Aufgabe 4: Nandu

Team-ID: 01305

Team-Name: RecursionLimitExceeded

Bearbeiter/-innen dieser Aufgabe:
Tim Himmelsbach

20. November 2023

Inhaltsverzeichnis

1 Lösungsidee

Es fällt schwer bei den verschieden farbigen Bausteinen nicht an logische Gatter zu denken.

Sei $Q_i \in \{0, 1\}$ eine Taschenlampe (in der Aussagelogik auch Literal genannt). Die Taschenlampe kann zwei Zustände annehmen: Licht an oder Licht aus. Dies wird in der Aussagelogik mit 0 (falsch)

und 1 (wahr) dargestellt. Um eine Wahrheitstabelle zu erstellen, müssen alle möglichen Kombinationen betrachtet werden. Bei $Q_1, Q_2, ..., Q_n$ sind das $2 * 2 * ... * 2$, also 2^n Kombinationsmöglichkeiten.

Der rote Baustein ist die logische Negation, auch NOT Gatter genannt. Ist der Sensor dunkel, dann emittiert der Baustein Licht und umgekehrt. Man könnte sagen, dass der Baustein das Gegenteil des Sensors abgibt.

$$LED_1, LED_2 = \neg SENSOR_1 \tag{1}$$

Die weißen Bausteine sind äquivalent zu den NAND Gattern aus der Aussagelogik. Es wird nur dann ein Licht emmitiert, wenn beide Sensoren S_1 und S_2 dunkel sind.

$$LED_1, LED_2 = \neg(SENSOR_1 \wedge SENSOR_2) \tag{2}$$

Der blaue Baustein gibt das Licht einfach weiter.

$$LED_1 = SENSOR_1 \quad LED_2 = SENSOR_2 \tag{3}$$

Die Schaltung ist als Raster aufgebaut. In der obersten Reihen befinden sich die Eingaben $Q_1, Q_2, ...Q_n$ und in der untersten Reihe die Ausgaben $L_1, L_2, ..., L_m$. Dazwischen befinden sich die einzelnen Bausteine. Die Sensoren der Bausteine beziehen sich auf die Reihe oberhalb. Für alle möglichkeiten der Taschenlampe wir die Schaltung druchlaufen und dei Ergebnisse werden ausgegeben.

1.1 Mögliche Optimierungen

Für alle Beispieleingaben $2 \leq n \leq 6$ (n ist die Anzahl der Taschenlampen) wird die Schaltung im Bruchteil einer Sekunde berechnet. Doch die theoretische Laufzeit $\mathcal{O}(2^n)$ skaliert exponentiell. Somit kann eine Schaltung mit $n = 32$ schon mal mehrere Stunden dauern. Schon das Problem nur eine Konfiguration zu finden, für die ein Output einen bestimmten Wert annimmt, ist NP-schwer.

In vielen Fällen können Optimierungen helfen. Die praktische Laufzeit kann dramatisch reduziert werden, wärend die theoretische Laufzeit aber gleich bleibt.

Die Schaltung wird als azyklischer Graph $G = (V, E)$ dargestellt, dabei repräsentiert ein Knoten einen AND oder NOT Gatter und eine gerichtete Kante die Verbindung zwischen einem Sensor und der LED eines anderen Bausteins. So haben die Ausgänge keine eingehenden Kanten und die Eingänge keine ausgehenden Kanten.

Gibt es keinen Pfad von Knoten L_m zum Knoten Q_n, dann hat Q_n auch keinen Einfluss auf das Ergebniss von L_m und darf somit vernachlässigt werden. Für jeden Output eine eigene Wahrheitstabelle erstellt. Die Laufzeit beträgt immernoch $\mathcal{O}(2^n)$, wobei n die maximale Anzahl an Taschenlampen ist, von denen ein Output abhängig ist. In Abb. 1 ist zum Beispie zu sehen, dass nur Q_5 einfluss auf L_5 hat. Die Wahrheitstabelle für L_5 hat somit nur 2^1 Einträge. Weiter, wäre es noch möglich den Logikgraphen nach den Regeln der Aussagelogik zu vereinfachen und Tautologien, Aussagen die immer wahr sind und Kontradiktionen, Aussagen die immer falsch sind, zu entfernen.

