Theoretische Informatik

Florin Manea
(basierend auf den Folien von Carsten Damm)

Stand: 28. April 2025

Jetzt:

1. Einführung
    - Bücher
    - Motivation
    - Symbole, Wörter und Sprachen
    - Operationen auf Sprachen
    - Sprachklassen
    - Reguläre Ausdrücke

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

> *Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?*

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

> *Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?*

Konsequenz: Es entstehen grundsätzliche Probleme, wenn man *uneingeschränkte* Mengenbildungen zulässt:

- Sei R die „Menge“ aller Mengen, die sich nicht selbst als Element enhalten: $R = \{x | x \notin x\}$. Enthält R sich selbst, d.h. gilt $R \in R$?

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?

Konsequenz: Es entstehen grundsätzliche Probleme, wenn man *uneingeschränkte* Mengenbildungen zulässt:

- Sei R die „Menge" aller Mengen, die sich nicht selbst als Element enhalten: $R = \{x | x \notin x\}$. Enthält R sich selbst, d.h. gilt $R \in R$?

Ausweg: Mengenbildung nur aus Elementen „niederer Stufe"

Wir betrachten ein Universum von „Urelementen". Mengen *erster Stufe* sind Mengen, deren Elemente Urelemente sind.

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?

Konsequenz: Es entstehen grundsätzliche Probleme, wenn man *uneingeschränkte* Mengenbildungen zulässt:

- Sei R die „Menge“ aller Mengen, die sich nicht selbst als Element enhalten: $R = \{x | x \notin x\}$. Enthält R sich selbst, d.h. gilt $R \in R$?

Ausweg: Mengenbildung nur aus Elementen „niederer Stufe“

Wir betrachten ein Universum von „Urelementen“. Mengen *erster Stufe* sind Mengen, deren Elemente Urelemente sind. Mengen *zweiter Stufe* (im folgenden Klassen genannt) haben als Elemente Mengen erster Stufe, usw.

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

> *Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?*

Konsequenz: Es entstehen grundsätzliche Probleme, wenn man *uneingeschränkte* Mengenbildungen zulässt:

- Sei R die „Menge“ aller Mengen, die sich nicht selbst als Element enhalten: $R = \{x | x \notin x\}$. Enthält R sich selbst, d.h. gilt $R \in R$?

Ausweg: Mengenbildung nur aus Elementen „niederer Stufe“

Wir betrachten ein Universum von „Urelementen“. Mengen *erster Stufe* sind Mengen, deren Elemente Urelemente sind. Mengen *zweiter Stufe* (im folgenden Klassen genannt) haben als Elemente Mengen erster Stufe, usw.

Beispiel (fixiertes Alphabet Σ und Universum Σ^*)

- Mengen erster Stufe = formale Sprachen über Σ

Stufenaufbau der Mengenlehre

Das Barbier-Paradoxon (Bertrand Russel, 1872–1970)

Den Barbier rasiert genau die Leute, die sich nicht selbst rasieren. Rasiert der Barbier sich selbst?

Konsequenz: Es entstehen grundsätzliche Probleme, wenn man *uneingeschränkte* Mengenbildungen zulässt:

- Sei R die „Menge“ aller Mengen, die sich nicht selbst als Element enhalten: $R = \{x | x \notin x\}$. Enthält R sich selbst, d.h. gilt $R \in R$?

Ausweg: Mengenbildung nur aus Elementen „niederer Stufe“

Wir betrachten ein Universum von „Urelementen“. Mengen *erster Stufe* sind Mengen, deren Elemente Urelemente sind. Mengen *zweiter Stufe* (im folgenden Klassen genannt) haben als Elemente Mengen erster Stufe, usw.

Beispiel (fixiertes Alphabet Σ und Universum Σ^*)

- Mengen erster Stufe = formale Sprachen über Σ
- Klassen = bestimmte Mengen von formalen Sprachen über Σ, z.B.
  - Klasse Fin_Σ der endlichen Sprachen über Σ
  - Klasse $\varepsilon\text{-}Free_\Sigma$ der Sprachen über Σ, die ε nicht enthalten.

Formale Sprachklassen

Die Zeichen praktisch aller Schriftsprachen sind im Unicode-Zeichensatz enthalten. Unicode wird ständig weiterentwickelt und ist im Prinzip beliebig erweiterbar.

Formale Sprachklassen

Die Zeichen praktisch aller Schriftsprachen sind im Unicode-Zeichensatz enthalten. Unicode wird ständig weiterentwickelt und ist im Prinzip beliebig erweiterbar.

Ähnlich gehen wir von nun an davon aus, dass alle Alphabete, die wir betrachten, Teilmengen einer abzählbar unendlichen *universellen Symbolmenge* $\Gamma_S = \{z_1, z_2, z_3, \ldots\}$ sind[2].

[2] Für uns ist nur wichtig, dass es eine universelle Symbolmenge gibt, nicht, welche es genau ist.

Formale Sprachklassen

Die Zeichen praktisch aller Schriftsprachen sind im Unicode-Zeichensatz enthalten. Unicode wird ständig weiterentwickelt und ist im Prinzip beliebig erweiterbar.

Ähnlich gehen wir von nun an davon aus, dass alle Alphabete, die wir betrachten, Teilmengen einer abzählbar unendlichen *universellen Symbolmenge* $\Gamma_S = \{z_1, z_2, z_3, \dots\}$ sind[2].

Definition

Eine formale Sprachklasse ist eine *nichtleere* Klasse $\mathcal{L}$ von formalen Sprachen, so dass folgende Eigenschaften erfüllt sind:

- für jedes $L \in \mathcal{L}$ gibt es ein endliches Alphabet $\Sigma \subseteq \Gamma_S$ mit $L \subseteq \Sigma^*$,
- es gibt ein $L \in \mathcal{L}$ mit $L \neq \emptyset$.

Formale Sprachklassen

Die Zeichen praktisch aller Schriftsprachen sind im Unicode-Zeichensatz enthalten. Unicode wird ständig weiterentwickelt und ist im Prinzip beliebig erweiterbar.

