

Berechenbarkeit und Komplexitätstheorie

Wintersemester 2021/2022

Aufgabenblatt 10

Abgabe: –

Ankündigung: Dieses Übungsblatt wird nicht bewertet. Dennoch findet die Übung am 8. Februar statt um die Aufgaben zu besprechen.

Definition(en)

- FÄRBBARKEIT:
 - **gegeben:** Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \mathbb{N}$
 - **gefragt:** Gibt es eine Funktion $f: V \rightarrow \{1, \dots, k\}$, sodass für alle $v, u \in V$ (mit $vu \in E$) gilt: $f(v) \neq f(u)$?
(Anders ausgedrückt: Kann man die Knoten des Graphen so färben, dass keine benachbarten Knoten dieselbe Farbe haben?)
- EULERPFAD-Problem:
 - **gegeben:** Ein ungerichteter Graph $G = (V, E)$
 - **gefragt:** Gibt es einen Pfad, der jede *Kante* im Graphen genau einmal besucht?
- RUCKSACK-Problem:
 - **gegeben:** Eine Kapazität $K \in \mathbb{N}$ und Gewichte $g_1, \dots, g_k \in \mathbb{N}$
 - **gefragt:** Gibt es eine Teilmenge $R \subseteq \{1, \dots, k\}$ mit $\sum_{i \in R} g_i = K$?
- s-t-ERREICHBARKEIT:
 - **gegeben:** Ein gerichteter Graph und zwei Knoten s und t
 - **gefragt:** Gibt es einen Pfad von s nach t ?
- FAKTORISIERUNG:
 - **gegeben:** Zwei natürliche Zahlen n und k
 - **gefragt:** Hat n einen Primfaktor, der kleiner als k ist?
- SUPERMARIOBROS:
 - **gegeben:** Ein beliebiges Level aus *Super Mario Bros.*
 - **gefragt:** Ist es möglich, das Ziel zu erreichen?
- SOKOBAN:
 - **gegeben:** Ein beliebiges *Sokoban*-Puzzle
 - **gefragt:** Ist es möglich, alle Kisten auf Zielfelder zu schieben?
- INFINITEMINESWEEPER:
 - **gegeben:** Ein beliebiges, teilweise gelöstes *Minesweeper*-Feld
 - **gefragt:** Ist es möglich die Lösung auf eine unendlich große Ebene auszuweiten?

Aufgabe 10.1 (0 Punkte)

- (i) Welche der folgenden Entscheidungsprobleme liegen in P?
 - (ii) Welche liegen in NP? (Ohne, dass ein deterministischer Polynomzeitalgorithmus bekannt ist.)
 - (iii) Für welche der Probleme ist noch nicht einmal ein nichtdeterministischer Polynomzeitalgorithmus bekannt?
- EULERPFAD-Problem
 - RUCKSACK-PROBLEM
 - s - t -ERREICHBARKEIT
 - FAKTORISIERUNG
 - k -FÄRBBARKEIT
 - SUPERMARIOBROS
 - SOKOBAN
 - INFINITEMINESWEEPER

Lösung

- (i) s - t -ERREICHBARKEIT, EULERPFADPROBLEM
- (ii) k -FÄRBBARKEIT, RUCKSACK-Problem, FAKTORISIERUNG, SUPERMARIOBROS
- (iii) SOKOBAN, INFINITEMINESWEEPER

Etwas mehr Informationen:

- EULERPFAD-Problem: Ein Graph hat einen Eulerpfad, gdw. er zusammenhängend ist und jeder Knoten (bis auf maximal zwei) eine gerade Valenz hat. Zusammenhang kann man mit Tiefensuche testen (Laufzeit linear in $\mathcal{O}(|V|+|E|)$) und Valenzen zählen ist möglich in $\mathcal{O}(|V|^2)$ ($|V|$ Knoten mit jeweils maximal $|V|$ Nachbarn). Das Problem liegt also (im Gegensatz zum HAMILTONKREIS-Problem) in P.
- s - t -ERREICHBARKEIT kann man sogar nichtdeterministisch in logarithmischem Platz (NL) lösen. Auch hier ist wieder ein Guess-and-Check-Algorithmus das Mittel der Wahl. Die einzigen Informationen, die gespeichert werden müssen sind der Knoten, der als nächstes betreten werden soll und die Anzahl der besuchten Knoten. Der Algorithmus endet, wenn Knoten t erreicht wird ("akzeptieren") oder der Pfad länger als n Knoten ist ("ablehnen"). Wenn man binär (oder zumindest nicht unär) zählt, dann kann man eine Zahl n in Platz $\mathcal{O}(\log n)$ aufschreiben.

