1 Entscheiden, Konstruieren und Optimieren

Komplexitätsklassen enthalten formale Sprachen über einem Alphabet, die als Entscheidungsprobleme (gehört das Eingabewort zur Sprache?) aufgefasst werden können. Praktische Rechenprobleme hingegen haben oft den Charakter von Konstruktions- oder Optimierungsproblemen. In diesem Abschnitt wollen wir zeigen, dass die Frage der polynomiellen Lösbarkeit aller dieser Problemtypen weitgehend anhand von geeignet definierten formalen Sprachen diskutiert werden kann. Wir erhalten dadurch eine Berechtigung uns in der Komplexitätstheorie auf das Studium von Komplexitätsklassen zurückzuziehen.

Ein wichtiges Werkzeug, um verschiedene Probleme zueinander in Beziehung zu setzen, ist die Problemreduktion. Aus diesem Grund werden wir neben der uns bereits bekannten polynomiellen Reduktion zwei weitere Reduktionstypen einführen: die *Cook-Reduktion* und die *Levin-Reduktion* (jeweils benannt nach ihrem Erfinder).

Dieser Abschnitt ist aufgebaut wie folgt. Zunächst schildern wir das *Problem des Handelsreisenden*, auch *Travelling Salesman Problem* oder kurz *TSP* genannt. Alle wesentlichen Ideen tauchen bereits in diesem Beispiel auf. Danach bringen wir die im TSP-Beispiel genannten Problemtypen in eine allgemeinere begrifflich gestraffte Form und führen die drei genannten Reduktionstypen formal ein.

1.1 Das Problem des Handelsreisenden

TSP liest sich als Optimierungsproblem wie folgt. Gegeben sei eine $(n \times n)$-Distanzmatrix $D = (d_{i,j})_{0 \leq i,j \leq n-1}$, wobei $d_{i,j}$ die Distanz von einer Stadt i zu einer Stadt j angibt. Gesucht ist die kürzeste Rundreise durch alle n Städte, also eine Permutation σ von $1, \ldots, n$, die die Kostenfunktion

$$|\sigma| := \sum_{i=0}^{n-1} d_{\sigma(i),\sigma(i+1 \bmod n)}$$

minimiert. σ_* bezeichne im Folgenden eine Permutation, die eine kürzeste Rundreise repräsentiert.
Eine Abschwächung des TSP-Optimierungsproblems ist das TSP-Wertoptimierungsproblem, bei welchem nur nach der Länge einer kürzesten Rundreise gefragt ist.

TSP als Konstruktionsproblem hat als weiteren Eingabeparameter neben der Distanzmatrix D eine Kostenschranke K. Gesucht ist eine Rundreise, deren Kosten K nicht überschreiten bzw. die Meldung, dass eine solche Rundreise nicht existiert.
Eine Abschwächung des TSP-Konstruktionsproblems ist das TSP-Entscheidungsproblem, bei welchem nur gefragt ist, ob eine Rundreise mit Kosten höchstens K existiert. Rundreisen (Permutationen), die die Kostenschranke K respektieren, nennen wir im Folgenden *zulässig*.

Im folgenden skizzieren wir den Beweis, daß alle vier Problemvarianten polynomiell verknüpft sind, d.h., ein deterministischer Polynomialzeitalgorithmus für eine Variante kann in einen nicht wesentlich aufwendigeren Algorithmus für alle anderen Varianten transformiert werden.

Die in Abbildung 1 dargestellte Hierarchie ist offensichtlich. Hierbei bedeutet $A \to B$, daß eine polynomiell zeitbeschränkte DTM M', die Problem (oder Problemvariante) B löst, in eine polynomiell zeitbeschränkte DTM M für Problem (oder Problemvariante) A transformiert werden kann. Die einfache Argumentation zu den Reduktionen 1,2,3,4 überlassen wir dem Leser und der Leserin. Auf eine griffige Formel gebracht haben wir bis jetzt gezeigt: Entscheiden ist nicht schwerer als Konstruieren, und Konstruieren ist nicht schwerer als Optimieren. Überraschender ist, daß die Problemreduktionen in der obigen Hierarchie auch in der umgekehrten Richtung möglich sind, wie es in Abbildung 2 zu sehen ist. Griffig formuliert: Optimieren ist nicht schwerer als Konstruieren, und Konstruieren ist nicht schwerer als Entscheiden.