Ähnlich gehen wir von nun an davon aus, dass alle Alphabete, die wir betrachten, Teilmengen einer abzählbar unendlichen *universellen Symbolmenge* $\Gamma_S = \{z_1, z_2, z_3, \dots\}$ sind[2].

Definition

Eine formale Sprachklasse ist eine *nichtleere* Klasse $\mathcal{L}$ von formalen Sprachen, so dass folgende Eigenschaften erfüllt sind:

- für jedes $L \in \mathcal{L}$ gibt es ein endliches Alphabet $\Sigma \subseteq \Gamma_S$ mit $L \subseteq \Sigma^*$,
- es gibt ein $L \in \mathcal{L}$ mit $L \neq \emptyset$.

Beispiele für formale Sprachklassen

- Klasse *Fin* der endlichen Sprachen
- Klasse ε-*Free* der ε-freien Sprachen

Reguläre Mengen

Die betrachteten *Sprachoperationen* (einfache Mengenoperationen, Produkt, Hülle und Bild/Urbild unter Homomorphismen und Reflexion) bilden (Paare von) Sprachen auf Sprachen ab.

Reguläre Mengen

Die betrachteten *Sprachoperationen* (einfache Mengenoperationen, Produkt, Hülle und Bild/Urbild unter Homomorphismen und Reflexion) bilden (Paare von) Sprachen auf Sprachen ab.

Eine Sprachklasse $\mathcal{C}$ heißt abgeschlossen unter einer Sprachoperation, wenn ausgehend von Sprachen aus $\mathcal{C}$, das Ergebnis ebenfalls in $\mathcal{C}$ liegt.

Reguläre Mengen

Die betrachteten *Sprachoperationen* (einfache Mengenoperationen, Produkt, Hülle und Bild/Urbild unter Homomorphismen und Reflexion) bilden (Paare von) Sprachen auf Sprachen ab.

Eine Sprachklasse $\mathcal{C}$ heißt abgeschlossen unter einer Sprachoperation, wenn ausgehend von Sprachen aus $\mathcal{C}$, das Ergebnis ebenfalls in $\mathcal{C}$ liegt.

Definition

Die Klasse $\mathcal{R}eg$ der regulären Mengen ist die kleinste Sprachklasse, die $\mathcal{F}in$ enthält und abgeschlossen ist unter endlich vielen Anwendungen von Vereinigung, Konkatenation und Kleenescher Hülle.

Das Wortproblem

Das Wortproblem besteht darin, zu entscheiden, ob ein gegebenes Wort w zu einer gegebenen formalen Sprache L gehört.

Reguläre Mengen haben eine endliche Beschreibung

Um das Wortproblem algorithmisch zu untersuchen, muss L durch eine endliche Beschreibung gegeben sein.

Das Wortproblem

Das Wortproblem besteht darin, zu entscheiden, ob ein gegebenes Wort w zu einer gegebenen formalen Sprache L gehört.

Reguläre Mengen haben eine endliche Beschreibung

Um das Wortproblem algorithmisch zu untersuchen, muss L durch eine endliche Beschreibung gegeben sein. Nach Definition ist dies zum Beispiel für reguläre Mengen möglich.

$\mathcal{R}eg$ ist sehr robust

Wir werden zeigen: Die Klasse ist abgeschlossen unter *allen* bisher genannten Sprachoperationen.

$\mathcal{R}eg$ ist sehr robust

Wir werden zeigen: Die Klasse ist abgeschlossen unter *allen* bisher genannten Sprachoperationen. Darüber hinaus auch unter diesen:

Quotient bzgl. Wort

Sei $L \subseteq \Sigma^*$ eine formale Sprache und $x \in \Sigma^*$ ein Wort. Rechts- bzw. Linksquotient von L bzgl. x sind definiert durch

$$L/x := \{z \in \Sigma^* : zx \in L\} \text{ bzw. } x\backslash L := \{z \in \Sigma^* : xz \in L\}.$$

$\mathcal{R}eg$ ist sehr robust

Wir werden zeigen: Die Klasse ist abgeschlossen unter *allen* bisher genannten Sprachoperationen. Darüber hinaus auch unter diesen:

Quotient bzgl. Wort

Sei $L \subseteq \Sigma^*$ eine formale Sprache und $x \in \Sigma^*$ ein Wort. Rechts- bzw. Linksquotient von L bzgl. x sind definiert durch

$$L/x := \{z \in \Sigma^* : zx \in L\} \text{ bzw. } x\backslash L := \{z \in \Sigma^* : xz \in L\}.$$

Quotient bzgl. Sprache

Sei $K \subseteq \Sigma^*$ eine weitere formale Sprache. Der Quotient der beiden Sprachen ist definiert durch

$$L/K := \{x \in \Sigma^* : \exists z \in K \wedge xz \in L\}.$$

$\mathcal{R}eg$ ist sehr robust

Wir werden zeigen: Die Klasse ist abgeschlossen unter *allen* bisher genannten Sprachoperationen. Darüber hinaus auch unter diesen:

Quotient bzgl. Wort

Sei $L \subseteq \Sigma^*$ eine formale Sprache und $x \in \Sigma^*$ ein Wort. Rechts- bzw. Linksquotient von L bzgl. x sind definiert durch

$$L/x := \{z \in \Sigma^* : zx \in L\} \text{ bzw. } x\backslash L := \{z \in \Sigma^* : xz \in L\}.$$

Quotient bzgl. Sprache

Sei $K \subseteq \Sigma^*$ eine weitere formale Sprache. Der Quotient der beiden Sprachen ist definiert durch

$$L/K := \{x \in \Sigma^* : \exists z \in K \wedge xz \in L\}.$$

Shuffle-Produkt

Das Shuffle-Produkt von $L, K \subseteq \Sigma^*$ ist definiert durch

$$L\#K := \{x_1 z_1 x_2 z_2 \ldots x_n z_n : \exists n \in \mathbb{N}, x_1, \ldots, x_n, z_1, \ldots, z_n \in \Sigma^*, x_1 x_2 \ldots x_n \in L, z_1 z_2 \ldots z_n \in K\}.$$

Jetzt:

1. Einführung
   - Bücher
   - Motivation
   - Symbole, Wörter und Sprachen
   - Operationen auf Sprachen
   - Sprachklassen
   - Reguläre Ausdrücke

Wie notiert man eine reguläre Menge?