- **FAKTORISIERUNG** ist in **NP**. Ein Guess-and-Check-Algorithmus (Faktor raten und überprüfen ob der Faktor prim ist) zeigt die Zugehörigkeit. Weiter ist das Problem auch in **co-NP** (Menge aller Probleme, deren *Komplement* in **NP** liegt). Hier kann man auch einen Guess-and-Check-Algorithmus angeben (“rate” die Primfaktorzerlegung und überprüfe, ob jeder Faktor größer als k ist) um das Problem zu lösen. Ob das Problem auch **NP**-schwer ist, ist noch unbekannt. Die landläufige Vermutung ist, dass das Problem *nicht* **NP**-schwer ist. Wäre **FAKTORISIERUNG** **NP**-vollständig, dann würde $\mathbf{NP} = \mathbf{co-NP}$ folgen. ($\mathbf{NP} \stackrel{?}{=} \mathbf{co-NP}$ ist ebenfalls eine “große” Frage der theoretischen Informatik.)
- Das **RUCKSACK**-Problem ist sogar **NP**-vollständig. Die Zugehörigkeit zu **NP** zeigt sich einfach über einen Guess-and-Check-Algorithmus. Die **NP**-Schwere kann man über eine Reduktion von 3-SAT zeigen. Sei also Φ eine aussagenlogische Formel in konjunktiver Normalform. O. B. d. A. können wir annehmen, dass jede Klausel *genau* drei Literale beinhaltet. Φ ist also von der Form $(z_{11} \wedge z_{12} \wedge z_{13}) \vee \dots \vee (z_{m1} \wedge z_{m2} \wedge z_{m3})$ wobei alle $z_{ij} \in \{x_1, \dots, x_n, \neg x_1, \dots, \neg x_n\}$ sind. Wir konstruieren aus dieser Formel eine Instanz des **RUCKSACK**-Problems wie folgt:
 - Sei K eine Dezimalzahl, die mit m Vieren anfängt und mit n Einsen endet, also $K := \underbrace{4 \dots 4}_m \underbrace{1 \dots 1}_n$ (bzw. $K = 4 \cdot 10^m \cdot \sum_{i=0}^{m-1} 10^i + \sum_{i=0}^{n-1} 10^i$).
 - Seien die Gewichte $\{v_1, \dots, v_n, v'_1, \dots, v'_n, c_1, \dots, c_m, d_1, \dots, d_m\}$; wobei die einzelnen Zahlen wie folgt definiert sind:
 - v_i ist eine $m+n$ -stellige Dezimalzahl, die an Stelle k (von links) eine 1 enthält, wenn das Literal x_i in Klausel C_k einmal vorkommt. (Analog steht an Stelle k eine 2 oder 3, wenn die Variable zwei- bzw. dreimal in der Klausel vorkommt.) Weiter beinhaltet die Zahl eine 1 an Position $m+i$ (d. h. im *hinteren* Ziffernblock an Stelle i). Alle anderen Stellen sind 0.
 - v'_i ist analog definiert, mit dem Unterschied, dass die Ziffern im linken Block dem Literal $\neg x_i$ entsprechen.
 - c_i ist eine $m+n$ -stellige Dezimalzahl, die an Stelle i eine 1 enthält. Alle anderen Ziffern sind 0.
 - d_i ist eine $m+n$ -stellige Dezimalzahl, die analog an Stelle i eine 2 enthält. Alle anderen Ziffern sind 0.

Wir müssen nun zeigen, dass Φ genau dann eine erfüllende Belegung hat, wenn es eine Teilmenge der oben definierten Zahlen gibt, die sich genau zu K aufsummiert.

$\Rightarrow$: Sei eine erfüllende Belegung für Φ gegeben. Wir wählen von den oben definierten Zahlen alle v_k für die gilt, dass x_k mit WAHR belegt ist und alle v'_k für die gilt, dass x_k mit FALSCH belegt ist. Da in einer validen Belegung keine Variable mit beiden Wahrheitswerten gleichzeitig belegt sein kann, gibt es auch kein k , so dass sowohl v_k als auch v'_k gewählt werden. Daraus folgt, dass die letzten n Ziffern unserer Summe sich genau zu $\underbrace{1 \dots 1}_n$ aufsummieren. Die vorderen m Ziffern sind jeweils höchstens 3,

da Φ in 3-KNF gegeben ist und damit ein einzelnes Literal nicht häufiger als dreimal in einer Klausel vorkommen kann. Da die Belegung Φ erfüllt, gilt ebenfalls, dass jede einzelne Klausel erfüllt ist und darum in jeder Klausel mindestens ein Literal WAHR ist. Das heißt, keine der vorderen m Ziffern der Summe ist 0. Man kann jetzt jede dieser einzelnen Ziffern zu genau 4 aufsummieren, in dem man passend c_k bzw. d_k

wählt. Zusammengefasst hat man aus der erfüllenden Belegung eine Teilmenge R gewählt, die sich genau zu $K(= \underbrace{4 \dots 4}_m \underbrace{1 \dots 1}_n)$ aufsummiert.

⇐: Sei eine Teilmenge R gegeben, die aus den oben definierten Zahlen besteht und sich zu $\underbrace{4 \dots 4}_m \underbrace{1 \dots 1}_n$ aufsummiert. Um die “hinteren” n Einsen zu erreichen muss für jedes $1 \leq k \leq n$ genau eine der Zahlen v_k oder v'_k gewählt sein. Keine andere Kombination liefert den $\underbrace{1 \dots 1}_n$ -Teil am Ende. Weiter liefert das Aufsummieren der c_k und d_k höchstens 3, also muss für jedes $1 \leq j \leq m$ zusätzlich ein passendes v_k oder v'_k gewählt werden. Die Wahl der v_k bzw. v'_k beschreibt somit eine erfüllende Belegung für Φ .

