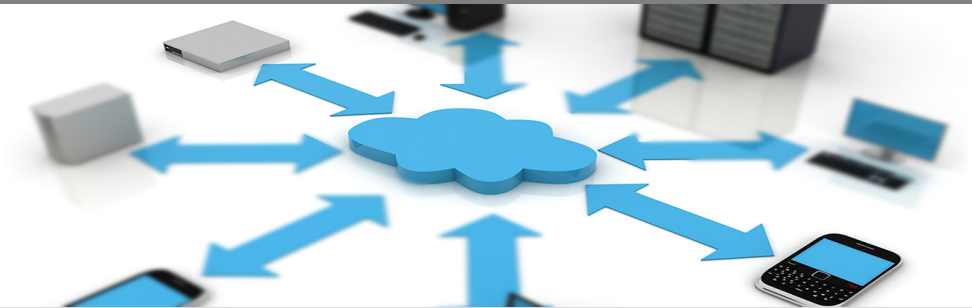
Karlsruher Institut für Technologie

GBI Tutorium Nr. 41

Foliensatz 13

Vincent Hahn – vincent.hahn@student.kit.edu | 31. Januar 2013



Outline/Gliederung

Vincent Hahn – vincent.hahn@student.kit.edu

1. Wiederholung
2. Unentscheidbare Probleme
3. Äquivalenzrelationen

Überblick

Wiederholung

- Was gehört zur formalen Definition einer Turingmaschine?
- Welche Eigenschaften haben Äquivalenzrelationen?
- Und was heißt das?

Wiederholung

Vincent Hahn – vincent.hahn@student.kit.edu

- Was gehört zur formalen Definition einer Turingmaschine?
  $T = (Z, z_0, X, f, g, m)$
- Welche Eigenschaften haben Äquivalenzrelationen?

- Und was heißt das?

Wiederholung

Vincent Hahn – vincent.hahn@student.kit.edu

- Was gehört zur formalen Definition einer Turingmaschine?
  $T = (Z, z_0, X, f, g, m)$
- Welche Eigenschaften haben Äquivalenzrelationen?
  - Reflexiv
  - Symmetrisch
  - Transitiv
- Und was heißt das?

Überblick

Vincent Hahn – vincent.hahn@student.kit.edu

Unentscheidbare Probleme

Es gibt Probleme, die lassen sich mit einer Turing-Maschine (oder äquivalent: einem Java-Programm) nicht lösen. (Auch nicht mit unendlich viel Zeit und Platz.)
Ein solches Problem ist nicht **entscheidbar**

Entscheidbarkeit

Für ein entscheidbares Problem gibt es eine Turingmaschine, die für jede Eingabe hält und das Eingabewort entweder akzeptiert oder nicht.

Codierung von Turingmaschinen

Bisher haben wir eine Turingmaschine formal so geschrieben $T = (Z, Z_0, X, f, g, m)$. Wir bauen uns eine Codierung, die die ganze Turingmaschine in ein Wort w_1 "packt".

Universelle Turingmaschine (UTM)

Dieses Wort w_1 übergeben wir dann einer universellen Turingmaschine U,

- die überprüft, ob w_1 eine Turingmaschine T codiert
- dann die Turingmaschine T "simuliert" und als Eingabe w_2 verwendet
- und schließlich das Ergebnis davon ausgibt

Codierung von Turingmaschinen

Bisher haben wir eine Turingmaschine formal so geschrieben $T = (Z, Z_0, X, f, g, m)$. Wir bauen uns eine Codierung, die die ganze Turingmaschine in ein Wort w_1 "packt".

Universelle Turingmaschine (UTM)

Dieses Wort w_1 übergeben wir dann einer universellen Turingmaschine U,

- die überprüft, ob w_1 eine Turingmaschine T codiert
- dann die Turingmaschine T "simuliert" und als Eingabe w_2 verwendet
- und schließlich das Ergebnis davon ausgibt

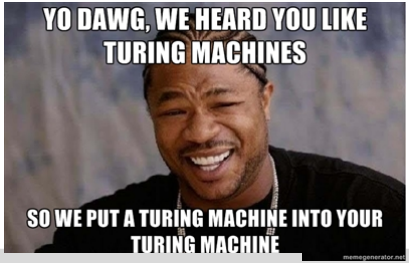
Codierung von Turingmaschinen

Bisher haben wir eine Turingmaschine formal so geschrieben $T = (Z, Z_0, X, f, g, m)$. Wir bauen uns eine Codierung, die die ganze Turingmaschine in ein Wort w_1 "packt".