Wir wissen nun über reguläre Mengen:

Definition Sie werden aus endlichen Mengen gebildet durch einfachste Sprachoperationen (Vereinigung, Produkt und Stern)

Robustheit Die Klasse der regulären Mengen ist abgeschlossen unter vielen weiteren Sprachoperationen.

Wie notiert man eine reguläre Menge?

Wir wissen nun über reguläre Mengen:

Definition Sie werden aus endlichen Mengen gebildet durch einfachste Sprachoperationen (Vereinigung, Produkt und Stern)

Robustheit Die Klasse der regulären Mengen ist abgeschlossen unter vielen weiteren Sprachoperationen.

Die Robustheit macht $\mathcal{R}eg$ zu einer sehr vielseitigen Sprachklasse.

Wie notiert man eine reguläre Menge?

Wir wissen nun über reguläre Mengen:

Definition Sie werden aus endlichen Mengen gebildet durch einfachste Sprachoperationen (Vereinigung, Produkt und Stern)

Robustheit Die Klasse der regulären Mengen ist abgeschlossen unter vielen weiteren Sprachoperationen.

Die Robustheit macht $\mathcal{R}eg$ zu einer sehr vielseitigen Sprachklasse.

Für bequeme Anwendung brauchen wir aber noch eine kompaktere Notation anstelle vieler Mengenklammern und Operationszeichen, Beispiel:

$$(\{2\} \circ \{0, 1, 2, 3\} \cup \{\varepsilon, 1\} \circ \{0, 1, ..., 9\}) \circ \{:\} \circ \{0, 1, ..., 5\} \circ \{0, 1, ..., 9\}$$

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, $(,),|,*$ nicht enthält.

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, $(,),|,*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, (,),|,$*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Semantik: *durch regulären Ausdruck beschriebene reguläre Menge*

$\emptyset$, ε und σ beschreiben die Mengen $\emptyset, \{\varepsilon\}, \{\sigma\} \subset \Sigma^*$.

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, (,),|,$*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Semantik: *durch regulären Ausdruck beschriebene reguläre Menge*

$\emptyset$, ε und σ beschreiben die Mengen $\emptyset, \{\varepsilon\}, \{\sigma\} \subset \Sigma^*$.
Beschreiben die regulären Ausdrücke r, s die regulären Mengen M_r, M_s, so beschreiben $(r|s), (rs)$ und $(r*)$ die Mengen $M_r \cup M_s, M_r \circ M_s$ und M_r^*.

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, (,),|,$*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Semantik: *durch regulären Ausdruck beschriebene reguläre Menge*

$\emptyset$, ε und σ beschreiben die Mengen $\emptyset, \{\varepsilon\}, \{\sigma\} \subset \Sigma^*$.
Beschreiben die regulären Ausdrücke r, s die regulären Mengen M_r, M_s, so beschreiben $(r|s), (rs)$ und $(r*)$ die Mengen $M_r \cup M_s, M_r \circ M_s$ und M_r^*. Durch Assoziativität und Vorrang ($*$ vor $\cdot$, $\cdot$ vor $|$) können Klammern entfallen.

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, (,),|,$*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Semantik: *durch regulären Ausdruck beschriebene reguläre Menge*

$\emptyset$, ε und σ beschreiben die Mengen $\emptyset, \{\varepsilon\}, \{\sigma\} \subset \Sigma^*$.
Beschreiben die regulären Ausdrücke r, s die regulären Mengen M_r, M_s, so beschreiben $(r|s), (rs)$ und $(r*)$ die Mengen $M_r \cup M_s, M_r \circ M_s$ und M_r^*. Durch Assoziativität und Vorrang ($*$ vor $\cdot$, $\cdot$ vor $|$) können Klammern entfallen.

Beispiele

- Uhrzeiten: $(2(0|1|2|3)|(\varepsilon|1)(0|1|\ldots|9)) : (0|1|\ldots|5)(0|1|\ldots|9)$

Reguläre Ausdrücke

Syntax: reguläre Ausdrücke (r.A. oder auch r.e. — regular expressions)

Sei Σ ein beliebiges Alphabet, das die Metasymbole $\emptyset$, ε, (,),|,$*$ nicht enthält.

- $\emptyset$ und ε sind reguläre Ausdrücke.
- Für jedes $\sigma \in \Sigma$ ist σ ein regulärer Ausdruck.
- Sind r und s reguläre Ausdrücke, so auch $(r|s), (rs)$ und $(r*)$.

Semantik: *durch regulären Ausdruck beschriebene reguläre Menge*

$\emptyset$, ε und σ beschreiben die Mengen $\emptyset, \{\varepsilon\}, \{\sigma\} \subset \Sigma^*$.
Beschreiben die regulären Ausdrücke r, s die regulären Mengen M_r, M_s, so beschreiben $(r|s), (rs)$ und $(r*)$ die Mengen $M_r \cup M_s, M_r \circ M_s$ und M_r^*. Durch Assoziativität und Vorrang ($*$ vor $\cdot$, $\cdot$ vor $|$) können Klammern entfallen.

Beispiele

- Uhrzeiten: $(2(0|1|2|3)|(\varepsilon|1)(0|1|...|9)) : (0|1|...|5)(0|1|...|9)$
- Bitstrings mit 1 an vor-vorletzer Stelle: $(1+0)*1(1+0)(1+0)$
  Erklärung: oft wird $+$ anstelle $|$ verwendet (auch hier im Skript).

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird.

