Satz von Rice: Grundlagen, Beweis und Implikationen

René Filip[†]

***Abstract*—Der Satz von Rice hat in der Informatik weitreichende Konsequenzen, denn er sagt aus, dass es kein allgemeines Computerprogramm gibt, das überprüft, ob der eigene Code eine gewünschte Eigenschaft bezüglich seiner Funktion berechnet oder nicht. Nachdem wir den Satz von Rice mittels Reduktion auf das Halteproblem beweisen, zeigen wir die weitrechenden und zunächst nicht ersichtlichen Konsequenzen. Als Einführung zum Reduktionsbeweis zeigen wir jedoch zunächst, dass selbst die Sprache, die Programmcode in einer polynomiellen Laufzeit beinhaltet, nicht entscheidbar ist und beweisen dies ebenfalls mittels Reduktion auf das Halteproblem.**

***Index Terms*—Satz von Rice, Theoretische Informatik, Entscheidbarkeit, Halteproblem.**

I. Einführung

Computer sind heutzutage mächtige Werkzeuge, denn sie können bestimmte Probleme und Aufgaben bereits schneller lösen als das menschliche Gehirn. 1997 schlug der Schachcomputer "Deep Blue" den Weltmeister "Garry Kasparov" im berühmten "Game 6" [TODO]. 19 Jahre später entwickelte Google das erste Computerprogramm "AlphaGo", das einer der besten Go-Spielern "Lee Sedol" besiegte [TODO]. Man könnte meinen, dass mit nur genügend Zeit und genug schlauen Entwicklern, jedes Problem von einem Computer gelöst werden könnte. Das Halteproblem zeigt jedoch, dass dies nicht immer der Fall ist. Der **Satz von Rice** geht dabei sogar noch einen Schritt weiter, denn er erschlägt viele Probleme und Fragestellungen, die für einen Informatiker sehr interessant gewesen wären. Anstatt eine explizite Lösung zu geben, sagt er aus, dass bestimmte Sprachen nicht entscheidbar sind und es daher niemals ein kluges Computerprogramm geben wird, dass jede Eingabe richtig bearbeiten könnte.

Genauer gesagt: Sobald wir überprüfen möchten, ob unser Programmcode eine Eigenschaft bezüglich seiner zu berechnenden Funktion erfüllt oder nicht, können wir darüber keine Aussage treffen. In Kapitel 6 werden wir diese Aussage formalisieren und beweisen. Danach zeigen wir die Konsequenzen die aus dem Satz von Rice folgen in Kapitel IV. Bevor wir jedoch den Satz mittels Reduktion beweisen, stellen wir in Kapitel II noch einen anderen Reduktionsbeweis vor.

II. Entscheidbarkeit der Sprache $O(n^k)$

Zunächst beweisen wir, dass man im allgemeinen nicht entscheiden kann, ob Programmcode (z.b. in Java) eine polynomielle Laufzeit besitzt oder nicht.

[†]DHBW Karlsruhe, TINF13B1, Seminar Theoretische Informatik 2016

Definition 1 (Sprache P^k_{Java})**.** *Seien die Sprachen P^1_{Java}, P^2_{Java}, …, P^k_{Java} wie folgt definiert:*

$$P^1_{Java} = \{P \mid \textit{Java-Programm } P \textit{ hat Zeitkomplexität} \subseteq O(n)\}$$
$$P^2_{Java} = \{P \mid \textit{Java-Programm } P \textit{ hat Zeitkomplexität} \subseteq O(n^2)\}$$
$$\vdots$$
$$P^k_{Java} = \{P \mid \textit{Java-Programm } P \textit{ hat Zeitkomplexität} \subseteq O(n^k)\}$$

Das bedeutet, dass die Sprache P^1_{Java} alle Javaprogramme beinhaltet, für die dessen Programme eine Laufzeit von $O(n)$ oder besser besitzen. Gleichbedeutendes gilt für andere Werte für k.

Satz 2. *Die Sprachen $P^1_{Java}, P^2_{Java}, \ldots, P^k_{Java}$ sind nicht entscheidbar.*

Wir können nicht entscheiden, ob ein gegebener Programmcode zu einen dieser Sprachen gehört oder nicht. Das heißt, es ist nicht möglich, ein kluges Computerprogramm zu schreiben, das die (polynomielle) Laufzeit eines beliebigen Programmcodes berechnet und ausgibt.

