Ex2_LuisZüttel_GionRubitschung_D1P

March 4, 2024

1 Exercise 1 (Operationen auf Mengen)

Gegeben sind die Mengen $Starters = \{salad, soup\}$, $Main = \{pizza, steak\}$, and $Dessert = \{fruit, icecream\}$, sowie die Relationen $Father = \{(adam, bert), (bert, carl)\}$ und $Car = \{(adam, audi), (bert, bmw), (carl, chevy)\}$.

1. $Starters \times Main \times Dessert$ $Starters \times Main = \{(salad, pizza), (salad, steak), (soup, pizza), (soup, steak)\}$

$Starters \times Main \times Dessert = \{((salad, pizza), fruit), (salad, steak), fruit), (soup, pizza), fruit), (soup, steak), fru$

2. $Father.Father$

$Father.Father = \{(adam, carl)\}$

3. $Father.Car$

$Father.Car = \{(adam, bmw), (bert, chevy)\}$

4. $Car.Father$

$Car.Father = \emptyset$

2 Exercise 2 (Aussagen über Mengen)

Welche der folgenden Aussagen sind wahr? Begründen Sie jeweils, warum. Wenn eine Aussage falsch ist, dann zeigen Sie das mit einem Gegenbeispiel. Die Mengen A, B, C seien Teilmengen einer beliebigen endlichen Grundmenge U.

1. Wenn $A \cup B = A \cup C$ dann $B = C$

- $a \in A$ oder $a \in B$
- $a \in A$ oder $a \in C$

Dann ist $B = C$, weil die Teilmenge A mit jeweils B und C diesselbe Menge gibt.

2. Es gilt $A \subseteq B$ genau dann wenn $\overline{B} \subseteq \overline{A}$

$A = \{1, 2\}$ $\overline{A} = \{3, ..., \infty\}$

$B = \{1, 2, 3\}$ $\overline{B} = \{4, ..., \infty\}$

$A \subseteq B =$ wahr

$\overline{B} \subseteq \overline{A} =$ wahr, weil in $\overline{B}$ alles nach und mit 4, auch in $\overline{A}$ vorkommt

$\Longrightarrow A \subseteq B \Longleftrightarrow \overline{B} \subseteq \overline{A}$

3. $A \times B = B \times A$

Nein, denn wenn wir beispielsweise die Mengen $A = \{(1), (5), (20)\}$ und $B = \{(2), (3), (4)\}$ nehmen und die kartesischen Produkte beider Gleichungen rechnen, bekommen wir:

- $A \times B = \{(1, 2), (1, 3), (1, 4), (5, 2), (5, 3), (5, 4), (20, 2), (20, 3), (20, 4)\}$
- $B \times A = \{(2, 1), (2, 5), (2, 20), (3, 1), (3, 5), (3, 20), (4, 1), (4, 5), (4, 20)\}$

Also ist die Gleichung falsch, denn $A \times B \neq B \times A$

4. $|A \times B| = |A| \cdot |B|$

Diese Aussage stimmt, denn wir können für A und B die Kardinalität 3 setzen. Also beide Mengen besitzen jeweils 3 Elemente.

Das bedeutet das die Kardinalität des kartesischem Produkt der beiden Mengen $3 \cdot 3 = 9$ wäre. Genau wie wenn man die Kardinalität beider Mengen einzeln miteinander multipliziert.

3 Exercise 3 (Mengen als Listen)

Mengen werden häufig mit Hilfe von Listen implementiert. Geben Sie in einer Programmiersprache ihrer Wahl oder in Pseudocode eine Funktion `boolean equals(list l1,list l2)` an, die genau dann `true` liefert, wenn die Mengen der Elemente der beiden Listen gleich sind. Gegeben sind die Funktionen `int head(list l)` und `list tail(list l)`, die für nichtleere Listen jeweils das erste Element und die Restliste liefern, sowie die Funktion `boolean empty(list l)`. Geben Sie möglichst einfachen, kurzen und offensichtlich korrekten Code an, die Laufzeit spielt keine Rolle. Definieren Sie sich geeignete Hilfsfunktionen.

[ ]:
```python
def head(l: list) -> int:
    return l[0]

def tail(l: list) -> list:
    return l[1:]

def empty(l: list) -> bool:
    return len(l) == 0

def contains(l: list, e: int) -> bool:
    if empty(l):
        return False
    elif head(l) == e:
        return True
    else:
        return contains(tail(l), e)

def remove(l: list, e: int) -> list:
    if empty(l):
        return []
    elif head(l) == e:
        return tail(l)
```

```
    else:
        return [head(l)] + remove(tail(l), e)

def equals(l1: list, l2: list) -> bool:
    if empty(l1) and empty(l2):
        return True
    elif empty(l1) or empty(l2):
        return False
    elif contains(l2, head(l1)):
        return equals(tail(l1), remove(l2, head(l1)))
    else:
        return False

l1 = [2, 1, 3]
l2 = [1, 2, 3]
print(equals(l1, l2))

l1 = [1, 2, 3]
l2 = [2, 1, 4]
print(equals(l1, l2))
```

```
True
False
```