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Basisfälle:

1. $h(M_\emptyset) = h(\emptyset) = \emptyset = M_\emptyset = M_{h(\emptyset)}$

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Basisfälle:

1. $h(M_\emptyset) = h(\emptyset) = \emptyset = M_\emptyset = M_{h(\emptyset)}$
2. $h(M_\varepsilon) = h(\{\varepsilon\}) = \{\varepsilon\} = M_\varepsilon = M_{h(\varepsilon)}$

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Basisfälle:

1. $h(M_\emptyset) = h(\emptyset) = \emptyset = M_\emptyset = M_{h(\emptyset)}$
2. $h(M_\varepsilon) = h(\{\varepsilon\}) = \{\varepsilon\} = M_\varepsilon = M_{h(\varepsilon)}$
3. Sei $\sigma \in \Sigma$ und $h(\sigma) = \gamma_1 \cdots \gamma_n$ mit $\gamma_i \in \Gamma$. Dann gilt
   $h(M_\sigma) = h(\{\sigma\}) = \{h(\sigma)\} = \{\gamma_1 \cdots \gamma_n\} = M_{\gamma_1} \cdots M_{\gamma_n} = M_{\gamma_1 \cdots \gamma_n} = M_{h(\sigma)}$

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Basisfälle:

1. $h(M_\emptyset) = h(\emptyset) = \emptyset = M_\emptyset = M_{h(\emptyset)}$
2. $h(M_\varepsilon) = h(\{\varepsilon\}) = \{\varepsilon\} = M_\varepsilon = M_{h(\varepsilon)}$
3. Sei $\sigma \in \Sigma$ und $h(\sigma) = \gamma_1 \cdots \gamma_n$ mit $\gamma_i \in \Gamma$. Dann gilt
   $h(M_\sigma) = h(\{\sigma\}) = \{h(\sigma)\} = \{\gamma_1 \cdots \gamma_n\} = M_{\gamma_1} \cdots M_{\gamma_n} = M_{\gamma_1 \cdots \gamma_n} = M_{h(\sigma)}$

Reguläre Ausdrücke und Homomorphismen

Satz (Abschluss unter Homomorphismen)

$\mathcal{R}eg$ ist abgeschlossen unter Homomorphismen, d.h. das Bild $h(M)$ einer regulären Menge $M \subseteq \Sigma^*$ unter einem Homomorphismus $h : \Sigma \to \Gamma^*$ ist auch eine reguläre Menge.

Beweisskizze (mittels regulärer Ausdrücke)

Sei r regulärer Ausdruck über Σ und $h(r)$ der daraus entstehende r.A., indem jedes Zeichen $\sigma \in \Sigma$ durch $h(\sigma)$ ersetzt wird. Es genügt, $h(M_r) = M_{h(r)}$ für alle r zu beweisen. Das gelingt durch strukturelle Induktion.

Basisfälle:

1. $h(M_\emptyset) = h(\emptyset) = \emptyset = M_\emptyset = M_{h(\emptyset)}$
2. $h(M_\varepsilon) = h(\{\varepsilon\}) = \{\varepsilon\} = M_\varepsilon = M_{h(\varepsilon)}$
3. Sei $\sigma \in \Sigma$ und $h(\sigma) = \gamma_1 \cdots \gamma_n$ mit $\gamma_i \in \Gamma$. Dann gilt $h(M_\sigma) = h(\{\sigma\}) = \{h(\sigma)\} = \{\gamma_1 \cdots \gamma_n\} = M_{\gamma_1} \cdots M_{\gamma_n} = M_{\gamma_1 \cdots \gamma_n} = M_{h(\sigma)}$

Induktionsschritt: — hier nur für $r|s$ gezeigt, Rest analog.

$h(M_{r|s}) = h(M_r \cup M_s) = h(M_r) \cup h(M_s) = M_{h(r)} \cup M_{h(s)} = M_{h(r)|h(s)} = M_{h(r|s)}$ □

1. Einführung
2. Endliche Automaten und reguläre Sprachen
3. Kontextfreie Sprachen und Kellerautomaten
4. Typ1 und Typ0-Sprachen, Turing-Maschinen
5. Berechenbarkeit

2 Endliche Automaten und reguläre Sprachen

- Definition endlicher Automaten
- Reguläre Sprachen
- Automatenminimierung
- Nichtdeterministische endliche Automaten
- Abschlusseigenschaften
- Reguläre Grammatiken
- String Matching mit endlichen Automaten
- Entscheidungsprobleme für reguläre Sprachen

Jetzt:

2. Endliche Automaten und reguläre Sprachen
   - Definition endlicher Automaten
   - Reguläre Sprachen
   - Automatenminimierung
   - Nichtdeterministische endliche Automaten
   - Abschlusseigenschaften
   - Reguläre Grammatiken
   - String Matching mit endlichen Automaten
   - Entscheidungsprobleme für reguläre Sprachen

Motivation

Grundidee: „Finite state machine“

- endliche viele Zustände, darunter ein Startzustand
- Einwirkungen von außen bewirken Zustandsänderung

Motivation

Grundidee: „Finite state machine“

- endliche viele Zustände, darunter ein Startzustand
- Einwirkungen von außen bewirken Zustandsänderung

Anwendungen

- Getränkeautomat (Münze, Getränk/Abbruch wählen, entnehmen)
- Handlungsabläufe: Bedienungsanleitung, Notfallplan
- Ampelschaltung: Farbwechsel nach Zeitablauf (ggf. angeregt durch Fußgängersignal, beeinflusst durch Verkehrsdichte)
- Textanalyse: ist *dieses Wort* enthalten?, ...
- Bestandteil von Compilern: Lexer — extrahiert und klassifiziert Tokens aus dem Quelltext (Bezeichner, Schlüsselwörter, Literale, ...)

usw.

Motivation

Grundidee: „Finite state machine“

- endliche viele Zustände, darunter ein Startzustand
- Einwirkungen von außen bewirken Zustandsänderung

Anwendungen

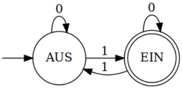
- Getränkeautomat (Münze, Getränk/Abbruch wählen, entnehmen)
- Handlungsabläufe: Bedienungsanleitung, Notfallplan
- Ampelschaltung: Farbwechsel nach Zeitablauf (ggf. angeregt durch Fußgängersignal, beeinflusst durch Verkehrsdichte)
- Textanalyse: ist *dieses Wort* enthalten?, ...
- Bestandteil von Compilern: Lexer — extrahiert und klassifiziert Tokens aus dem Quelltext (Bezeichner, Schlüsselwörter, Literale, ...)

usw.

Endliche Automaten in theoretischer Informatik

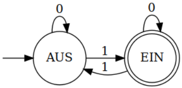
erkennende Automaten: keine Ausgabe,
Resultat wird durch erreichten Zustand signalisiert

Informales Beispiel: Ein Eingabe-gesteuerter Kippschalter



Lesen von „1“ löst Schaltvorgang aus, Lesen von „0“ wird ignoriert.