Universelle Turingmaschine (UTM)

Dieses Wort w_1 übergeben wir dann einer universellen Turingmaschine U,

- die überprüft, ob w_1 eine Turingmaschine T codiert
- dann die Turingmaschine T "simuliert" und als Eingabe w_2 verwendet
- und schließlich das Ergebnis davon ausgibt



Codierungen von Turingmaschinen: Gödelisierung

Wir codieren eine Turingmaschine so:

- Das Alphabet ist $\{0, 1, [,]\}$.
- Die Zustände werden durchnummeriert, Startzustand mit 0, Zustände haben gleich viele Stellen, eingeklammert in []. Dafür schreiben wir $\text{cod}_Z(Z)$.
- Bandalphabet wird auch durchnummeriert, Blanket ist 0. Dafür schreiben wir $\text{cod}_X(x)$.
- Bewegungsrichtungen werden mit [10], [00], [01] codiert (links, stehen bleiben, rechts). Dafür schreiben wir $\text{cod}_M(r)$.
- Auch die partiellen Funktionen f, g und m werden codiert. (Skript)

Das ganze nennen wir **Gödelisierung**. Jede Turingmaschine hat dann eine **Gödelnummer**.

Codierungen von Turingmaschinen: Gödelisierung

Wir codieren eine Turingmaschine so:

- Das Alphabet ist $\{0, 1, [,]\}$.
- Die Zustände werden durchnummeriert, Startzustand mit 0, Zustände haben gleich viele Stellen, eingeklammert in []. Dafür schreiben wir $\text{cod}_Z(Z)$.
- Bandalphabet wird auch durchnummeriert, Blanket ist 0. Dafür schreiben wir $\text{cod}_X(x)$.
- Bewegungsrichtungen werden mit [10], [00], [01] codiert (links, stehen bleiben, rechts). Dafür schreiben wir $\text{cod}_M(r)$.
- Auch die partiellen Funktionen f, g und m werden codiert. (Skript)

Das ganze nennen wir **Gödelisierung**. Jede Turingmaschine hat dann eine **Gödelnummer**.

Codierungen von Turingmaschinen: Gödelisierung

Wir codieren eine Turingmaschine so:

- Das Alphabet ist $\{0, 1, [,]\}$.
- Die Zustände werden durchnummeriert, Startzustand mit 0, Zustände haben gleich viele Stellen, eingeklammert in []. Dafür schreiben wir $\text{cod}_z(Z)$.
- Bandalphabet wird auch durchnummeriert, Blanket ist 0. Dafür schreiben wir $\text{cod}_x(x)$.
- Bewegungsrichtungen werden mit [10], [00], [01] codiert (links, stehen bleiben, rechts). Dafür schreiben wir $\text{cod}_M(r)$.
- Auch die partiellen Funktionen f, g und m werden codiert. (Skript)

Das ganze nennen wir **Gödelisierung**. Jede Turingmaschine hat dann eine **Gödelnummer**.

Codierungen von Turingmaschinen: Gödelisierung

Wir codieren eine Turingmaschine so:

- Das Alphabet ist $\{0, 1, [,]\}$.
- Die Zustände werden durchnummeriert, Startzustand mit 0, Zustände haben gleich viele Stellen, eingeklammert in []. Dafür schreiben wir $\text{cod}_z(Z)$.
- Bandalphabet wird auch durchnummeriert, Blanket ist 0. Dafür schreiben wir $\text{cod}_x(x)$.
- Bewegungsrichtungen werden mit [10], [00], [01] codiert (links, stehen bleiben, rechts). Dafür schreiben wir $\text{cod}_M(r)$.
- Auch die partiellen Funktionen f, g und m werden codiert. (Skript)

Das ganze nennen wir **Gödelisierung**. Jede Turingmaschine hat dann eine **Gödelnummer**.

Codierungen von Turingmaschinen: Gödelisierung

Wir codieren eine Turingmaschine so:

- Das Alphabet ist $\{0, 1, [,]\}$.
- Die Zustände werden durchnummeriert, Startzustand mit 0, Zustände haben gleich viele Stellen, eingeklammert in []. Dafür schreiben wir $\text{cod}_z(Z)$.
- Bandalphabet wird auch durchnummeriert, Blanket ist 0. Dafür schreiben wir $\text{cod}_x(x)$.
- Bewegungsrichtungen werden mit [10], [00], [01] codiert (links, stehen bleiben, rechts). Dafür schreiben wir $\text{cod}_M(r)$.
- Auch die partiellen Funktionen f, g und m werden codiert. (Skript)

Das ganze nennen wir **Gödelisierung**. Jede Turingmaschine hat dann eine **Gödelnummer**.

