Lösungen zu den Aufgaben zur Vorlesung “*Grundlagen der Informatik*”

Lösung Aufgabe 1:

(a) Folgende Prozedur leistet das Gewünschte:

```

both := procedure(l1, l2) {
    return { x in l1 | x in l2 };
};
```

(b) Folgende Prozedur leistet das Gewünschte:

```
procedure pythagoras(n);
    return { [ x, y, z ] : x in {1..n}, y in {1..n}, z in {1..n}
                         | x*x + y*y = z*z
           };
end pythagoras;
```

(c) Folgende Prozedur leistet das Gewünschte:

```
subsets := procedure(n, k) {
    return { m in pow({1 .. n }) | #m == k };
};
```

(d) Folgende Prozedur leistet das Gewünschte:

```
% pythagoras(+Number, -List(Triple)).
pythagoras(N, L) :-
        pythagoras(N, [], L).

% pythagoras(+Number, +List(Triple), -List(Triple)).
pythagoras(N, Lin, Lout) :-
        pythagorasTriple(N, T),
        \+ member(T, Lin),
        !,
        pythagoras(N, [ T | Lin ], Lout).
pythagoras(_, L, L).

% pythagorasTriple(+Number, -Triple) :-
pythagorasTriple(N, [X, Y, Z]) :-
        M is N - 1,
        between(1, M, X),
        between(1, M, Y),
        between(1, M, Z),
        Lhs is X * X + Y * Y,
        Rhs is Z * Z,
        Lhs == Rhs.
```

Lösung Aufgabe 2:

(a) Folgende Prozedur leistet das Gewünschte:

```
compose := procedure(a1, a2) {
    return { [[x,z], d1 + d2] : [[x,y1], d1] in a1, [[y2,z], d2] in a2
                              | y1 == y2
           };
};
```

(b) Folgende Prozedur leistet das Gewünschte:

```
closure := procedure(a) {
    c    := a;
    oldC := {};
    while (c != oldC) {
        oldC := c;
        c    := c + compose(c, c);
    }
    return c;
};
```

Lösung Aufgabe 3: Die folgende Implementierung leistet das Gewünschte:

```
toLists := procedure(m) {
    if (m == {}) {
        return { [] };
    }
    return { [ x ] + l : x in m, l in toLists(m - {x}) };
};
```

Lösung Aufgabe 4:

(a) Wir führen die folgenden aussagenlogischen Variablen als Abkürzungen ein.

1. Linker Motor fällt aus: l
2. Rechter Motor fällt aus: r
3. Flugzeug stürzt ab: a

Die gesuchte Schluss-Regel ist:

$$\frac{l \vee r \rightarrow a \qquad a \qquad \neg l}{r}$$

(b) Die Schluss-Regel ist genau dann korrekt, wenn die Formel

$$(l \vee r \rightarrow a) \wedge a \wedge \neg l \rightarrow r$$

eine Tautologie ist. Wir formen diese Formel in KNF um:

$$\begin{array}{ll} & (l \vee r \rightarrow a) \wedge a \wedge \neg l \rightarrow r \\ \Leftrightarrow & \neg\big((l \vee r \rightarrow a) \wedge a \wedge \neg l\big) \vee r \\ \Leftrightarrow & \neg\big((\neg(l \vee r) \vee a) \wedge a \wedge \neg l\big) \vee r \\ \Leftrightarrow & \neg(\neg(l \vee r) \vee a) \vee \neg a \vee l \vee r \\ \Leftrightarrow & ((l \vee r) \wedge \neg a) \vee \neg a \vee l \vee r \\ \Leftrightarrow & (l \vee r \vee \neg a \vee l \vee r) \wedge (\neg a \vee \neg a \vee l \vee r) \\ \Leftrightarrow & (l \vee r \vee \neg a) \end{array}$$

Diese Formel ist keine Tautologie. Ein Gegenbeispiel ist durch die Belegung

$$\mathcal{I} := \big\{\langle l, \texttt{false}\rangle, \langle r, \texttt{false}\rangle, \langle a, \texttt{true}\rangle\big\}$$

gegeben. Diese Belegung hätte man auch ohne die Rechnung angeben dürfen. Anschaulich ist das Ergebnis klar, denn das Flugzeug kann auch abstürzen, wenn beide Motoren funktionieren. Nur mit der Prämisse

$$l \vee r \leftrightarrow a$$

hätte man aus dem Absturz auch auf den Ausfall eines Motors schließen können.

