Unentscheidbarkeit von Problemen mittels Turingmaschinen

Daniel Roßberg 0356177
Roland Schatz 0355521

2. Juni 2004

Zusammenfassung

In dieser Arbeit befassen wir uns mit der Unentscheidbarkeit von Problemen mittels des formalen Konzepts der Turingmaschine. Dazu führen wir zuerst die dafür notwendigen Grundlagen ein. Anschließend beweisen wir die Unentscheidbarkeit ausgewählter Probleme. Weiters stellt sich eine ganze Klasse von praxisrelevanten Problemen als unentscheidbar heraus.

1 Einleitung

Ob es für ein konkretes mathematisches Problem in endlich vielen Schritten eine Lösung gibt, ist für das Finden eines Algorithmus entscheidend. Die Berechenbarkeitstheorie befasst sich mit eben dieser Frage.

Sie trennt berechenbares von nicht berechenbarem, sie hat die Aufgabe einen formalen Berechenbarkeits- bzw. Algorithmenbegriff festzulegen, um damit Aussagen über spezielle Algorithmen treffen zu können. Ein Werkzeug um algorithmische Abläufe in einzelne vereinfachte Schritte zu zerlegen und dadurch einer mathematischen Betrachtung zugänglich zu machen ist die sogenannte Turingmaschine. Benannt ist sie nach ihrem geistigen Schöpfer Alan Mathison Turing (1912–1954).

Die Frage nach der Lösbarkeit eines Problems ist im Wesentlichen die Frage nach der Existenz eines zur Lösung geeigneten Algorithmus. Dass solche Lösungen nicht für jede Art von mathematischen Problemen existieren können wurde von Kurt Gödel (1906–1978) mittels seiner Unvollständigkeitssätze bewiesen[1].

Im folgenden Text befassen wir uns mit dem Problem der Entscheidbarkeit bzw. Nichtentscheidbarkeit. Dazu führen wir zuerst den Begriff der Sprache formal ein und definieren dazu wichtige Termini wie Alphabet und Wort. Anschließend geben wir eine allgemeine Definition einer Turingmaschine, welche wir an einem konkreten Beispiel genauer erläutern.

Mit Hilfe dieser formalen Werkzeuge beweisen wir die Unentscheidbarkeit einiger exemplarischer Probleme und geben einen Ausblick auf weitere Klassen von unentscheidbaren Problemen. Die dafür verwendete Beweistechnik stammt aus [3].

2 Sprachen

2.1 Definitionen

Definition 2.1 (Alphabet) Ein Alphabet Σ ist eine nicht-leere endliche Menge. Die Elemente dieser Menge heißen Buchstaben oder Symbole.

Definition 2.2 (Wort) Ein *Wort* über einem Alphabet Σ ist eine *endliche* Folge von Buchstaben aus dem Alphabet Σ. Die Anzahl der Buchstaben heißt die *Länge* eines Wortes.

Definition 2.3 Sei Σ ein Alphabet. Σ^n ist die Menge aller Wörter der Länge n über Σ. Weiters ist $\Sigma^* := \bigcup_{n\in\mathbb{N}_0} \Sigma^n$, also die Menge *aller* Wörter über Σ.

Definition 2.4 (Sprache) Jede Teilmenge von Σ^* heißt eine *Sprache* über dem Alphabet Σ.

2.2 Schreibweisen

Sei $\Sigma := \{a, b, c, d\}$ unser Alphabet, $n \in \mathbb{N}_0$ und w ein Wort:

- Das Wort (a, b, c) schreiben wir kurz als abc.
- a^n steht für n Wiederholungen des Buchstaben a, analog steht $(w)^n$ für n Wiederholungen des Wortes w.
- Für das leere Wort (mit der Länge 0) schreiben wir ϵ.

Beispielsweise steht $ab^3(cd)^2$ für das Wort $abbbcdcd$. $L := \{0^n1^n \mid n \in \mathbb{N}\}$ ist die Sprache aller Wörter die zuerst eine beliebige Anzahl Nuller und dann die selbe Anzahl Einser haben. 0011 ist in dieser Sprache ($0011 \in L$), 0010 ist nicht in der Sprache ($0010 \notin L$).

3 Turingmaschinen

3.1 Einführung

Eine Turingmaschine besteht aus einer Kontrolleinheit, einem Ein-/Ausgabeband und einem Lese-/Schreibkopf (siehe Abbildung 1).

