DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

# Seminar Theoretische Informatik

# Postsches Korrespondenzproblem

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

**Simon Schrodi**

Vortragsdatum 5. April 2019

| | |
|---|---|
| Bearbeitungszeitraum | 1 Semester |
| Matrikelnummer | 9746027 |
| Kurs | TINF 16B3 |
| Gutachter der Studienakademie | Prof. Dr. Heinrich Braun |

## Motivation

Ausgehend von Turingmaschinen und Automaten hat sich der Begriff der (Un)entscheidbarkeit eingeführt. Das Halteproblem ist das Standardbeispiel von unentscheidbaren Problemen. Mit Hilfe des Postschen Korrespondenzproblems (PKP) soll für andere Probleme die (Un)entscheidbarkeit bewiesen werden, indem das PKP auf die Probleme reduziert wird. Das PKP wurde von Emil Leon Post entwickelt. Emil Leon Post ist ein polnisch-amerikanischer Mathematiker und Logiker. 1936 entwickelte er ein Automatenmodell, das gleichmächtig wie die Turingmaschine ist. 1947 zeigt er, die Unentscheidbarkeit des PKP.

Mittels Reduktion können viele andere unentscheidbare Grammatikprobleme gezeigt werden. Gegeben zwei kontextfreie Grammatiken $G_1$, $G_2$, so sind folgende Probleme unentscheidbar:

- Ist $L(G_1) \cap L(G_2) = \emptyset$? (Schnittproblem)
- Ist $|L(G_1) \cap L(G_2) = \infty$? (Endlichkeitsproblem)
- Ist $L(G_1) \subseteq L(G_2)$? (Inklusionsproblem)
- Ist $L(G_1) = L(G_2)$? (Äquivalenzproblem)
- Ist $L(G_1)$ mehrdeutig? (Mehrdeutigkeitsproblem)
- ...

## Wiederholung

### Reduktion

Mit Hilfe der Reduktion wird ein Problem auf ein anderes Problem zurückgeführt. Die Reduktion geht ebenfalls wie das PKP auf Emil Leon Post zurück. Formal kann die Reduktion wie folgt definiert werden:

Seien $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ Sprachen. $A \leq B$ gdw. es gibt eine berechenbare Funktion $f : \Sigma^* \to \Gamma^*$, so dass $\forall x \in \Sigma^*$ gilt:

$$x \in A \Leftrightarrow f(x) \in B. \tag{0}$$

Falls $A \leq B$ und $A$ unentscheidbar, so ist auch B unentscheidbar.

### Konfiguration einer Turingmaschine

Die Konfiguration einer Turingmaschine ist ein Wort $k = \alpha z a \beta \in \Gamma^* Z \Gamma^*$, dabei ist $\alpha$ der Linkskontext, $z$ der momentane Zustand, $a$ das aktuell gelesene Zeichen und $\beta$ der Rechtskontext. Aufeinanderfolgende Konfigurationen werden durch # getrennt. Abb. 1 zeigt eine bildliche Darstellung einer Wortkonfiguration.

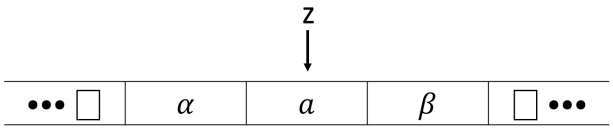


**Abbildung 1:** Bildliche Darstellung d. Konfiguration einer Turingmaschine

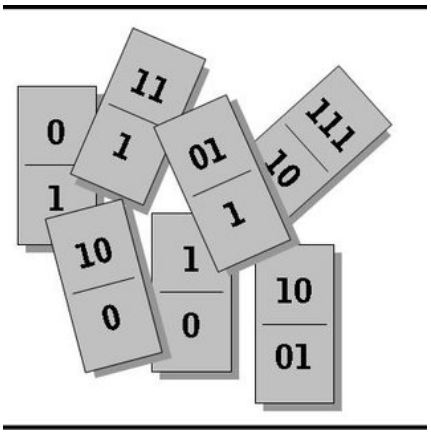


**Abbildung 2:** Bildliche Darstellung d. PKP mittels Dominosteine

## Einführung

**Definition.** *Eine Instanz des PKP besteht aus einer endlichen Folge*

$$K = [(x_1, y_1), ..., (x_k, y_k)], \tag{1.1}$$

*wobei $x_i, y_i \neq \epsilon$ über einem endlichen Alphabet $\Sigma$ sind. Es soll entschieden werden, ob es eine korrespondierende Folge*

