Modellierung

Modell

Abstraktion der eines komplexen Systems - beschrieben mit Modellierungssprachen.

1. Deskriptive Modelle

   komplexes System → Modell

   Zum Analysieren

2. Präskriptive Modelle

   Modell → komplexes System

   Zum Planen / Konstruieren

Sprache

1. Natürliche Sprachen

   komplex, mehrdeutig, unscharf, universell einsetzbar

2. Formale Sprachen

   künstlich, präzise, beschränkte Einsetzungsgebiete

   zB: mathematische Notation, Programmiersprachen, logische Formalsprachen

Aspekte einer formalen Sprache:

1. Syntax — Welche Äußerungen sind zugelassen?
2. Semantik — Was bedeutet eine zulässige Äußerung?
3. Ausdrucksstärke — Was kann ausgedrückt werden, was nicht?
4. Verwendung — Wie setzt man diese Sprache zum Modellieren ein?

Formale Modellierung

= Formalisierung / Formale Spezifikation

Ist eine Übersetzung:

Problemstellungen in natürlicher Sprache → Beschreibungen in einer formalen Sprache

Formale Modellierungs-Sprachen in dieser LVA können textuell oder graphisch sein.

1. Textuelle Sprachen:
   - Logische Sprachen: Aussagenlogik, Prädikatenlogik
   - Reguläre Ausdrücke
   - Formale Grammatiken
2. Graphische Sprachen
   - Endliche Automaten
   - Petri-Netze (quasi eine Erweiterung von endlichen Automaten)

Logik

Logik

beschäftigt sich mit Schlussfolgerungen und besteht aus:

1. logische Symbole, Variablen — Notation für logische Formel (Zeichenfolge) / Term
2. Syntax — Regeln für Wohlgeformtheit einer Formel
3. Semantik — Bedeutung - Wahrheitswerte, Wahrheit von Formeln
4. Konsequenzrelation — Logischer Zusammenhang zwischen Formeln
5. Inferenzregeln (logischer Kalkül) — Beweis von Formeln - nicht Teil dieser LVA

Beispiel: Inferenzregeln

Junktoren

Logische Junktoren = Operatoren, Konnektive, Funktionen

Bildung zusammengesetzter Aussagen

Bestimmen die Ausdrucksmöglichkeiten.

Junktoren in der

- Aussagenlogik — und, entweder-oder, genau dann-wenn, . . .
- Zeitlogik — morgen, gestern, im nächsten Moment, bis, . . .
- Modallogik — notwendigerweise, möglicherweise, . . .

Quantoren

Aussage über Anzahl

- Elementare Logik — und, oder, nicht, für alle, . . .
- Zeitlogiken — im nächsten Moment, für immer, irgendwann später, . . .
- Modallogiken — ich glaube/weiß, dass, es ist möglich/notwendig, dass, . . .

Unterschiede zwischen Logiken

Anzahl der Wahrheitswerte

Zweiwertige (boolsche) Logik: wahr / falsch

Mehrwertige Logiken: wahr / falsch / unbekannt / widersprüchlich, . . .

Fuzzy logic: [0,1] (alle reellen Zahlen zwischen 0 und 1)

Quantoren

Aussagenlogik: keine Quantoren

Quantifizierte Aussagenlogik: Quantoren über Aussagenvariablen

Prädikatenlogik: Quantoren über Individuenvariablen

Logiken höherer Stufen: Quantoren über Funktionen und Prädikate

Syntax vs. Semantik

Hinter verschiedener Syntax ("one", "eins", "1") kann die selbe Bedeutung stehen (das Konzept).

Syntax

- Regeln für Gültigkeit der notierten Zeichen- / Symbol-Ketten = Formeln
- Regeln dafür welche Zeichenketten gültig sind
- Formellsprache = Metasprache um eine andere Sprache zu beschreiben / Syntax zu induktiv zu bestimmen.

Semantik (Funktion)

- Zeichenkette $\mapsto$ Bedeutung

Beispiel: Inferenzregeln

Beispiel: Inferenzregeln

Inferenzen / Inferenzregeln

> Prämisse $\wedge$ Prämisse $\wedge$ ... $\wedge$ Prämisse $\Rightarrow$ Konklusion
>
> Logische Konsequenz: $F_1, \ldots, F_n \models, G$ genau dann wenn $(F_1 \wedge ... \wedge F_n) \supset G$

Konklusion kann falsch sein obwohl Inferenzregel an sich gültig ist indem die eingesetzten Aussagen für die Prämissen nicht stimmen.

Gültig

Diese Inferenzregel ist gültig, aber mit falschen Formeln als Prämisse ist auch die Konklusion falsch.

Alle x sind y	Prämisse
z ist ein x	Prämisse
z ist ein y	Konklusion

Alle Menschen sind sterblich	wahr
Sokrates ist ein Mensch	wahr
Sokrates ist sterblich	wahr

▼ Als prädikatenlogische Formel

Alle Menschen sind sterblich	$\forall x\ (Mensch(x)) \supset Sterblich(x)$
Sokrates ist ein Mensch	$Mensch(sokrates)$
Sokrates ist sterblich	$Sterblich(sokrates)$

Alle US-Präsidenten sind Amerikaner.	wahr
Schwarzenegger ist US-Präsident.	falsch (stimmt nicht)
Schwarzenegger ist Amerikaner.	falsch

Ungültig

Diese Inferenzregel ist ungültig (Generalisierung)

Alle x sind y	Prämisse
z ist ein y	Prämisse
z ist ein x	Konklusion

Aussagenlogik

Aussagenlogik / Propositional-Logik

Proposition = Aussage

Wahrheitswerte	wahr / falsch
Aussagenvariablen	können wahr / falsch sein
Junktoren	elementare Operatoren (Siehe unten)
Quantoren	keine

Operationen

Sind Funktionen und beschreiben Operatoren-Symbole auf semantischer Ebene.

¬ Negation

∧ Konjunktion

∨ Disjunktion

$\not\equiv$, $\not\Leftrightarrow$ Antivalenz, XOR

≡, ⇔ Äquivalenz, ¬XOR, iff

Beispiel:

Immer wenn du hüpfst, dann hüpfe ich auch. *Sonst nicht*.

⊃, ⇒ Implikation

Beispiel:

Immer wenn du hüpfst, dann hüpfe ich auch. *Sonst vielleicht aber auch*.

Beispiel:

Wenn / Falls ich ins Kino gehe, dann esse ich dort immer Popkorn.

(Gleiche Aussage wie) Ich gehe nur dann ins Kino, wenn ich dort Popkorn esse.

Das liegt daran, dass ich nicht im Kino sein kann ohne Popkorn zu essen.

↑ Negierte Konjunktion, nand

↓ Negierte Disjunktion, nor

Funktionale Vollständigkeit

Eine Menge von Funktionen mit denen alle Funktionen (einer Funktions-Klasse) ausgedrückt werden können.