2 Umsetzung

2.1 Ohne Optimierungen

Die Umsetzung ohne Optimierungen ist primitiv. Da das Licht nur "nach unten"weitergegeben werden kann ist es ausgeschlossen, dass Bausteine in einer Zeile sich gegeneinander Beinflussen können. Der Algorithmus erstellt alle möglichen Kombinationen von $Q_1, Q_2, ...Q_n$ in der ersten Zeile. Die Schaltung wird zeilenweise durchlaufen die Bausteine werden in-place angewendet. Ist die letzte Zeile erreicht, werden die Wahrheitswerte für $L_1, L_2, ..., L_m$ am entsprechenden Index ausgelesen, eine Wahrheitstabelle wird erstellt.

2.2 Mit Optimierungen

Im ersten Schritt wird der logische Graph aus der Eingabe erstellt. Im folgenden Schritt wird von einem Output Knoten rekursiv eine Wahrheitstabelle erstellt. Der Datentyp Wahrheitstabelle speichert die Ausgaben für alle Konfigurationen einer Menge von Eingängen. Alle Einträge einer Wahrheitstabelle können mit einer boolschen Funktion modifiziert werden. Desweiter können zwei Wahrheitstabellen mit einer boolschen Funktion kombiniert werden. Rekursiv wird definiert, dass die Wahrheitstabelle eines AND-Gatters die kombinierte Wahrheitstabellen der Eingänge, die Wahrheitstabelle eines NOT-Gatters