4 Exercise 4 (Mengen von Natürlichen Zahlen als Boolean Arrays)

Eine endliche Menge S von natürlichen Zahlen kann man als Array `boolean[] s` repräsentieren, wie folgt:

`s[n] = true` falls $n \in S$

`s[n] = false` falls $n \notin S$

Die Menge $S = \{1, 3, 4\}$ zum Beispiel ist dann durch folgendes Array repräsentiert:

```
s[0] = false
s[1] = true
s[2] = false
s[3] = true
s[4] = true
```

Geben Sie möglichst einfache Algorithmen in Pseudocode an für folgende Funktionen:

1. `boolean subset(boolean[] s, boolean[] t)`, die genau dann `true` liefert, wenn S eine Teilmenge von T ist

[ ]:
```
def subset(s: list[bool], t: list[bool]) -> bool:
    for n in range(len(s)):
        if s[n] and (n >= len(t) or not t[n]):
            return False
```

```
    return True
```

Testen wir diese Funktion mit den Mengen $S = \{1, 3, 4\}$ und $S = \{1, 3, 4\}$. Hierbei ist $S = T$ und somit $S \in T$, die Funktion muss also `True` zurückgeben.

[ ]:
```
s = [False, True, False, True, True]
t = [False, True, False, True, True]

subset(s, t)
```

[ ]: True

Was ist wenn $S \neq T$? Setzen wir $T = \{1, 2, 3, 4, 5\}$. Hierbei ist immernoch $S \in T$, die Funktion muss also wieder `True` zurückgeben.

[ ]:
```
s = [False, True, False, True, True]
t = [False, True, True, True, True, True]

subset(s, t)
```

[ ]: True

Was ist wenn $S \notin T$? Setzen wir $S = \{1, 3\}$ und $T = \{2, 4\}$, die Funktion muss also `False` zurückgeben.

[ ]:
```
s = [False, True, False, True]
t = [False, False, True, False, True]

subset(s, t)
```

[ ]: False

2. `boolean[] union(boolean[] s, boolean[] t)`, die die (Repräsentation der) Mengenvereinigung von S und S liefert.

[ ]:
```
def union(s: list[bool], t: list[bool]) -> list[bool]:
    max_length = max(len(s), len(t))
    result = [False] * max_length

    for n in range(max_length):
        if n < len(s) and s[n]:
            result[n] = True
        if n < len(t) and t[n]:
            result[n] = True

    return result
```

Testen wir die Funktion mit $S = \{1, 3\}$ und T={2,3,4}$, wobei $S \cup T = \{1, 2, 3, 4\}$ gibt, die Funktion muss also `[False, True, True, True, True]` zurück geben.

[ ]:
```
s = [False, True, False, True]
t = [False, False, True, True, True]

union(s, t)
```

[ ]:
```
[False, True, True, True, True]
```

5 Exercise 5 (Relationen als Boolean Arrays)

Eine endliche binäre Relation R auf den natürlichen Zahlen sei als ein zweidimensionales Array `boolean[][] r` repräsentiert, so dass gilt:

`r[m][n] = True`

falls $(m, n) \in R$

`r[m][n] = False`

falls $(m, n) \notin R$

Geben Sie einen möglichst einfachen Algorithmus in Pseudocode an für die Funktion `boolean[][] compose(boolean[][] r, boolean[][] t)`, die die (Repräsentation der) Komposition der Relation R und T liefert.

[ ]:
```
r = [[True, False], [False, True], [False, False], [True, True]]
t = [[False, False], [False, True]]

def compose(r: list[list[bool]], t: list[list[bool]]) -> list[list[bool]]:
    result = []
    for i in range(len(r)):
        for j in range(len(t[0])):
            if r[i][1] == t[j][0]:
                result.append([r[i][0], t[j][1]])
    return result

print(compose(r, t))
```

```
[[True, False], [True, True], [False, False], [False, True]]
```

6 Exercise 6 (Anzahl Relationen zwischen Mengen)

Gegeben seien zwei Mengen A und B mit $|A| = m$ und $|B| = n$. Wieviele Relationen zwischen A und B gibt es? Betrachten Sie Beispiele. Begründen Sie Ihre Antwort.

Die Anzahl Relationen zwischen 2 Mengen, kann man ausrechnen indem man die Kardinalität beider Mengen miteinander multipliziert. Sowie wir es bereits in der Aufgabe 2.4 gemacht haben.

Was jedoch zu berücksichtigen ist wohl, dass die 2 Mengen entweder als $A \times B$ oder \$ B ×A\$ gerechnet werden können und nicht dasselbe Resultat zurückgeben, wie in Aufgabe 2.3.

Das bedeutet, das die Anzahl möglicher Relationen zwischen den Beiden Mengen $mn + nm = \underline{\underline{2mn}}$ ist.