Beispiele:

{not, and}, {nand}, {nor}, {not, or}, {not, implies}, {implies, false}

Induktive Definition

Allgemeiner Vorgang zur induktiven Definition unendlicher Mengen:

$\mathcal{U}$ Universum, Menge aller relevanten Elemente

$\mathcal{M}_0 \subseteq \mathcal{U}$ Menge von Grundelementen

$f_1 : \mathcal{U}^{n_1} \mapsto \mathcal{U}, \quad f_2 : \mathcal{U}^{n_2} \mapsto \mathcal{U}$ Konstruktionsfunktionen

1. Stufenweise Konstruktion der Menge $\mathcal{M}$

   $$\begin{aligned}\mathcal{M}_{i+1} = \mathcal{M}_i \cup &\{f_1(x_1, \ldots, x_{n_1}) \mid x_1, \ldots, x_{n_1} \in \mathcal{M}_i\} \\ &\cup \{f_2(x_1, \ldots, x_{n_2}) \mid x_1, \ldots, x_{n_2} \in \mathcal{M}_i\} \\ &\cup \cdots\end{aligned}$$

   $$\mathcal{M} = \lim_{i\to\infty} \mathcal{M}_i = \bigcup_{i\geq 0} \mathcal{M}_i$$

2. Induktive Definition der Menge $\mathcal{M}$

   $\mathcal{M}$ ist die kleinste Menge für die gilt:

   - $\mathcal{M}_0 \subseteq \mathcal{M}$
   - Wenn $x_1, \ldots, x_{n_1} \in \mathcal{M}$, dann $f_1(x_1, \ldots, x_{n_1}) \in \mathcal{M}$
   - Wenn $x_1, \ldots, x_{n_1} \in \mathcal{M}$, dann $f_2(x_1, \ldots, x_{n_1}) \in \mathcal{M}$
   - $\ldots$

Beispiele:

▼ Menge der geraden Zahlen unter $\mathbb{N}$

1. Stufenweise Konstruktion (Konstruktiv)
   - 0 ist eine gerade Zahl: $G_0 = \{0\}$
   - $G_1 = G_0 \cup \{n+2 \mid n \in G_0\} = \{0, 2\}$
   - $G_{i+1} = G_i \cup \{n+2 \mid n \in G_i\} = \{0, 2, 4, ..., 2(i+1)\}$
   - $\mathbb{G} = lim_{i\to\infty} G_i = \bigcup_{i\geq 0} G_i$

2. Induktive Konstruktion (Nicht konstruktiv - greift auf bestehende Menge $\mathbb{G}$ zurück)

   $\mathbb{G}$ ist die kleinste Menge für die gilt:

   - $0 \in \mathbb{G}$
   - Wenn $n \in \mathbb{G}$ dann auch $n + 2 \in \mathbb{G}$

▼ Geschachtelte Klammern

Gesucht: Menge der syntaktisch richtigen Folgen von runden, eckigen, geschwungenen Klammern. wie zb `[{()}]`.

Induktive Definition

$\mathcal{K}$ ist die kleinste Menge der Klammerfolgen für die gilt:

1. $\{\texttt{()}, \texttt{[]}, \texttt{\{\}}\} \subseteq \mathcal{K}$
2. Wenn $x \in \mathcal{K}$, dann auch $(x) \in \mathcal{K}$
3. Wenn $x \in \mathcal{K}$, dann auch $[x] \in \mathcal{K}$
4. Wenn $x \in \mathcal{K}$, dann auch $\{x\} \in \mathcal{K}$

Um zu beweisen dass `[{()}]` $\in \mathcal{K}$

$() \in \mathcal{K}$
Da $() \in \mathcal{K}$, gilt auch $\{()\} \in \mathcal{K}$. wegen $(k4)$
Da $\{()\} \in \mathcal{K}$, gilt auch $[\{()\}] \in \mathcal{K}$. wegen$(k3)$
Da $[\{()\}] \in \mathcal{K}$, gilt auch $[[\{()\}]] \in \mathcal{K}$. wegen $(k3)$
Da $[[\{()\}]] \in \mathcal{K}$, gilt auch $([[\{()\}]]) \in \mathcal{K}$. wegen $(k2)$

Induktive Definition der Syntax in der Aussagenlogik

$\mathcal{U}$ Menge aller Zeichenketten (aus Variablen, Operatoren, Klammern)

$\mathcal{V} := \{A, B, C, ...A_0, A_1, ..., \top, \bot\}$ Aussagenlogische Variablen als Grundelemente

Konstruktionsfunktionen:

$$
\begin{aligned}
&f_{\neg}(F) = \neg F \\
&f_{\wedge}(F, G) = (F \wedge G_n) \\
&f_{\uparrow}(F, G) = (F_n \uparrow G_n) \\
&\vdots \\
&f_{\subset}(F, G) = (F \subset G_n)
\end{aligned}
$$

Induktive Definition der Menge $\mathcal{A}$

Die Menge $\mathcal{A}$ der aussagenlogischen Formeln F_i ist die kleinste für die gilt:

1. $\mathcal{V} \subseteq \mathcal{A}$ (eine Variable ist eine gültige Formel)
2. $\{\top, \bot\} \subseteq \mathcal{A}$
3. Wenn $\mathsf{F} \in \mathcal{A}$ dann auch $\neg\mathsf{F} \in \mathcal{A}$
4. Wenn $\mathsf{F}, \mathsf{G} \in \mathcal{A}$ dann auch $(\mathsf{F} * \mathsf{G}) \in \mathcal{A}$

   wobei $* := \{\wedge, \vee, \uparrow, \downarrow, \equiv, \not\equiv, \subset, \supset\}$

Semantik

Die Semantik ist eine Funktion die Zeichenkette $\mapsto$ Bedeutung.

$I : \mathcal{V} \mapsto \mathbb{B}$ Wahrheitsbelegung / Interpretation (für einzelne Variablen)

Mit Wahrheitswerten $\mathbb{B} = \{1, 0\}$

$\mathcal{I} = \{I \mid I : \mathcal{V} \mapsto \mathbb{B}\}$ Menge aller Interpretationen

Semantik von Formeln (Auswertung)

Wahrheits-Wert einer Formel mit der Belegung / Interpretation I festgelegt durch die Funktion:

$\mathsf{val}: \mathcal{I} \times \mathcal{A} \mapsto \mathbb{B}$

$\mathsf{val}(I, A) = \mathsf{val}_I(A)$

Für jede Operation auf der syntaktischen Ebene gibt es auch eine auf der semantischen Ebene (Metaebene):

$$
\begin{aligned}
&\mathrm{val}_I(A) = I(A) \text{ für } A \in \mathcal{V} \\
&\mathrm{val}_I(\top) = 1 \text{ und } \mathrm{val}_I(\bot) = 0 \text{ ;} \\
&\mathrm{val}_I(\neg F) = \mathrm{not}\, \mathrm{val}_I(F) \\
&\mathrm{val}_I((F * G)) = \mathrm{val}_I(F) \circledast \mathrm{val}_I(G)
\end{aligned}
$$

wobei $\circledast$ die logische Funktion zum Operator $*$ ist.

Beispiele

Beispiele für $\circledast$: Ausdrücken von Semantik in der Aussagenlogik. (Alles darunter bezieht sich nur auf Aussagenlogik).

Semantische Äquivalenz

Semantik $=$ ist durch $\equiv$ ausdrückbar.

$F = G$ sind semantisch äquivalent wenn $\mathsf{val}_I(F) = \mathsf{val}_I(G)$ gültig ist.

Logische Konsequenz

Die Semantik $\models$ ist durch $\supset$ ausdrückbar

$F_1, \ldots, F_n \models, G$ genau dann wenn $F_1 \supset (F_2 \supset \cdots (F_n \supset G) \cdots)$

Und weil $A \supset (B \supset C) = (A \wedge B) \supset C$ folgt daraus die äquivalenz mit $(F_1 \wedge ... \wedge F_n) \supset G$.

Anwendungen:

- "Falls in der Wahrheitsbelegung I alle Prämissen wahr sind, dann ist auch die Konklusion wahr in I." (Kriterium für die Gültigkeit von Inferenzregeln)
- "Die Formel G ist eine logische Konsequenz der Formeln $F_1, ..., F_n$"

  $\models F$ ist eine Abkürzug dafür, dass F gültig ist. Man muss es quasi als $\text{true} \models F$ lesen.