Lösung Aufgabe 5:

(a) Die gesuchte Formel ist

$$(q \wedge p) \wedge (r \vee \neg q) \rightarrow r \wedge p.$$

(b) Wir formen diese Formel in konjunktive Normalform um:

$$
\begin{array}{cl}
 & (q \wedge p) \wedge (r \vee \neg q) \rightarrow r \wedge p \\
\Leftrightarrow & \neg\big((q \wedge p) \wedge (r \vee \neg q)\big) \vee (r \wedge p) \\
\Leftrightarrow & \big(\neg(q \wedge p) \vee \neg(r \vee \neg q)\big) \vee (r \wedge p) \\
\Leftrightarrow & \big((\neg q \vee \neg p) \vee (\neg r \wedge q)\big) \vee (r \wedge p) \\
\Leftrightarrow & \big((\neg q \vee \neg p \vee \neg r) \wedge (\neg q \vee \neg p \vee q)\big) \vee (r \wedge p) \\
\Leftrightarrow & \big((\neg q \vee \neg p \vee \neg r) \wedge \top\big) \vee (r \wedge p) \\
\Leftrightarrow & (\neg q \vee \neg p \vee \neg r) \vee (r \wedge p) \\
\Leftrightarrow & (\neg q \vee \neg p \vee \neg r \vee r) \wedge (\neg q \vee \neg p \vee \neg r \vee p) \\
\Leftrightarrow & \top \wedge \top \\
\Leftrightarrow & \top \\
\Leftrightarrow & \{\}
\end{array}
$$

Lösung Aufgabe 6:

(a) Die folgende Prozedur leistet das Gewünschte:

```
allVars := procedure(e) {
    operators := {"+", "-", "*", "/"};
    switch {
        case isString(e):       return { e };
        case e(2) in operators: return allVars(e(1)) + allVars(e(3));
        default:                abort("Error in allVars($e$)");
    }
};
```

(b) Die folgende Prozedur leistet das Gewünschte:

```
countVars := procedure(e, x) {
    operators := {"+", "-", "*", "/"};
    switch {
        case x == e:            return 1;
        case isString(e):       return 0;
        case e(2) in operators: return countVars(e(1), x) + countVars(e(3), x);
        default:                abort("Error in countVars($e$)");
    }
};
```

(c) Die folgende Prozedur leistet das Gewünschte:

```
singleVars := procedure(e) {
    return { x in allVars(e) | countVars(e,x) == 1 };
};
```

Lösung Aufgabe 7: Die folgende Kette von Anwendungen der Schnitt-Regel weist die Behauptung nach:

1. $\{\neg t, \neg p\}, \{\neg t, p\} \vdash \{\neg t\}$,
2. $\{\neg t\}, \{t, \neg q\} \vdash \{\neg q\}$,
3. $\{\neg q\}, \{q, \neg p\} \vdash \{\neg p\}$,
4. $\{\neg p\}, \{t, p, \neg r\} \vdash \{t, \neg r\}$,
5. $\{t, \neg r\}, \{\neg t\} \vdash \{\neg r\}$,
6. $\{\neg r\}, \{t, q, r\} \vdash \{t, q\}$,
7. $\{t, q\}, \{\neg t\} \vdash \{q\}$,
8. $\{q\}, \{\neg q\} \vdash \{\}$.

Lösung Aufgabe 8: Da die Menge M keine Unit-Klausel enthält, beginnen wir mit einer Fall-Unterscheidung. Wir führen die Fall-Unterscheidung bezüglich p durch:

1. Als erstes betrachten wir daher die Menge $M_0 = M \cup \big\{\{p\}\big\}$. Wir berechnen

   $$M_1 := \mathit{reduce}(M_0, p) = \big\{\ \{r, \neg t\}, \{\neg r, q\}, \{\neg q, s\}, \{\neg r, \neg s\}, \{\neg s\}, \{q, t\}, \{p\}\ \big\}.$$

   M_1 enthält die neue Unit-Klausel $\{\neg s\}$. Daher berechnen wir nun

   $$M_2 := \mathit{reduce}(M_1, \neg s) = \big\{\ \{r, \neg t\}, \{\neg r, q\}, \{\neg q\}, \{q, t\}, \{p\}, \{\neg s\}\ \big\}.$$

   M_2 enthält die neue Unit-Klausel $\{\neg q\}$. Daher berechnen wir nun

   $$M_3 := \mathit{reduce}(M_2, \neg q) = \big\{\ \{r, \neg t\}, \{\neg r\}, \{t\}, \{p\}, \{\neg s\}, \{\neg q\}\ \big\}.$$

   M_3 enthält die neue Unit-Klausel $\{t\}$. Daher berechnen wir nun

   $$M_4 := \mathit{reduce}(M_3, t) = \big\{\ \{r\}, \{\neg r\}, \{p\}, \{\neg s\}, \{\neg q\}, \{t\}\ \big\}.$$

   Da M_4 sowohl die Unit-Klausel $\{r\}$ als auch die Unit-Klausel $\{\neg r\}$ enthält, ist M_4 nicht lösbar.