Beispiel zur Gödelisierung

Gegeben ist die Turingmaschine
$T = (\{z_0, z_1, z_2\}, z_0, \{\square, a, b, c, d\}, f, g, m)$. Codiert alles außer f, g, m.

Muss alles angegeben werden?

Beispiel zur Gödelisierung

Gegeben ist die Turingmaschine
$T = (\{z_0, z_1, z_2\}, z_0, \{\square, a, b, c, d\}, f, g, m)$. Codiert alles außer f, g, m.

- $\text{cod}_z(z_0) = [00]$

Muss alles angegeben werden?

Beispiel zur Gödelisierung

Gegeben ist die Turingmaschine
$T = (\{z_0, z_1, z_2\}, z_0, \{\square, a, b, c, d\}, f, g, m)$. Codiert alles außer f, g, m.

- $\text{cod}_z(z_0) = [00]$
- $\text{cod}_z(z_1) = [01]$
- $\text{cod}_z(z_2) = [10]$
- $\text{cod}_x(\square) = [000]$
- $\text{cod}_x(a) = [001]$
- $\text{cod}_x(b) = [010]$
- $\text{cod}_x(c) = [011]$
- $\text{cod}_x(d) = [100]$

Muss alles angegeben werden?

Beispiel zur Gödelisierung

Gegeben ist die Turingmaschine
$T = (\{z_0, z_1, z_2\}, z_0, \{\square, a, b, c, d\}, f, g, m)$. Codiert alles außer f, g, m.

- $\text{cod}_z(z_0) = [00]$
- $\text{cod}_z(z_1) = [01]$
- $\text{cod}_z(z_2) = [10]$
- $\text{cod}_x(\square) = [000]$
- $\text{cod}_x(a) = [001]$
- $\text{cod}_x(b) = [010]$
- $\text{cod}_x(c) = [011]$
- $\text{cod}_x(d) = [100]$

Muss alles angegeben werden?
Nein, es reicht jeweils das größte Element.

Halteproblem-Beweis 1

Satz

Es ist nicht möglich, eine Turingmaschine U zu bauen, die für jede Turingmaschine T (codiert als w_1) und jede Eingabe w_2 entscheidet, ob T bei der Eingabe von w_2 hält.

Das lässt sich auch beweisen.

Beweis des Halteproblems

Annahme: es gibt eine Super-Turingmaschine H. H bekommt als Eingabe:

- eine andere Turingmaschine T und
- ein Eingabewort w.

Die Super-Turingmaschine H

- simuliert die “normale” Turingmaschine T und
- benutzt als Eingabe für T das Wort w.

Die Super-Turingmaschine H gibt aus

- 1, wenn T mit w als Eingabe hält und
- 0, wenn T mit w als Eingabe nicht hält.

Beweis des Halteproblems

Annahme: es gibt eine Super-Turingmaschine H. H bekommt als Eingabe:

- eine andere Turingmaschine T und
- ein Eingabewort w.

Die Super-Turingmaschine H

- simuliert die “normale” Turingmaschine T und
- benutzt als Eingabe für T das Wort w.

Die Super-Turingmaschine H gibt aus

- 1, wenn T mit w als Eingabe hält und
- 0, wenn T mit w als Eingabe nicht hält.

Beweis des Halteproblems

Annahme: es gibt eine Super-Turingmaschine H. H bekommt als Eingabe:

- eine andere Turingmaschine T und
- ein Eingabewort w.

Die Super-Turingmaschine H

- simuliert die “normale” Turingmaschine T und
- benutzt als Eingabe für T das Wort w.

Die Super-Turingmaschine H gibt aus

- 1, wenn T mit w als Eingabe hält und
- 0, wenn T mit w als Eingabe nicht hält.

Halteproblem-Beweis 2

Wir bauen uns eine unendlich große Tabelle, die

- nach rechts (in den Spalten) alle möglichen Worte w enthält und
- nach unten (in den Zeilen) die codierte Turingmaschine T_w zum Wort w enthält.

In den Zeilen sind also **alle möglichen Turingmaschinen**. Unsere Super-Turingmaschine hat die Tabelle ausgefüllt mit

- 1, wenn $T_w(w)$ hält und
- 0, wenn $T_w(w)$ nicht hält.

	w_0	w_1	w_2	...
T_{w_0}	1	0	1	
T_{w_1}	0	0	0	
T_{w_2}	0	0	1	
...				