Eigenschaften von Formeln

gültig wenn für alle Wahrheitsb. immer $\text{val}_I(F) = 1$

Immer wahr

erfüllbar wenn für mindestens eine Wahrheitsb. $\text{val}_I(F) = 1$

Mind. bei 1 Fall wahr

widerlegbar wenn für mindestens eine Wahrheitsb. $\text{val}_I(F) = 0$

Mind. bei 1 Fall falsch

unerfüllbar wenn für alle Wahrheitsb. immer $\text{val}_I(F) = 0$

Immer falsch

Bool'sche Algebra

In der boolschen Algebra gelten die Eigenschaften: Assoziativität, Kommutativität, Idempotenz,...

Beispiele für boolsche Algebra:

$\langle \mathbb{B}, \text{ and}, \text{or}, \text{ not}, 0, 1 \rangle$

$\langle 2^M, \cap, \cup, ^-, \emptyset, M \rangle$ (2^M steht für alle möglichen Mengen)

DNF / KNF

Manche meinen mit DNF / KNF die maximale Form bei der alle Variablen von der Funktion vorkommen müssen - in dieser LVA ist nicht diese gemeint.

Anwendung

Von der Funktion f zur dazugehörigen Formel.

Defintion

$\vec{b} = (b_1, \ldots, b_n) \in \mathbb{B}^n$ sind Tupel mit bool'schen Werten.

Wir haben eine Funktion $f : \mathbb{B}^n \mapsto \mathbb{B}$ mit Argumenten A_i die durch Interpretation $I_{\vec{b}}(A_i) = b_i$ belegt werden.

Also $f(A_i) = b \in \mathbb{B}$

Konjunkte / Disjunkte einer Menge von Variablen:

$\bigwedge\{F, G, H, \ldots\} = F \wedge G \wedge H \wedge \cdots$

$\bigvee\{F, G, H, \ldots\} = F \vee G \vee H \vee \cdots$

Wobei $\bigwedge\{\} = \top$ und $\bigvee\{\} = \bot$

DNF: Disjunktionen von Konjunktionen von Literalen (= Variablen oder ihre negierte Form)

Charakteristisches Konjunkt für $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{B}^n$:

$K_{\vec{b}} = \bigwedge\{ A_i \mid b_i = 1,\ i = 1..n\} \wedge \bigwedge\{ \neg A_i \mid b_i = 0,\ i = 1..n\}$

Wobei A_i die Variablen sind die als Argumente der Funktion dienen.

Beispiel:

$I_{\vec{b}} : A_1 \mapsto 1, A_2 \mapsto 0, A_3 \mapsto 1, A_4 \mapsto 1 \quad \Longrightarrow \quad \mathrm{val}_{I_b}\left(K_{\vec{b}}\right) = 1$

$\vec{b} = (1,0,1,1) \quad \Longrightarrow \quad K_{\vec{b}} = A_1 \wedge \neg A_2 \wedge A_3 \wedge A_4$

Für Wahrheitsbelegung I_b hat $K_{\vec{b}}$ den Wert 1 und sonst 0.

Die Konjunkte werden disjunktiv verbunden.

$$\mathrm{DNF}_f = \bigvee \left\{ K_{\vec{b}} \mid f(\vec{b}) = 1, \vec{b} \in \mathbb{B}^n \right\}$$

Dadurch repräsentiert die DNF die Funktion f.

$$\forall \vec{b} \in \mathbb{B}^n : \mathrm{val}_{\vec{b}}\left(\mathrm{DNF}_f\right) = f(\vec{b})$$

DNF: Konjunktionen von Disjunktionen von Literalen

Charakteristisches Disjunkt für $\vec{b} = (b_1, \ldots, b_n) \in \mathbb{B}^n$:

$D_{\vec{b}} = \bigvee \{ A_i \mid b_i = 0,\ i = 1..n \} \vee \bigvee \{ \neg A_i \mid b_i = 1,\ i = 1..n \}$

Wobei A_i die Variablen sind die als Argumente der Funktion dienen.

Beispiel:

$I_{\vec{b}} : A_1 \mapsto 1, A_2 \mapsto 0, A_3 \mapsto 1, A_4 \mapsto 1 \quad \Longrightarrow \quad \mathrm{val}_{I_b}\left(D_{\vec{b}}\right) = 0$

$\vec{b} = (1,0,1,1) \quad \Longrightarrow \quad D_{\vec{b}} = \neg A_1 \vee A_2 \vee \neg A_3 \vee \neg A_4$

Für Wahrheitsbelegung I_b hat $D_{\vec{b}}$ den Wert 0 und sonst 1.

Die Konjunkte werden disjunktiv verbunden.

$$\mathrm{KNF}_f = \bigwedge \left\{ D_{\vec{b}} \mid f(\vec{b}) = 0, \vec{b} \in \mathbb{B}^n \right\}$$

Dadurch repräsentiert die DNF die Funktion f.

$$\forall \vec{b} \in \mathbb{B}^n : \mathrm{val}_{\vec{b}}\left(\mathrm{KNF}_f\right) = f(\vec{b})$$

Laufzeiten

von einer beliebigen Formel zu DNF / KNF kommen:

Beide exponentielle obere Schranke für Laufzeit

1. Semantische Methode (Siehe oben) - übersichtlicher, einfacher, aber *immer* exponentielle Laufzeit $2^{Anz.Variablen}$ weil für jedes element von $\vec{b}$ ein Konj. / Disjunkt gebildet werden muss.
2. Algebraische Methode (Algebraisch umformen / kürzen bis man erwünschte Form erreicht) - durchschnittlich effizienter, aber obere Schranke wie semantische Methode

Normalformen

Formeln unter gewissen Einschränkungen. - An der Semantik ändert sich nichts.

Normalformen sind meistens nicht eindeutig.

Beispiele

- Disjunktive Normalform DNF
- Konjunktive Normalform KNF
- Negations Normalform NNF

  (DNF, KNF sind auch gültige NNF)

  kleinstmögliche Menge für die gilt:

  1. $(F \vee G)$ und $(F \wedge G)$ sind in der Menge wenn $\{F, G\} \in$ NNF.
  2. Literale (= negierte / nicht negierte Variablen) sowie $\top$, $\bot$ sind NNF

Andere Einschränkungen auch möglich für Normalformen, zb nur Nand-Operator $\uparrow$ erlaubt. (Operator muss funktional vollständig sein)

SAT-Problem

P	Problemklasse, die sich polynomiell lösen lässt.
NP	Problemklasse, die sich polynomiell verifizieren lässt.
NP-vollständig	Problemklasse, auf die sich alle NP-Probleme reduzieren lassen.

Erfüllbarkeitsproblem / Satisfiability / SAT (NP-vollständig)

Wir haben eine Formel F und wollen wissen ob sie erfüllbar ist.

Wir wollen wissen ob $\exists\, I \in \mathcal{I} : \mathsf{val}_I(F) = 1$

Aussagenlogische Frage übersetzt zu (Un)Erfüllbarkeitsproblem:

$$\begin{aligned} G \text{ gültig} &\iff \neg G \text{ unerfüllbar} \\ G \text{ widerlegbar} &\iff \neg G \text{ erfüllbar} \\ G = H \text{ gültig} &\iff G \not\equiv H \text{ unerfüllbar} \\ F_1, \ldots, F_n \models G &\iff F_1 \wedge \cdots \wedge F_n \wedge \neg G \text{ unerfüllbar} \end{aligned}$$

SAT lösen:

1. Wahrheitstafel (ineffizient)
2. DNF (ineffizient, Probleme meistens fast in KNF Form)
3. SAT-Solver

SAT-Solver

Mögliche Resultate:

a) "SAT" $\exists\, I \in \mathcal{I} : \mathsf{val}_I(F) = 1$ (Formel ist erfüllbar)

b) "UNSAT" $\not\exists\, I \in \mathcal{I} : \mathsf{val}_I(F) = 1$ (Formel ist unerfüllbar)

SAT Solver Beispiel

Dualität

Dualität ≠ Negation

Duale Funktionen

Zwei n-stellige Funktionen f und g heißen dual zueinander, wenn gilt:

$$\text{not}\, f(x_1, \ldots, x_n) = g(\text{not}\, x_1, \ldots, \text{not}\, x_n)$$

Beispiel: `and` und `or` bei gespiegelten Wahrheitsbelegungen

DeMorgan Gesetz: not(x and y) = (not x) or (not y)

x	y	x and y	x or y
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

x	y	x or y	x and y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Duale Operatoren

Operatoren (semantisch) sind dual wenn ihre Funktionen dual sind.