$$i_1, ..., i_n \in [1, ..., k], n \geq 1 \tag{1.2}$$

*von Indizes, genannt* ***Lösung****, gibt, so dass gilt*

$$x_{i_1} x_{i_2} ... x_{i_n} = y_{i_1} y_{i_2} ... y_{i_n}. \tag{1.3}$$

Bildlich kann man sich das PKP als Dominosteine vorstellen (s. Abb. 2). Dabei befindet sich auf der oberen Seite eines Dominosteins ein Wort $x_i$ und auf der unteren Seite das Wort $y_i$. Nun gibt es so viele Dominosteine, wie es Wortpaare in einer Instanz des PKP gibt. Einfachheitshalber wird angenommen, dass es von den Dominosteinen beliebig viele gibt. Ziel ist es nun die Dominosteine aneinanderzulegen, so dass die Konkatenation der Worte der oberen Seiten der Dominosteine genau gleich der unteren Seite der Dominosteine ist.

Im folgenden werden einige Beispiele für das PKP beschrieben, die die Definition veranschaulichen.

Gegeben: $K = [(1, 111), (10111, 10), (10, 0)]$. Gesucht: Korrespondierende Indexfolge. Lösung: Die Indexfolge ist $(2, 1, 1, 3)$:

$$\underbrace{10111}_{x_2}\underbrace{1}_{x_1}\underbrace{1}_{x_1}\underbrace{10}_{x_2} = 101111110 = \underbrace{10}_{y_2}\underbrace{111}_{y_1}\underbrace{111}_{y_1}\underbrace{0}_{y_2}$$

Gegeben: $K = [(10, 101), (011, 11), (101, 011)]$. Lösung: Es gibt keine passende Indexfolge (**Zugzwangargument**)!

- Jede potentielle Lösung muss mit $i_1 = 1$ beginnen.
- Immer wenn $y$-Sequenz eine 1 Vorsprung hat, ist die einzig mögliche Fortsetzung:

$$x - Sequenz: \ ... \ \underbrace{101}_{x_3}$$
$$y - Sequenz: \ ... \ 1\underbrace{011}_{y_3}.$$

  Die $y$-Sequenz hat wiederum eine 1 Vorsprung. Dadurch wiederholt sich dieser Schritt unendlich häufig. Daher gibt es für das PKP keine Lösung.

Gegeben: $K = [(001, 0), (01, 011), (01, 101), (10, 001)]$. Lösung: $(2, 4, 3, 4, 4, 2, 1, 2, 4, 3, 4, 3, ...)$ mit 66 Indizes.

In den Beispielen zeigt sich, dass dieses scheinbar einfache Problem, gar nicht so einfach lösbar ist.

**Definition.** *Das modifizierte PKP (MPKP) ist definiert wie das PKP. Zusätzlich wird die Bedingung* $i_1 = 1$ *festgelegt.*

*Bemerkung:* **PKP und MPKP sind semi-entscheidbar.** Bei den Beispielen verwenden wir einen naiven Algorithmus. Wir bauen einen kombinatorischen Entscheidungsbaum auf, den wir absuchen. Die Baumsuche führen wir mittels Breitensuche durch. Tiefensuche ist nicht möglich, da sich die Tiefensuche in einem Zweig des Baums aufhängt, falls es in diesem Zweig keine Lösung gibt. Bei der Breitensuche dahingegen wird der Baum Schicht für Schicht durchsucht, so dass falls es eine Lösung gibt, diese auch gefunden wird. Falls es eine Lösung gibt, stoppt der Algorithmus. Falls es keine Lösung gibt, stoppt der Algorithmus hingegen nicht. Dies ist genau die Definition der Semi-Entscheidbarkeit.