Halteproblem-Beweis 2

Wir bauen uns eine unendlich große Tabelle, die

- nach rechts (in den Spalten) alle möglichen Worte w enthält und
- nach unten (in den Zeilen) die codierte Turingmaschine T_w zum Wort w enthält.

In den Zeilen sind also **alle möglichen Turingmaschinen**. Unsere Super-Turingmaschine hat die Tabelle ausgefüllt mit

- 1, wenn $T_w(w)$ hält und
- 0, wenn $T_w(w)$ nicht hält.

	w_0	w_1	w_2	...
T_{w_0}	1	0	1	
T_{w_1}	0	0	0	
T_{w_2}	0	0	1	
...				

Halteproblem-Beweis 2

Wir bauen uns eine unendlich große Tabelle, die

- nach rechts (in den Spalten) alle möglichen Worte w enthält und
- nach unten (in den Zeilen) die codierte Turingmaschine T_w zum Wort w enthält.

In den Zeilen sind also **alle möglichen Turingmaschinen**. Unsere Super-Turingmaschine hat die Tabelle ausgefüllt mit

- 1, wenn $T_w(w)$ hält und
- 0, wenn $T_w(w)$ nicht hält.

	w_0	w_1	w_2	...
T_{w_0}	1	0	1	
T_{w_1}	0	0	0	
T_{w_2}	0	0	1	
...				

Halteproblem-Beweis 3

Nun nehmen wir die Diagonale und schreiben sie auch in die Tabelle (hier blau). Außerdem schreiben wir darunter das “Komplementär” der Diagonale (1 wird zu 0 und umgekehrt, hier in rot).

	w_0	w_1	w_2	...	
T_{w_0}	1	0	1	...	
T_{w_1}	0	0	0	...	
T_{w_2}	0	0	1	...	
...	...	...	...	...	
T_d	1	0	1	...	(← das ist die Diagonale)
$T_{\overline{d}}$	0	1	0	...	(← die Zeile war sicher noch nirgends)

Warum gibt es $T_{\overline{d}}$ nicht schon vorher?

Halteproblem-Beweis 3

Nun nehmen wir die Diagonale und schreiben sie auch in die Tabelle (hier blau). Außerdem schreiben wir darunter das “Komplementär” der Diagonale (1 wird zu 0 und umgekehrt, hier in rot).

	w_0	w_1	w_2	...	
T_{w_0}	1	0	1	...	
T_{w_1}	0	0	0	...	
T_{w_2}	0	0	1	...	
...	...	...	...	...	
T_d	1	0	1	...	(← das ist die Diagonale)
$T_{\overline{d}}$	0	1	0	...	(← die Zeile war sicher noch nirgends)

Warum gibt es $T_{\overline{d}}$ nicht schon vorher?
Sie unterscheidet sich von jeder Zeile um ein Element (Diagonalelement).

Halteproblem-Beweis 4

	w_0	w_1	w_2	...	
T_{w_0}	1	0	1	...	
T_{w_1}	0	0	0	...	
T_{w_2}	0	0	1	...	
...	...	...	...	...	
T_d	1	0	1	...	(← die Zeile war schon irgendwo)
$T_{\overline{d}}$	0	1	0	...	(← die Zeile war sicher noch nirgends)

Obwohl $T_{\overline{d}}$ sicher nirgends vorkam, könnten wir sie bauen:

- Wir wissen, dass T_d hält (sagt uns die Super-Turingmaschine), also gehen wir mit $T_{\overline{d}}$ in eine Endlosschleife.
- Wir wissen, dass T_d nicht hält (sagt uns die Super-Turingmaschine), also halten wir mit $T_{\overline{d}}$.

Verrückt: Wenn es die Super-Turingmaschine gibt, dann könnten wir die Turing-Maschine $T_{\overline{d}}$ bauen, die es eigentlich nicht gibt. Das ist ein Widerspruch, also kann es die Super-Turingmaschine nicht geben.

Halteproblem-Beweis 4

	w_0	w_1	w_2	...	
T_{w_0}	1	0	1	...	
T_{w_1}	0	0	0	...	
T_{w_2}	0	0	1	...	
...	...	...	...	...	
T_d	1	0	1	...	(← die Zeile war schon irgendwo)
$T_{\overline{d}}$	0	1	0	...	(← die Zeile war sicher noch nirgends)

Obwohl $T_{\overline{d}}$ sicher nirgends vorkam, könnten wir sie bauen:

- Wir wissen, dass T_d hält (sagt uns die Super-Turingmaschine), also gehen wir mit $T_{\overline{d}}$ in eine Endlosschleife.
- Wir wissen, dass T_d nicht hält (sagt uns die Super-Turingmaschine), also halten wir mit $T_{\overline{d}}$.