Beispiel: Sei $\Phi = (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_4 \vee x_4)$. Damit ist $n = 4$ und $m = 3$ und wir wählen als Kapazität $K = 4441111$. Weiter wählen wir die anderen Zahlen wie folgt:

$v_1 = 101\,1000$	$v'_1 = 010\,1000$
$v_2 = 000\,0100$	$v'_2 = 110\,0100$
$v_3 = 100\,0010$	$v'_3 = 000\,0010$
$v_4 = 012\,0001$	$v'_4 = 000\,0001$
$c_1 = 100\,0000$	$d_1 = 200\,0000$
$c_2 = 010\,0000$	$d_2 = 020\,0000$
$c_3 = 001\,0000$	$d_3 = 002\,0000$

Eine erfüllende Belegung für Φ ist $x_1 = x_4 = \text{WAHR}$ und $x_2 = x_3 = \text{FALSCH}$. Wenn wir die v_i bzw. v'_i entsprechend der Belegung wählen, ergibt sich:

$$\begin{aligned}
 V &:= v_1 + v_4 + v'_2 + v'_3 \\
 &= 101\,1000 \\
 &\quad + 012\,0001 \\
 &\quad + 110\,0100 \\
 &\quad + 000\,0010 \\
 &= 223\,1111
 \end{aligned}$$

Wenn wir nun noch zusätzlich d_1 , d_2 und c_3 zur Summe hinzufügen, ergibt sich:

$$\begin{aligned}
 V + d_1 + d_2 + c_3 \\
 &= 223\,1111 \\
 &\quad + 200\,0000 \\
 &\quad + 020\,0000 \\
 &\quad + 001\,0000 \\
 &= 444\,1111 = K
 \end{aligned}$$

Andererseits sieht man, wenn man eine nicht-erfüllende Belegung, wie bpsw. $x_1 = x_2 = x_3 = x_4 = \text{FALSCH}$ wählt, dass sich als Summe der v_i und v'_i 1201111 ergibt, welche man nicht mit Wahl von c_i und d_i zu K aufsummieren kann, dann die 0 an dritter Position höchstens zu 3 aufsummiert wird.

- k -FÄRBBARKEIT liegt in NP, wie auf dem letzten Übungsblatt gezeigt. Das Problem ist auch NP-schwer, sogar mit $k = 3$. Dies kann man durch eine Reduktion von 3-SAT

zeigen (vgl. [Richard M. Karp: “Reducibility Among Combinatorial Problems”]). Grob erklärt: Aus einer gegebenen Formel (in 3KNF) erzeugt man einen Graphen, der genau dann 3-färbbar ist, wenn die Formel erfüllbar ist. Abbildung 1 zeigt wie man den Graphen aufbaut.

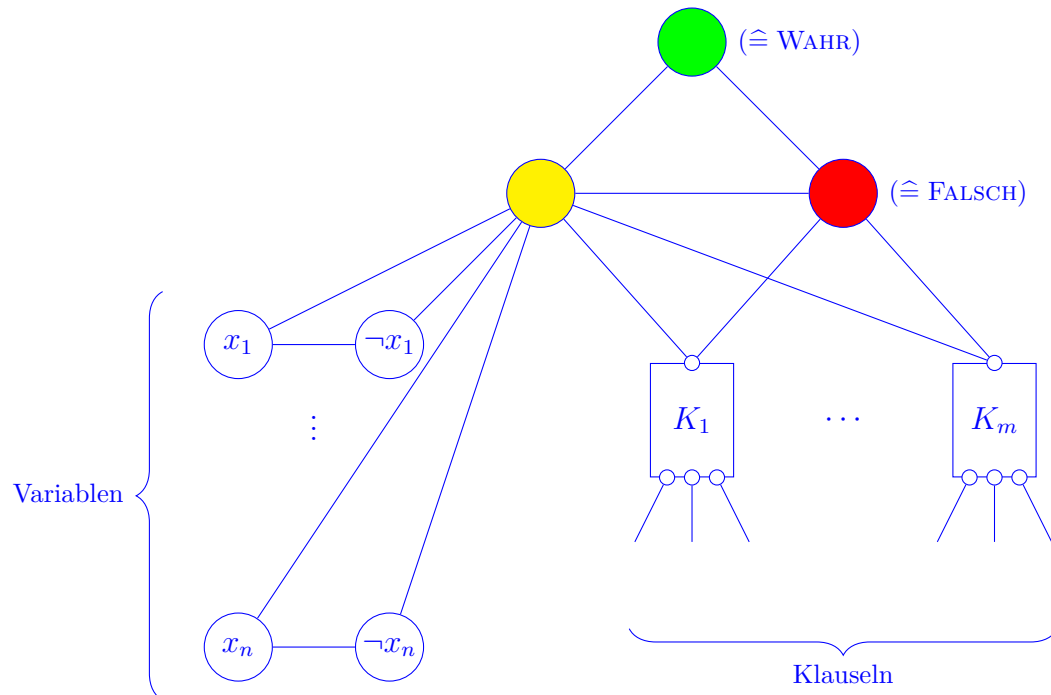


Abbildung 1: Skizze eines zu einer aussagenlogischen Formel gehörenden Graphen

Der Graph besteht aus zwei Knoten für jede Variable der Formel (wobei einer der Knoten der negierten Variable entspricht), einem Untergraphen (siehe Abbildung 2) für jede Klausel der Formel und einem Dreieck, welches “vorgefärbt” ist. Jeden Variablen-Knoten verbindet man mit dem gelben Knoten des Dreiecks und zusätzlich mit dem zugehörigen, negierten Variablen-Knoten. Außerdem wird jeder Variablen-Knoten und jeder negierte Variablen-Knoten mit dem zugehörigen unteren Knoten (“Eingang”) des zugehörigen Klausel-Untergraphen verbunden (für eine Klausel $K_x = (x_i \wedge \neg x_j \wedge x_k)$ beinhaltet der Graph eine Kante vom Knoten x_i zum ersten Eingang in K_x , vom Knoten $\neg x_j$ zum zweiten Eingang und vom Knoten x_k zum dritten Eingang). Die oberen Knoten (“Ausgänge”) der Klausel-Untergraphen verbindet man jeweils mit dem gelben und dem roten Knoten des Dreiecks. Das Dreieck erzwingt zum einen, dass jede Variable und ihre jeweilige Negierung entweder grün oder rot gefärbt (also mit WAHR oder FALSCH belegt) wird und zum anderen, dass jeder Klausel-Untergraph am Ausgang grün gefärbt werden muss.