Duale Formeln

Formel $F[A_1, \ldots, A_n]$ und $G[A_1, \ldots, A_n]$ sind dual zu einander wenn gilt:

$$\neg F[A_1, \ldots, A_n] = G[\neg A_1, \ldots, \neg A_n]$$

Quasi wenn überall in der Formel die Operatoren mit ihrem Dual ausgetauscht werden - aber eine Formel kann zu mehreren anderen dual sein.

Beispiel:

$\neg(((A \wedge B) \not\equiv \neg(B \uparrow C)) \not\supset ((A \wedge \top) \vee B))$ ist dual zu

$\neg(((A \vee B) \equiv \neg(B \downarrow C)) \subset ((A \vee \bot) \wedge B))$

Aussagenlogische Modellierung

deklarativ-statisch, nicht prozedural-dynamisch

+ deklarativ (man sagt welchen Zustand man sich unter welchen Bedingungen wünscht)
+ modular

− Erraten von Parametern oft notwendig (siehe Simpsons bsp)
− Große Zahl an Variablen und Formeln
− Frame Problem: Bei jeder Aktion muss auch definiert werden, was sich nicht ändert.
− unintuitiv - menschen denken bei Modellieren nicht an statischen Bedingungen

Beispiele

Beispiele

Beispiel: Dr. House Diagnose

F_1 : Max wird mit hohem Fieber und ausgeprägten Gliederschmerzen in das Spital eingeliefert. Dr. House diskutiert die Diagnose mit einer Kollegin.

- F_2 : House: „Wenn der Patient Fieber hat, handelt es sich um Grippe oder Erkältung.“
- F_3 : Cameron: „Wenn er keine starken Gliederschmerzen hat, dann hat er auch keine Grippe.“
- F_4 : House: „Jedenfalls weisen hohes Fieber und starke Gliederschmerzen immer auf Grippe hin.“
- F_5 : Cameron: “Er hat sicher nicht beide Krankheiten gleichzeitig.“

Variablen erstellen:

F Patient hat (hohes) Fieber

S Patient hat Gliederschmerzen

G Patient hat eine Grippe

E Patient hat eine Erkältung

Gültige Formeln erstellen:

(Wir gehen davon aus, dass die Ärzte Implikation und nicht Äquivalenz meinen)

$F_1 := F \wedge S$
$F_2 := F \supset (G \vee E)$
$F_3 := \neg S \supset \neg G$
$F_4 := (F \wedge S) \supset G$
$F_5 := \neg(G \wedge E)$

Wir suchen eine Interpretation / Wahrheitsbelegung bei der alle Formeln wahr sind:

1. Mittels Wahrheitstafel

F	S	G	E	F_1	$F \supset (G \vee E)$	$\neg S \supset \neg G$	$(F \wedge S) \supset G$	$\neg(G \wedge E)$
1	1	0	0	1	0	1	0	1
1	1	0	1	1	1	1	0	1
1	1	1	0	1	1	1	1	1
1	1	1	1	1	1	1	1	0

2. (Algebraische) Umwandlung in DNF und Vereinfachung

   Liefert dann $F \wedge S \wedge G \wedge \neg E$

3. (Algebraische) Umwandlung in KNF und SAT-Solver nützen

$$\underbrace{F \wedge S}_{F_1} \wedge \underbrace{(\neg F \vee G \vee E)}_{F_2} \wedge \underbrace{(S \vee \neg G)}_{F_3} \wedge \underbrace{(\neg F \vee \neg S \vee G)}_{F_4} \wedge \underbrace{(\neg G \vee \neg E)}_{F_5}$$

   Um zu überprüfen ob es andere Lösungen gibt, erstellen wir zusätzlich noch F_6

   $F_6 = \neg(F \wedge S \wedge G \wedge \neg E) = \neg F \vee \neg S \vee \neg G \vee E$

   SAT-Solver liefert dann das Ergebnis „unerfüllbar“ - es gibt keine weiteren Lösungen.

Unsere Formel ist erfüllt wenn $I(G) = 1,\ I(E) = 0$

Diagnose: Grippe

Beispiel: Gone Maggie gone

Problem: Homer will mit einem Boot, seiner Tochter Maggie, dem Hund Knecht Ruprecht zur anderen Fluss-Seite.

Das Boot kann nur gleichzeitig 2 Objekte tragen.

Nicht von Homer alleine gelassen werden dürfen:

- Maggie und Gift
- Maggie und Hund (Knecht Ruprecht)

Variablen erstellen:

Es gibt *diese* und *die andere* Fluss-Seite.

Wenn eines dieser Variablen true ist dann:

$$\left.\begin{array}{ll} M_i & \dots \text{ Maggie} \\ K_i & \dots \text{ Knecht Ruprecht} \\ G_i & \dots \text{ Gift} \\ H_i & \dots \text{ Homer} \end{array}\right\} \text{ zum Zeitpunkt } i \text{ auf der anderen Fluss-Seite}$$

Wenn alle hier sind (alle false):

$$\text{AlleHier}(i) := \neg M_i \wedge \neg K_i \wedge \neg G_i \wedge \neg H_i$$

Wenn alle dort sind (alle true):

$$\text{AlleDort}(i) := M_i \wedge K_i \wedge G_i \wedge H_i$$

Homer muss immer bei Maggie sein, wenn sie bei Hund oder Gift ist

$$\text{Sicher}(i) := ((M_i \equiv K_i) \vee (M_i \equiv G_i)) \supset (M_i \equiv H_i)$$

Wenn eines dieser Variablen true ist, dann:

$$\left.\begin{array}{ll} MH_i & \dots \text{ Maggie} \\ KH_i & \dots \text{ Knecht Ruprecht} \\ GH_i & \dots \text{ Gift} \\ HH_i & \dots \text{ Homer} \end{array}\right\} \text{ fährt Homer zwischen } [i-1; i] \text{ zur anderen Seite}$$

Pro Zeitpunkt *nur eine* Überfahrt erlaubt (nur eine Var. darf true sein):

$$\begin{array}{l} \text{Überfahrt}(i) := \\ (\text{MH}_i \wedge \neg\text{KH}_i \wedge \neg\text{GH}_i \wedge \neg\text{HH}_i) \vee (\neg\text{MH}_i \wedge \text{KH}_i \wedge \neg\text{GH}_i \wedge \neg\text{HH}_i) \vee \\ (\neg\text{MH}_i \wedge \neg\text{KH}_i \wedge \text{GH}_i \wedge \neg\text{HH}_i) \vee (\neg\text{MH}_i \wedge \neg\text{KH}_i \wedge \neg\text{GH}_i \wedge \text{HH}_i) \end{array}$$

Erlaubte Unterschiede zwischen Zeitpunkten i und $i-1$:

(Frame Problem)

$$\begin{array}{l} \text{DefÜberfahrt}(i) := \\ \quad (MH_i \supset ((M_{i-1} \not\equiv M_i) \wedge (K_{i-1} \equiv K_i) \wedge (G_{i-1} \equiv G_i) \wedge (H_{i-1} \not\equiv H_i) \wedge (H_i \equiv M_i))) \\ \wedge (KH_i \supset ((M_{i-1} \equiv M_i) \wedge (K_{i-1} \not\equiv K_i) \wedge (G_{i-1} \equiv G_i) \wedge (H_{i-1} \not\equiv H_i) \wedge (H_i \equiv K_i))) \\ \wedge (GH_i \supset ((M_{i-1} \equiv M_i) \wedge (K_{i-1} \equiv K_i) \wedge (G_{i-1} \not\equiv G_i) \wedge (H_{i-1} \not\equiv H_i) \wedge (H_i \equiv G_i))) \\ \wedge (HH_i \supset ((M_{i-1} \equiv M_i) \wedge (K_{i-1} \equiv K_i) \wedge (G_{i-1} \equiv G_i) \wedge (H_{i-1} \neq H_i)) \end{array}$$