Verrückt: Wenn es die Super-Turingmaschine gibt, dann könnten wir die Turing-Maschine $T_{\overline{d}}$ bauen, die es eigentlich nicht gibt. Das ist ein Widerspruch, also kann es die Super-Turingmaschine nicht geben.

Die Busy-Beaver-Funktion

Definition: Busy-Beaver-Funktion

Der Busy-Beaver ("fleißiger Biber") ist eine Turingmaschine, die $n+1$ Zustände, wobei ein Anfangszustand und ein Haltezustand darunter sind. Diese Turingmaschine kann nur 1 auf das Band schreiben.
Die Busy-Beaver-Funktion $\mathrm{bb}(n)$ wird der fleißige Biber genannt, der die maximale Anzahl an Einsen auf das Band schreiben kann (also der fleißigste Biber).

Theorem

Für jede berechenbare Funktion $f : \mathbb{N}_+ \to \mathbb{N}_+$ gibt es ein n_0, so dass für alle $n \geq n_0$ gilt:

$$\mathrm{bb}(n) > f(n)$$

Kurz: Die Busy-Beaver-Funktion ist die am schnellsten wachsende Funktion. Und nicht berechenbar.

Die Busy-Beaver-Funktion

Definition: Busy-Beaver-Funktion

Der Busy-Beaver ("fleißiger Biber") ist eine Turingmaschine, die $n+1$ Zustände, wobei ein Anfangszustand und ein Haltezustand darunter sind. Diese Turingmaschine kann nur 1 auf das Band schreiben.
Die Busy-Beaver-Funktion $\mathrm{bb}(n)$ wird der fleißige Biber genannt, der die maximale Anzahl an Einsen auf das Band schreiben kann (also der fleißigste Biber).

Theorem

Für jede berechenbare Funktion $f : \mathbb{N}_+ \to \mathbb{N}_+$ gibt es ein n_0, so dass für alle $n \geq n_0$ gilt:

$$\mathrm{bb}(n) > f(n)$$

Kurz: Die Busy-Beaver-Funktion ist die am schnellsten wachsende Funktion. Und nicht berechenbar.

Beispiel zum Busy-Beaver

Wie viele Einsen produziert diese Turingmaschine? Übrigens: Das ist $\mathrm{bb}(4)$.

	A	B	C	D	H
□	1, *R*, *B*	1, *L*, *A*	1, *R*, *H*	1, *R*, *D*	
1	1, *L*, *B*	□, *L*, *C*	1, *L*, *D*	□, *R*, *A*	

Beispiel zum Busy-Beaver

Wie viele Einsen produziert diese Turingmaschine? Übrigens: Das ist $\mathrm{bb}(4)$.

	A	B	C	D	H
□	1, *R*, *B*	1, *L*, *A*	1, *R*, *H*	1, *R*, *D*	
1	1, *L*, *B*	□, *L*, *C*	1, *L*, *D*	□, *R*, *A*	

Diese Turingmaschine produziert 13 Einsen.

Überblick

Vincent Hahn – vincent.hahn@student.kit.edu

1 Wiederholung

2 Unentscheidbare Probleme

3 Äquivalenzrelationen

Äquivalenzrelation

Definition: Äquivalenzrelation

Eine Relation R ist genau dann eine **Äquivalenzrelation**, wenn sie

- symmetrisch,
- reflexiv und
- transitiv

ist.

Eigenschaften von Relationen

Welche Eigenschaften haben diese Relationen (stets auf ganze Zahlen)?

- $\leq$
- $>$
- $=$

Wie sieht das in einem Graphen aus? (Tafel)

Könnt ihr euch noch weitere Äquivalenzrelationen vorstellen?

Eigenschaften von Relationen

Welche Eigenschaften haben diese Relationen (stets auf ganze Zahlen)?

- $\leq$ reflexiv, transitiv
- $>$
- $=$

Wie sieht das in einem Graphen aus? (Tafel)
Könnt ihr euch noch weitere Äquivalenzrelationen vorstellen?

Eigenschaften von Relationen

Welche Eigenschaften haben diese Relationen (stets auf ganze Zahlen)?