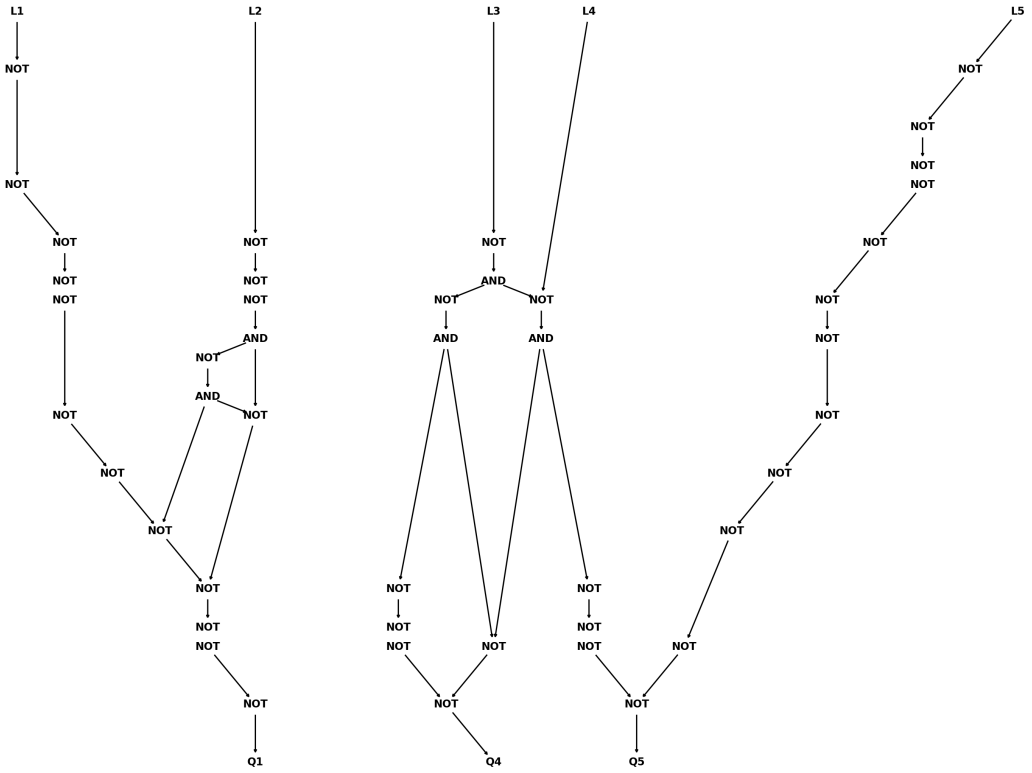

Abbildung 1: Beispiel 5 als Graph

die negierte Wahrheitstabelle des Eingangs ist. Der Basisfall ist erreicht, wenn eine Taschenlampe erreicht wird. Die Wahrheitstabelle wird in einem Cache gespeichert. Wird eine Wahrheitstabelle erneut benötigt, wird diese aus dem Cache gelesen. Dies ist dann nützlich, wenn ein Knoten zwei eingehende Kanten hat. Die Ausgabe unterscheidet sich von der ohne Optimierung. Für jeden Output wird eine eigene Wahrheitstabelle erstellt.

2.3 Implementation

Sowohl der Algorithmus mit `mitOptimierung.py` als auch ohne optimierung `ohneOptimierung.py` wurden implementiert. Die Implementierung ist in Python geschrieben. Beim Ausführen der Datei muss ein Pfad zu gewünschten Beispieleingabe als Argument übergeben werden.

3 Beispiele

Bemerkung: In der Ausgabe der `mitOptimierung.py` sind leider noch Bugs die ich auf den letzten Drücker nicht mehr beheben konnte, deswegen ist die Ausgabe teilweise inkorrekt. Verstehen Sie das eher als Beweis, dass eine Erweiterung auch in der Praxis funktionieren würde (wären da nicht die Bugs). Die Ausgaben der `ohneOptimierung.py` sind korrekt. Für jede Beispieldatei wird die Ausgabe mit und ohne Optimierung zum Vergleich aufgehführt.

3.1 nandu1.txt

3.1.1 ohne Optimierung

```
Q1|Q2||L1|L2|
0 |0 ||1 |1 |
```

```
1 |0 ||1 |1 |
0 |1 ||1 |1 |
1 |1 ||0 |0 |
Berechngszeit: 0.00011706352233886719
```

3.1.2 mit Optimierung

```
Tabelle fuer L1
Q1|Q2|L
0 |0 |1
1 |0 |1
0 |1 |1
1 |1 |0

Tabelle fuer L2
Q1|Q2|L
0 |0 |1
1 |0 |1
0 |1 |1
1 |1 |0

Berechnungszeit: 0.0002827644348144531
```

3.2 nandu2.txt

3.2.1 ohne Optimierung

```
Q1|Q2||L1|L2|
0 |0 ||0 |1 |
1 |0 ||0 |1 |
0 |1 ||0 |1 |
1 |1 ||1 |0 |
Berechngszeit: 0.00014781951904296875
```

3.2.2 mit Optimierung

```
Tabelle fuer L1
Q1|Q2|L
0 |0 |0
1 |0 |0
0 |1 |0
1 |1 |1

Tabelle fuer L2
Q1|Q2|L
0 |0 |1
1 |0 |1
0 |1 |1
1 |1 |0

Berechnungszeit: 0.0003120899200439453
```

3.3 nandu3.txt

3.3.1 ohne Optimierung

```
Q1|Q2|Q3||L1|L2|L3|L4|
0 |0 |0 ||1 |0 |0 |1 |
1 |0 |0 ||0 |1 |0 |1 |
0 |1 |0 ||1 |0 |1 |1 |
1 |1 |0 ||0 |1 |1 |1 |
0 |0 |1 ||1 |0 |0 |0 |
1 |0 |1 ||0 |1 |0 |0 |
0 |1 |1 ||1 |0 |1 |0 |
1 |1 |1 ||0 |1 |1 |0 |
Berechngszeit: 0.00021719932556152344
```

3.3.2 mit Optimierung

```
Tabelle fuer L1
Q1|L
0 |0
```

```
1 |1

Tabelle fuer L2
Q1|L
0 |1
1 |0

Tabelle fuer L3
Q2|L
0 |1
1 |0

Tabelle fuer L4
Q3|L
0 |1
1 |0

Berechnungszeit: 0.0002849102020263672s
```