Daraus folgt die Gesamtformel:

Nach n Überfahrten sollen alle auf der anderen Seite sein.

$$\begin{array}{l} \text{Gesamtformel}\ (n) := \text{AlleHier}\ (0) \wedge\ \text{AlleDort}\ (n) \wedge \bigwedge_{i=0}^{n} \text{Sicher}\ (i) \\ \qquad\qquad \wedge \bigwedge_{i=1}^{n} \text{Überfahrt}\ (i) \wedge \bigwedge_{i=1}^{n} \text{DefÜberfahrt}\ (i) \end{array}$$

Vorgang:

1. Die benötigte Zahl n der Überfahrten erraten oder iterativ erhöhen
2. SAT Solver nützen

Lösung: $n = 7$

$(MH_1) = I\,(HH_2) = I\,(GH_3) = I\,(MH_4) = I\,(KH_5) = I\,(HH_6) = I\,(MH_7) = 1$

Prädikatenlogik

Terme

Zur Bildung von Ausdrücken die ausgewertet werden können.

Alles lässt sich als Term auffassen.

Syntax

Menge $\mathcal{T}(\mathcal{F}, \mathcal{V})$ wobei

$\mathcal{F}$ Funktions-Symbole

$\mathcal{V}$ Individuen-Variablen-Symbole

ist kleinste Menge für die gilt:

1. $\mathcal{V} \subseteq \mathcal{T}$
2. Wenn $f/0 \in \mathcal{F}$ dann $f \in \mathcal{T}$ (Konstanten)
3. Wenn $f/n \in \mathcal{F}$ und $t_1, \ldots, t_n \in \mathcal{T}$ dann $f\,(t_1, \ldots, t_n) \in \mathcal{T}$

▼ Beispiel: Menge Aussagenlogischer Formeln über Σ

$\mathcal{V} = \{A, B, C, \ldots\}$

$\mathcal{F} = \{\top/0, \bot/0, \neg/1, \wedge/2, \vee/2, \ldots\}$

Mit Infix-Notation:

$(A \wedge B) \supset C \qquad \supset (\wedge(A, B), C)$
$A \wedge (B \supset C) \qquad \wedge (A, \supset (B, C))$

▼ Beispiel: Reguläre Ausdrücke über dem Alphabet Σ

$\mathcal{V} = \{\}$

$\mathcal{F} = \{\varepsilon/0, \emptyset/0, +/2, \cdot/2, ^*/1\} \cup \{s/0 \mid s \in \Sigma\}$

Semantik

In Domäne / Wertebereich / Universum $\mathcal{U}$:

n ... Anzahl der Stellen / benötigten Argumente

$\mathcal{V}$ Wert abhängig von Werte-Belegung

$f/0$ Konstanten, vordefinierte Werte fest eingebaut in Term-Semantik

f/n Freie Symbole, Wert kann interpretiert werden

Interpretation I (für Funktionen f)

Ordnet jeder Funktion f einen Term zu.

$I(f) \in \mathcal{U}$ für $f/0 \in \mathcal{F}$

$I(f) : \mathcal{U}^n \mapsto \mathcal{U}$ für $f/n \in \mathcal{F}, n > 0$

Wertebelegung σ (für Variablen $\mathcal{V}$) unter einer Interpretation I

$\sigma : \mathcal{V} \mapsto \mathcal{U}$

$\text{val}_{I,\sigma}(v) = \sigma(v)$ für $v \in \mathcal{V}$

$\text{val}_{I,\sigma}(f) = I(f)$ für $f/0 \in \mathcal{F}$

$\text{val}_{I,\sigma}(f(t_1,\ldots,t_n)) = I(f)(\text{val}_{I,\sigma}(t_1),\ldots,\text{val}_{l,\sigma}(t_n))$ für $f/n \in \mathcal{F}, n > 0$

▼ Beispiel: Artihmetische Ausdrücke

Ausdruck: `(x+1)*(x+1+1)`

$\mathcal{U} = \mathbb{Z}, \quad I_1(1) = 1, \quad I_1(+) = +, \quad I_1(*) = \cdot$

$\text{val}_{1_1,\sigma}(\texttt{(x+1)*(x+1+1)}) = (0+1)\cdot(0+1+1) = 2$

▼ Beispiel: Aussagenlogik

Ausdruck: `(x+1)*(x+1+1)`

$\mathcal{U} = \mathbb{B}, \quad l_2(1) = 1, \quad l_2(+) = \text{ or}, \quad l_2(*) = \text{ and}$

$\text{val}_{I_2,\sigma}(\texttt{(x+1)*(x+1+1)}) = \text{ and }(\text{ or }(0,1),\text{ or }(0,\text{ or }(1,1))) = 1$

Prädikatenlogik

$\mathcal{F}$ Funktions-Symbole

I ordnet jedem Funktionssymbol $f \in F$ einen Term / eine Funktion die ein bestimmtes Element $\in \mathcal{U}$ retourniert zu.

$f/n \in \mathcal{F}$

$f/0$ Konstanten (Konstante Funktion)

$\mathcal{P}$ Prädikaten-Symbole / Atomformeln

I ordnet jedem Prädikatsymbol $P \in \mathcal{P}$ einen Wert $\in \{0,1\}$ zu.

$P/n \in \mathcal{P}$

$P/1$ Typ / Klasse

$P/0$ Aussagenvariablen, Elementaraussagen

$\mathcal{V}$ Individuen-Variablen-Symbole

Stehen für ein *beliebiges* Element $\in \mathcal{U}$, werden im Zusammenhang mit Quantoren eingesetzt.

$\mathcal{V} = \{x, y, z, x_0, x_1, \ldots\}$

Beispiel: $Mensch(mutter(sokrates)), \quad Sterblich(x)$

$\mathcal{F}$: mutter/1, sokrates/0

$\mathcal{P}$: Mensch/1, Sterblich/1

$\mathcal{V}$: x

Syntax

Die Regeln (1,3,4) wären die gesamte Aussagenlogik.

Die Menge $\mathcal{PF}$ der prädikatenlogischen Formeln über Signatur $(\mathcal{F}, \mathcal{P})$ ist die kleinste Menge für die gilt:

1. $P \in \mathcal{PF}$ wenn $P/0 \in \mathcal{P}$
2. $P(t_1,\ldots,t_n) \in \mathcal{PF}$ wenn $P/n \in \mathcal{P}$ und $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{F},\mathcal{V})$
3. $\top, \bot \in \mathcal{PF}$

4. $\neg F \in \mathcal{PF}$ wenn $F \in \mathcal{PF}$
5. $(F * G) \in \mathcal{PF}$ wenn $F, G \in \mathcal{PF}, \quad * \in \{\wedge, \uparrow, \vee, \downarrow, \equiv, \not\equiv, \supset, \subset\}$
6. $\forall x F \in \mathcal{PF}$ wenn $x \in \mathcal{V}, \quad F \in \mathcal{PF}$
7. $\exists x F \in \mathcal{PF}$ wenn $x \in \mathcal{V}, \quad F \in \mathcal{PF}$

Semantik

(Prädikatenlogische) Interpretation I über $\mathcal{U}$

$\mathcal{F}$ Funktions-Symbole

$I(f) \in \mathcal{U}$ für $f/0 \in \mathcal{F}$

$I(f) : \mathcal{U}^n \mapsto \mathcal{U}$ für $f/n \in \mathcal{F}, n > 0$

$\mathcal{P}$ Prädikaten-Symbole

$I(P) \in \{0, 1\}$ für $P/0 \in \mathcal{P}$ (Aussagenvariablen)

$I(P) : \mathcal{U}^n \mapsto \{0, 1\}$ für $P/n \in \mathcal{P}, \quad n > 0$ (Als Funktion - Sichtweise 1)

$I(P) \subseteq \mathcal{U}^n$ für $P/n \in \mathcal{P}, \quad n > 0$ (Als Menge - Sichtweise 2)

$\mathcal{P}$ stehen für Relationen, Elementaraussagen.

