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 Funktion berechnet oder nicht. Nachdem wir die Grundlagen und Definitionen geklärt haben, beweisen wir den Satz mittels "Many-to-One" Reduktion auf ein einfacheres Problem, für das sich ein direkter Beweis finden lässt. Am Ende gehen wir nochmal kurz auf dessen Implikationen und Bedeutung ein.**

***Index Terms*—Rice's Theorem, Theoretical computation, computability, decidability, Post correspondence problem, Halting problem.**

I. Introduction

A. Definitions and Notation

Definition 1 (Turingmaschine): Eine Turingmaschine (TM) ist definiert durch ein Septuple $(Q, \Gamma, B, \Sigma, \delta, q_0, F)$:

1) endliche Zustandsmenge Q
2) endliche Bandalphabet Γ
3) Leerzeichen $B \in \Gamma$
4) endliche Eingabealphabet $\Sigma \subseteq \Gamma \setminus B$
5) Zustandsübergangsfunktion $\delta\colon Q \times \Gamma \to Q \times \Gamma \times (L, R, N)$
6) Anfangszustand $q_0 \in Q$
7) Menge der akzeptierenden Endzustände $F \subseteq Q$

Although we can show that a Turing machine can simulate any computer program [1], their "programs" (i.e. their transition functions δ) are static. Thus, these Turing machines are only "special purpose computers". Instead, we would rather like to work with "general purpose computer". By creating an universal Turing machine, which takes as input a computer program $\langle M\rangle$ *and* a word $w \in \{0,1\}^*$ to accept, we can solve our problem. In specific, $\langle M\rangle$ will be a Gödel number defined as followed:

Definition 2 (Gödel Number): Let there be a Turing machine $M = (\{q_1, \ldots, q_t\}, \{0,1,B\}, B, \{0,1\}, \delta, q_1, \{q_2\})$. Furthermore, let $X_1 = 0$, $X_2 = 1$, $X_3 = B$, $D_1 = L$, $D_2 = N$, $D_3 = R$. We can describe the the z-th row of δ's transition table of length s as following:

$$\delta(q_i, X_j) = (q_k, X_l, D_m) \Longrightarrow \text{code}(z) = 0^i 1 0^j 1 0^k 1 0^l 1 0^m$$

where $i, k \geq 1$ and $j, l, m \in \{1,2,3\}$. Now, the Gödel Number $\langle M\rangle$ of the Turing machine M is given by

$$111\,\text{code}(1)11\,\text{code}(2)11 \ldots 11\,\text{code}(s)111$$

We can easily see that a program starts and ends with the string 111 and each "line" of code is separated by the string 11. [2] describes the functionality and properties of this Turing machine more in detail.

B. Principles

First, we define what computability and decidability is in perspective of theoretical computation. Then, we summarize what we have learned from previous presentations and outline the most important aspects from them.

Definition 3 (Computability): A function $f\colon \Sigma^* \to \Gamma^*$ is called *computable* if there is a Turing machine M that takes $w \in \Sigma^*$ as input and calculates $f(w) \in \Gamma^*$ as output. [3]

Definition 4 (Decidability): A language $L \subseteq \Sigma^*$ is called *decidable* if there is a total Turing machine that accepts input w iff $w \in L$. [4]

A total Turing machine (also known as decider) halts (i.e. it's state becomes a final state) for every input it receives. That way, it always determines if a word w is part of the language L or not.

Note that decidability refers to languages and computability refers to functions. Although they sound equal, they are used in different contexts

Difference

Appendix A
Proof of the First Zonklar Equation

Some text for the appendix.

Acknowledgment

The author would like to thank Martin Holzer for reviewing the related presentation as well as this document.

The authors would like to thank…

References

[1] I. Wegener, "Theoretische informatik - eine algorithmenorientierte einführung." Berlin Heidelberg New York: Springer-Verlag, 2013, ch. 2, pp. 16 – 17.

[2] ——, "Theoretische informatik - eine algorithmenorientierte einführung." Berlin Heidelberg New York: Springer-Verlag, 2013, ch. 2, p. 18.

[3] ——, "Theoretische informatik - eine algorithmenorientierte einführung." Berlin Heidelberg New York: Springer-Verlag, 2013, ch. 2, p. 10.

[4] ——, "Theoretische informatik - eine algorithmenorientierte einführung." Berlin Heidelberg New York: Springer-Verlag, 2013, ch. 2, p. 11.

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