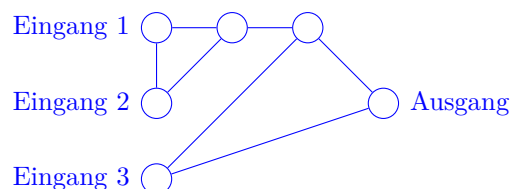


Abbildung 2: Klausel-Untergraph

Die Klausel-Untergraphen sind so aufgebaut, dass der Ausgang nur dann grün gefärbt werden kann, wenn mindestens einer der mit den drei Eingängen verbundenen Variablen-Knoten grün gefärbt ist. Das bedeutet, dass der Klausel-Untergraph

nur dann am Ausgangsknoten grün gefärbt werden kann, wenn die Klausel erfüllt werden kann. Und da der Graph nur dann 3-färbbar ist, wenn alle Ausgänge der Klausel-Untergraphen grün gefärbt werden können, gilt die Reduktionseigenschaft.

Interessant ist, dass k -FÄRBBARKEIT auch für festes k nicht zwingend in P liegt. Ein Bruteforce-Algorithmus (anders als bei bspw. k -CLIQUE) hat eine Laufzeit in $\mathcal{O}(k^n)$. Je nach Eigenschaften des Graphen kann man das Problem aber dennoch “einfach” lösen. Es sind z.B. alle bipartiten Graphen 2-färbbar und alle planaren Graphen sind 4-färbbar (vgl. [Kenneth Appel und Wolfgang Haken: “Every Planar Map is Four Colorable”]).

- SUPERMARIOBROS ist mit den üblichen Spielbedingungen (Zeitlimit, komprimierte Levelkodierung, ...) NP-vollständig. Dies kann man bspw. mit einer Reduktion von 3-SAT zeigen (vgl. [Greg Aloupis et al.: “Classic Nintendo Games are (Computationally) Hard”, 2012]). Die Reduktion funktioniert ähnlich wie bei der Reduktion von 3-SAT zu 3-FÄRBBARKEIT: Man konstruiert Variablen- und Klausel-Gadgets als SMB-Teillevel. Diese setzt man dann so zusammen, dass man aus einer gegebenen Formel in 3-KNF ein SMB-Level erhält. (Für jede Variable ein Variablen-Gadget und für jede Klausel ein Klausel-Gadget, die man passend verbindet.) Im Fall von SUPERMARIOBROS (und auch bei den meisten anderen solcher Reduktionen) muss noch ein Start- und ein Ziel-Gadget konstruiert werden. Prinzipiell müssen auch Gadgets konstruiert werden, die das Verbinden der einzelnen Gadgets ermöglichen – im Fall von 3-FÄRBBARKEIT waren das die Kanten des Graphen.

Das Start-Gadget (Abbildung 3a) ist ein Levelabschnitt, der Marios initiale Position und einen Fragezeichenblock mit einem Pilz beinhaltet. Das Ziel-Gadget (Abbildung 3b) besteht aus einem Gang, bei dem ein Block zerstörbar ist, wenn Mario groß ist und aus dem eigentlich Ziel – der Flagge. Das Variablen-Gadget (Abbildung 3c) besteht aus zwei Eingängen (oben) und zwei Ausgängen (unten). Das Gadget ist so aufgebaut, dass wenn Mario von der oberen Kante runterspringt, er nicht mehr hoch springen kann. Die beiden Ausgänge entsprechen dem Belegen der Variable mit WAHR oder FALSCH und sind mit den Klausel-Gadgets verbunden, welche das Literal enthalten. Das Klausel-Gadget (Abbildung 3d) besteht aus zwei Teilen und funktioniert prinzipiell so, dass Mario das Gadget durch einen der drei Eingänge oben (diese entsprechen den Literalen) betritt und auf den Koopa hüpfte. Dann kann er den Panzer des Koopas nach unten schießen und damit die zerstörbaren Blöcke zerstören. Danach kann Mario das Gadget wieder nach oben verlassen. Wichtig ist hier, dass Mario das Gadget wieder oben verlassen muss: Wenn er nach unten springt, dann kann er nicht nach oben zu einem der anderen Koopas springen, da der Abstand zu hoch ist. Weiter kann Mario nicht den selben Weg wie der Koopa-Panzer gehen, da der Gang nur ein Block hoch ist, Mario aber größer. Prinzipiell könnte Mario sich von einem Koopa treffen lassen (oder den Pilz am Anfang ignorieren) und dann durch den Gang gehen; wenn Mario es so bis zum Ziel-Gadget schafft, dann kann er allerdings dort die Flagge nicht erreichen. Das Ziel-Gadget erzwingt, dass Mario groß sein muss um das Level zu schaffen. Und da im Start-Gadget der einzige Pilz im ganzen Level ist, darf Mario sich von keinem der Koopas treffen lassen. Der zweite Abschnitt des Gadgets ist der untere (drei Blöcke hohe) Gang, der durch zerstörbare Blöcke zugesperrt ist. Dieser kann nur durchquert werden, wenn diese Blöcke zerstört wurden sind; also wenn Mario vorher das Gadget durch einen der oberen Eingänge betreten und einen Koopa-Panzer runtergeschossen hat. Da im Gegensatz zu 3-FÄRBBARKEIT die “Reihenfolge” wichtig ist (Mario durchläuft das Level Gadget für Gadget und ist nicht überall gleichzeitig), muss man