Wir beweisen den Satz mittels Reduktion auf das Halteproblem. Wir nehmen an, dass ein kluges Computerprogramm P_P existiert, welches überprüft, ob ein beliebiges Java-Programm P' die Laufzeit $O(n)$ oder besser besitzt oder nicht. Das heißt, P_P entscheidet die Sprache P^1_{Java} für $k = 1$. Dann transformieren wir aus einem Halteproblem (P, E) ein Java-Programm $P' = f(P, E)$, für welches P_P das Halteproblem anhand seiner Ausgabe lösen würde. Würde P_P die Sprache P^1_{Java} entscheiden, dann hätten wir auch das Halteproblem (P, E) entschieden. Wir wissen jedoch, dass das Halteproblem nicht entscheidbar ist und daher dann auch nicht die Sprache P^1_{Java} entscheidbar.

Beweis. Gegeben sei ein Schritt-Simulator, der wie folgt funktioniert:

Algorithm 1 Schritt-Simulator

Input: Programm P, Eingabe E, Anzahl an Schritte n

```
for i = 1 to n do
    Simuliere Schritt i von P auf E
    if P hält then
        Gehe in Endlosschleife
    end if
end for
```

Als Eingabe nimmt er ein Computerprogramm P, eine dazu zugehörige Eingabe E und eine Anzahl an Schritten n und simuliert P auf E in diesen n Schritten. Hält P auf E

innerhalb den n Schritten, geht der Schritt-Simulator in eine Endlosschleife. Andernfalls ist der Schritt-Simulator genau n Schritte beschäftigt.

Wir können aus einem Halteproblem (P, E) nun das Programm $P' = f(P, E)$ konstruieren und dessen Verhalten analysieren:

Algorithm 2 Programm P'

Input: Eingabe x

1: Schritt-Simulator(P, E, Länge(x))
2: **return** 1

Zunächst rufen wir den Schritt-Simulator(P, E, n) auf und machen die Anzahl der Schritte n abhängig von der Eingabe $x = \text{Länge}(x)$. Bei einer längeren Eingabe sind dann auch die Anzahl an Schritte größer. Falls der Schritt-Simulator terminiert, geben wir 1 zurück.

Das Verhalten von P' können wir nun leicht beschreiben:

1) P_P entscheidet $P' \in P^1_{Java}$: P' besitzt eine Laufzeit von $O(n)$ oder besser. Daraus folgt, dass für kein n (und daher auch x) der Schritt-Simulator in eine Endlosschleife gegangen ist, denn sonst wäre die Laufzeit sicher nicht $\subseteq O(n)$. Dies bedeutet jedoch wiederum, dass das Programm P unter der Eingabe E niemals anhält, denn sonst hätten wir irgendwann eine endliche Anzahl an Schritte gefunden, für die P unter E gehalten hätte. Es gilt also:

$$\begin{aligned} P' \in P^1_{\text{Java}} &\Leftrightarrow P' \subseteq O(n) \subseteq O(n^k) \\ &\Leftrightarrow P' = f(P, E) \text{ geht für kein } n \text{ in Endlosschleife} \\ &\Leftrightarrow (\text{Programm } P, \text{Eingabe } E) \in \overline{H_{\text{Java}}} \end{aligned}$$

2) P_P entscheidet $P' \notin P^1_{Java}$: P' besitzt eine schlechtere Laufzeit als $O(n)$. Da der Schritt-Simulator nur eine schlechtere Laufzeit als $O(n)$ "erzeugen" kann, wenn er in eine Endlosschleife gerät, muss das Programm P unter Eingabe E in endlich vielen Schritten n ab einer bestimmten Länge von x angehalten haben. Demnach:

$$\begin{aligned} P' \notin P^1_{\text{Java}} &\Leftrightarrow P' \notin O(n) \subseteq O(n^k) \\ &\Leftrightarrow P' = f(P, E) \text{ geht für } n \text{ in Endlosschleife} \\ &\Leftrightarrow (\text{Programm } P, \text{Eingabe } E) \in H_{\text{Java}} \end{aligned}$$

Wenn also P' in P^1_{Java} liegt, dann hält P nicht auf E. Wenn P' nicht in P^1_{Java} liegt, dann hält P auf E. Das Halteproblem ist jedoch nicht entscheidbar, also ist auch P^1_{Java} nicht entscheidbar.