- $\leq$ reflexiv, transitiv
- $>$ transitiv
- $=$

Wie sieht das in einem Graphen aus? (Tafel)
Könnt ihr euch noch weitere Äquivalenzrelationen vorstellen?

Eigenschaften von Relationen

Welche Eigenschaften haben diese Relationen (stets auf ganze Zahlen)?

- $\leq$ reflexiv, transitiv
- $>$ transitiv
- $=$ reflexiv, transitiv, symmetrisch

Wie sieht das in einem Graphen aus? (Tafel)
Könnt ihr euch noch weitere Äquivalenzrelationen vorstellen?

Eigenschaften von Relationen

Welche Eigenschaften haben diese Relationen (stets auf ganze Zahlen)?

- $\leq$ reflexiv, transitiv
- $>$ transitiv
- $=$ reflexiv, transitiv, symmetrisch

Wie sieht das in einem Graphen aus? (Tafel)
Könnt ihr euch noch weitere Äquivalenzrelationen vorstellen? (Denkt an Betrag, die Quersumme, Modulo)

Äquivalenzrelation Kongruenz Modulo

Definition: Kongruent Modulo

Zwei Zahlen $x, y \in \mathbb{N}_+$ heißen **kongruent modulo n**, wenn die Differenz $x - y$ durch n teilbar, also ein Vielfaches von n ist. Es wird geschrieben:

$$x \equiv y \ (\mod n)$$

Beweis (Auszug):

Äquivalenzrelation Kongruenz Modulo

Vincent Hahn – vincent.hahn@student.kit.edu

Definition: Kongruent Modulo

Zwei Zahlen $x, y \in \mathbb{N}_+$ heißen **kongruent modulo n**, wenn die Differenz $x - y$ durch n teilbar, also ein Vielfaches von n ist. Es wird geschrieben:

$$x \equiv y \,(\mod n)$$

Beweis (Auszug):

- Reflexivität: $x - x = 0$ ist Vielfaches von n.
- Symmetrie: $x - y$ ist Vielfaches von n, $\Rightarrow y - x = -(x - y)$ ist auch Vielfaches von n
- Transitivität: $x - y = k_1 n$ und $y - z = k_2 n$ $(k_1, k_2 \in \mathbb{Z})$, dannn ist $x - z = (x - y) + (y - z) = (k_1 + k_2) \cdot n$ ein Vielfaches von n.

Äquivalenzrelation Kongruenz Modulo

Definition: Kongruent Modulo

Zwei Zahlen $x, y \in \mathbb{N}_+$ heißen **kongruent modulo n**, wenn die Differenz $x - y$ durch n teilbar, also ein Vielfaches von n ist. Es wird geschrieben:

$$x \equiv y \,(\mod n)$$

Beweis (Auszug):

- Reflexivität: $x - x = 0$ ist Vielfaches von n.
- Symmetrie: $x - y$ ist Vielfaches von n, $\Rightarrow y - x = -(x - y)$ ist auch Vielfaches von n
- Transitivität: $x - y = k_1 n$ und $y - z = k_2 n$ $(k_1, k_2 \in \mathbb{Z})$, dannn ist $x - z = (x - y) + (y - z) = (k_1 + k_2) \cdot n$ ein Vielfaches von n.

Äquivalenzklasse

Vincent Hahn – vincent.hahn@student.kit.edu

Definition: Äquivalenzklasse

Sind zwei Elemente $(x, y) \in R$, so schreibt man auch xRy (Infixschreibweise). Alle Elemente, die miteinander in Relation stehen, befinden sich in der selben **Äquivalenzklasse**:

$$[x]_{\mathrm{R}} = \{y | yRx\}$$

Quiz

Vincent Hahn – vincent.hahn@student.kit.edu

- Stimmt es, dass auch xRy folgt: $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$?
- Existiert ein $z \in [x]_{\mathrm{R}}$ und $z \in [y]_{\mathrm{R}}$, so ist $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$.
- Wieviele Äquivalenzklassen gibt es zu $R = \mod 6$?

Quiz

Vincent Hahn – vincent.hahn@student.kit.edu

- Stimmt es, dass auch xRy folgt: $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$? Ja.
- Existiert ein $z \in [x]_{\mathrm{R}}$ und $z \in [y]_{\mathrm{R}}$, so ist $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$.
- Wieviele Äquivalenzklassen gibt es zu $R = \mod 6$?