## Beweis der Unentscheidbarkeit des Postschen Korrespondenzproblems

Um zu zeigen, dass $PKP$ unentscheidbar ist, wird das allgemeine Halteproblem für Turingmaschinen $H$ zunächst auf MPKP reduziert. Das MPKP wird dann auf $PKP$ reduziert. Zunächst wird gezeigt, dass:

$$H \leq MPKP, \tag{2.1}$$

indem eine Reduktionsabbildung $f$ gegeben wird, die die Eingaben vom allgemeinen Halteproblem $H = \{(p, w)|\text{Turingprogramm p angesetzt auf w hält}\}$, so auf Eingaben von MPKP abbildet,

| $\delta$ | 0 | 1 | $\square$ |
|---|---|---|---|
| $z_1$ | $(z_2, 1, R)$ | $(z_2, 0, L)$ | $(z_2, 1, L)$ |
| $z_2$ | $(z_f, 0, L)$ | $(z_1, 0, R)$ | $(z_2, 0, R)$ |
| $z_f$ | – | – | – |

dass:

$$(p, w) \in H(M_p) \Leftrightarrow f(p, w) \in MPKP \tag{2.2}$$

Die Idee des Beweises kann in folgende drei Ideen zusammengefasst werden:

- Stelle $x$- und $y$-Sequenzen als Konfigurationsfolgen von der Turingmaschine dar
- Die $y$-Sequenz hat immer eine Konfiguration Vorsprung
- Die $x$-Sequenz holt nach dem Stoppen von der Turingmaschine den Vorsprung ein

### Herleitung der Abbildungsregeln anhand eines Beispiels

Die Abbildungsregeln der Reduktion werden mittels eines Beispiels hergeleitet. Die Beschreibungen werden in den Bildunterschriften gegeben. Gegeben sei nun eine Turingmaschine $M_w = (Z, \Sigma, \Gamma, \delta, z_0, \square, E)$ und ein Eingabewort $w \in \Sigma^*$. Die Regeln $\delta$ sind durch obige Tabelle gegeben. Mittels von den Abbildungen 3 bis 12 werden die Abbildungsregeln hergeleitet.

x-Sequenz: #

y-Sequenz: #$z_1$01#

$M_w$: $z_1$01

**Abbildung 3:** Zunächst betrachten wir die Konfiguration der Turingmaschine $M_w$. Die Turingmaschine hat am Anfang die Konfiguration $z_1 01$. Verglichen zur vorherigen Konfigurationen (genauer keine Konfiguration) haben sich alle Zeichen verändert. Um nun der $y$-Sequenz einen Vorsprung von genau einer Konfiguration gegenüber der $x$-Sequenz zu geben, wird $\#z_1 01\#$ in die $y$-Sequenz geschrieben. Für die $x$-Sequenz wird # geschrieben. Diese Regel wird **Anfangsregel** $(\#, \#z_1 01\#)$ genannt. Nun könnte man natürlich annehmen, dass die # (bei der $y$-Sequenz das hintere) weggelassen werden könnten. Jedoch wird bei der Definition des PKP gesagt, dass die Worte nicht leer sein dürfen. Daher ist ein # notwendig.

x-Sequenz: $\#\mathbf{z_1 0}$

y-Sequenz: $\#z_1 01\#\mathbf{1z_2}$

$M_w: z_1 01 \vdash \mathbf{1z_2 1}$

**Abbildung 4:** Nun macht die Turingmaschine einen Übergang. Bei der Konfiguration hat sich $z_1 0$ zu $1z_2$ verändert. Der weitere Links -bzw. Rechtskontext bleibt unverändert. Daher wird $1z_2$ an die $y$-Sequenz angehängt. Nun soll die $x$-Sequenz genau eine Konfiguration Vorsprung haben, daher muss ebenfalls eine Zeichenkette der gleichen Länge an der $x$-Sequenz angehängt werden. Die ist in diesem Fall $z_1 0$. Dies wird als Rechts-**Überführungsregel** bezeichnet und kann generell mit $(za, cz'),\ falls\ \delta(z, a) = (z', c, R)$ formal definiert werden.

x-Sequenz: $\#z_1 01$

y-Sequenz: $\#z_1 01\#1z_2 1$

$M_w: z_1 01 \vdash \mathbf{1z_2 1}$

**Abbildung 5:** Nun wird die Konfiguration noch nicht ganz in den Sequenzen abgebildet. Um diese zu Komplettieren kann die 1 an beide Sequenzen angehängt werden. Diese Regel wird als **Kopierregel** bezeichnet und formal definiert durch $(a, a)$.