□

Der Beweis für andere Programmiersprachen und Laufzeiten läuft äquivalent ab und ist dem aufmerksamen Leser als Übung überlassen.

III. Satz von Rice

Definition 3 (Sprache $L_{\text{Java}}(S)$)**.** *Sei $\mathcal{R}$ die Menge aller berechenbaren Funktionen und S eine nicht-triviale Teilmenge von R. Das heißt, $\emptyset \neq S \subset R$. Dann können wir folgende Sprache $L_{Java}(S)$ konstruieren:*

$$L_{Java}(S) = \{\textit{Javacode} \mid \textit{Programm berechnet Funktion aus } S\}$$

Die Definition für andere Programmiersprachen verläuft nach dem gleichen Schema.

Die Menge S beinhaltet also mindestens ein Element (eine Funktion) und ist niemals gleich $\mathcal{R}$. Dabei ist wichtig zu verstehen, dass es sich hier tatsächlich um Funktionen handelt und nicht um die Wertepaare der Funktionen.

Beispiel 4. *S besteht aus einem Element:*

$$S = \{f(x) = x^2\}$$

Die Menge S besteht aus einem Element, nämlich der Funktion $f(x) = x^2$. Für $f(x)$ können bei $f\colon \mathbb{R} \to \mathbb{R}$ aber unendlich viele Wertepaare besitzen.

Beispiel 5. *S besteht aus mehr als einem Element:*

$$S = \{f \mid f \textit{ ist total}\}$$

S darf auch mehrere Funktionen aus $\mathcal{R}$ beinhalten. Dadurch können wir in Kapitel IV interessante Aussagen treffen.

Aus S konstruieren wir die Sprache $L_{\text{Java}}(S)$. Diese beinhaltet jeden Javacode, dessen Programm irgendeine Funktion aus S berechnet. Im Beispiel 4 enthält $L_{\text{Java}}(S)$ jeden möglichen Javacode, der x^2 berechnet.

Satz 6 (Satz von Rice)**.** *Die Sprache $L_{Java}(S)$ ist nicht entscheidbar.*

Der Satz ist äquivalent für andere Programmiersprachen.

Für kein mögliches S ist also $L_{\text{Java}}(S)$ entscheidbar. Damit erschlägt der Satz von Rice viele Sprachen, für die man sonst immer einen eigenen Beweis bezüglich der Entscheidbarkeit/Unentscheidbarkeit finden hätte müssen.

Der Beweis von 6 verwendet wieder die Reduktion auf das Halteproblem.

Beweis. Wir nehmen an, dass P_S die Sprache $L(S)$ entscheiden kann, also ob ein bestimmter Programmcode P' zu der Sprache $L(S)$ gehört oder nicht. Dann konstuieren wir ein Programmcode P', für das P_S ein Halteproblem (P, E) lösen würde. Das Halteproblem ist jedoch nicht entscheidbar, also kann auch P_S das Halteproblem nicht entscheiden und daher ist die Sprache $L(S)$ ist nicht entscheidbar.

Der Programmcode für P' ist im Vergleich zum oberen Beweis noch einfacher. Eine kleine Schwierigkeit ist jedoch, dass wir eine Fallunterscheidung durchführen müssen. Sei u die überall undefinierte Funktion. u gibt für jede Eingabe, die sie erhält, undef zurück. Es ist leicht zu sehen, dass u in $\mathcal{R}$ liegt. Nun kann u entweder in S liegen oder nicht. Abhängig vom Fall, können wir dann sehen, wie die Reduktion verläuft.

Desweiteren nehmen wir wieder an, dass wir ein Java-Simulator zur Verfügung haben, der ein Programm P unter der Eingabe E simulieren kann.