▼ Sichtweisen auf das Prädikat-Symbol

Beispiel: Einstellig

Einstellige Prädikate geben Typ / Klasse.

$\mathcal{U} = \{ \text{Andrea}, \text{Anna}, \text{Hans}, \text{Maria}, \text{Tom} \}$

Prädikate: $\text{Mann}/1, \ \text{Frau}/1 \in \mathcal{P}$

Sichtweise 1: Menge

$I(\text{Mann}) = \{ \text{Andrea}, \text{Hans}, \text{Tom} \}$

$I(\text{ Frau }) = \{ \text{Andrea}, \ \text{Anna}, \text{Maria} \}$

Sichtweise 2: Funktion

x	$I(\text{Mann})(x)$	$I(\text{ Frau })(x)$
Andrea	1	1
Anna	0	1
Hans	1	0
Maria	0	1
Tom	1	0

Beispiel: Mehrstellig

Mehrstellige Prädikate geben Relation / Beziehung

$\mathcal{U} = \mathbb{N}$

Prädikat: $</2 \in \mathcal{P}$

Sichtweise 1: Menge

$I(<) = \{(m, n) \mid m < n\} = \{(0,1), (0,2), (1,2), (0,3), (1,3), \ldots\}$

Sichtweise 2: Funktion

$$I(<)(m, n) = \begin{cases} 1 & \text{falls } m < n \\ 0 & \text{sonst} \end{cases}$$

Vorteil der rekursiven Definition: Keine unendliche Menge

Beispiel: Mehrstellig

$\mathcal{U} = \mathbb{N}$

$\text{Plus}/3 \in \mathcal{P}$

Sichtweise 1: Menge

$$I(\text{Plus}) = \{(k,l,m) \mid k+l=m\}$$
$$= \{(0,0,0),(1,0,1),(0,1,1),(2,0,2)(1,1,2)(0,2,2),\ldots\}$$

Sichtweise 2: Funktion

$$I(\text{Plus})(k,I,m) = \begin{cases} 1 & \text{falls } k+I=m \\ 0 & \text{sonst} \end{cases}$$

Wertebelegung für Variablensymbole σ

Wertet Prädikaten-Formeln und Prädikatensymbole aus.

$\text{val}_{I,\sigma}(\top) = 1, \quad \text{val}_{I,\sigma}(\bot) = 0$

$\text{val}_{I,\sigma}(P) = I(P)$ für $P/0 \in \mathcal{P}$

$\text{val}_{I,\sigma}(P(t_1,\ldots,t_n)) = I(P)(\text{val}_{t,\sigma}(t_1),\ldots,\text{val}_{I,\sigma}(t_n))$ für $P/n \in \mathcal{P}$

$\text{val}_{I,\sigma}(\neg F) = \text{not}\,\text{val}_{l,\sigma}(F)$ für $F \in \mathcal{PF}$

$\text{val}_{I,\sigma}((F * G)) = \text{val}_{I,\sigma}(F) \circledast \text{val}_{I,\sigma}(G)$ (wobei $\circledast$ die Interpretation für $*$ ist)

$$\text{val}_{I,\sigma}(\forall x F) = \begin{cases} 1 & \text{falls val}_{l,\sigma'}(F) = 1 \text{ für alle } \sigma' \overset{x}{\sim} \sigma \\ 0 & \text{sonst} \end{cases}$$

$$\text{val}_{l,\sigma}(\exists x F) = \begin{cases} 1 & \text{falls val}_{,\sigma'}(F) = 1 \text{ für mind. ein } \sigma' \overset{x}{\sim} \sigma \\ 0 & \text{sonst} \end{cases}$$

$\sigma \overset{x}{\sim} \sigma' \ldots \sigma(v) = \sigma'(v)$ für alle $v \in \mathcal{V}$ mit $v \neq x$

Alle Variablenbelegungen für die der Ausdruck gleich ist aber aber x unterschiedlich ist

Gültigkeit beweisen

Beweisen sehr schwer und braucht eigenes Kalkül. Aber es geht bei einfachen Beispielen.

Beispiel: $\forall x \exists y : x = y - 1$

Idee: So würde man es intuitiv lösen

$y = x + 1$

$x = (x+1) - 1$

$x = x$

Aber man muss es formalisieren:

▼ Beispiel 1: $\forall x \exists y : x = y - 1$

$\mathcal{U} = \mathbb{N}$

$I(s)(n) := n - 1$

$I(P)(m,n) := (m = n)$

$\forall x \exists y P(x, s(y))$

Wenn es eine Variablenbelegung σ gibt mit der dieser Ausdruck wahr ist

$\text{val}_{l,\sigma}(\forall x \exists y P(x,s(y))) = 1$

$\iff \text{val}_{l,\sigma'}(\exists y P(x,s(y))) = 1$ für alle x, daher $\sigma' \overset{x}{\sim} \sigma$

$\iff \text{val}_{I,\sigma''}(P(x,s(y))) = 1$ für mind. ein y und alle x, daher zusätzlich $\sigma'' \overset{y}{\sim} \sigma'$

$\iff I(P)(\text{val}_{I,\sigma''}(x), \text{val}_{I,\sigma''}(s(y))) = 1$

$\iff \sigma''(x) = \sigma''(y) - 1$

Wir suchen also jene Variablenbelegung σ''

für die $\sigma''(y) = \sigma''(x) + 1$ für alle $\sigma''(x) = \sigma'(x) \in \mathbb{N}$ gilt.

Wenn

$\sigma''(x) = \sigma''(y) - 1$

$\sigma''(y) = \sigma''(x) + 1$

Dann muss

$\sigma''(x) = (\sigma''(x) + 1) - 1$

Und das trifft immer zu.

▼ Beispiel 2: $\forall x \exists y : x = y + 1$

$\mathcal{U} = \mathbb{N}$

$I(s)(n) := n - 1$

$I(P)(m, n) := (m = n)$

$\forall x \exists y P(x, s(y))$

Wenn es eine Variablenbelegung σ gibt mit der dieser Ausdruck wahr ist

$\text{val}_{I,\sigma}(\forall x \exists y P(x, s(y))) = 1$

$\iff \text{val}_{I,\sigma'}(\exists y P(x, s(y))) = 1$ für alle x, daher $\sigma' \overset{x}{\sim} \sigma$

$\iff \text{val}_{I,\sigma''}(P(x, s(y))) = 1$ für mind. ein y und alle x, daher zusätzlich $\sigma'' \overset{y}{\sim} \sigma'$

$\iff I(P)\,(\text{val}_{I,\sigma''}(x), \text{val}_{I,\sigma''}(s(y))) = 1$

$\iff \sigma''(x) = \sigma''(y) + 1$

Wir suchen also jene Variablenbelegung σ''

für die $\sigma''(y) = \sigma''(x) - 1$ für alle $\sigma''(x) = \sigma'(x) \in \mathbb{N}$ gilt.

Wenn

$\sigma''(x) = \sigma''(y) + 1$

$\sigma''(y) = \sigma''(x) - 1$

Dann muss

$\sigma''(x) = (\sigma''(x) - 1) + 1$

Diese Gleichung wäre aber nur in $\mathbb{Z}$ gültig, nicht in $\mathbb{N}$.