2. Nun betrachten wir die Menge $M \cup \big\{\{\neg p\}\big\}$.

   $$M_5 := \mathit{reduce}(M_0, \neg p) = \big\{\ \{q\}, \{r, s\}, \{r, \neg s\}, \{\neg q, s\}, \{\neg r, \neg s\}, \{\neg p\}\ \big\}$$

   M_5 enthält die neue Unit-Klausel $\{q\}$. Daher berechnen wir nun

   $$M_6 := \mathit{reduce}(M_5, q) = \big\{\ \{r, s\}, \{r, \neg s\}, \{s\}, \{\neg r, \neg s\}, \{\neg p\}, \{q\}\ \big\}$$

   M_6 enthält die neue Unit-Klausel $\{s\}$. Daher berechnen wir nun

   $$M_7 := \mathit{reduce}(M_6, s) = \big\{\ \{r\}, \{\neg r\}, \{\neg p\}, \{q\}, \{s\}\ \big\}$$

   Da M_7 sowohl die Unit-Klausel $\{r\}$ als auch die Unit-Klausel $\{\neg r\}$ enthält, ist M_7 nicht lösbar.

Da sowohl $M \cup \big\{\{p\}\big\}$ als auch $M \cup \big\{\{\neg p\}\big\}$ unlösbar sind, ist auch die Menge M unlösbar. $\Box$

Lösung Aufgabe 9: Wir setzen $\mathcal{U} := \{a, b\}$ und definieren die Interpretation $p^{\mathcal{J}}$ des Prädikats-Zeichens p als

$$p^{\mathcal{J}} := \bigl\{\langle a, a\rangle, \langle b, b\rangle\bigr\}.$$

Dann gibt es in U zwar für jedes x ein y, so dass $p(x, y)$ gilt, denn wir können $y = x$ setzen. Aber es gibt kein y, so dass für alle x die Formel $p(x, y)$ gilt. Wollten wir dies mit dem in der Vorlesung gezeigten Programm testen, so könnten wir folgendes definieren:

```
a := "a";
b := "b";
u := { a, b };  // the universe
// pJ is the interpretation of the predicate symbol "p".
pJ := { [ a, a ], [ b, b ] };
// The structure consists of the universe and the interpretation of
// the function and predicate symbols.
j := { [ "p", pJ ]};
s := [ u, j ];
// I is a variable assignment.
i := { [ "x", a ], [ "y", b ] };
f := parse("(forall x: exists y: p(x,y)) -> (exists y: forall x: p(x,y))");
print(f);
print(evalFormula(f, s, i));
```

Dann würde der Aufruf “`evalFormula(f, S, I)`” den Wert `false` liefern.

Lösung Aufgabe 10:

$$\begin{array}{ll}
 & \Bigl(\forall x : \neg\exists y : \bigl(q(x) \rightarrow p(x, y)\bigr)\Bigr) \rightarrow \bigl(\forall z : q(z)\bigr) \\
\Leftrightarrow & \neg\Bigl(\forall x : \neg\bigl(\exists y : q(x) \rightarrow p(x, y)\bigr)\Bigr) \vee \bigl(\forall z : q(z)\bigr) \\
\Leftrightarrow & \Bigl(\exists x : \exists y : \bigl(q(x) \rightarrow p(x, y)\bigr)\Bigr) \vee \bigl(\forall z : q(z)\bigr) \\
\Leftrightarrow & \exists x : \Bigl(\exists y : \bigl(q(x) \rightarrow p(x, y)\bigr) \vee \bigl(\forall z : q(z)\bigr)\Bigr) \\
\Leftrightarrow & \exists x : \exists y : \Bigl(\bigl(q(x) \rightarrow p(x, y)\bigr) \vee \bigl(\forall z : q(z)\bigr)\Bigr) \\
\Leftrightarrow & \exists x : \exists y : \forall z : \Bigl(\bigl(q(x) \rightarrow p(x, y)\bigr) \vee q(z)\Bigr) \\
\approx_e & \exists y : \forall z : \Bigl(\bigl(q(s_1) \rightarrow p(s_1, y)\bigr) \vee q(z)\Bigr) \qquad \text{mit der Skolem-Konstante } s_1 \text{ für } x \\
\approx_e & \forall z : \Bigl(\bigl(q(s_1) \rightarrow p(s_1, s_2)\bigr) \vee q(z)\Bigr) \qquad \text{mit der Skolem-Konstante } s_2 \text{ für } y \\
\Leftrightarrow & \forall z : \neg q(s_1) \vee p(s_1, s_2) \vee q(z) \\
\Leftrightarrow & \bigl\{\{\neg q(s_1),\ p(s_1, s_2),\ q(z)\}\bigr\}
\end{array}$$

Lösung Aufgabe 11:

```
% is_contained(+List(T), +List(T)).

is_contained([], L).

is_contained([ X | R ], L) :-
    member(X, L),
    is_contained(R, L).
```

Lösung Aufgabe 12:

```
% all_lists(+Number, +Number, -List(Number)).

all_lists(_N, 0, []).

all_lists(N, K, [ X | T ]) :-
        K > 0,
        K1 is K - 1,
        all_lists(N, K1, T),
        between(1, N, X),
        \+ member(X, T).
```