x-Sequenz: $\#z_1 01\#$

y-Sequenz: $\#z_1 01\#1z_2 1\#$

$M_w: z_1 01 \vdash \mathbf{1z_2 1}$

**Abbildung 6:** Um die Konfiguration wird nochmals die Kopieregel angewandt und # an beide Sequenzen angehängt.

x-Sequenz: $\#z_1 01\#1$

y-Sequenz: $\#z_1 01\#1z_2 1\#1$

$M_w: z_1 01 \vdash 1z_2 1 \vdash \mathbf{10z_1}$

**Abbildung 7:** Die vorherige Konfiguration ist durch die Sequenzen abgebildet. Nun macht die Turingmaschine eine weitere Überführung. Die Überführungsregel entspricht der zuvorigen Überführungsregel. Daher kann wiederum die Rechts-Überführungsregel angewendet werden. Zunächst wird der unveränderte Linkskontext (1) mit Hilfe der Kopieregel an beide Sequenzen angehängt.

x-Sequenz: $\#z_1 01 \# 1 \mathbf{z_2 1}$

y-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 1 \mathbf{0 z_1}$

$M_w$: $z_1 01 \vdash 1 z_2 1 \vdash \mathbf{1 0 z_1}$

**Abbildung 8:** Hier kann nun die Rechtsüberführungsregel angewendet werden, d.h. an die $x$-Sequenz wird $z_2 1$ und an die $y$-Sequenz $0 z_1$ angehängt.

x-Sequenz: $\#z_1 01 \# 1 z_2 1 \mathbf{\# 1 0 z_1 \#}$

y-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \mathbf{\# 1 z_2 01 \#}$

$M_w$: $z_1 01 \vdash \cdots \vdash 10 z_1 \vdash \mathbf{1 z_2 01}$

**Abbildung 9:** Zunächst wird die vorherige Konfiguration durch Anwendung der Kopieregel in den Sequenzen durch Anhängen von # komplettiert. Nun macht die Turingmaschine wieder eine Überführung. Dabei kann diesmal die Rechts-Überführungsregel nicht angewendet werden, da der Kopf der Turingmaschine ein Blank liest. Zunächst kann festgehalten werden, dass der Linkskontext bis auf die 0 unverändert bleibt. Der Rechtskontext ist nicht vorhanden und bleibt daher auch unberührt. Daher kann die 1 kopiert werden. Bei der Konfiguration hat sich $0 z_1$ zu $z_2 01$ verändert, daher wird zweites in die $y$-Sequenz geschrieben. Zusätzlich wird # noch geschrieben, da klar ist, dass kein weiteres Zeichen links folgt. Nun soll die $y$-Sequenz genau eine Konfiguration Vorsprung haben. Da sie dich Konfiguration um eins verlängert und der Vorsprung eingehalten werden muss, müssen drei Zeichen in die $x$-Sequenz ($0 z_1 \#$) geschrieben werden. Bei der Regel handelt es sich um eine Sonderregel. Formal ist diese Regel durch $(bz\#, z'bc\#),\ falls\ \delta(z, \square) = (z', c, L),\ \forall b \in \Gamma \backslash \{\square\}$ definiert.

x-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \mathbf{\# 1 z_2 01 \#}$

y-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \# 1 z_2 01 \mathbf{\# z_f 101 \#}$

$M_w$: $z_1 01 \vdash \cdots \vdash 1 z_2 01 \vdash \mathbf{z_f 101}$

**Abbildung 10:** Nun macht die Turingmaschine eine weiteren Schritt. Hier sind beide kennengelernten Überführungsregeln nicht anwendbar. Hier hat sich $1 z_2 0$ zu $z_f 10$ verändert. Der weitere Linkskontext (hier keiner) und Rechtskontext (1) bleibt unverändert. Daher wird $z_f 10$ zunächst an die $y$-Sequenz angehängt. Da der Vorsprung von einer Konfiguration eingehalten werden muss, muss an die $x$-Sequenz ebenfalls an drei Zeichen ($z_2 01$) angehängt werden. Diese Überführungsregel wird als **Links-Überführungsregel** bezeichnet und generell gilt: $(bza, z'bc),\ falls\ \delta(z, a) = (z', c, L),\ \forall b \in \Gamma$. Abschließend werden die 1 und # kopiert, um die Konfigurationen in den Sequenzen zu komplettieren.

x-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \# 1 z_2 01 \mathbf{\# z_f 1}$

y-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \# 1 z_2 01 \# z_f 101 \mathbf{\# z_f}$