Eine Formel *F* heißt

(allgemein) gültig wenn für alle I, σ immer $\text{val}_{I,\sigma}(F) = 1$

Immer wahr - dann nennt man es "Tautologie"

erfüllbar wenn für mindestens eine I, σ gilt $\text{val}_{I,\sigma}(F) = 1$

Mind. bei 1 Fall wahr - dann nennt man es "Modell von *F*"

widerlegbar wenn für mindestens eine I, σ gilt $\text{val}_{I,\sigma}(F) = 0$

Mind. bei 1 Fall falsch - dann nennt man es "Gegenbeispiel" / "Gegenmodell"

unerfüllbar wenn für alle I, σ immer $\text{val}_{I,\sigma}(F) = 0$

Immer falsch

Gültigkeitsproblem

Gegeben: prädikatenlogische Formel F

Frage: Ist F gültig für alle I und σ? → Unentscheidbar!

Beweiser terminieren Fall oft nicht wenn Formel ungültig.

Äquivalenz

Semantische Äquivalenz

$F = G$ wenn für alle I, σ gilt $\text{val}_{I,\sigma}(F) = \text{val}_{I,\sigma}(G)$

Dualität von $\forall$ und $\exists$

$\neg \forall x F = \exists x \neg F$
$\neg \exists x F = \forall x \neg F$

Distributivität

$\forall$ ist wie ein großes $\wedge$ - und $\exists$ ist wie ein großes $\vee$.

$\forall x (F \wedge G) = \forall x F \wedge \forall x G$
$\exists x (F \vee G) = \exists x F \vee \exists x G$

Vertauschung von $\forall$ und $\exists$

Nur wenn sie hintereinander vorkommen:

$\forall x \forall y F = \forall y \forall x F$

$\exists x \exists y F = \exists y \exists x F$

Ansonsten (alternierende Quantoren) ändert sich die Bedeutung:

Angenommen $\text{Mutter}(x, y)$ bedeutet x ist Mutter von y:

Fall 1:

$\forall x \exists y \text{ Mutter } (y, x)$

"Jeder hat eine Mutter."

y hängt vom gewählten x ab.

Fall 2:

$\exists y \forall x \text{ Mutter } (y, x)$

"Jemand ist die Mutter von allen."

y ist unabhängig vom gewählten x.

Prädikatenlogische-Modellierung

Beispiel

Beispiel

Beispiel: Auf dem Spielplatz

$\mathcal{U} = \{\ \text{Albrecht}, \text{Bogdan}, \text{Erich}, \text{Frieda}, \text{Kathrin}, \text{Nina}, \text{Tamara}\ \}$

$\mathcal{P} = \{\ \text{Vater}\ /1,\ \text{Kind}\ /1,\ \text{SpieltMit}\ /2\}$

$\mathcal{F} = \{\ \text{albrecht}\ /0,\ \text{frieda}\ /0\}$ (eingeführte Konstanten - klein geschrieben)

$I(\ \text{albrecht}\ ) = \ \text{Albrecht}$
$I(\ \text{frieda}\ ) = \ \text{Frieda}$

$I(\ \text{Vater}\ ) = \{\ \text{Bogdan}, \text{Erich}\ \}$

$I(\ \text{Kind}\ ) = \{\ \text{Albrecht}, \text{Frieda}, \text{Nina}\ \}$

$I(\ \text{SpieltMit}\ ) = \{\ (\text{Albrecht}, \text{Frieda}), (\text{Bogdan}, \text{Albrecht}),\ (\text{Erich}, \text{Albrecht}), (\text{Erich}, \text{Frieda}),$
$(\text{Frieda}, \text{Nina}), (\text{Frieda}, \text{Kathrin}), (\text{Frieda}, \text{Albrecht}), (\text{Nina}, \text{Frieda}),$
$(\text{Tamara}, \text{Erich}), (\text{Tamara}, \text{Bogdan}), (\text{Tamara}, \text{Albrecht})\ \}$

Behauptung: $\exists x(\text{Kind}(x) \wedge \forall y(\text{Kind}(y) \supset \text{SpieltMit}(x, y)))$

„Es gibt (mindestens) ein Kind, das mit allen Kindern spielt."

Falsch in *I*, da Albrecht nicht mit Nina, Frieda nicht mit Frieda und Nina nicht mit Albrecht spielt.

Behauptung: $\forall x(\ \text{SpieltMit}\ (x,\ \text{frieda}\ ) \not\equiv\ \text{SpieltMit}\ (x,\ \text{albrecht}\ ))$

„Alle spielen entweder mit Frieda oder Albrecht (aber nicht mit beiden)."

Falsch in *I*, da Erich sowohl mit Frieda als auch mit Albrecht spielt.

Formale Sprachen

Formale Sprachen

$\sum$ Alphabet, nicht leere endliche Menge an Symbolen

ε Leerwort (Leer-String (`""`), nicht Leerzeichen (`(" ")`) oder leere Menge (`{}`))

w Wort über Σ, endliche Verkettung von Symbolen

Verkettung: $w \cdot w' = ww'$

Wort Mengen

$\sum^+$ alle möglichen Wörter ohne ε

$\sum^*$ alle möglichen Wörter mit ε

$\sum^\omega$ alle möglichen unendlichen Wörter = "ω-Wörter"

$L \subseteq \Sigma^*$ - Formale Sprache über Σ^*

Sprache ist eine Teilmenge aller Wörter des Alphabets potenziert mit Kleene Stern

- auch $\{\}, \{\varepsilon\}, \sum^*$
- die Menge aller deutschen Sätze (Alphabet: Buchstaben, Satzzeichen, Leerzeichen)
- die Menge aller Java-Programme (Alphabet: ASCII-Zeichen inkl. Leerzeichen)

Mehrere Sprachen können dasselbe Alphabet nutzen.

$2^{\sum^*}$ - Menge aller formalen Sprachen

Potenzmenge = Menge die *alle* möglichen Teilemengen einer Menge enthält

Operationen auf formale Sprachen

$L, L' \subseteq \Sigma^*$

Vereinigung $L \cup L' = \{w \mid w \in L \vee w \in L'\}$

Verkettung $L \cdot L' = \{w \cdot w' \mid w \in L, w' \in L'\}$ (nicht kommutativ)

Potenzierung $L^0 = \{\varepsilon\}$

$L^{n+1} = L \cdot L^n$ (n≥0)

$L^+ = \bigcup_{n \geq 1} L^n$ (Alle Wörter der Länge n für n≥1)

$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^+ = \{\varepsilon\} \cup L^+$ "Kleene Stern"

▼ Beispiel: Verkettung

$L_1 = \{a, b\} \qquad L_2 = \{b, c, d\}$

$L_1 \cdot L_2 = \{ab, ac, ad, bb, bc, bd\}$

$L_2 \cdot L_1 = \{ba, bb, ca, cb, da, db\}$

▼ Beispiel: Potenzierung

$L = \{\mathrm{a}, 42\}$

$L^0 = \{\varepsilon\}$

$$L^+ = \bigcup_{n \geq 1} L^n = L^1 \cup L^2 \cup L^3 \cup \cdots$$
$$= \{a, 42,\ \mathrm{aa}, a42, 42a, 4242,\ \mathrm{aaa}, aa42, a42a,\ \mathrm{a4242}\ , 42aa, \ldots\}$$

$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \cdots$$
$$= \{\varepsilon, a, 42,\ \mathrm{aa}, a42, 42\mathrm{a}, 4242, \mathrm{aaa}, \mathrm{aa42}, \mathrm{a42a}, \mathrm{a4242}, 42\mathrm{aa}, \ldots\}$$

Einselement $\{\epsilon\}$

$$\{a, b\} \cdot \{\varepsilon\} = \{a \cdot \varepsilon, b \cdot \varepsilon\} = \{a, b\}$$

Nullelement $\{\}$

$$\{a, b\} \cdot \{\} = \{\}$$

Reguläre Sprachen

Regex Übung: https://alf.nu/RegexGolf

EBNF converter: https://www.bottlecaps.de/rr/ui

Reguläre Sprachen

$\mathcal{L}_{\text{reg}}(\Sigma)$ Menge der Sprachen $L \subseteq \Sigma^*$ die regulär sind.