Informales Beispiel: Ein Eingabe-gesteuerter Kippschalter

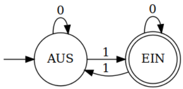


Lesen von „1“ löst Schaltvorgang aus, Lesen von „0“ wird ignoriert.

Bestandteile

- *Zustände* (eingekreist) darunter der *Startzustand* (Pfeil von außen)
- *Zustandsübergänge* (Pfeile, markiert mit verarbeitbaren Zeichen)
- *Endzustand* (Doppelkreis)

Informales Beispiel: Ein Eingabe-gesteuerter Kippschalter



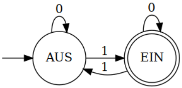
Lesen von „1“ löst Schaltvorgang aus, Lesen von „0“ wird ignoriert.

Bestandteile

- *Zustände* (eingekreist) darunter der *Startzustand* (Pfeil von außen)
- *Zustandsübergänge* (Pfeile, markiert mit verarbeitbaren Zeichen)
- *Endzustand* (Doppelkreis)

Das Bild (Zustandsdiagramm) zeigt nicht den Schalter selbst, sondern ein *formales Berechnungsmodell* davon.

Informales Beispiel: Ein Eingabe-gesteuerter Kippschalter



Lesen von „1“ löst Schaltvorgang aus, Lesen von „0“ wird ignoriert.

Bestandteile

- *Zustände* (eingekreist) darunter der *Startzustand* (Pfeil von außen)
- *Zustandsübergänge* (Pfeile, markiert mit verarbeitbaren Zeichen)
- *Endzustand* (Doppelkreis)

Das Bild (Zustandsdiagramm) zeigt nicht den Schalter selbst, sondern ein *formales Berechnungsmodell* davon.

Arbeitsweise

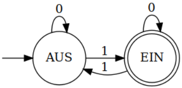
- beginne im Startzustand
- nacheinander Zeichen verarbeiten und Zustand ändern wie angegeben („bedingte Sprünge“)
- Eingaben, die zu Endzustand führen, heißen akzeptiert (ein sugesstiverer Name für Endzustand ist Akzeptierungszustand)

Die erkannte Sprache

Die vom Automaten erkannte Sprache (auch „die Sprache des Automaten“) ist die Menge akzeptierter Eingaben.

Die erkannte Sprache

Die vom Automaten erkannte Sprache (auch „die Sprache des Automaten“) ist die Menge akzeptierter Eingaben.

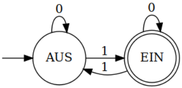


Frage

Welche Sprache wird von obigem Automaten erkannt?

Die erkannte Sprache

Die vom Automaten erkannte Sprache (auch „die Sprache des Automaten“) ist die Menge akzeptierter Eingaben.



Frage

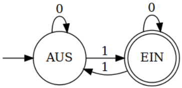
Welche Sprache wird von obigem Automaten erkannt?

Antwort

Menge der Bitmuster mit ungerader Anzahl an Einsen

Die erkannte Sprache

Die vom Automaten erkannte Sprache (auch „die Sprache des Automaten") ist die Menge akzeptierter Eingaben.



Frage

Welche Sprache wird von obigem Automaten erkannt?

Antwort

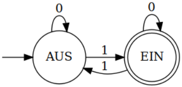
Menge der Bitmuster mit ungerader Anzahl an Einsen

Beobachtung

Für die erkannte Sprache sind die Zustandsnamen unwichtig, anstelle AUS und EIN könnte man genauso gut auch q_0 und q_1 verwenden.

Die erkannte Sprache

Die vom Automaten erkannte Sprache (auch „die Sprache des Automaten") ist die Menge akzeptierter Eingaben.



Frage

Welche Sprache wird von obigem Automaten erkannt?

Antwort

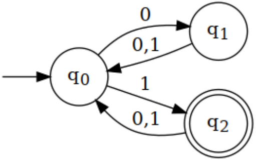
Menge der Bitmuster mit ungerader Anzahl an Einsen

Beobachtung

Für die erkannte Sprache sind die Zustandsnamen unwichtig, anstelle AUS und EIN könnte man genauso gut auch q_0 und q_1 verwenden.

Wichtig sind nur ihre Rollen (Start-, End- oder anderer Zustand) und die Übergänge zwischen ihnen.

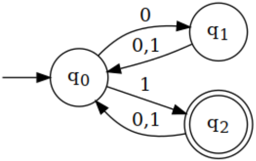
Informales Beispiel 2



Frage

Welche Sprache wird von dem Automaten erkannt?

Informales Beispiel 2



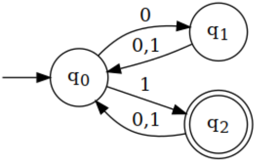
Frage

Welche Sprache wird von dem Automaten erkannt?

Antwort

- Akzeptierte Wörter müssen eine Länge $2k + 1$ haben und auf 1 enden.

Informales Beispiel 2



Frage

Welche Sprache wird von dem Automaten erkannt?

Antwort

- Akzeptierte Wörter müssen eine Länge $2k + 1$ haben und auf 1 enden.
- Jedes solche Wort wird akzeptiert: Nach den ersten $2k$ Zeichen befindet sich der Automat in q_0. Nur eine 1 am Ende führt zur Akzeptierung.

Vorbereitung: Etwas Mengenlehre

Theoretische Berechnungsmodelle beruhen nur auf mengentheoretischen Begriffen, insbesondere Funktionen und (geordneten) Paaren bzw. n-Tupeln.

Vorbereitung: Etwas Mengenlehre

Theoretische Berechnungsmodelle beruhen nur auf mengentheoretischen Begriffen, insbesondere Funktionen und (geordneten) Paaren bzw. n-Tupeln. Letztere benutzen wir als Grundbegriffe:

Das geordnete Paar (a, b) ist die Zusammenfassung von a und b zu einer Einheit.