Die Kontrolleinheit kann eine endliche Anzahl von Zuständen aus der Zustandsmenge Q annehmen. Der Anfangszustand wird üblicherweise als q_0 bezeichnet. Eine Untermenge $F \subset Q$ von Zuständen heißen "akzeptierende Zustände".

Das in eine Richtung unendliche Band ist in Zellen eingeteilt, wobei in jeder Zelle ein Symbol aus dem *Bandalphabet* Γ steht. Am Anfang der Berechnung steht auf dem Band ein Wort aus dem Eingabealphabet $\Sigma \subset \Gamma$. Ein Zeichen $\gamma_0 \in \Gamma\backslash\Sigma$ steht für "leere" Zellen, es wird im Folgenden durch $\sqcup$ dargestellt.

Der Lese-/Schreibkopf (LS-Kopf) zeigt immer genau auf eine Zelle, am Anfang einer Berechnung steht er auf der ersten Zelle.

Bei jedem Berechnungsschritt schreibt die TM in Abhängigkeit vom aktuellen Zustand und dem Symbol auf das der LS-Kopf zeigt ein neues Symbol auf das Band, worauf der LS-Kopf evtl. nach links oder rechts bewegt wird und die Kontrolleinheit einen neuen Zustand annimmt.

Definition 3.1 (Turingmaschine) Eine *Turingmaschine* ist ein 6-Tupel $(Q, \Sigma, \Gamma, q_0, F, \delta)$. Q ist die Zustandsmenge, Γ das Bandalphabet, $\Sigma \subset \Gamma$ das Eingabealphabet, $q_0 \in Q$ der Anfangszustand, $F \subset Q$ die Menge der akzeptierende Zustände und $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$ die Überführungsfunktion. L, R und S stehen für die Bewegungen nach links, rechts bzw. stationär. δ kann eine partielle Funktion sein.

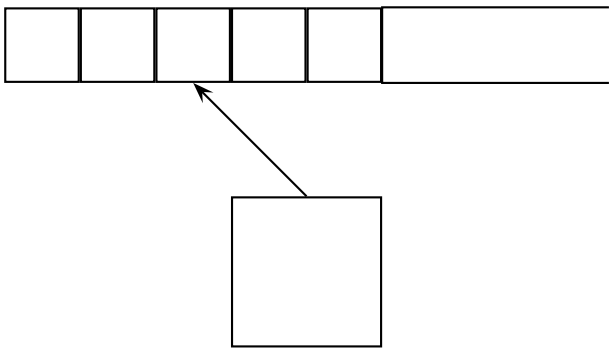

Abbildung 1: Turingmaschine

Definition 3.2 (Konfiguration) Eine *Konfiguration* k_i einer TM besteht aus einer endlichen Folge von Bandsymbolen $\alpha_0, \ldots, \alpha_k$ links vom LS-Kopf, dem aktuellen Zustand q und einer endlichen Folge von Bandsymbolen $\alpha_{k+1}, \alpha_{k+2}, \ldots, \alpha_n$ rechts vom LS-Kopf (incl. dem Symbol auf das der LS-Kopf zeigt). Rechts davon werden unendlich viele leere Zellen ($\sqcup$) angenommen. Wir schreiben dafür:

$$k_i = \alpha_0 \ldots \alpha_k q \alpha_{k+1} \ldots \alpha_n$$

Definition 3.3 (Berechnungsschritt) Ein *Berechnungsschritt* einer TM ist der durch δ angegeben Übergang von einer Konfiguration k_i in die nächste k_{i+1}. Wir sagen k_{i+1} geht aus k_i hervor und schreiben dafür $k_i \vdash k_{i+1}$.

Seien $p, q \in Q$ Zustände und $\alpha_i, \beta_i \in \Gamma$ Bandsymbole. Abhängig von δ sind folgende Berechnungsschritte möglich:

- $\delta(p, \alpha_k) = (q, \beta_k, L) \iff \alpha_0 \ldots \alpha_{k-1} p \alpha_k \ldots \alpha_n \vdash \alpha_0 \ldots \alpha_{k-2} q \alpha_{k-1} \beta_k \ldots \alpha_n$
- $\delta(p, \alpha_k) = (q, \beta_k, R) \iff \alpha_0 \ldots \alpha_{k-1} p \alpha_k \ldots \alpha_n \vdash \alpha_0 \ldots \beta_k q \alpha_{k+1} \ldots \alpha_n$
- $\delta(p, \alpha_k) = (q, \beta_k, S) \iff \alpha_0 \ldots \alpha_{k-1} p \alpha_k \ldots \alpha_n \vdash \alpha_0 \ldots \alpha_{k-1} q \beta_k \ldots \alpha_n$