Quiz

Vincent Hahn – vincent.hahn@student.kit.edu

- Stimmt es, dass auch xRy folgt: $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$? Ja.
- Existiert ein $z \in [x]_{\mathrm{R}}$ und $z \in [y]_{\mathrm{R}}$, so ist $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$. Ja.
- Wieviele Äquivalenzklassen gibt es zu $R = \mod 6$?

Quiz

- Stimmt es, dass auch xRy folgt: $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$? Ja.
- Existiert ein $z \in [x]_{\mathrm{R}}$ und $z \in [y]_{\mathrm{R}}$, so ist $[x]_{\mathrm{R}} = [y]_{\mathrm{R}}$. Ja.
- Wieviele Äquivalenzklassen gibt es zu $R =$ mod 6? 6

Nerode-Äquivalenzrelation

Vincent Hahn – vincent.hahn@student.kit.edu

Definition: Nerode-Äquivalenzrelation

Sei $L \subseteq A^*$ eine formale Sprache. w_1 und w_2 seien Wörter $\in A^*$. Die Wörter heißen **Nerode-Äquivalent** ($\equiv_L$), falls gilt:

$$w_1 \equiv_L w_2 \Leftrightarrow (\forall w \in A^* : w_1 w \in L \leftrightarrow w_2 w \in L)$$

Beispiel zur Nerode Äquivalenz

- Alphabet $A = \{a, b\}$
- Sprache $L \subset A*$, L enthält alle Wörter ohne das Teilwort ba: $L = \langle a^* b^* \rangle$

Wie sieht der zugehörige Automat aus?

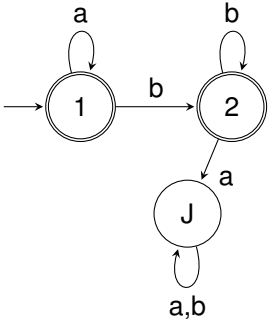
Nerode-Äquivalenzklassen:

Vincent Hahn – vincent.hahn@student.kit.edu

Beispiel zur Nerode Äquivalenz

- Alphabet $A = \{a, b\}$
- Sprache $L \subset A*$, L enthält alle Wörter ohne das Teilwort ba: $L = \langle a^* b^* \rangle$

Wie sieht der zugehörige Automat aus?



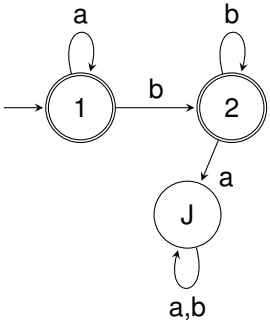
Nerode-Äquivalenzklassen:

Vincent Hahn – vincent.hahn@student.kit.edu

Beispiel zur Nerode Äquivalenz

- Alphabet $A = \{a, b\}$
- Sprache $L \subset A*$, L enthält alle Wörter ohne das Teilwort ba: $L = \langle a^* b^* \rangle$

Wie sieht der zugehörige Automat aus?



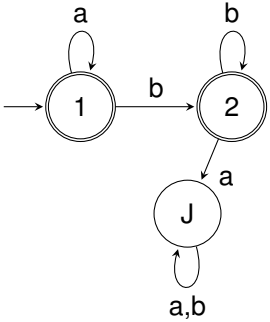
Wie kann jeder Zustand erreicht werden?

Nerode-Äquivalenzklassen:

Beispiel zur Nerode Äquivalenz

- Alphabet $A = \{a, b\}$
- Sprache $L \subset A*$, L enthält alle Wörter ohne das Teilwort ba: $L = \langle a^* b^* \rangle$

Wie sieht der zugehörige Automat aus?



Wie kann jeder Zustand erreicht werden?

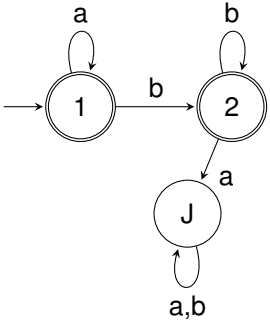
- a^*
- $a^* bb^*$
- $a^* bb^* a \{a, b\}^*$

Nerode-Äquivalenzklassen:

Beispiel zur Nerode Äquivalenz

- Alphabet $A = \{a, b\}$
- Sprache $L \subset A*$, L enthält alle Wörter ohne das Teilwort ba: $L = \langle a^* b^* \rangle$

Wie sieht der zugehörige Automat aus?



Wie kann jeder Zustand erreicht werden?

- a^*
- $a^* b b^*$
- $a^* b b^* a \{a, b\}^*$

Nerode-Äquivalenzklassen:
$[\epsilon]$, $[b]$, $[ba]$.