Vorbereitung: Etwas Mengenlehre

Theoretische Berechnungsmodelle beruhen nur auf mengentheoretischen Begriffen, insbesondere Funktionen und (geordneten) Paaren bzw. n-Tupeln. Letztere benutzen wir als Grundbegriffe:

Das geordnete Paar (a, b) ist die Zusammenfassung von a und b zu einer Einheit. Allgemeiner ist ein geordnetes n-Tupel $(a_1, a_2, ..., a_n)$ die Zusammenfassung von n Elementen $a_1, a_2, ..., a_n$ zu einer Einheit. a_1 ist die erste, . . . , a_n die n-te Komponente des Tupels.

Vorbereitung: Etwas Mengenlehre

Theoretische Berechnungsmodelle beruhen nur auf mengentheoretischen Begriffen, insbesondere Funktionen und (geordneten) Paaren bzw. n-Tupeln. Letztere benutzen wir als Grundbegriffe:

Das geordnete Paar (a, b) ist die Zusammenfassung von a und b zu einer Einheit. Allgemeiner ist ein geordnetes n-Tupel $(a_1, a_2, ..., a_n)$ die Zusammenfassung von n Elementen $a_1, a_2, ..., a_n$ zu einer Einheit. a_1 ist die erste, . . . , a_n die n-te Komponente des Tupels.

Definition

Das Kreuzprodukt (*kartesisches Produkt*) der Mengen $A_1, A_2, ..., A_n$ ist die Menge

$$A_1 \times \cdots \times A_n := \{(a_1, ..., a_n) : a_1 \in A_1 \wedge ... \wedge a_n \in A_n\}$$

Vorbereitung: Etwas Mengenlehre

Theoretische Berechnungsmodelle beruhen nur auf mengentheoretischen Begriffen, insbesondere Funktionen und (geordneten) Paaren bzw. n-Tupeln. Letztere benutzen wir als Grundbegriffe:

Das geordnete Paar (a, b) ist die Zusammenfassung von a und b zu einer Einheit. Allgemeiner ist ein geordnetes n-Tupel $(a_1, a_2, \ldots, a_n)$ die Zusammenfassung von n Elementen $a_1, a_2, \ldots, a_n$ zu einer Einheit. a_1 ist die erste, . . . , a_n die n-te Komponente des Tupels.

Definition

Das Kreuzprodukt (*kartesisches Produkt*) der Mengen $A_1, A_2, \ldots, A_n$ ist die Menge

$$A_1 \times \cdots \times A_n := \{(a_1, \ldots, a_n) : a_1 \in A_1 \wedge \ldots \wedge a_n \in A_n\}$$

Umsetzung in Python

```
pair = (a,b) = (4,(5,6,7))  # ein Paar als Einheit aus Zahl und 3-Tupel
pair[0] == a and pair[1] == b == (5,6,7) # Komponentenzugriff
A = {1,2}; B = {3,4}                     # zwei endliche Mengen und ihr ..
{(x,y) for x in A for y in B} == {(1,3),(1,4),(2,3),(2,4)} #.. Kreuzprodukt
```

Der Begriff der Funktion

Definition

Eine Funktion oder *Abbildung* $f : X \to Y$ aus einer Menge X in eine Menge Y (Urbild- und Bildbereich) ist eine Teilmenge $f \subseteq X \times Y$ von Paaren (x, y) mit:

Für jedes $x \in X$ gibt es höchstens ein y, so dass $(x, y) \in f$.

Der Begriff der Funktion

Definition

Eine Funktion oder *Abbildung* $f : X \to Y$ aus einer Menge X in eine Menge Y (Urbild- und Bildbereich) ist eine Teilmenge $f \subseteq X \times Y$ von Paaren (x, y) mit:

Für jedes $x \in X$ gibt es höchstens ein y, so dass $(x, y) \in f$.

Sprechweise: f ordnet x den Funktionswert y zu
(Notation: $f(x) = y$ oder $f : x \mapsto y$)

Der Begriff der Funktion

Definition

Eine Funktion oder *Abbildung* $f : X \to Y$ aus einer Menge X in eine Menge Y (Urbild- und Bildbereich) ist eine Teilmenge $f \subseteq X \times Y$ von Paaren (x, y) mit:

Für jedes $x \in X$ gibt es höchstens ein y, so dass $(x, y) \in f$.

Sprechweise: f ordnet x den Funktionswert y zu
(Notation: $f(x) = y$ oder $f : x \mapsto y$)

Beispiel zur Realisierung als Python-Dictionary

Voraussetzung: Es gibt nur endlich viele Zuordnungspaare.

```
f = { # Die Zuordnungspaare (x,y) werden in der Form x:y aufgefuehrt:
0:'zero', 1: 'one',       # Festlegung der Zuordnung fuer ..
2:'digit', 3:'digit'      # .. Elemente 0, 1, 2, 3
}

f[4] = 'digit'            # weitere Zuordnung festlegen
b = f[3]                  # ACHTUNG: auch beim Abruf eckige Klammern [] verwenden
```

Der Begriff der Funktion

Definition

Eine Funktion oder *Abbildung* $f : X \to Y$ aus einer Menge X in eine Menge Y (Urbild- und Bildbereich) ist eine Teilmenge $f \subseteq X \times Y$ von Paaren (x, y) mit:

Für jedes $x \in X$ gibt es höchstens ein y, so dass $(x, y) \in f$.

Sprechweise: f ordnet x den Funktionswert y zu
(Notation: $f(x) = y$ oder $f : x \mapsto y$)

Beispiel zur Realisierung als Python-Dictionary

Voraussetzung: Es gibt nur endlich viele Zuordnungspaare.

```
f = { # Die Zuordnungspaare (x,y) werden in der Form x:y aufgefuehrt:
0:'zero', 1: 'one',      # Festlegung der Zuordnung fuer ..
2:'digit', 3:'digit'     # .. Elemente 0, 1, 2, 3
}

f[4] = 'digit'           # weitere Zuordnung festlegen
b = f[3]                 # ACHTUNG: auch beim Abruf eckige Klammern [] verwenden
```

Der Urbildbereich von `f` ist `{0, 1, 2, 3, 4}`. Das sind die *Schlüssel* des Dictionaries und `'zero'`, `'one'`, 'digits' sind seine *Werte*.