3.4 nandu4.txt

3.4.1 ohne Optimierung

```
Q1|Q2|Q3|Q4||L1|L2|
0 |0 |0 |0 ||0 |0 |
1 |0 |0 |0 ||0 |0 |
0 |1 |0 |0 ||1 |0 |
1 |1 |0 |0 ||0 |0 |
0 |0 |1 |0 ||0 |1 |
1 |0 |1 |0 ||0 |1 |
0 |1 |1 |0 ||1 |1 |
1 |1 |1 |0 ||0 |1 |
0 |0 |0 |1 ||0 |0 |
1 |0 |0 |1 ||0 |0 |
0 |1 |0 |1 ||1 |0 |
1 |1 |0 |1 ||0 |0 |
0 |0 |1 |1 ||0 |0 |
1 |0 |1 |1 ||0 |0 |
0 |1 |1 |1 ||1 |0 |
1 |1 |1 |1 ||0 |0 |
Berechngszeit: 0.0004391670227050781
```

3.4.2 mit Optimierung

```
Tabelle fuer L1
Q1|Q2|L
0 |0 |0
1 |0 |0
0 |1 |1
1 |1 |0

Tabelle fuer L2
Q1|Q2|Q3|Q4|L
0 |0 |0 |0 |0
1 |0 |0 |0 |0
0 |1 |0 |0 |0
1 |1 |0 |0 |0
0 |0 |1 |0 |1
1 |0 |1 |0 |1
0 |1 |1 |0 |1
1 |1 |1 |0 |1
0 |0 |0 |1 |0
1 |0 |0 |1 |0
0 |1 |0 |1 |0
1 |1 |0 |1 |0
0 |0 |1 |1 |0
1 |0 |1 |1 |0
0 |1 |1 |1 |0
1 |1 |1 |1 |0

Berechnungszeit: 0.0004589557647705078
```