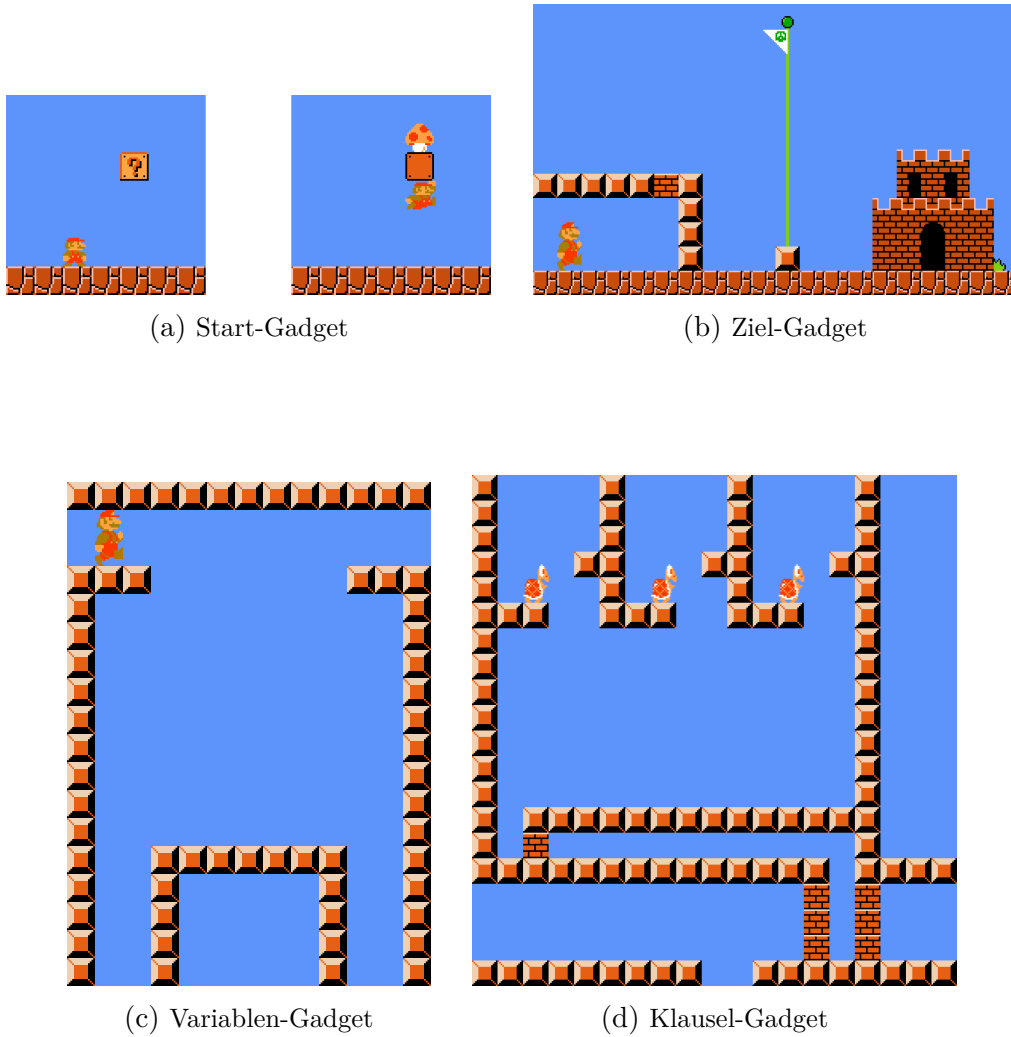


Abbildung 3: Gadgets

die Gadgets passend verbinden. Aus einer aussagenlogischen Formel in 3-KNF (mit Variablen x_1 bis x_n und Klauseln K_1 bis K_m) konstruiert man ein SMB-Level folgendermaßen: Das Level startet mit dem Start-Gadget. Dieses führt in ein Variablen-Gadget welches Variable x_1 entspricht. Der linke Ausgang wird mit der ersten Klausel-Gadget K_i verbunden, welches die Variable enthält und der zweite Ausgang wird mit dem ersten Klausel-Gadget K_j verbunden, welches $\neg x_1$ enthält. Klausel-Gadget K_i wird dann mit dem nächsten Klausel-Gadget verbunden, welches x_1 enthält; K_j mit dem nächsten Klausel-Gadget, welches $\neg x_1$ enthält. Die jeweils letzten Klausel-Gadgets die x_1 bzw. $\neg x_1$ enthalten werden danach mit Variablen-Gadget x_2 verbunden. Dieses dann wieder mit den entsprechenden Klausel-Gadgets usw. Die oberen Ein-/Ausgänge der jeweils letzten Klausel-Gadgets der letzten Variable x_n werden dann mit dem unteren Gang des Klausel-Gadgets K_m verbunden. Die Gänge werden dann untereinander verbunden (also K_m nach K_{m-1} usw. bis K_2 nach K_1). Der untere Gang in Klausel-Gadget K_1 wird dann zum Schluss noch mit dem Ziel-Gadget verbunden. Damit Mario in dem Level das Ziel erreichen kann, muss er also jedes Klausel-Gadget mindestens einmal betreten. Er betritt ein Klausel-Gadget genau dann, wenn er sich für die “richtige” Belegung der Variable entschieden hat. Mario kann also genau dann das Ziel erreichen, wenn die Formel erfüllbar ist.