Algorithm 3 Java-Simulator

Input: Programm P, Eingabe E

1: Simmuliere P auf E

Falls das Programm P unter E in eine Endlosschleife gerät, verfängt sich auch der Java-Simulator in einer Endlosschleife.

1) Fall 1: $u \notin S$: Fall 1 entspricht dem Beispiel 4, denn dort besteht S nur aus f und nicht auch noch u. Wir konstruieren das Programm P':

Algorithm 4 Programm P' für Fall 1 $u \notin S$

Input: Eingabe x
1: Java-Simulator(P, E)
2: Berechne f // $f \in S$

f wählen wir so, dass es in S liegt. Dies ist erlaubt, da $S \neq \emptyset$ per Definiton 3 gilt.

Wie verhält sich P'? Hält der Java-Simulator, dann berechnet P' die Funktion $f \in S$. Hält er nicht, dann verfängt sich P' in einer Endlosschleife und berechnet daher $u \notin S$.

Falls P_S als Ergebnis zurückliefert, dass P' in $L(S)$ liegt, dann muss das Programm P' in die zweite Zeile gelangt sein und f berechnen, da er andernfalls in Zeile 1 in einer Endlosschleife wäre und $u \notin S$ berechnen würde. Das bedeutet wiederum, dass das Programm P unter Eingabe E gehalten haben muss. Wir stellen fest:

$$P' \in L(S) \Leftrightarrow P' \text{ berechnet } f \in S \Leftrightarrow (P, E) \in H$$

Falls P' nicht in $L(S)$ liegt, dann erreichte P' auch nicht Zeile 2 und P hält nicht auf E:

$$P' \notin L(S) \Leftrightarrow P' \text{ berechnet } u \notin S \Leftrightarrow (P, E) \in \overline{H}$$

Algorithm 5 Programm P' für Fall 2 $u \in S$

Input: Programm P, Eingabe E, Anzahl an Schritte n
1: Java-Simulator(P, E)
2: Berechne f // $f \notin S$

2) Fall 2: $u \in S$: hier steht noch mehr text

□

IV. Konsequenzen

Die Konsequenzen von Satz von Rice sind weitreichender als man zunächst vermuten würde. Allgemein lässt sich nämlich sagen, dass sobald ein Programmcode eine bestimmte Eigenschaft bezüglich einer (mathematischen) Funktion erfüllt (d.h. nicht von der Syntax o.ä. abhängig ist) und ein anderer Programmcode diese Eigenschaft nicht erfüllt, dann lässt sich im allgemeinen keine Aussage dazu treffen, welcher Programmcode nun welche Eigenschaft (nicht) erfüllt.

Im Folgenden wollen wir diese Aussage mit mehreren Beispielen für S erläutern:

A. Berechnung einer mathematischen Funktion

TODO

$$S = \{f(x) = x^2\} \qquad S = \{f(x) = e^x\}$$
$$S = \{f(x) = x!\} \qquad S = \{f(x) = 1\}$$

B. Eigenschaften von mathematischen Funktionen

TODO

$$S = \{f \mid f \text{ ist injektiv}\}$$
$$S = \{f \mid f \text{ ist surjektiv}\}$$
$$S = \{f \mid f \text{ ist bijektiv}\}$$
$$S = \{f \mid f \text{ ist eine konstante Funktion}\}$$
$$S = \{f \mid f \text{ ist ein Prädikat}\}$$
$$S = \{f \mid f(x) \text{ ist durch 3 teilbar}\}$$

C. Halteproblem

TODO

$$S = \{f \mid f \text{ ist definiert bei Eingabe } E\}$$

TODO

$$S = \{f \mid f \text{ ist total}\}$$

TODO

D. Korrektheit bzw. Äquivalität

TODO definiere f

$$S = f$$

TODO konsequenz

E. Berechnung spezieller Funktionen

Collatz Funktion

$$f(n) = \begin{cases} n/2, & \text{wenn } n \text{ gerade,} \\ 3n+1, & \text{wenn } n \text{ ungerade.} \end{cases}$$

TODO

$$S = \{1\}$$

Fibonacci Zahlen

$$\text{Fibonacci}(n) = n\text{-te Fibonacci Zahl}$$

TODO

$$S = \{\text{Fibonacci}(n)\}$$

References