent(Z,X), parent(Z, Y).
uncle(X,Y) :- parent(Z,Y), silbling(Z,X), maennlich
   (X).
niece(X,Y) :- parent(Z,X), silbling(Y,Z), weiblich(
   X).

end_module.
```

ad f) Die natürlichsprachige Definition wäre “Ein einer vorherigen Generation angehöriger Verwandter”, das Datalog Programm sieht so aus:

```
module serie.
export parent(ff,fb,bf), grandparent(ff,bf,fb),
   sister(ff,fb,bf), uncle(ff,fb,bf), niece(ff,fb,
   bf), ancestor(ff,fb,bf).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sister(X,Y) :- parent(Z, X), parent(Z, Y), weiblich
   (Y).
silbling(X,Y) :- parent(Z,X), parent(Z, Y).
uncle(X,Y) :- parent(Z,Y), silbling(Z,X), maennlich
   (X).
niece(X,Y) :- parent(Z,X), silbling(Y,Z), weiblich(
   X).
ancestor(X,Y) :- parent(X, Y).
ancestor(X,Y) :- parent(X, Z), ancestor(Z,Y).

end_module.
```

ad g)

Die natürlichsprachige Definition “es gibt einen gemeinsamen Vorfahren” ist schon in der Aufgabenstellung gegeben, das Datalog Programm sähe entsprechend so aus:

```
module serie.
```

```
export parent(ff,fb,bf), grandparent(ff,bf,fb),
    sister(ff,fb,bf), uncle(ff,fb,bf), niece(ff,fb,
    bf), ancestor(ff,fb,bf), is_related(ff,fb,bf).

parent(X,Y) :- vater(X,Y).
parent(X,Y) :- mutter(X,Y).

grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sister(X,Y) :- parent(Z, X), parent(Z, Y), weiblich
    (Y).
silbling(X,Y) :- parent(Z,X), parent(Z, Y).
uncle(X,Y) :- parent(Z,Y), silbling(Z,X), maennlich
    (X).
niece(X,Y) :- parent(Z,X), silbling(Y,Z), weiblich(
   X).
ancestor(X,Y) :- parent(X, Y).
ancestor(X,Y) :- parent(X, Z), ancestor(Z,Y).

is_related(X,Y) :- ancestor(X,Z), ancestor(Y,Z),
    not equal(X,Y).

end_module.
```

Dementsprechend kommt dieses Beispiel nicht ohne build-in prädikate aus.

Aufgabe 4.

Das folgende Programm setzt die Funktionaltät um:

```
module serie.
export d(bbb,fbb,bfb,bbf,fbf,bff).



d(X,Y,0) :- X = Y.
d(X,Y,Z) :- length(X,Z).
d(X,Y,Z) :- length(Y,Z).



d(X,Y,Z) :- extract(X,0,CX), extract(Y,0,CY), CX =
   CY, substr(X,CX,RX), substr(Y,CY,RY), d(RX,RY,Z)
   .
d(X,Y,Z) :- extract(X,0,CX), extract(Y,0,CY), CX !=
     CY, substr(X,CX,RX), substr(Y,CY,RY), d(RX,RY,Z
     - 1).
d(X,Y,Z) :- extract(X,0,CX), substr(X,CX,RX), d(RX,
   Y, Z-1).
```

```
d(X,Y,Z) :- extract(Y,0,CY), substr(Y,CY,RY), d(X,
    RY, Z-1).

end_module.
```