Ein neueres Paper ([Erik D. Demaine et al.: “Super Mario Bros. Is Harder/Easier than We Thought”, 2016]) analysiert SUPERMARIOBROS im Speziellen noch-

mal und kommt zu Ergebnissen abhängig der Verallgemeinerung des Spiels: In P wenn der Bildschirm “beschränkt” ist und die Level unkomprimiert als Eingabe vorliegen (SMB-STANDARD); (schwach) NP-vollständig wenn der Bildschirm “unbeschränkt” ist und die Level komprimiert als Eingabe vorliegen (SMB-RLE); PSPACE-vollständig wenn zusätzlich zu den vorigen Eigenschaften es Mario noch erlaubt ist, wieder nach links zu gehen (SMB-GENERAL). Für SMB-STANDARD kann man einen Polynomzeitalgorithmus angeben; für SMB-RLE kann man eine Reduktion vom Rucksackproblem angeben und für SMD-GENERAL kann man eine Reduktion vom sogenannten “open-close door”-Framework angeben.

- SOKOBAN ist PSPACE-vollständig. Dies kann man durch eine Reduktion vom *Erfüllbarkeitsproblem für quantifizierte boolesche Formeln* (QBF) zeigen. Eine quantifizierte boolesche Formel ist eine Formel der Form $\exists x_1 \forall x_2 \dots \forall x_k \phi(x_1, x_2, \dots, x_k)$. Wegen dem Allquantor kann man eine vorgegebene Lösung (Belegung der Existenzquantor-Variablen) *nicht* auf naive Art in Polynomzeit testen. QBF ist analog zu SAT das “kanonische” PSPACE-vollständige Problem. (Genauer wurde eigentlich ein spezielles Framework (“Nondeterministic Constraint Logic” [vgl. Robert A. Hearn und Erik D. Demaine: “PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation”, 2005]) entwickelt, das für die Reduktion zu solchen Bewegungsplanungs-Problemen geeignet ist. Und die Komplexität dieses Frameworks wurde mit einer Reduktion von QBF gezeigt.)
- MINESWEEPER ist Turing-vollständig; vgl. [Richard Kaye: “Infinite versions of minesweeper are Turing complete”, 2007]). Die grundlegende Idee ist es, die Arbeitsweise einer beliebigen Turingmaschine durch eine Instanz des Minesweeper-Problems darzustellen. Die Startkonfiguration der TM entspricht dem vorgegebenen Muster des Minesweeper-Felds und die Konfigurationsübergänge werden durch Regeln simuliert, wie das vorgegebene Muster an bestimmten Stellen fortgesetzt werden kann. Die Reduktion ist so aufgebaut, dass die TM genau dann hält, wenn das Muster *nicht* unendlich fortgesetzt werden kann (darum co-RE-vollständig).

Randnotizen:

- $NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq NPSPACE \subseteq EXP$
- $NL \subsetneq PSPACE$ und $P \subsetneq EXP$, also müssen manche der Inklusionen oben *echt* sein
- $PSPACE = NPSPACE$ (Satz von Savitch). Intuition: Platz kann man wiederverwenden. Unter der Annahme, dass man Zeit wiederverwenden kann gilt auch $P = NP$. “Zeit wiederverwenden” könnte man bspw. durch Zeitreisen; die zugehörige Komplexitätsklasse nennt sich P_{CTC} und für diese gilt $P_{CTC} = PSPACE$ (vgl. [Scott Aaronson: “NP-complete Problems and Physical Reality”, 2005] oder [Todd A. Brun: “Computers with closed timelike curves can solve hard problems”, 2002]).

Aufgabe 10.2 (0 Punkte)

- Geben Sie eine (Mehrband-)Turingmaschine M an, die die Funktion $g: \mathbb{N} \rightarrow \mathbb{N}$ mit $g(n) = 11n$ berechnet. (*Hinweis:* Sie müssen nicht unbedingt das Binärsystem nutzen.)
- Geben Sie eine Funktion $f: \mathbb{N} \rightarrow \mathbb{N}$ an, so dass $time_M(w) \leq f(|w|)$. (Begründen Sie Ihre Antwort.)

Lösung

Eine Lösung ist es, wenn man als Ein- und Ausgabe Zahlen zur Basis 11 zulässt. Dann muss man nur eine Null ans Ende der Eingabe hängen. Bei dieser Lösung muss die TM ans Ende der Eingabe laufen, eine 0 schreiben und dann wieder an den Anfang der Eingabe laufen. Das heißt, sie braucht $|w|+1 + |w \cdot 0| = 2|w|+2$ Schritte. Wir setzen also $f(n) := 2n + 2$.