**Abbildung 11:** Nun ist der Endzustand $z_f$ erreicht und die $x$-Sequenz soll den Vorsprung der $y$-Sequenz einholen. Hierfür werden Zeichen durch die **Löschregeln** $((az_f, z_f), (z_f a, z_f))$ gelöscht. Dabei wird an der $x$-Sequenz ein Zeichen mehr angehängt als bei der $y$-Sequenz. Dadurch wird der Vorsprung eingeholt.

x-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \# 1 z_2 01 \# z_f 101 \# z_f 01 \# z_f 1 \# z_f \# \#$

y-Sequenz: $\#z_1 01 \# 1 z_2 1 \# 10 z_1 \# 1 z_2 01 \# z_f 101 \# z_f\ 01 \# z_f 1 \# z_f \# \#$

**Abbildung 12:** Nun werden Kopierregeln und Löschregeln angewandt, bis nur noch der Endzustand $z_f$ in der Konfiguration vorkommt. Hier kann keine Löschregeln mehr angewandt werden und es wird daher eine sogenannte **Abschlussregel** $(z_f \# \#, \#)$ definiert, die dessen Anhängung die x-Sequenz gleich der y-Sequenz ist.

### Zusammenfassung

Mit Hilfe der folgenden Regeln wird jedes beliebige Paar $(M, w)$ in eine Folge $(x_1, y_1), ..., (x_k, y_k)$ überführt:

(i) **Anfangsregel:** $(\#, \#z_1x\#)$.

(ii) **Kopierregeln:** $\forall a \in \Gamma \cup \{\#\} : (a, a)$

(iii) **Überführungsregeln:** $\forall z \in Z \backslash E; \forall z' \in Z; \forall a, c \in \Gamma \backslash \{\square\}$:

$$\begin{aligned}
(za, cz'),\ & falls\ \delta(z, a) = (z', c, R) \\
(bza, z'bc),\ & falls\ \delta(z, a) = (z', c, L),\ \forall b \in \Gamma \\
(z\#, cz'\#),\ & falls\ \delta(z, \square) = (z', c, R) \\
(bz\#, z'bc\#),\ & falls\ \delta(z, \square) = (z', c, L),\ \forall b \in \Gamma \backslash \{\square\}
\end{aligned}$$

(iv) **Löschregeln:** $\forall z_f \in E; \forall a \in \Gamma \backslash \{\square\}$: $(az_f, z_f)$, $(z_fa, z_f)$

(v) **Abschlussregeln:** $\forall z_f \in E : (z_f\#\#, \#)$

### Nachweis der Reduktion

Es ist klar, dass diese Reduktionsabbildung berechnbar ist, da diese von einer Turingmaschine simuliert wird. Nun ist noch zu zeigen, dass die Äquivalenz $(p, w) \in H \Leftrightarrow f(p, w) \in MPKP$ gilt:

- $(p, w) \in H \Rightarrow f(p, w) \in MPKP$
  - Falls $(p, w) \in H$, erhalten wir irgendwann eine Lösung der Form $(k, k\alpha z_f\beta\#)$ mit $z_f \in E, \alpha, \beta \in \Gamma^*$
  - Mittels Kopierpaare und Löschpaare kann der Vorsprung $\alpha z_f\beta\#$ vermindert werden,
  - bis das Abschlusspaar angewendet werden kann, so dass $(k'z_f\#\#, k'z_f\#\#)$
- $f(p, w) \in MPKP \Rightarrow (p, w) \in H$
  - Es sei angenommen: $(p, w) \notin H \Rightarrow f(p, w) \notin MPKP$
  - Da kein Zustand $z_f \in E$ erreicht wird, werden keine Löschpaare angewendet, und
  - somit hat die $y$-Sequenz stets eine Konfiguration Vorsprung

### Beweis der Unentscheidbarkeit von PKP

Für den Beweis wird die Reduktion $MPKP \leq PKP$ gezeigt. Es sei ein Eingabeinstanz des MPKP über einem Alphabet $\Sigma$ gegeben. Für die Reduktionabbildung werden $\#, \$ \notin \Sigma$ als neue Symbole eingeführt. Die Reduktionabbildung kann wie folgt definiert werden:

$$f(K) = [(x'_0, y'_0), (x'_1, y'_1), ..., (x'_k, y'_k), (x'_{k+1}, y'_{k+1})]$$

mit

- $x'_0 = \#x'_1, x_{k+1} = \$, y'_0 = y'_1, y'_{k+1} = \#\$$

- $\forall i \in \{1, ..., k\}$ *gilt* $x'_i = x_i\#$ *bzw.* $y'_i = \#y_i$

Nun ist noch die Äquivalenz zu zeigen. Hier wird wiederum in zwei Schritten vorgegangen