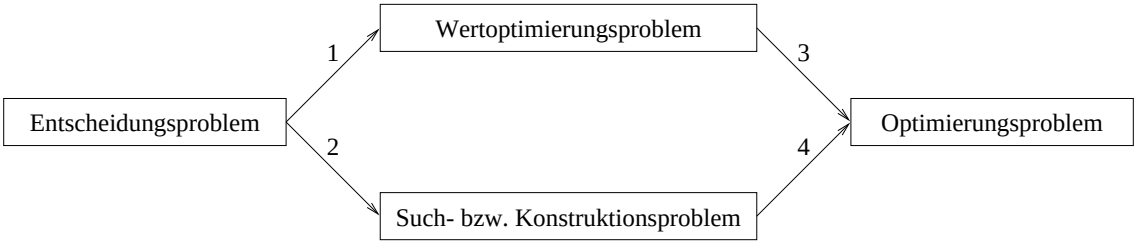

Abbildung 1: Reduktion des Entscheidungsproblems auf das Optimierungsproblem.

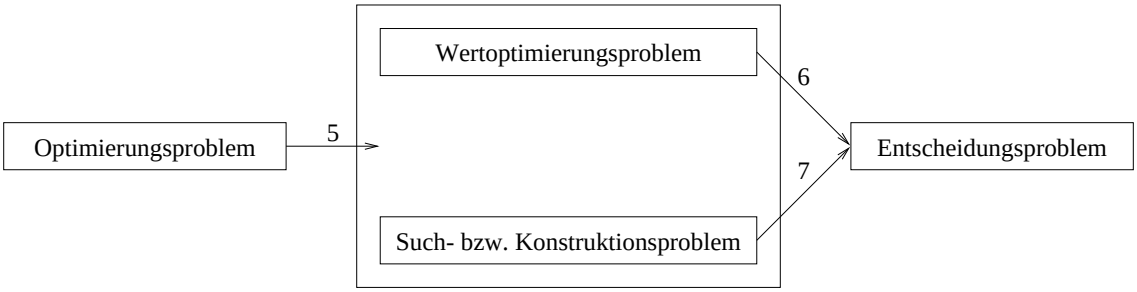

Abbildung 2: Reduktion des Optimierungproblems auf das Entscheidungsproblem.

Reduktion 5 Gegeben eine Prozedur zur Lösung des TSP-Wertoptimierungsproblems sowie eine Prozedur zur Lösung des TSP-Konstruktionsproblems. Das TSP-Optimierungsproblem kann dann wie folgt gelöst werden:

1. Bestimme die Länge K_* einer kürzesten Rundreise.
2. Bestimme eine zulässige Rundreise σ_* zu Distanzmatrix D und Kostenschranke K_*.

Reduktion 6 Gegeben eine Prozedur zur Lösung des TSP-Entscheidungsproblems, so erhalten wir die Länge K_* einer kürzesten Rundreise mit Hilfe von Binärsuche.

Reduktion 7 Die Reduktion des Konstruktionsproblems auf das Entscheidungsproblem ist die interessanteste von allen. Sie macht sich die Eigenschaft der "Selbstreduzierbarkeit" von TSP zunutze. Selbstreduzierbarkeit ist ein wichtiges Konzept der Komplexitätstheorie, das später noch formal sauber besprochen werden wird. Nehmen wir im Falle von TSP oBdA an, dass eine zulässige Rundreise mit Kosten höchstens K existiert. (Andernfalls gibt es nichts zu konstruieren.) Die entscheidende Idee ist, bei jeder einzelnen "Kante" (i, j) von Stadt i nach Stadt j zu testen, ob sie für eine zulässige Rundreise entbehrlich ist. Dazu werden ihre Kosten $d_{i,j}$ versuchsweise auf $K + 1$ gesetzt. Die Entscheidungsprozedur verrät uns dann, ob trotzdem noch eine zulässige Rundreise (logischerweise dann ohne Kante (i, j)) existiert. Falls ja, dann lassen wir $d_{i,j}$ auf seinem hohen Wert. Dies ist nicht weiter schlimm, da wir uns zuvor von der Entbehrlichkeit der Kante (i, j) überzeugt haben. Falls aber die Entscheidungsprozedur signalisiert, dass keine zulässige Lösung mehr existiert, dann setzen wir $d_{i,j}$ auf seinen alten Wert zurück und nehmen Kante (i, j) in die die zu konstruierende Rundreise σ auf. Da wir nur unter Zugzwang Kanten in σ aufnehmen, entsteht auf diese Weise eine zulässige Rundreise.

1.2 Allgemeine Suchprobleme

Formale Sprachen und Entscheidungsprobleme sind zwei Seiten der gleichen Münze. Zu einer formalen Sprache $L \subseteq \Sigma^*$ können wir nämlich das Entscheidungsproblem assoziieren, ob ein vorgegebenes Wort $x \in \Sigma^*$ zu L gehört (das sogenannte *Mitgliedschaftsproblem* zu L). Umgekehrt können wir zu einem Entscheidungsproblem (mit den möglichen Antworten "Ja" und "Nein") die formale Sprache aller Eingaben assoziieren, welche zu der Antwort "Ja" führen.