Definition 3.4 (Berechnung) Eine *Berechnung* einer TM ist eine Folge von Konfigurationen $k_0 \vdash k_1 \vdash \cdots \vdash k_n$. Die Startkonfiguration k_0 ist von der Form $q_0 w$ wobei w ein Wort aus dem Eingabealphabet Σ ist. Die Berechnung *terminiert* wenn für eine Konfiguration k_n die partielle Funktion δ nicht definiert ist.

Definition 3.5 Sei $q_0 w \vdash l_1 q_1 r_1 \vdash \cdots \vdash l_n k_n r_n$ eine Berechnung einer TM M.
M akzeptiert $w :\iff \exists i \leq n : q_i \in F$, also genau dann wenn in der Berechnung ein akzeptierender Zustand vorkommt. M heißt dann eine *akzeptierende TM*.

3.2 Beispiel

Als Beispiel bauen wir eine TM, die erkennt ob ein Wort die Form $0^n 1^n$ hat:
Zustandsmenge $Q = \{q_0, q_1, q_2, q_3, q_4\}$
Eingabealphabet $\Sigma = \{0, 1\}$
Bandalphabet $\Gamma = \{\sqcup, 0, 1, X, Y\}$
Anfangszustand q_0
Endzustände $F = \{q_4\}$

	⊔	0	1	X	Y
q_0	—	(q_1, X, R)	—	—	(q_3, Y, R)
q_1	—	$(q_1, 0, R)$	(q_2, Y, L)	—	(q_1, Y, R)
q_2	—	$(q_2, 0, L)$	—	(q_0, X, R)	(q_2, Y, L)
q_3	$(q_4, ⊔, R)$	—	—	—	(q_3, Y, R)
q_4	—	—	—	—	—

Abbildung 2: Überführungsfunktion δ

Überführungsfunktion δ: siehe Abbildung 2

Ausgehend vom Eingabewort 0011 ergibt sich folgende Berechnung:

$$q_0 0011 \vdash X q_1 011 \vdash X 0 q_1 11 \vdash X q_2 0 Y 1 \vdash$$
$$q_2 X 0 Y 1 \vdash X q_0 0 Y 1 \vdash X X q_1 Y 1 \vdash X X Y q_1 1 \vdash$$
$$X X q_2 Y Y \vdash X q_2 X Y Y \vdash X X q_0 Y Y \vdash X X Y q_3 Y \vdash$$
$$X X Y Y q_3 ⊔ \vdash X X Y Y ⊔ q_4 ⊔$$

An dieser Stelle terminiert die Berechnung, da $\delta(q_4, ⊔)$ nicht definiert ist. $q_4 \in F$, also wird das Wort 0011 von der TM akzeptiert.

Andererseits wird zum Beispiel das Wort 100 nicht akzeptiert, da bereits für die erste Konfiguration $q_0 100$ kein Berechnungsschritt mehr definiert ist ($\delta(q_0, 1)$ ist nicht definiert) und daher nie ein akzeptierender Zustand erreicht wird.

3.3 Eigenschaften von Turingmaschinen

Es folgen einige simple Eigenschaften von Turingmaschinen, die bei späteren Beweisen benötigt werden. Die Beweise dazu sind relativ aufwändig, aber nicht schwer zu verstehen. Deswegen werden sie hier nur kurz skizziert.

Definition 3.6 (Mächtigkeit) Die *Mächtigkeit* einer Maschine ist die Menge aller Probleme[1], die diese Maschine lösen kann. Unter Maschine ist allgemein ein mathematisches Konstrukt zur Lösung von Problemen zu vertehen.

Lemma 3.7 Turingmaschinen und normale Computerprogramme[2] sind gleich mächtig.

Beweisidee Offensichtlich kann man ganz einfach ein Computerprogramm schreiben, welches die Berechnungsschritte einer TM simuliert. Außerdem ist es möglich eine TM zu konstruieren, die ein Computerprogramm einliest und ausführt.

Lemma 3.8 Wir erweitern das Konzept einer TM derart, dass wir beliebig viele Bänder haben, die in beide Richtungen unendlich sind. Dieses erweiterte Konzept ist gleich mächtig wie das herkömmlicher Turingmaschinen.