Faktormenge

Definition: Faktormenge

Die Menge aller Äquivalenzklassen einer Menge zur Relation R bezeichnet man als **Faktormenge** und schreibt $M_{/\mathrm{R}}$.

Beispiel zur Faktormenge

- Nennt die Äquivalenzklassen zu Kongruenz modulo 5. (Natürlich auf ganzen Zahlen.)
- Nennt die Faktormenge dazu.

Beispiel zur Faktormenge

- Nennt die Äquivalenzklassen zu Kongruenz modulo 5. (Natürlich auf ganzen Zahlen.)
  $[0], [1], [2], [3], [4]$
- Nennt die Faktormenge dazu.

Beispiel zur Faktormenge

Vincent Hahn – vincent.hahn@student.kit.edu

- Nennt die Äquivalenzklassen zu Kongruenz modulo 5. (Natürlich auf ganzen Zahlen.)
  $[0], [1], [2], [3], [4]$
- Nennt die Faktormenge dazu.
  $\mathbb{Z}_{/\equiv_5} = \{[0], [1], [2], [3], [4]\}$

Beispiel 2 zur Nerode-Äquivalenz

Vincent Hahn – vincent.hahn@student.kit.edu

Gegeben sei die Sprache $L = \{a^k b^k | k \in \mathbb{N}_0\}$.
Wie sieht hier ein endlicher Akzeptor aus?

Beispiel 2 zur Nerode-Äquivalenz

Gegeben sei die Sprache $L = \{a^k b^k | k \in \mathbb{N}_0\}$.
Wie sieht hier ein endlicher Akzeptor aus? Es gibt keinen.

Beispiel 2 zur Nerode-Äquivalenz

Gegeben sei die Sprache $L = \left\{a^k b^k | k \in \mathbb{N}_0\right\}$.
Wie sieht hier ein endlicher Akzeptor aus? Es gibt keinen.
Nennt einige Nerode-Äquivalenzklassen.

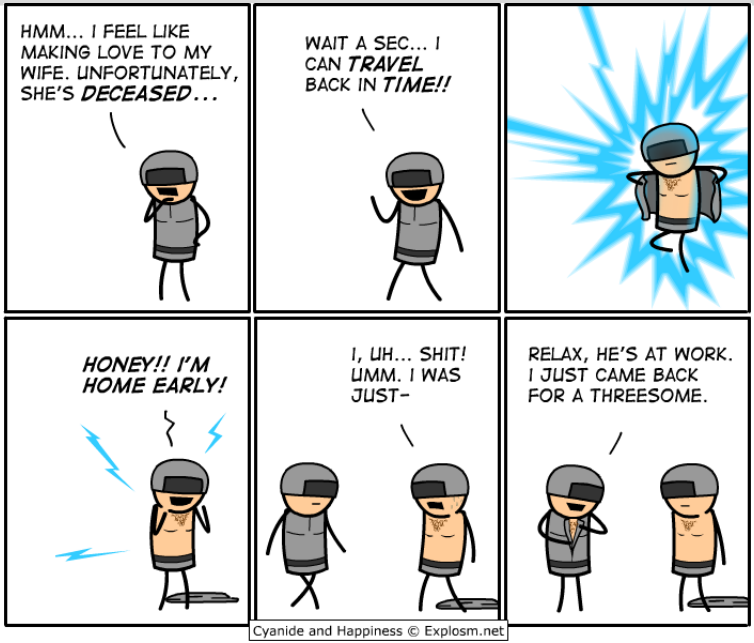
Beispiel 2 zur Nerode-Äquivalenz

Gegeben sei die Sprache $L = \{a^k b^k | k \in \mathbb{N}_0\}$.
Wie sieht hier ein endlicher Akzeptor aus? Es gibt keinen.
Nennt einige Nerode-Äquivalenzklassen.
Wieviele gibt es?

Beispiel 2 zur Nerode-Äquivalenz

Vincent Hahn – vincent.hahn@student.kit.edu

Gegeben sei die Sprache $L = \{a^k b^k | k \in \mathbb{N}_0\}$.
Wie sieht hier ein endlicher Akzeptor aus? Es gibt keinen.
Nennt einige Nerode-Äquivalenzklassen.
Wieviele gibt es?
Es gibt unendlich viele Nerode-Äquivalenzklassen. Die Faktormenge hat also unendlich viele Elemente.



http://www.explosm.net/comics/1633/