Eine andere Lösung ist es, wenn man das Unärsystem nutzt. Dann muss man zehn Kopien der Eingabe hinter die Eingabe schreiben. Zum kopieren nutzt man ein zweites Band. Man läuft über die Eingabe und kopiert jede gelesene 1 auf das zweite Band. Wenn man am Ende der Eingabe angekommen ist, überschreibt man jede 1 auf dem zweiten Band mit einer Markierung 1_2 und schreibt eine 1 auf das erste Band. Wenn man am Ende (bzw. Anfang) des zweiten Bands angekommen ist, dann läuft man wieder über das zweite Band und überschreibt jede Markierung mit 1_3 . Außerdem schreibt man eine 1 für jede gefundene Markierung auf das erste Band. Diese Schritte wiederholt man solange, bis man die Eingabe insgesamt zehn (weitere) Male auf das Eingabeband geschrieben hat. Dann läuft man auf dem ersten Band wieder an den Anfang des Wortes (und löscht “unterwegs” das zweite Band). Die Maschine braucht n Schritte um an das Ende des ersten Bandes zu kommen. Dann braucht sie jeweils weitere n Schritte für jede Kopie der Eingabe (Also um jeweils die Markierungen 1_2 bis 1_{11} zu verarbeiten). Am Ende braucht sie nochmal $11n$ Schritte um wieder an den Anfang des ersten Bandes zu kommen. Als Funktion ergibt sich also $f(n) := n + 10n + 11n = 22n$.

Eine weitere Lösung arbeitet im Dezimalsystem: Man kopiert die Zahl auf ein zweites Arbeitsband (um eine Stelle verschoben) und hängt an die Eingabe eine 0. Dann steht auf dem ersten Band $10n$ und auf dem zweiten Band n . Dann addiert man beide Bänder wie bei der schriftlichen Addition. Hier ergibt sich ebenfalls als Funktion $f(n) := 2n + 2$, da die TM ans Ende der Eingabe laufen muss, ein Zeichen hinzufügen muss, und dann wieder an den Anfang (der nun eins längeren) Eingabe laufen muss.

Im Binärsystem kann man ähnlich verfahren, denn $11n = 8n + 2n + n$. Also kann man drei Arbeitsbänder nutzen. Eines der Bänder enthält die Eingabe, ein zweites die Eingabe um eins nach links verschoben (mit einer zusätzlichen Null am Ende) und ein drittes die Eingabe um drei nach links verschoben (mit drei zusätzlichen Nullen am Ende). Dann kann man auch hier wie beim schriftlichen Addieren die Bänder zusammenrechnen. Als Laufzeit ergibt sich analog $f(n) := 2n + 6$.

Aufgabe 10.3 (0 Punkte)

Gegeben ist eine Turingmaschine $M = (S, E, A, \delta, z_0, \square, F)$ mit Zustandsmenge $S = \{z_0, z_1, z_2, z_e\}$, Eingabealphabet $E = \{0, 1\}$, Bandalphabet $A = \{0, 1, \square\}$, Blanksymbol $\square$, Startzustand z_0 , Endzustandsmenge $F = \{z_e\}$ und Zustandsüberföhrungsfunktion δ mit:

$\delta(z, w)$	0	1	$\square$
z_0	$(z_0, \square, R)$	$(z_1, 1, R)$	$(z_2, 1, N)$
z_1	$(z_1, 0, R)$	$(z_1, 1, R)$	$(z_2, 1, N)$
z_2	$(z_2, 0, L)$	$(z_2, 1, L)$	$(z_e, 1, N)$

- Geben Sie die Folge der Konfigurationen an, die M bei Eingabe 010 durchläuft.
- Welche Funktion $s : \{0, 1\}^* \rightarrow \{0, 1\}^*$ und welche Funktion $f : \mathbb{N} \rightarrow \mathbb{N}$ berechnet M ?

Lösung

Die Maschine löscht alle führenden Nullen und schreibt dann eine Eins ans Ende und an den Anfang des übrigen Wortes.

a) $z_0 010 \vdash \Box z_0 10 \vdash \Box 1 z_1 0 \vdash \Box 10 z_1 \Box \vdash \Box 10 z_2 1 \vdash \Box 1 z_2 01 \vdash \Box z_2 101 \vdash z_2 \Box 101 \vdash z_e 1101$

b) • $s : \{0, 1\}^* \rightarrow \{0, 1\}^*$, $w \mapsto 1 \cdot w' \cdot 1$ (mit $w = 0^* \cdot w'$).

Bzw. etwas "formaler": $w \mapsto \begin{cases} 1 \cdot w \cdot 1 & (w \neq 0 \cdot w') \\ s(w') & (w = 0 \cdot w') \end{cases}$

• $f : \mathbb{N} \rightarrow \mathbb{N}$, $n \mapsto (2n + 1) + 2^{\ell(2n+1)}$, wobei $\ell(n) =$ "Die Länge der Zahl im Binärsystem". (Es gilt $\ell(n) = \lfloor \log(n) \rfloor + 1$.)

Aufgabe 10.4 (0 Punkte)

Schreiben Sie ein LOOP- oder WHILE-Programm, das x^y berechnet. Vermeiden Sie die explizite Multiplikation.

Lösung

```

x0 := 1
LOOP y DO
  s := 0
  LOOP x DO
    s := s + x0
  END
  x0 := s
END
END

```

Aufgabe 10.5 (0 Punkte)

Sind die folgenden Aussagen wahr oder falsch? (Begründen Sie Ihre Antworten.)

- i) Jede nicht-entscheidbare Menge enthält eine entscheidbare Teilmenge.
- ii) Jede Teilmenge einer entscheidbaren Menge ist entscheidbar.
- iii) Aus A entscheidbar und $A \cap B$ entscheidbar folgt B entscheidbar.
- iv) Die Menge $\{(w, x) \in E^* \times \mathbb{N} : M_w(x) = x^2\}$ ist entscheidbar.
- v) Für $h_M(w) : E^* \rightarrow \{0, 1\}$ mit $(h_M(w) = 1 \equiv M(w) \text{ hält})$ gilt $A = \{h_M(w)\}$ ist entscheidbar.