3.5 nandu5.txt

3.5.1 ohne Optimierung

```
Q1|Q2|Q3|Q4|Q5|Q6||L1|L2|L3|L4|L5|
0 |0 |0 |0 |0 |0 ||0 |0 |0 |1 |0 |
1 |0 |0 |0 |0 |0 ||1 |0 |0 |1 |0 |
0 |1 |0 |0 |0 |0 ||0 |0 |0 |1 |0 |
1 |1 |0 |0 |0 |0 ||1 |0 |0 |1 |0 |
0 |0 |1 |0 |0 |0 ||0 |0 |0 |1 |0 |
1 |0 |1 |0 |0 |0 ||1 |0 |0 |1 |0 |
0 |1 |1 |0 |0 |0 ||0 |0 |0 |1 |0 |
1 |1 |1 |0 |0 |0 ||1 |0 |0 |1 |0 |
0 |0 |0 |1 |0 |0 ||0 |0 |1 |0 |0 |
1 |0 |0 |1 |0 |0 ||1 |0 |1 |0 |0 |
0 |1 |0 |1 |0 |0 ||0 |0 |1 |0 |0 |
1 |1 |0 |1 |0 |0 ||1 |0 |1 |0 |0 |
0 |0 |1 |1 |0 |0 ||0 |0 |1 |0 |0 |
1 |0 |1 |1 |0 |0 ||1 |0 |1 |0 |0 |
0 |1 |1 |1 |0 |0 ||0 |0 |1 |0 |0 |
1 |1 |1 |1 |0 |0 ||1 |0 |1 |0 |0 |
0 |0 |0 |0 |1 |0 ||0 |0 |0 |1 |1 |
1 |0 |0 |0 |1 |0 ||1 |0 |0 |1 |1 |
0 |1 |0 |0 |1 |0 ||0 |0 |0 |1 |1 |
1 |1 |0 |0 |1 |0 ||1 |0 |0 |1 |1 |
0 |0 |1 |0 |1 |0 ||0 |0 |0 |1 |1 |
1 |0 |1 |0 |1 |0 ||1 |0 |0 |1 |1 |
0 |1 |1 |0 |1 |0 ||0 |0 |0 |1 |1 |
1 |1 |1 |0 |1 |0 ||1 |0 |0 |1 |1 |
0 |0 |0 |1 |1 |0 ||0 |0 |0 |1 |1 |
1 |0 |0 |1 |1 |0 ||1 |0 |0 |1 |1 |
0 |1 |0 |1 |1 |0 ||0 |0 |0 |1 |1 |
1 |1 |0 |1 |1 |0 ||1 |0 |0 |1 |1 |
0 |0 |1 |1 |1 |0 ||0 |0 |0 |1 |1 |
1 |0 |1 |1 |1 |0 ||1 |0 |0 |1 |1 |
0 |1 |1 |1 |1 |0 ||0 |0 |0 |1 |1 |
1 |1 |1 |1 |1 |0 ||1 |0 |0 |1 |1 |
0 |0 |0 |0 |0 |1 ||0 |0 |0 |1 |0 |
1 |0 |0 |0 |0 |1 ||1 |0 |0 |1 |0 |
0 |1 |0 |0 |0 |1 ||0 |0 |0 |1 |0 |
1 |1 |0 |0 |0 |1 ||1 |0 |0 |1 |0 |
0 |0 |1 |0 |0 |1 ||0 |0 |0 |1 |0 |
1 |0 |1 |0 |0 |1 ||1 |0 |0 |1 |0 |
0 |1 |1 |0 |0 |1 ||0 |0 |0 |1 |0 |
1 |1 |1 |0 |0 |1 ||1 |0 |0 |1 |0 |
0 |0 |0 |1 |0 |1 ||0 |0 |1 |0 |0 |
1 |0 |0 |1 |0 |1 ||1 |0 |1 |0 |0 |
0 |1 |0 |1 |0 |1 ||0 |0 |1 |0 |0 |
1 |1 |0 |1 |0 |1 ||1 |0 |1 |0 |0 |
0 |0 |1 |1 |0 |1 ||0 |0 |1 |0 |0 |
1 |0 |1 |1 |0 |1 ||1 |0 |1 |0 |0 |
0 |1 |1 |1 |0 |1 ||0 |0 |1 |0 |0 |
1 |1 |1 |1 |0 |1 ||1 |0 |1 |0 |0 |
0 |0 |0 |0 |1 |1 ||0 |0 |0 |1 |1 |
1 |0 |0 |0 |1 |1 ||1 |0 |0 |1 |1 |
0 |1 |0 |0 |1 |1 ||0 |0 |0 |1 |1 |
1 |1 |0 |0 |1 |1 ||1 |0 |0 |1 |1 |
0 |0 |1 |0 |1 |1 ||0 |0 |0 |1 |1 |
1 |0 |1 |0 |1 |1 ||1 |0 |0 |1 |1 |
0 |1 |1 |0 |1 |1 ||0 |0 |0 |1 |1 |
1 |1 |1 |0 |1 |1 ||1 |0 |0 |1 |1 |
0 |0 |0 |1 |1 |1 ||0 |0 |0 |1 |1 |
1 |0 |0 |1 |1 |1 ||1 |0 |0 |1 |1 |
0 |1 |0 |1 |1 |1 ||0 |0 |0 |1 |1 |
1 |1 |0 |1 |1 |1 ||1 |0 |0 |1 |1 |
0 |0 |1 |1 |1 |1 ||0 |0 |0 |1 |1 |
1 |0 |1 |1 |1 |1 ||1 |0 |0 |1 |1 |
0 |1 |1 |1 |1 |1 ||0 |0 |0 |1 |1 |
1 |1 |1 |1 |1 |1 ||1 |0 |0 |1 |1 |
Berechngszeit: 0.004221200942993164
```