- Instanz $K$ eines MPKP hat eine Lösung $\Rightarrow$ Instanz $f(K)$ eines PKP hat eine Lösung
  - Dies ist intuitiv klar, da MPKP wie PKP definiert ist mit der Zusatzbedingung $i_1 = 1$. Bei einem Wegfall dieser Zusatzbedingung hat das PKP die gleiche Lösung
  - Formaler gilt die obige Implikation, da die Wörter der Lösung des MPKP im PKP mit Rauten eingerahmt werden und das Wort mit dem Dollarzeichen endet
- Instanz $f(K)$ eines PKP hat eine Lösung $\Rightarrow$ Instanz $K$ eines MPKP hat eine Lösung
  - PKP muss aus Konstruktion mit erstem Wortpaar beginnen, da nur dieses Wortpaar gleiche Anfangssymbole hat
  - Durch Weglassen aller Rauten und dem Dollarzeichen entsteht die Lösung für eine Instanz des MPKP

## Unentscheidbarkeit von Grammatikproblemen

Die Unentscheidbarkeitsbeweise werden o.B.d.A. mittels Reduktion des MPKP auf die Grammatikprobleme bewiesen.

### Unentscheidbarkeit des Mehrdeutigkeitproblems

Zunächst wird motiviert, warum wir uns für die Mehrdeutigkeit respektive die Eindeutigkeit einer Grammatik interessieren. Die Mehrdeutigkeit einer Grammatik einer Programmiersprache würde dazu führen, dass mehrere Semantiken existieren.

Nun zum Beweis der Unentscheidbarkeit des Mehrdeutigkeitproblems. Es sei eine Instanz $K = [(x_1, y_1), ..., (x_k, y_k)]$ des MPKP über einem endlichen Alphabet $\Sigma$ und $I = \{i_1, ..., i_k\} \notin \Sigma$ gegeben. Es werden die Grammatiken $G_x = (V_x, T, P_x, S_x)$ und $G_y = (V_y, T, P_y, S_y)$ definiert. Dabei gilt für $T_x = \Sigma \cup I$ und $P_x = \{S_x \to i_1 S_x x_1 | ... | i_k S_x x_k | i_1 x_1\}$. Analog sind die Produktionsregeln für $G_y$ definiert, wobei $x_i$ durch $y_i$ ersetzt werden. Es ist zu bemerken, dass die Indexfolge gespiegelt wird. Eine weitere Bemerkung ist, dass die Grammatiken die $LL(2)$-Bedingung erfüllen. Die $LL(1)$-Bedingung wird nicht wegen der Regeln $i_1 S_x x_1$ und $i_1 x_1$ nicht erfüllt. Die $LL(2)$-Bedingung wird erfüllt, da dann für erstere $i_1 i_j$ für ein beliebiges $j$ und für zweitere Produktionregel $i_1 x_1$ in der First-Menge ist. D.h. die Schnittmenge der First-Follow-Mengen ist disjunkt.

Zusätzlich wird eine Grammatik $G_z$ aus den Grammatiken $G_x$ und $G_y$ erstellt. Dabei ssei $L(G_z) = L(G_x) \cup L(G_y)$ mit $P_z = \{S \to S_x | S_y\} \cup P_x \cup P_y$. Es ist klar, dass die Reduktionsabbildung berechenbar ist, da die Grammatik durch eine Turingmaschine simuliert werden kann. Nun ist die Äquivalenz zu zeigen:

- K hat eine Lösung $\Rightarrow$ G ist mehrdeutig
  - Beweis durch Gegenbeispiel: Angenommen $G_z$ ist eindeutig
  - MPKP aus Beispiel 1: $K = [(10111, 10), (1, 111), (10, 0)]$ hat Lösung $(1, 2, 2, 3)$