[1] Der Begriff "Problem" wird in Sektion 4 definiert. Für die Betrachtungen in dieser Sektion ist die genaue Definition nicht relevant.

[2] Für eine mathematisch einwandfreie Definition von Computerprogrammen sei auf das Kaptiel "RAM und RASP" in [4] verwiesen

Beweisidee Man kann natürlich auch ein Computerprogramm schreiben, welches eine erweiterte TM simuliert. Anschließend Lemma 3.7 anwenden...

Lemma 3.9 Jede TM $(Q, \Sigma, \Gamma, q_0, F, \delta)$ lässt sich auch als TM $(Q, \Sigma', \Gamma', q_0, \{q_f\}, \delta')$ darstellen, wobei $\Sigma' = \{0, 1\}$, $\Gamma' = \{\sqcup, 0, 1\}$ und $\delta'(q_f, \alpha)$ ist nicht definiert.

Beweisidee Jedes Zeichen aus Σ bzw. Γ kann als Binärcode dargestellt werden. Außerdem kann man einen neuen Zustand q_f einführen und von jedem Zustand aus F unabhängig vom Zeichen am Band einfach nach q_f wechseln. Dadurch kommt man mit q_f als einzigen akzeptierenden Zustand aus.

Vereinbarung Ab jetzt werden wo nicht anders festgelegt nur mehr Turingmaschinen mit diesen Eigenschaften betrachtet. Aus Lemma 3.9 folgt außerdem dass jede TM sofort terminiert werden kann, sobald ein akzeptierender Zustand erreicht wird.

Lemma 3.10 Jede TM kann als ein Wort über $\Sigma = \{0, 1\}$ dargestellt werden.

Beweisidee Eine konkrete Möglichkeit der Darstellung wird in [4] vorgeschlagen.

3.4 Turingmaschinen und Sprachen

Definition 3.11 Sei M_a eine akzeptierende Turingmaschine (vgl. Definition 3.5).
$L(M_a) := \{w \mid M_a \text{ akzeptiert } w\}$, M_a akzeptiert die Sprache $L(M_a)$.

Definition 3.12 Sei M_g eine TM (evtl. mit mehreren Bändern), die auf dem ersten Band nie nach links geht und mit leeren Bändern startet. So eine Turingmaschine heißt *generierende Turingmaschine*. $L(M_g) := \{w \mid M_g \text{ schreibt } \sqcup\, w \sqcup \text{ auf das erste Band}\}$, M_g generiert die Sprache $L(M_g)$.

Satz 3.13 Sei L eine Sprache. Es existiert eine *akzeptierende* Turingmaschine M_a sodass $L(M_a) = L$ genau dann, wenn eine *generierende* Turingmaschine M_g existiert sodass $L(M_g) = L$.

Beweisidee '$\Rightarrow$': Man kann eine generierende TM aus einer akzeptierenden TM bauen, indem man auf einem Hilfsband systematisch Wörter aus Σ^* generiert, mit der akzeptierenden TM testet und auf das Ausgabeband kopiert.

'$\Leftarrow$': Man kann eine akzeptierende TM bauen, indem man auf einem Hilfsband eine generierende TM laufen lässt und alle generierten Wörter mit dem Eingabewort vergleicht.

Definition 3.14 Eine Sprache L heißt *rekursiv aufzählbar* (kurz r.a.) genau dann, wenn eine Turingmaschine existiert die diese Sprache akzeptiert bzw. generiert.

Definition 3.15 Eine Sprache L heißt *rekursiv*, wenn sowohl L als auch ihr Komplement $\Sigma^* \backslash L$ rekursiv aufzählbar sind.

Beispiel Die Sprache $\{0^n 1^n \mid n \in \mathbb{N}\}$ ist r.a., da eine TM existiert die diese Sprache akzeptiert (siehe Sektion 3.2). Wegen Satz 3.13 muss daher auch eine generierende TM existieren.

Auch das Komplement ist r.a., man muss nur in der TM die "—"-Einträge und die "$(q_4, \sqcup, R)$"-Einträge vertauschen. Also ist diese Sprache auch rekursiv.

4 Probleme und Entscheidbarkeit

4.1 Definitionen

Definition 4.1 (Problem) Ein *Problem* P ist eine aussagenlogische Formel (siehe [3]) mit den freien Variablen $(v_1, \ldots, v_n)$. Eine *Instanz* eines Problems ist eine konkrete Belegung dieser Variablen mit Werten. Die *Antwort* auf eine Instanz eines Problems ist der Wahrheitswert der Formel. Sie wird mit $P(v_1, \ldots, v_n)$ bezeichnet und kann die Werte "JA" und "NEIN" annehmen.