3.5.2 mit Optimierung

```
Tabelle fuer L1
Q1|L
0 |0
1 |1

Tabelle fuer L2
Q1|L
0 |1
1 |1

Tabelle fuer L3
Q4|Q5|L
0 |0 |0
1 |0 |1
0 |1 |0
1 |1 |1

Tabelle fuer L4
Q4|Q5|L
0 |0 |1
1 |0 |1
0 |1 |1
1 |1 |0

Tabelle fuer L5
Q5|L
0 |0
1 |1

Berechnungszeit: 0.001928091049194336
```

4 Quellcode

4.1 ohneOptimierung.py

```
f = open(sys.argv[1])
x, y = map(int, f.readline().split())

# Eine Zeile wird als Ganzzahl gespeichert, jedes Bit repraesentiert ein Feld
assign_bits = lambda n, i, b: n | (1 << i) | (1 << i+1) if b else n & ~(1 << i) & ~(1 << i+1)
get_bit = lambda n, i: (n >> i) & 1

t = time.time()

# Array mit den Indizes der Q-Variablen
z = [i for i, e in enumerate(f.readline().split()) if e.startswith("Q")]
# Erstellen aller moeglichen Kombinationen der Taschenlampen
s = []
for i in range(2**len(z)):
    e = 0
    for j in range(len(z)):
        e |= ((i >> j) & 1) << z[j]
    s.append(e)

i, j = 0, 2
while j < y:
    while i < x:
        c = f.read(1)
        while c == "␣" or c == "\n":
            c = f.read(1)
        # Da das Licht nicht ohne Baustein wandern kann, wird das Feld dunkel
        if c == 'X':
            s = [e & ~(1 << i) for e in s]
            i += 1
            continue
        # Weisser Baustein
        elif c == 'W':
            s[:] = map(lambda e: assign_bits(e, i, not (get_bit(e, i) and get_bit(e, i+1))), s)
        # Roter Baustein
        elif c == 'r':
```

```
            s[:] = map(lambda e: assign_bits(e, i, not get_bit(e, i+1)), s)
        elif c == 'R':
            s[:] = map(lambda e: assign_bits(e, i, not get_bit(e, i)), s)
        # Beim blauen Baustein veraendert sich nicht, das Licht wird weitergegeben
        f.read(3)
        i += 2
    i = 0
    j += 1

f.readline()
# Array mit den Indizes der L-Variablen
k = [i for i, e in enumerate(f.readline().split()) if e.startswith("L")]
# Ausgabe der Wahrheitstabelle
print("".join([f"Q{i+1}|" for i in range(len(z))]) + "|" + "".join([f"L{i+1}|" for i in
    range(len(k))]))
for index, e in enumerate(s):
    print("".join([f"{get_bit(index, i)} |" for i in range(len(z))]) + "|" + "".join([f"{
    get_bit(e, k[i])} |" for i in range(len(k))]))

print(f"Berechngszeit: {time.time() - t}")
```

4.2 mitOptimierung.py

4.2.1 Datenstrukturen

```
class Node:
    def __init__(self):
        self.number = 0
        self.cache = None
        self.children = set()

class TruthTable:
    def __init__(self, literals, arr):
        self.arr = arr
        self.literals = literals # Q1, Q2, ..., Qn

    # Hilfsmethode
    def calculate_index(self, s):
        index = 0
        for i,v in enumerate(self.literals):
            if s & (1 << (v - 1)):
                index += 2 ** i
        return index

    # Gibt einen Bestimmenten Wert aus der Wahrheitstabelle zurueck
    def get(self, s):
        return self.arr[self.calculate_index(s)]

    # Fuegt zwei Wahrheitstabellen zusammen
    def merge(self, other, func):
        new_literals = sorted(list(set(self.literals + other.literals)))
        new_arr = [False] * (2 ** len(new_literals))
        for i in range(len(new_arr)):
            s = 0
            for j in range(len(new_literals)):
                s |= ((i >> j) & 1) << (new_literals[j] - 1)
            new_arr[i] = func(self.get(s), other.get(s))
        return TruthTable(new_literals, new_arr)

    # Wendet eine Funktion auf alle Werte der Wahrheitstabelle an und gibt eine neue
    Wahrheitstabelle zurueck
    def modify(self, func):
        new_literals = self.literals.copy()
        new_arr = self.arr.copy()
        for i in range(len(self.arr)):
            new_arr[i] = func(new_arr[i])
        return TruthTable(new_literals, new_arr)

    # Gibt die Wahrheitstabelle als String zurueck
    def __str__(self) -> str:
```

```
        string = "".join([f"Q{i}|" for i in self.literals]) + "L" + "\n"
        for i in range(len(self.arr)):
            for j in range(len(self.literals)):
                string += str((i >> j) & 1) + "␣|"
            string += str(bin(self.arr[i])[2:]) + "\n"
        return string
```