Formale Beschreibung endlicher Automaten

Definition

Ein deterministischer endlicher Automat (*deterministic finite automaton* — DFA) ist ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ bestehend aus

- einer endlichen Menge Q (Zustandsmenge)
- einem Alphabet Σ (Eingabealphabet)
- einer Funktion $\delta : Q \times \Sigma \to Q$ (Übergangsfunktion)
- einem Element $q_0 \in Q$ (Startzustand) und
- einer Menge $F \subseteq Q$ (Endzustände, *Akzeptierungszustände*, ...).

Formale Beschreibung endlicher Automaten

Definition

Ein deterministischer endlicher Automat (*deterministic finite automaton* — DFA) ist ein 5-Tupel $(Q, \Sigma, \delta, q_0, F)$ bestehend aus

- einer endlichen Menge Q (Zustandsmenge)
- einem Alphabet Σ (Eingabealphabet)
- einer Funktion $\delta : Q \times \Sigma \to Q$ (Übergangsfunktion)
- einem Element $q_0 \in Q$ (Startzustand) und
- einer Menge $F \subseteq Q$ (Endzustände, *Akzeptierungszustände*, ...).

Maschinenlesbare Spezifikation zu Beispiel 2

```
{'Q': {'q1', 'q2', 'q0'},                    # 'Q': Wert = eine Menge von Strings
 'Sigma': {'0', '1'},                        # 'Sigma': Wert = eine Menge von Strings
 'Delta': {('q1', '0'): 'q0',                # 'Delta': Wert = ein Dictionary
           ('q1', '1'): 'q0',                # Schluessel:
           ('q2', '0'): 'q0',                #  (Zustand,Symbol)-Paare
           ('q2', '1'): 'q0',
           ('q0', '0'): 'q1',
           ('q0', '1'): 'q2'},
 'q0': 'q0',                                 # 'q0': Wert = ein String
 'F': {'q2'}                                 # 'F': Wert = eine Menge von Strings
}
```

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...
  - beliebige Wörter der Form $(111)^n$

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...
  - beliebige Wörter der Form $(111)^n$
  - eingestreute 0en „stören nicht“

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...
  - beliebige Wörter der Form $(111)^n$
  - eingestreute 0en „stören nicht“
- da ε akzeptiert wird, ist der Startzustand gleichzeitig auch Endzustand

Beispiel: Einen String-Klassifizierer konstruieren

Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...
  - beliebige Wörter der Form $(111)^n$
  - eingestreute 0en „stören nicht“
- da ε akzeptiert wird, ist der Startzustand gleichzeitig auch Endzustand
- ergänze Übergänge, so dass jeder 0/1-String verarbeitet werden kann

Beispiel: Einen String-Klassifizierer konstruieren

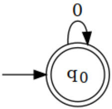
Aufgabe

DFA angeben, der die 0/1-Strings mit durch 3 teilbarer Anzahl von 1en akzeptiert, andere *verwirft*.

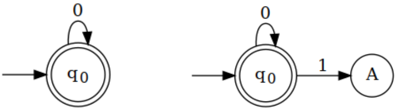
Idee

- bestimme charakteristische akzeptierte Wörter
  - ε, 111 sind die kürzesten, dann 0111, 1011, ...
  - beliebige Wörter der Form $(111)^n$
  - eingestreute 0en „stören nicht"
- da ε akzeptiert wird, ist der Startzustand gleichzeitig auch Endzustand
- ergänze Übergänge, so dass jeder 0/1-String verarbeitet werden kann
- finde klarere Darstellung/Bedeutung der Zustände

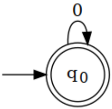
Umsetzung der Idee

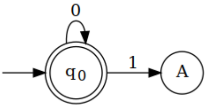


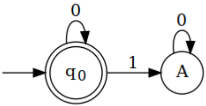
Umsetzung der Idee



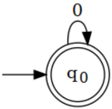
Umsetzung der Idee

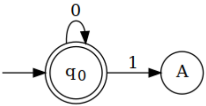


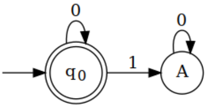


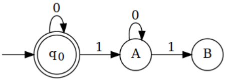


Umsetzung der Idee

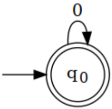


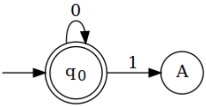


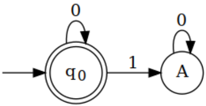


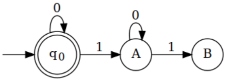


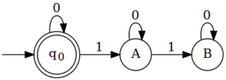
Umsetzung der Idee



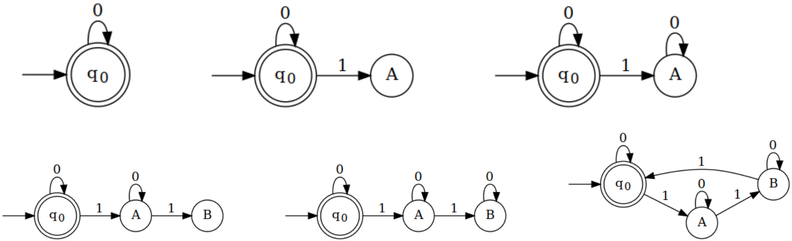




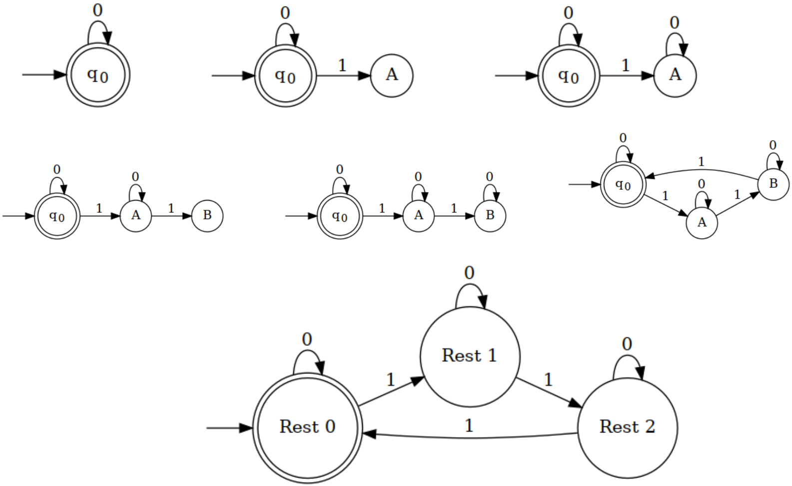




Umsetzung der Idee



Umsetzung der Idee



Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion von X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$.