Definition 4.2 Sei P ein Problem. $L_P := \{(v_1, \ldots, v_n) \mid P(v_1, \ldots, v_n) = \text{"JA"}\}$ ist die Sprache eines Problems.

Definition 4.3 Ein Problem dessen Sprache rekursiv ist heißt *entscheidbar*, sonst heißt es *unentscheidbar*. Ist die Sprache nur r.a., nennt man das Problem auch *semientscheidbar*.

Oft nennt man auch eine Sprache oder Menge L (un)entscheidbar, gemeint ist dann das Problem $x \in L$.

Beispiel $P := x$ ist eine Quadratzahl. Eine Instanz diese Problems ist "4 ist eine Quadratzahl", die Antwort auf diese Instanz ist "JA". Die Sprache L_P ist die Menge aller Quadratzahlen. Da man ganz einfach ein Computerprogramm schreiben kann dass bestimmt ob eine Zahl eine Quadratzahl ist oder nicht ist L_P rekursiv und damit P entscheidbar.

Nach dem letzten Beispiel stellt sich die Frage: Kann man nicht jedes Problem so einfach lösen? Gibt es überhaupt unentscheidbare Probleme? Um diese Frage zu beantworten müssen wir noch etwas ausholen.

4.2 Die Diagonalsprache

Lemma 4.4 $\|\Sigma^*\| = \aleph_0$

folgt direkt aus der Definition von Σ^*. Daraus und aus Lemma 3.10 folgt

Lemma 4.5 Es gibt abzählbar viele Turingmaschinen.

Also können jedem Wort aus Σ^* bzw. jeder TM eine Zahl $i \in \mathbb{N}$ bijektiv zuordnen, indem wir sie zum Beispiel zuerst der Länge nach und dann alphabetisch sortieren. Wir bezeichnen diese Wörter mit w_i und die Turingmaschinen mit M_i.

Definition 4.6 (Diagonalsprache) $L_d := \{w_i \mid i \in \mathbb{N} \wedge w_i \notin L(M_i)\}$

Satz 4.7 Die Diagonalsprache L_d ist nicht rekursiv aufzählbar.

Beweis (Wiederspruchsbeweis)

Wir nehmen an L_d wäre r.a., es existiert also eine TM M mit $L(M) = L_d$. Diese Turingmaschine muss irgendwo in der Ordnung von Lemma 4.5 sein, d. h. $\exists i \in \mathbb{N} : M = M_i$.

Wir nehmen nun dieses i und betrachten das Wort w_i.

$$w_i \in L_d \underset{4.6}{\iff} w_i \notin L(M_i) \underset{L(M_i)=L_d}{\iff} w_i \notin L_d$$

führt auf einen Wiederspruch. Also muss die Annahme dass L_d r.a. ist falsch sein.

4.3 Das Akzeptierungsproblem

Definition 4.8 Sei M eine TM, $\langle M \rangle$ ihre Codierung in Σ^* und w ein Wort. "M akzeptiert w" heißt das *Akzeptierungsproblem*. Die Sprache dazu ist $L_u := \{\langle M \rangle w \mid M \text{ akzeptiert } w\}$ und heißt *Universalsprache*.

Satz 4.9 Das Akzeptierungsproblem ist unentscheidbar.

Beweis (1. Teil) Die Universalsprache ist natürlich r.a., da wir eine TM bauen können, die M simuliert und auf der Eingabe w laufen lässt. Wenn w von M akzeptiert wird, akzeptieren wir $\langle M \rangle w$.

Das Komplement der Universalsprache ist aber nicht r.a.!

falscher Beweis Das Komplement der Universalsprache ist r.a., da wir eine TM bauen können, die M simuliert und auf der Eingabe w laufen lässt. Wenn w von M nicht akzeptiert wird, akzeptieren wir $\langle M \rangle w$.

Wo ist der Fehler? Wenn eine TM einen akzeptierenden Zustand erreicht wissen wir sofort, dass sie das Wort akzeptiert. Wenn sie allerdings weder einen akzeptierenden Zustand erreicht, noch terminiert, wissen wir nicht ob sie das Wort wirklich nicht akzeptiert oder ob wir einfach nur länger warten müssen.