Lösung

Sind die folgenden Aussagen wahr oder falsch? (Begründen Sie Ihre Antworten.)

- i) Ja, denn $\emptyset \subseteq X$ für alle X .
- ii) Nein. E^* ist entscheidbar, aber $H \subseteq E^*$ ist nicht entscheidbar.
- iii) Nein. Sei $A = \emptyset$, dann ist $A \cap B = \emptyset$ ebenfalls entscheidbar, auch für nicht-entscheidbares B .
- iv) Nein, wegen des Satzes von Rice.
- v) Ja, denn jede endliche Menge ist entscheidbar und $A \subsetneq \{0, 1\}$ ist endlich.

Aufgabe 10.6 (0 Punkte)

Sei h eine totale und surjektive Funktion, die als Bildbereich die *komplette* Menge aller einstelligen berechenbaren Zahlenfunktionen hat, also:

$$h: \mathbb{N} \rightarrow \{f: \mathbb{N} \rightarrow \mathbb{N} : f \text{ ist eine totale berechenbare Funktion}\}$$

Das heißt, für jedes $n \in \mathbb{N}$ ist $h(n)$ eine Funktion von $\mathbb{N}$ nach $\mathbb{N}$ und $h(n)(x)$ ist der Wert der Funktion $h(n)$ an Stelle x .

Zeigen Sie, dass die folgende Funktion nicht berechenbar ist:

$$g: \mathbb{N}^2 \rightarrow \mathbb{N}, \quad g(n, x) = h(n)(x)$$

(*Hinweis:* Konstruieren Sie aus g eine einstellige totale Funktion, welche im Bildbereich von h liegen würde und zeigen Sie mithilfe von Diagonalisierung (ähnlich zu Kapitel 5 in den Folien) einen Widerspruch auf.)

Lösung

Angenommen g sei berechenbar. Dann wäre $g': \mathbb{N} \rightarrow \mathbb{N}$ mit $g'(x) = g(x, x) + 1 (= h(x)(x) + 1)$ auch berechenbar. Weiter gilt: g' ist total (da sowohl h als auch $h(x)$ total sind). Daraus folgt, dass g' im Bildbereich von h liegt – es gibt also ein $z \in \mathbb{N}$ mit $h(z) = g'$. Insbesondere gilt dann $g'(z) = h(z)(z) = g(z, z)$. Nach Definition von g' gilt allerdings $g'(z) = g(z, z) + 1$. $\perp$

Aufgabe 10.7 (0 Punkte)

Zeigen Sie die folgenden Aussagen:

- a) $\{(n, k) \in \mathbb{N}^2 : n \text{ hat einen Primfaktor kleiner } k\}$ ist entscheidbar.
- b) $\{(x, y, z) \in \mathbb{N}^3 : (\exists n \in \mathbb{N}) \ x^n + y^n = z^n\}$ ist semi-entscheidbar.

Lösung

- a) $\{(n, k) \in \mathbb{N}^2 : n \text{ hat einen Primfaktor kleiner } k\}$ kann man mit folgendem Algorithmus entscheiden:

```
1:  $i \leftarrow 2$ 
2: while  $i < k$  do
3:   if  $n \equiv 0 \pmod{i}$  then                                 $\triangleright i$  teilt  $n$ 
4:     return TRUE
5:   fi
6:    $i \leftarrow i + 1$ 
7: od
8: return FALSE
```

Der Algorithmus testet alle potentiellen Teiler und gibt TRUE zurück, wenn ein Teiler gefunden wird. Wenn kein Teiler (in $\{2, \dots, k-1\}$) gefunden wird, dann besitzt die Zahl keinen Primfaktor kleiner k und der Algorithmus liefert FALSE zurück. Prinzipiell könnte man noch testen, ob das gefundene i , welches n teilt eine Primzahl ist oder nicht. Falls i eine Primzahl ist, dann ist i ein Primfaktor. Falls i keine Primzahl ist, dann hat i selbst allerdings eine Primfaktorzerlegung, wobei jeder Primfaktor kleiner als i selbst ist und jeder der Primfaktoren von i auch n teilt (folgt alles mehr oder weniger aus dem *Fundamentalsatz der Arithmetik*).

b) $\{(x, y, z) \in \mathbb{N}^3: \exists n \in \mathbb{N} \ x^n + y^n = z^n\}$ lässt sich mit folgendem Algorithmus “semi-entscheiden”:

```
1:  $i \leftarrow 1$ 
2: while  $x^i + y^i \neq z^i$  do
3:    $i \leftarrow i + 1$ 
4: od
5: return TRUE
```

Der Algorithmus testet für alle natürlichen Zahlen, ob die Gleichung erfüllt ist. Wenn er eine Zahl findet, für die die Gleichung gilt, liefert er TRUE zurück, sonst rechnet er weiter. Da Addition und Potenzierung WHILE-berechenbar sind, ist auch die Schleifenbedingung WHILE-berechenbar und damit ist der Algorithmus ein Semi-Entscheidungsverfahren. Prinzipiell ist die Menge sogar entscheidbar, da man nur $n = 1$ und $n = 2$ testen muss. Für alle $n > 2$ ist die Gleichung immer falsch (*Großer Fermat'scher Satz*).