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion von X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$.
Anderenfalls heißt f partiell (oder Funktion aus X nach Y, Notation $f : X \overset{\supseteq}{\to} Y$).

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion von X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$. Anderenfalls heißt f partiell (oder Funktion aus X nach Y, Notation $f : X \overset{\supseteq}{\to} Y$).

Beispiele (werden später noch benötigt)

Die Vorgängerfunktion ist eine partielle Funktion $\mathrm{pred} : \mathbb{N} \overset{\supseteq}{\to} \mathbb{N}$.

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion von X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$. Anderenfalls heißt f partiell (oder Funktion aus X nach Y, Notation $f : X \overset{\supset}{\to} Y$).

Beispiele (werden später noch benötigt)

Die Vorgängerfunktion ist eine partielle Funktion $\mathrm{pred} : \mathbb{N} \overset{\supset}{\to} \mathbb{N}$. Durch die Festlegung $\mathrm{pred}(0) := 0$ wird sie „totalisiert“.

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion von X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$. Anderenfalls heißt f partiell (oder Funktion aus X nach Y, Notation $f : X \stackrel{\supseteq}{\to} Y$).

Beispiele (werden später noch benötigt)

Die Vorgängerfunktion ist eine partielle Funktion $\mathrm{pred} : \mathbb{N} \stackrel{\supseteq}{\to} \mathbb{N}$. Durch die Festlegung $\mathrm{pred}(0) := 0$ wird sie „totalisiert“.
Analog: Subtraktion wird totalisiert durch $\mathrm{sub}(x, y) := 0$, falls $x \leq y$.

Partielle und totale Funktionen

Sei $f : X \to Y$ eine Abbildung. Der Definitionsbereich von f ist die Menge der x, für die eine Zuordnung festgelegt ist:

$$D_f = \{x : x \in X \wedge \exists y \in Y \wedge f(x) = y\}.$$

Definition

f heißt total (Funktion <u>von</u> X nach Y, Bezeichnung $f : X \to Y$), falls $D_f = X$.
Anderenfalls heißt f partiell (oder Funktion <u>aus</u> X nach Y, Notation $f : X \overset{\supseteq}{\to} Y$).

Beispiele (werden später noch benötigt)

Die Vorgängerfunktion ist eine partielle Funktion $\mathrm{pred} : \mathbb{N} \overset{\supseteq}{\to} \mathbb{N}$. Durch die Festlegung $\mathrm{pred}(0) := 0$ wird sie „totalisiert".
Analog: Subtraktion wird totalisiert durch $\mathrm{sub}(x, y) := 0$, falls $x \leq y$.

```
def pred(x):                          # modifizierter Vorgaenger
    return x-1 if x>0 else 0
def sub(x,y):                         # modifizierte Subtraktion
    return x-y if x>y else 0
```

Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten.

Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \rightarrow Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert.

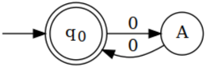
Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert. Zur Unterscheidung nennen wir solche DFAs partiell, die anderen total oder *vollständig*.

Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert. Zur Unterscheidung nennen wir solche DFAs partiell, die anderen total oder *vollständig*.

Beispiel



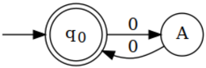
Erkannte Sprache:
„gerade Anzahl 0en“.

Sackgasse = ein Zustand von dem aus kein Akzeptierungszustand erreichbar ist.

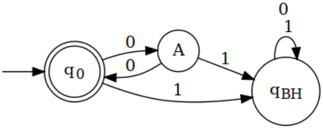
Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert. Zur Unterscheidung nennen wir solche DFAs partiell, die anderen total oder *vollständig*.

Beispiel



Erkannte Sprache:
„gerade Anzahl 0en“.



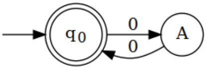
Für Eingabealphabet $\{0, 1\}$ totalisiert,
erkannte Sprache bleibt gleich.

Sackgasse = ein Zustand von dem aus kein Akzeptierungszustand erreichbar ist.

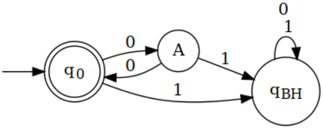
Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert. Zur Unterscheidung nennen wir solche DFAs partiell, die anderen total oder *vollständig*.

Beispiel



Erkannte Sprache: „gerade Anzahl 0en".



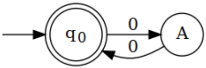
Für Eingabealphabet $\{0, 1\}$ totalisiert, erkannte Sprache bleibt gleich.

Sackgasse = ein Zustand von dem aus kein Akzeptierungszustand erreichbar ist.

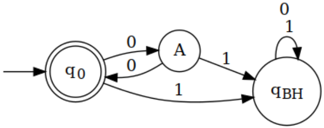
Unvollständige Automaten und ihre Vervollständigung

$\delta : Q \times \Sigma \to Q$ erfordert ein Zustandsdiagramm mit $|Q \times \Sigma|$ Kanten. *Partielle* Übergangsfunktionen sind oft übersichtlicher: Ist $\delta(q, \sigma)$ nicht definiert, so wird σ von q aus nicht verarbeitet, und die Eingabe nicht akzeptiert. Zur Unterscheidung nennen wir solche DFAs partiell, die anderen total oder *vollständig*.

Beispiel



Erkannte Sprache:
„gerade Anzahl 0en“.



Für Eingabealphabet $\{0, 1\}$ totalisiert,
erkannte Sprache bleibt gleich.

Sackgasse = ein Zustand von dem aus kein Akzeptierungszustand erreichbar ist.

Lemma (DFA-Vervollständigung)

Durch (eventuelle Hinzunahme von) Sackgassen, kann man jeden partiellen DFA totalisieren, ohne seine Sprache zu ändern.