Im vorigen Abschnitt haben wir das TSP-Entscheidungsproblem, das TSP-Konstruktionsproblem und das TSP-(Wert-)Optimierungsproblem diskutiert. In diesem Abschnitt werden wir den abstrakten Begriff des Suchproblems so allgemein fassen, dass alle Problemvarianten sich als Suchproblem darstellen lassen.

Es folgt die allgemeine Definition des Suchproblems.

Definition 1.1 *Ein* Suchproblem *ist gegeben durch eine Relation $R \subseteq \Sigma^* \times \Sigma^*$. Zu einer Eingabe $x \in \Sigma^*$ ist eine* R-zulässige Lösung[1] *$y \in \Sigma^*$ gesucht, d.h., ein y mit $(x, y) \in R$. Eine DTM löst das Suchproblem, wenn sie für jede Eingabe $x \in \Sigma^*$ ein zulässiges y ausgibt bzw. eine entsprechende Meldung, wenn keine zulässige Lösung existiert). Im folgenden bezeichne $\mathcal{S}(R)$ das zu Relation R assoziierte Suchproblem.*

Beispiel 1.2 *Eine formale Sprache $L \subseteq \Sigma^*$ kann als Suchproblem zur Relation*

$$R(L) = \{(x, 1) |\ x \in L\} \tag{1}$$

aufgefasst werden.

[1] Wenn R aus dem Kontext hervorgeht, sprechen wir auch einfach von einer zulässigen Lösung

Beispiel 1.3 *Das durch eine Relation R und eine Bewertungsfunktion val gegebene Minimierungsproblem — zu gegebener Eingabe x finde eine R-zulässige Lösung y mit minimalem Wert $val(x, y)$ — kann als Suchproblem zur Relation*

$$R_{val} = \{(x, y_*) \in R |\ \forall y :\ (x, y) \in R \implies val(x, y) \geq val(x, y_*)\} \tag{2}$$

aufgefasst werden. Eine analoge Bemerkung gilt für Maximierungsprobleme, Wertoptimierungsprobleme und Konstruktionsprobleme.

Ein Suchproblem könnte aus einem sehr banalen Grund schwer sein, zum Beispiel weil die zulässigen y für eine gegebene Eingabe x extrem lange Wörter sind. Evtl. braucht man dann superpolynomielle Zeit zum Lösen des Suchproblems, bloß weil das Aufschreiben der Ausgabe eine zeitraubende Angelegenheit ist. Die folgende Definition schließt diesen und weitere triviale Gründe für die Härte eines Suchproblems aus.

Definition 1.4 *Eine Relation R heisst* polynomiell beschränkt*, wenn ein Polynom p existiert so dass für alle $(x, y) \in R$: $|y| \leq p(|x|)$.*
R heisst polynomiell entscheidbar*, wenn eine polynomiell zeitbeschränkte DTM entscheiden kann, ob zwei vorgegebene Wörter x, y in Relation R zueinander stehen.*
R heisst polynomiell verifizierbar*, wenn R polynomiell beschränkt und polynomiell entscheidbar ist.*
Die Wertefunktion $val(x, y)$ heißt polynomiell auswertbar*, wenn zu gegebenem $(x, y) \in R$ die Binärdarstellung von $val(x, y)$ durch eine polynomiell zeitbeschränkte DTM berechenbar ist.*

Beispiel 1.5 *Die Relation R_{TSP}, die dem TSP-Konstruktionsproblem entspricht, enthält genau die Paare (x, y) mit $x = (D, K)$, $y = \sigma$ und Permutation σ repräsentiert eine Rundreise mit Kosten höchstens K bezüglich der Distanzmatrix D.*[2] *R_{TSP} ist offensichtlich polynomiell verifizierbar. Beim TSP-Optimierungsproblem verwenden wir die Länge einer Rundreise als Wertefunktion. Diese ist offensichtlich polynomiell auswertbar.*

Eine besondere Rolle spielen in der Komplexitätstheorie die sogenannten *NP*-Relationen:

Definition 1.6 *Zu einer Relation $R \subseteq \Sigma^* \times \Sigma^*$ assoziieren wir die formale Sprache*

$$L(R) := \{x |\ \exists y :\ (x, y) \in R\}. \tag{3}$$

*R heisst NP-*Relation*, falls $L(R) \in NP$.*

Das folgende Resultat basiert auf dem sogenannten *Rate-Verifikationsprinzip*:

Lemma 1.7 *R ist eine NP-Relation genau dann, wenn R polynomiell verifizierbar ist, d.h.,*

$$NP = \{L(R) |\ R \text{ ist polynomiell verifizierbar}\}. \tag{4}$$

[2]Streng genommen ist x ein Wort, das auf eine "natürliche Weise" D und K kodiert, und y ein Wort, das "auf natürliche Weise" σ kodiert. Wir gehen davon aus, dass Sie sich vorstellen können, wie man mathematische Objekte als Wörter über einem Alphabet kodiert und ziehen die informellere (aber weniger gestelzte) Formulierung vor.