4.2.2 Einlesen des Graphen

```
    # Modellieren des Graphen
with open(sys.argv[1]) as file:
    output = [] # Liste der L-Knoten
    x, y = map(int, file.readline().split())
    grid = [l.split() for l in file.read().splitlines()]
    # Speichert die Knoten in kombination der Koordinaten
    nodes = [[None] * x for _ in range(y)]
    for i in range(x):
        if grid[0][i].startswith("Q"):
            input_node = Node()
            input_node.number = int(grid[0][i][1:])
            nodes[0][i] = input_node
    for j in range(1, y):
        i = 0
        while i < x:
            if grid[j][i] == "X":
                i+=1
                continue
            else:
                # Der B Knoten speichert eine Referenz auf den Knoten, der sich ueber ihm
 befindet
                if grid[j][i] == "B":
                    nodes[j][i+1] = nodes[j-1][i+1]
                    nodes[j][i] = nodes[j-1][i]
                    i+=2
                    continue
                elif grid[j][i].startswith("L"):
                    nodes[j][i] = Node()
                    nodes[j][i].number = int(grid[j][i][1:])
                    nodes[j][i].children = {nodes[j-1][i]}
                    output.append(nodes[j][i])
                    i+=1
                    continue
                else:
                    not_node = Node()
                    # Ein NAND Gatter besteht aus einem NOT Knoten, der ein AND Gatter
    als Kind hat
                    if grid[j][i] == "W":
                        and_node = Node()
                        and_node.children = {nodes[j-1][i+1], nodes[j-1][i]}
                        not_node.children = {and_node}
                        nodes[j][i+1] = nodes[j][i] = not_node
                    elif grid[j][i] == "r":
                        not_node.children = {nodes[j-1][i+1]}
                    elif grid[j][i] == "R":
                        not_node.children = {nodes[j-1][i]}
                    nodes[j][i] = nodes[j][i+1] = not_node
                    i+=2
                    continue
```

4.2.3 Erstellen der Wahrheitstabellen

```
# Rekursive Funktion, die die Wahrheitstabelle eines Knotens berechnet
def calculate_truth_table(node) -> TruthTable:
    # Wenn die Wahrheitstabelle bereits berechnet wurde, wird sie aus dem Cache gelesen
    if node.cache != None:
        return node.cache
```

```
    # Basisfall, Wenn der Knoten ein Input Knoten ist, wird die Standartwahrheitstabelle
    zurueckgegeben
    if len(node.children) == 0:
        return TruthTable([node.number], [False, True])
    # Ist der Knoten ein NOT Knoten, wird die Wahrheitstabelle des Kindes negiert
    elif len(node.children) == 1:
        t = calculate_truth_table(list(node.children)[0])
        node.cache = t.modify(lambda e: not e)
    # Ist der Knoten ein AND Knoten, werden die Wahrheitstabellen der Kinder gemerged
    elif len(node.children) == 2:
        t_1 = calculate_truth_table(list(node.children)[0])
        t_2 = calculate_truth_table(list(node.children)[1])
        node.cache = t_1.merge(t_2, lambda a, b: a and b)
    return node.cache

# Berechnen der Wahrheitstabellen fuer alle L-Knoten
for node in output:
    print(f"Tabelle␣fuer␣L{node.number}")
    print(calculate_truth_table(list(node.children)[0]))
```