Beweis (2. Teil) (Wiederspruchsbeweis) Wir nehmen an das Akzeptierungsproblem wäre entscheidbar, also ist L_u rekursiv und wir haben eine TM M_u die L_u entscheidet, d. h. sie liefert immer in endlicher Zeit eine definitive Antwort "JA" oder "NEIN".

Jetzt konstruieren wir eine TM die die Diagonalsprache L_d akzeptiert. Sei w unser Eingabewort. Wir suchen uns den Index i sodass $w = w_i$ und die zugehörige TM M_i (w_i und M_i wie in 4.6 definiert).

Jetzt müssen wir nur noch unsere TM M_u auf der Eingabe $\langle M_i \rangle w_i$ laufen lassen und wir wissen ob $w_i \in L_d$. Das ist ein Wiederspruch zu Satz 4.7.

4.4 Das Halteproblem

Definition 4.10 Seien M, $\langle M \rangle$ und w wie in Definition 4.8. "Die Berechnung von M mit dem Wort w terminiert" heißt das *Halteproblem*. Die Sprache dazu ist:

$$L_h := \{\langle M \rangle w \mid M \text{ terminiert auf } w\}$$

Satz 4.11 Auch das Halteproblem ist unentscheidbar.

Beweis (Wiederspruchsbeweis)

L_h ist natürlich r.a., das Komplement jedoch nicht.

Wir nehmen an L_h wäre entscheidbar, d. h. wir können mit einer TM entscheiden ob M terminiert. Wenn M auf w nicht terminiert, kann sie auch w nicht akzeptieren (siehe Lemma 3.9). Wenn M jedoch auf w terminiert, funktioniert auf einmal der falsche Beweis bei Satz 4.9.

Damit haben wir durch Fallunterscheidung das Akzeptierungsproblem gelöst, was eigentlich nicht möglich sein sollte. Daher muss L_h unentscheidbar sein.

4.5 Weitere unentscheidbare Probleme

Analog kann man zeigen, dass auch das eingeschränkte Akzeptierungsproblem "M akzeptiert ϵ" und das eingeschränkte Halteproblem "M terminiert auf ϵ" unentscheidbar sind.

Definition 4.12 Eine *Eigenschaft* von Sprachen S ist eine Menge von Sprachen. Eine *nicht-triviale* Eigenschaft ist eine Eigenschaft, für die sowohl Sprachen L mit $L \in S$ als auch mit $L \notin S$ existieren.

Satz 4.13 (Satz von Rice) Jede nicht-triviale Eigenschaft S von Sprachen ist unentscheidbar.

Beweisidee Analog zu den vorhergehenden Beweisen kann man jede nicht-triviale Eigenschaft auf das eingeschränkte Halteproblem zurückführen. Details siehe [2].

Damit haben wir gleich eine ganze Klasse von unentscheidbaren Problemen gefunden.

Konsequenzen Aus dem Satz von Rice folgt wegen Lemma 3.7, dass die Korrektheit von Algorithmen (z. B. Computerprogrammen) unentscheidbar ist!

5 Zusammenfassung

In diesem Artikel wurde mit elementaren Zusammenhängen aus der Mengenlehre und der Prädikatenlogik plausibel dargelegt, dass Probleme existieren, die prinzipiell nicht lösbar sein können. Das wurde mittels des formalen Konzepts der Turingmaschine erreicht, exemplarisch dargestellt und in Folge anhand konkreter Beispiele bewiesen.

Weiters wurde mit Hilfe des Satzes von Rice ein Ausblick auf weitere unentscheidbare Probleme gegeben, von denen einige nicht nur rein theoretische Konstruktionen sind, sondern auch praktische Bedeutung haben. Das wohl überraschendste Resultat ist, dass die Korrektheit von Algorithmen im Allgemeinen unentscheidbar ist.

Literatur

[1] Kurt Gödel. Über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter Systeme. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

[2] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Trans. Amer. Math. Soc.*, 89:25–59, 1953.

[3] Wolfgang Windsteiger und Bruno Buchberger. Predicate Logic as a Working Language. Skriptum zur Vorlesung, Universität Linz, 2003.

[4] Franz Winkler. Formale Grundlagen der Informatik 2 (Theoretische Informatik). Skriptum zur Vorlesung, Universität Linz, 2002.

[5] Renate Winter. *Theoretische Informatik*. Oldenbourg Wissenschaftsverlag, 2002.