Beweis Falls $L \in NP$, dann gibt es ein Polynom T und eine T-zeitbeschränkte NTM M mit $L = L_M$. Wir können oBdA annehmen, dass M auf Eingabe $x \in \Sigma^n$ nach dem sogenannten Rate-Verifikationsprinzip arbeitet:

- In einer ersten Phase wir rät M einen binären Ratestring Wir bezeichnen den Ratestring, der von $a = (a_1, \ldots, a_{T(n)})$ und schreibt diesen auf die Zellen $-1, \ldots, -T(n)$. Auf Zelle 0 stehe Trennzeichen $\#$. T ist ein geeignetes Polynom in n.
- In der zweiten Phase arbeitet M deterministisch auf der um den Ratestring erweiterten Eingabe $a\#x$.

Wir setzen $M(x, a) = 1$, falls M in der deterministischen Phase 2 zu gegebener Eingabe x und gegebenem Ratestring a eine akzeptierende Rechnung vollzieht. Andernfalls setzen wir $M(x, a) = 0$. Mit

$$R = \cup_{n \geq 0} \{(x, a) \in \Sigma^n \times \Sigma^{T(n)} |\ M(x, a) = 1\}$$

ergibt sich dann $L = L(R)$. Offensichtlich ist Relation R polynomiell verifizierbar.
Sei nun R eine polynomiell verifizierbare Relation und $x \in \Sigma^n$. Dann gibt es zwei Polynome p, q und eine DTM V mit folgenden Eigenschaften:

- $x \in L(R) \Leftrightarrow (\exists y \in \Sigma^* : |y| \leq p(n) \wedge (x, y) \in R)$.
- V vollzieht auf einem Wortpaar (x, y) mit $|x| = n$ und $|y| \leq p(n)$ maximal $q(n + p(n))$ Rechenschritte.
- V akzeptiert das Wortpaar (x, y) genau dann, wenn $(x, y) \in R$.

Offensichtlich ist $T(n) = p(n) + q(n + p(n))$ ein Polynom in n. Die folgende NTM M ist ein T-zeitbeschränkter Akzeptor von L:

1. Zu gegebener Eingabe x mit $|x| = n$ rate einen String y mit $|y| \leq p(n)$.[3]
2. Setze V auf das Wortpaar (x, y) an.

Es folgt, dass $L = L_M \in NP$. **qed.**

1.3 Karp-, Cook- und Levin-Reduktionen

Die uns bereits bekannte Definition der polynomiellen Reduktion wurde von Richard Karp vorgeschlagen. In Würdigung ihres Erfinders und in Abgrenzung zu weiteren Reduktionstypen verwenden wir daher ab jetzt die Bezeichnung "Karp-Reduktion". Eine Karp-Reduktion setzt zwei formale Sprachen zueinander in Beziehung. Die Problemreduktionen, die wir im Zusammenhang mit TSP skizziert haben, verlaufen zwischen allgemeinen Suchproblemen. Um den hierfür geeigneten Reduktionsbegriff einzuführen, benötigen wir zunächst die Definition der Orakel Turing Maschine.

[3] Wir können oBdA annehmen, dass p platzkonstruierbar ist.

Definition 1.8 *Eine TM M mit einer Relation R als* Orakel*, genannt* Orakel Turing Maschine (OTM) *und notiert als $M^{[R]}$, verfügt über ein zusätzliches* Orakelband *und drei ausgezeichnete Zustände $q_?, q_+, q_-$. Auf das Orakelband kann M ein beliebiges Wort $x \in \Sigma^*$ schreiben und dann den* Fragezustand $q_?$ *annehmen. Das Orakel ersetzt dann Wort x durch ein Wort y mit $(x, y) \in R$ (falls möglich) und M wechselt in den Zustand q_+. Falls kein zulässiges y existiert, wird x nicht ersetzt und M wechselt in den Zustand q_-. Ansonsten ändert sich die Konfiguration der Maschine bei diesem Übergang nicht. Wenn die Maschine ihre Frage auf dem Orakelband spezifiziert hat, kostet sie das Befragen des Orakels und das Erhalten der Antwort nur zwei Schritte.*[4]
Falls eine Relation der Form $R(L)$ für eine formale Sprache L als Orakel verwendet wird, sprechen wir der Einfachheit halber von einem Orakel für L, das wir als $M^{[L]}$ notieren.
Deterministische (bzw. nichtdeterministische) Orakel Turing Maschinen bezeichnen wir kurz als DOTMs (bzw. NOTMs).

Mit Hilfe der Orakel-Maschinen definieren wir Cook-Reduktionen wie folgt.