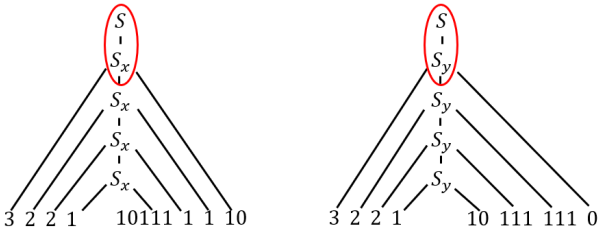


**Abbildung 13:** Ableitungsbäume mit selben Wort $w$

  - Die Ableitungsbäume in Abb. 13 sind unterschiedlich (rote Kreise). Dies steht im Widerspruch zur Annahme, dass $G_z$ eindeutig sei. Daher ist $G_z$ mehrdeutig

- $G$ ist mehrdeutig $\Rightarrow$ K hat eine Lösung
  - $G_z$ hat zwei **verschiedene** Ableitungsbäume mit **gleichem** Wort $w$
  - Fall 1: $S \to S_x \to w$ für beide Ableitungsbäume
    - D.h. $S_x$ ist mehrdeutig
    - Aber $G_x$ ist $LL(2)$. Widerspruch
  - Fall 2: $S \to S_x \to w$ bzw. $S \to S_y \to w$
    - Erster Teil der Worte ist gleich (Indexfolge)
    - Zweiter Teil der Worte ist gleich $\to$ Lösung für K

Damit ist die Unentscheidbarkeit des Mehrdeutigkeitproblems gezeigt.

### Unentscheidbarkeit des Schnittproblems

Analog zum vorherigen Beweis werden zwei Grammatiken $G_x$ und $G_y$ definiert. Nun ist noch die Äquivalenz zu zeigen:

- K hat eine Lösung $\Rightarrow G_x \cup G_y \neq \emptyset$
  - Analog zum vorherigen Beweis Fall 2 gibt es zwei Ableitungsbäume, bei denen der erster Teil gleich ist (Indexfolge) sowie der zweite Teil gleich ist (Lösung für K)
  - Daraus folgt, dass $G_x \cup G_y \neq \emptyset$
- K hat keine Lösung $\Rightarrow G_x \cup G_y = \emptyset$
  - Es gibt keine Ableitungsbäume, die das gleiche Wort produzieren, da es keine Lösung für K gibt
  - Daraus folgt, dass $G_x \cup G_y = \emptyset$

### Unentscheidbarkeit des Endlichkeitproblems

Analog zu den vorherigen Beweisen werden zwei Grammatiken $G_x$ und $G_y$ definiert. Nun ist noch die Äquivalenz zu zeigen:

- K hat eine Lösung $\Rightarrow G_x \cup G_y = \infty$

  - Falls K eine Lösung hat, dann hat K auch beliebig viele Lösungen, indem die Indexfolge wiederholt wird
  - MPKP aus Beispiel 1: $K = [(10111, 10), (1, 111), (10, 0)]$ hat Lösung $(1, 2, 2, 3)$
  - Dann sicher auch $(1, 2, 2, 3, 1, 2, 2, 3)$ usw.

- K hat keine Lösung $\Rightarrow G_x \cup G_y \neq \infty$
  - Der Beweis ist analog zum Beweis, der bei der Unentscheidbarkeit des Schnittproblems durchgeführt wird

### Weitere Unentscheidbarkeiten von Sprachproblemen

Wie in den obigen drei Beweisen zu sehen ist, sind die Beweis zueinander ähnlich. Nun gibt es weitere Sprachprobleme, deren Unentscheidbarkeit wir zeigen können. Hierbei gehen wir analog zu den obigen Beweisen vor.

# Literatur


[1] Uwe Schöning. *Theoretische Informatik - kurz gefasst*. 5. Aufl. Hochschultaschenbuch. Heidelberg: Spektrum Akademischer Verlag, 2012. ISBN: 978-3-8274-1824-1.

[2] Mirko Rahn. „Entscheidbare Fälle des Postschen Korrespondenzproblems“. Dissertation. Karlsruhe: Universität Fridericiana zu Karlsruhe, 2008.

[3] Hans. U. Simon und Maike Buchin. *Post'sche Korrespondenzproblem*. 2014. URL: https://www.ruhr-uni-bochum.de/lmi/lehre/materialien/ti/vorlesung-2014/kap2-3.pdf.

[4] Arik Rückemann. *Das Postsche Korrespondenzproblem*. 2012. URL: http://www.inf.fu-berlin.de/lehre/WS12/Prosem-ThInf/rueckemann.pdf.