Definition 1.9 *R heisst* Cook-reduzierbar *auf R' , in Zeichen $R \leq_T R'$, falls eine polynomiell zeitbeschränkte DTM M existiert, so dass die DOTM $M^{[R']}$ das Suchproblem $\mathcal{S}_R$ löst. Falls $R = R(L)$, dann schreiben wir der Einfachheit halber $L \leq_T R'$ statt $R(L) \leq_T R'$. Eine analoge Bemerkung gilt im Falle $R' = R(L')$.*

Offensichtlich ist eine Karp-Reduktion $L \leq_{pol} L'$ eine sehr spezielle Cook-Reduktion, bei der zunächst $f(x)$ aus x berechnet wird, um dann das Orakel für L' nach $f(x)$ zu befragen. Insbesondere wird auch nur eine einzige Frage an das Orakel gestellt. Es ist nicht schwer, die für Karp-Reduktionen bereits genannten Eigenschaften auch für Cook-Reduktionen zu beweisen:

Lemma 1.10 *1. Relation $\leq_T$ ist reflexiv und transitiv.*

2. Aus $R \leq_T R'$ und der Lösbarkeit von $\mathcal{S}_{R'}$ in Polynomialzeit folgt die Lösbarkeit von $\mathcal{S}_R$ in Polynomialzeit.

Mit Hilfe der Cook-Reduktionen können wir den Begriff der Selbstreduzierbarkeit präzise definieren.

Definition 1.11 *R heisst* selbstreduzierbar*, falls $R \leq_T L(R)$.*

Wir werden später sehen, dass **alle** *NPC*-Relationen[5] selbstreduzierbar sind.

Alle am TSP-Beispiel vorgeführten Reduktionen waren Cook-Reduktionen. Man überlegt sich leicht, dass diese Reduktionen verallgemeinert werden können auf beliebige selbstreduzierbare Relationen mit polynomiell auswertbaren Bewertungsfunktionen. Da alle *NPC*-Relationen selbstreduzierbar sind, ergibt sich für eine breite Klasse von Problemen, dass

[4] Das Orakel spendiert die Antwort sozusagen kostenlos wie eine gute Fee.

[5] *NPC* ist die Klasse der *NP*-vollständigen Probleme. Die formale Definition von *NPC*-Relationen holen wir später nach.

Entscheiden, Konstruieren (Suchen) und Optimieren ungefähr gleich schwer sind. Dadurch erhalten wir die Berechtigung, uns beim Studium der Problemkomplexität auf Entscheidungsprobleme (formale Sprachen) zurückzuziehen.

Stephen Cook (Universität Toronto) und Richard Karp (Universität Berkeley) waren zwei der westlichen Protagonisten der *NP*-Vollständigkeitstheorie in den siebziger Jahren des vorigen Jahrhunderts. Zur gleichen Zeit wurde die Theorie im Osten von Levin vorangetrieben. Wir beschließen diesen Abschnitt mit dem von Levin vorgeschlagenen Reduktionstyp.

Definition 1.12 *R heisst* Levin-reduzierbar *auf R' , in Zeichen $R \leq_L R'$, wenn drei in Polynomialzeit berechenbare Abbildungen $f, g, h : \Sigma^* \to \Sigma^*$ existieren, so dass für alle $x, y, z \in \Sigma^*$ folgendes gilt:*

$$x \in L(R) \quad \Leftrightarrow \quad f(x) \in L(R') \tag{5}$$

$$(x, y) \in R \quad \Leftrightarrow \quad (f(x), g(x, y)) \in R' \tag{6}$$

$$(f(x), y') \in R' \quad \Leftrightarrow \quad (x, h(x, y')) \in R \tag{7}$$

Wir können f als eine *Eingabetransformation* auffassen, die gemäß (5) eine Karp-Reduktion von $L(R)$ auf $L(R')$ repräsentiert. Abbildungen g und h sind *Lösungstransformationen*: zu gegebener Eingabe x transformiert g gemäß (6) eine R-zulässige Lösung für x in eine R'-zulässige Lösung für $f(x)$; umgekehrt transformiert h gemäß (7) eine R'-zulässige Lösung für $f(x)$ in eine R-zulässige Lösung für x.

Eine Levin-Reduktion von R auf R' impliziert nicht nur eine Karp-Reduktion von $L(R)$ auf $L(R')$, sondern auch eine Cook-Reduktion von R auf R':

Lemma 1.13 $R \leq_L R' \implies R \leq_T R'$.

Beweis Folgende DOTM mit einem Orakel für R' löst das Suchproblem $\mathcal{S}_R$ in Polynomialzeit:

1. Transformiere x in $f(x)$.
2. Befrage das Orakel für R' nach einer R'-zulässigen Lösung y' für $f(x)$. (Falls diese nicht existiert, so existiert wegen (5) auch keine R-zulässige Lösung für x.)
3. Transformiere (x, y') in $h(x, y')$ und gib $h(x, y')$ aus. (Wegen (7) ist $h(x, y')$ eine R-zulässige Lösung für x.

qed.

Es ist nicht schwer, die folgenden Aussagen zu zeigen:

Lemma 1.14 *1. Relation $\leq_L$ ist reflexiv und transitiv.*

2. Aus $R \leq_L R'$ und der Lösbarkeit von $\mathcal{S}_{R'}$ in Polynomialzeit folgt die Lösbarkeit von $\mathcal{S}_R$ in Polynomialzeit.

Die Relation R_{SAT} enthält alle Paare der Form (F, a), so dass folgende Bedingungen gelten:

- F repräsentiert eine Kollektion von Booleschen Klauseln, sagen wir über n Booleschen Variablen.
- $a \in \{0,1\}^n$ ist eine Boolesche Belegung dieser Variablen.
- Belegung a erfüllt alle in F enthaltenen Klauseln.

Offensichtlich gilt SAT $= L(R_{SAT})$.

Folgerung 1.15 *R_{SAT} ist eine unter Levin-Reduktionen NP-vollständige Relation.*

Beweis Sei R eine *NP*-Relation. Somit gilt $L = L(R) \in NP$. Der Beweis des Satzes von Cook liefert eine in Polynomialzeit berechenbare Eingabetransformation $w \mapsto F_w$, welche implizit eine Karp-Reduktion von $L = L(R)$ auf SAT $= L(R_{SAT})$ enthält. Zum Nachweis von Folgerung 1.15 fehlt also nur die Angabe zweier geeigneter Lösungstransformationen:

1. Es muss in Polynomialzeit möglich sein, ein Paar $(w, r) \in R$ in eine Boolesche Belegung a mit $(F_w, a) \in R_{SAT}$ zu transformieren.
   $(w, r) \in R$ bedeutet, dass r ein Zertifikat für $w \in L$ ist. Im Beweis des Satzes von Cook wurde gezeigt, wie man hieraus eine F_w erfüllende Boolesche Belegung a ableiten kann. Diese Belegung ergab sich im Wesentlichen aus der deterministischen Verifikation von $(r, w) \in R$. Da die Laufzeit der Verifikationsphase polynomiell in $|r| + |w|$, und somit polynomiell in $|w|$, beschränkt ist, erhalten wir die Transformation $(w, r) \mapsto a$ in Polynomialzeit.

2. Es muss in Polynomialzeit möglich sein, ein Paar (w, a) mit einer F_w erfüllenden Booleschen Belegung a in ein Zertifikat r mit $(w, r) \in R$ zu transformieren.
   Im Beweis des Satzes von Cook wurde demonstriert, wie sich r aus der Belegung der Booleschen Variablen der Form $S(0, -k, b)$ mit $k = 1, \ldots, p(n)$ und $b = 0, 1$ auf einfache Weise (und offensichtlich in Polynomialzeit) ablesen lässt.

Damit hat sich eine Levin-Reduktion von R auf R_{SAT} ergeben. **qed.**

2 Selbstreduzierbarkeit aller NPC-Relationen

Wir beginnen mit der Definition der $\mathcal{K}$-Relationen für eine Komplexitätsklasse $\mathcal{K}$.

Definition 2.1 *R heisst $\mathcal{K}$-Relation, falls $L(R) \in \mathcal{K}$.*

Im Spezialfall $\mathcal{K} = NP$ deckt sich diese Definition mit unserer alten Definition der *NP*-Relationen. Das Hauptresultat dieses Abschnittes ist die folgende Aussage über *NPC*-Relationen:

Theorem 2.2 *Jede NPC-Relation ist selbstreduzierbar.*

Beweis Wir beweisen zunächst die Selbstreduzierbarkeit von R_{SAT}:

Behauptung $R_{SAT} \leq_T$ SAT.

Sei $F = F(v_1, \ldots, v_n)$ eine Konjunktion von Klauseln über den Booleschen Variablen $v_1, \ldots, v_n$. Mit Hilfe eines SAT-Orakels können wir das Suchproblem zu R_{SAT} lösen wie folgt:

Vorabtest Wir fragen das Orakel, ob F erfüllbar ist. Falls nicht, können wir mit der Meldung, dass keine erfüllende Belegung existiert, abbrechen. Falls doch, dann weiter wie folgt.

Konstruktion einer erfüllenden Belegung Für $i \geq 0$ und eine partielle Belegung $(a_1, \ldots, a_i) \in \{0,1\}^i$ bezeichne $F_i = F(a_1, \ldots, a_i, v_{i+1}, \ldots, v_n)$ die vereinfachte Klauselkonjunktion, die entsteht, wenn wir für $j = 1, \ldots, i$ die Variable v_j durch die Boolesche Konstante a_j ersetzen. [6] Für $i = 0$ ist noch keine Variable belegt, d.h., $F_0 \equiv F$. Für $i = n$ sind alle Variablen belegt, d.h., $F_n \in \{\text{TRUE}, FALSE\}$. Falls $F_n \equiv$ TRUE, dann handelt es sich bei a um eine erfüllende Belegung. Eine partielle Belegung heisse *gut*, wenn sie zu einer erfüllenden Belegung fortgesetzt werden kann. Da der Vorabtest garantiert, dass $F \equiv F_0$ erfüllbar ist, ist die leere Belegung gut. Unsere Strategie besteht darin eine gute partielle Belegung $a_1, \ldots, a_i$, $0 \leq i \leq n-1$, mit Hilfe des SAT-Orakels zu einer guten partiellen Belegung $a_1, \ldots, a_i, a_{i+1}$ zu erweitern. Falls dies gelingt, können wir iterativ die gute leere Belegung zu einer guten vollständigen (und somit F erfüllenden) Belegung fortsetzen. Nehmen wir also an, dass $a_1, \ldots, a_i$ eine gute partielle Belegung ist. Beachte, dass dann $a_1, \ldots, a_i, 0$ oder $a_1, \ldots, a_i, 1$ eine gut partielle Belegung sein muss. Wir fragen das SAT-Orakel, ob $F(a_1, \ldots, a_i, 0, v_{i+2}, \ldots, v_n)$ erfüllbar ist. Falls ja, dann setzen wir $a_{i+1} = 0$. Falls nein, dann muss erzwungenermaßen $F(a_1, \ldots, a_i, 1, v_{i+2}, \ldots, v_n)$ erfüllbar sein, und wir setzen $a_{i+1} = 1$. Auf diese Weise gelangen wir nach n Iterationen zu einer F erfüllenden Belegung $a = (a_1, \ldots, a_n)$.

Um Theorem 2.2 zu beweisen haben wir zu zeigen:

Behauptung Sei R eine NPC-Relation. Dann gilt $R \leq_T L(R)$.

Sei x eine Eingabe für das Suchproblem zu R. Eine Anfrage an das $L(R)$-Orakel klärt wieder vorab, ob überhaupt eine R-zulässige Lösung für x existiert. OBdA sei dies der Fall.

Wir stehen nun vor dem Problem, mit Hilfe des $L(R)$-Orakels eine R-zulässige Lösung y mit $(x, y) \in R$ zu konstruieren. Dazu gehen wir in zwei Phasen vor. Wir argumentieren zunächst, dass die Konstruktion eines geeigneten y mit Hilfe des SAT-Orakels anstelle des $L(R)$-Orakels leicht durchzuführen ist. Danach zeigen wir, dass ein $L(R)$-Orakel es auf einfache Weise erlaubt, ein SAT-Orakel zu simulieren.

[6] Vereinfachungen: durch die partielle Belegung bereits erfüllte Klauseln können eliminiert bzw. durch TRUE ersetzt werden; in den verbleibenden Klauseln können alle Literale $v_j, \bar{v}_j$ für $j = 1, \ldots, i$ entfernt bzw. durch FALSE ersetzt werden.

Für **Phase 1** bedienen wir uns der polynomiellen Levin-Reduktion von R auf R_{SAT}, die gemäß Folgerung 1.15 existieren muss. Sei f die zur Reduktion gehörende Eingabetransformation und h die zweite Lösungstransformation. (Vgl. Definition 1.12.) Wir gehen dann vor wie folgt:

- Transformiere x vermöge f in eine Konjunktion F_x Boolescher Klauseln. Da x eine zulässige Lösung besitzt, muss F_x erfüllbar sein.
- Nutze die Selbstreduzierbarkeit von R_{SAT} aus, um mit Hilfe eines SAT-Orakels eine F_x erfüllende Belegung a zu konstruieren.
- Transformiere (x, a) vermöge h in eine R-zulässige Lösung y für x.

Dieses Verfahren ist offensichtlich polynomiell zeitbeschränkt. Einziger Schönheitsfehler: wir haben real kein SAT- sondern nur ein $L(R)$-Orakel.

In **Phase 2** demonstrieren wir, dass man mit Hilfe eines $L(R)$-Orakels ein virtuelles SAT-Orakel bereitstellen kann. Dazu bedienen wir uns der polynomiellen Karp-Reduktion von SAT nach $L(R)$, die wegen $L(R) \in NPC$ existieren muss. Sei f' die dazu gehörende Eingabetransformation. Eine Anfrage an das SAT-Orakel nach der Erfüllbarkeit von F beantworten wir wie folgt:

- Transformiere F vermöge f' in eine Eingabe x' für das Mitgliedschaftsproblem zu $L(R)$.
- Frage das $L(R)$-Orakel, ob $x \in L(R)$ und gib die Antwort aus.

Da die Karp-Reduktion garantiert, dass F genau dann erfüllbar ist, wenn $x' \in L(R)$, ist die von uns ausgegebene Antwort korrekt. **qed.**

Die Klasse *NPC* ist sehr reichhaltig. Die Eigenschaft der Selbstreduzierbarkeit gilt daher für eine sehr umfangreiche Klasse von Relationen. Für alle diese Relationen ist also das Konstruieren einer zulässigen Lösung nicht wesentlich aufwendiger als das Entscheiden, ob eine solche Lösung existiert.

Nichtsdestotrotz gibt es einige natürliche Relationen, deren Suchproblem vermutlich nicht in Polynomialzeit lösbar ist und die vermutlich dennoch keine *NPC*-Relationen sind:

COMPOSITES COMPOSITES bezeichnet die Menge der natürlichen Zahlen $N > 1$, die keine Primzahlen sind. Da ein Primzahltest in Polynomialzeit durchgeführt werden kann, gehört COMPOSITES zur Klasse P. Das zugehörige Suchproblem ist das *Faktorisierungsproblem*: zu gegebener Zahl N finde (falls möglich) eine Faktorisierung $N = N_1 \cdot N_2$ mit $N_1, N_2 \geq 2$. Die Frage der Selbstreduzierbarkeit ist offen. Im Falle der Selbstreduzierbarkeit würde ein COMPOSITES-Orakel erlauben, das Faktorisierungsproblem zu lösen. Wegen COMPOSITES $\in P$ könnte dann eine Zahl in Polynomialzeit in ihre Primfaktoren zerlegt werden.[7]

[7] Man vermutet, dass dies nicht möglich ist, und viele Public Key Kryptosysteme basieren auf der (vermuteten) Härte des Faktorisierungsproblems.

GRAPHENISOMORPHIE Das *Graphenisomorphieproblem* ist die Frage, ob zwei vorgegebene Graphen $G = (V, E), G' = (V', E')$ mit $|V| = |V'|$ *isomorph* sind, d.h., sind beide Graphen bis auf Umbenennung der Knoten gleich? Auch dieses Problem liegt vermutlich in $NP \setminus (P \cup NPC)$. Das zugehörige Suchproblem verlangt nach Angabe der *Isomorphie*, also der bijektiven Abbildung $h : V \to V'$, so dass $E' = \{(h(v), h(w))| \, (v, w) \in E\}$.

Im Unterschied zu COMPOSITES und PRIMES kann man bei GRAPHENISOMORPHIE einfach eine Cook-Reduktion vom Suchproblem auf das Entscheidungsproblem angeben. Die zentralen Ideen zum Nachweis der Selbstreduktion sind wie folgt:

- Eine partielle Lösung des Grpahenisomorphieproblems ist eine injektive Abbildung h einer Teilmenge U von V nach V', die zu einer Isomorphieabbildung fortsetzbar ist. Nehmen wir oBdA an, dass G und G' isomorph sind. Dann ist die *leere Abbildung* mit Definitionsbereich $U = \emptyset$ eine partielle Lösung.

- Die Idee ist, den Definitionsbereich einer partiellen Lösung mit Hilfe des GRAPHENISOMORPHIE-Orakels iterativ zu erweitern. Wenn wir testen könnten, ob eine Erweiterung der Form $h(w) = z$ mit $w \in V \setminus U$ immer noch eine partielle Lösung ist, dann könnten wir durch Ausprobieren aller Kandidaten z schließlich eine Fortsetzung der partiellen Lösung ausfindig machen. Schönheitsfehler: Wir verfügen nicht über einen solchen "Fortsetzungstest", sondern nur über ein GRAPHENISOMORPHIE-Orakel.

- Es verbleibt zu zeigen, dass man mit einem GRAPHENISOMORPHIE-Orakel einen virtuellen Fortsetzungstest bereitstellen kann. Sei h bisher auf $U = \{u_1, \ldots, u_i\}$ definiert und $u'_j = h(u_j)$ für $j = 1, \ldots, i$. Wir wollen testen, ob $h(u_{i+1}) = u'_{i+1}$ eine geeignete Fortsetzung ist. Zu diesem Zweck erzeugen wir zwei Hilfsgraphen H und H'. H geht aus G hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten u_j jn-viele neue Knoten als Nachbarn geben. Analog geht H' geht aus G' hervor, indem wir für $j = 1, \ldots, i+1$ dem Knoten u'_j jn-viele neue Knoten als Nachbarn geben. Offensichtlich ist h mit erweitertem Definitionsbereich $\{u_1, \ldots, u_i, u_{i+1}\}$ genau dann eine partielle Lösung, wenn H und H' isomorph sind. Der Fortsetzungstest wird also implementiert, indem wir das GRAPHENISOMORPHIE-Orakel nach der Isomorphie von H und H' befragen.

Wir fassen die letzte Diskussion zusammen:

Theorem 2.3 *Die Relation zum Graphenisomorphieproblem ist selbstreduzierbar.*