Kleinste Menge für die gilt:

1. $\{\}, \{\varepsilon\}, \{s\}$ für $s \in \Sigma$ ist eine reguläre Sprache
2. $L, L' \in \mathcal{L}_{\text{reg}}(\Sigma)$ dann sind auch $L \cup L', L \cdot L', L^*$ reguläre Sprachen

▼ Beispiel: Reele Numerale

$$\Sigma = \{0, \ldots, 9, ., E, +, -\}$$
$$\begin{aligned} \text{real} &= \text{digit} \cdot \text{digit}^* \cdot \{.\} \cdot \text{digit}^* \cdot (\{\varepsilon\} \cup \text{scale}) \\ \text{scale} &= \{\text{E}\} \cdot \{+, -, \varepsilon\} \cdot \text{digit} \cdot \text{digit}^* \\ \text{digit} &= \{0, \ldots, 9\} = \{0\} \cup \cdots \cup \{9\} \end{aligned}$$

Wobei $\varepsilon,\ \text{real}, \text{scale}, \text{digit}$ "Meta-Symbole" / Abkürzungen sind, keine Symbole.

Man könnte statt $\text{digit} \cdot \text{digit}^*$ auch digit^+ schreiben, aber nicht in unserer Definition.

Notationen für Reguläre Ausdrücke

Es gibt verschiedene Notationen für reguläre Ausdrücke, aber die Sprachen unter ihnen können äquivalent sein.

Äquivalenz von regulären Ausdrücken

Die reguläre Sprache $\mathcal{L}(r)$ von einem regulärem Ausdruck r definiert durch:

$r = r'$ wenn $\mathcal{L}(r) = \mathcal{L}(r')$

Also die Menge von allen Wörtern die sich mit dem regulären Ausdruck bilden lassen.

1) Algebraische Notation

Kleinste Menge für die gilt:

1. $\emptyset, \varepsilon, s \in \Sigma$ sind reguläre Ausdrücke
2. Wenn r, r' reguläre Ausdrücke sind, dann auch $(r + r'), (rr'), r^*$

▼ Beziehungsweise

$$\mathcal{L}(\emptyset) = \{\}$$
$$\mathcal{L}(\varepsilon) = \{\varepsilon\}$$
$$\mathcal{L}(s) = \{s\} \quad \text{für } s \in \Sigma$$
$$\mathcal{L}(r + r') = \mathcal{L}(r) \cup \mathcal{L}(r')$$
$$\mathcal{L}(rr') = \mathcal{L}(r) \cdot \mathcal{L}(r')$$
$$\mathcal{L}(r^*) = (\mathcal{L}(r))^*$$

▼ Stärke der Bindung

Wenn man Klammern auslässt:

1. Kleene Stern

2. Verkettung
3. Vereinigung

▼ Beispiel: Reele Numerale

$R = DD^* \cdot D^*(\varepsilon + S)$
$S = \mathrm{E}(+ + - + \varepsilon)DD^*$
$D = 0 + 1 + \cdots + 9$

2) Erweiterte Backus-Naur-Form EBNF Notation

für Programmiersprachen die reguläre Ausdrücke zulassen.

`AB`	$A \cdot B$
`A|B`	$A \cup B$
`[A]`	$\{\varepsilon\} \cup A$
`{A}`	A^*
`(A)`	A
`"s"`	$\{s\}$ wobei $s \in \Sigma$

▼ Beispiel: Reele Numerale

```
real = digit {digit} "." {digit} [scale]
scale = "E" ["+" | "-"] digit {digit}
digit = "0" | "1" | "2" | ... | "9"
```

3) Posix - Extended Regular Expressions ERE Notation

▼ Beispiel: Reele Numerale

Keine Modularisierung, gesamter Ausdruck in einem.

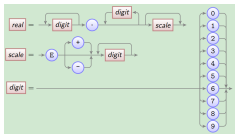
```
real = ^[0-9]+\.[0-9]*(E[+-]?[0-9]+)?$
scale = E[+-]?[0-9]+
digit = [0-9]
```

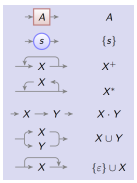
regexp	trifft zu auf	Reg. Sprache
$\backslash s$	Zeichen s	$\{s\}$
s	s, falls kein Sonderzeichen	$\{s\}$
.	alle Zeichen	Σ
^	Zeilenanfang	
$	Zeilenende	
$[s_1 \cdots s_n]$	eines der Zeichen s_i	$\{s_1, \ldots, s_n\}$
$[\hat{}s_1 \cdots s_n]$	alle Zeichen außer $s_1, \ldots, s_n$	$\Sigma - \{s_1, \ldots, s_n\}$
(X)	X	X
XY	X gefolgt von Y	$X \cdot Y$
$X \mid Y$	X oder Y	$X \cup Y$
$X*$	≥ 0 Mal X	X^*
$X+$	≥ 1 Mal X	X^+
$X?$	≤ 1 Mal X	$X \cup \{\varepsilon\}$
$X\{i\}$	i Mal X	X^i
$X\{i,\}$	$\geq i$ Mal X	$X^i \cdot X^*$
$X\{i,j\}$	i bis j Mal X	$X^i \cup X^{i+1} \cup \cdots \cup X^j$

4) Syntax Diagramm Notation

Graphische Form der EBNF

▼ Beispiel: Reele Numerale





Eigenschaften

Entscheidbarkeit

Ein ja/nein Problem ist entscheidbar wenn ein Algorithmus für jede Eingabe immer ja/nein liefern kann.

Beispiel: Halteproblem (Nicht entcheidbar)

> Man kann nicht algorithmisch bestimmen ob ein Algorithmus anhält oder nicht (Endlosschleife) weil sich der Prüfer selbst aufhängen könnte.

Entscheidbare Probleme bei Reg. Ausdrücken r

- Ist $w \in \mathcal{L}(r)$? (Wortproblem)
- Ist $\mathcal{L}(r) = \mathcal{L}(r')$? (Äquivalenzproblem)
- Ist $\mathcal{L}(r)$ leer / endlich / unendlich?

Ausdruckskraft

Gleich Ausdruckskräftig wie endliche automaten. $\mathcal{L}(r) = \mathcal{L}(\mathcal{A})$

Übersetzung in beide Richtungen möglich.

Nicht reguläre Sprachen

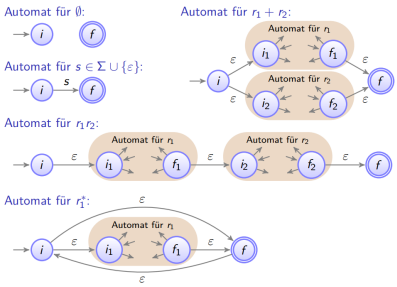
Lassen sich nicht durch reguläre Ausdrücke ausdrücken.

Beispiele: Wohlgeformte Klammerausdrücke:

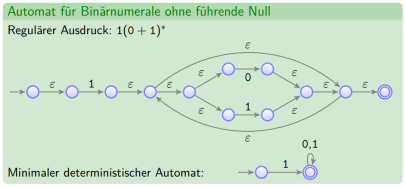
- $\{(), (()), ()(), ((())), (())(), ()()(), \ldots\}$
- $\{\mathrm{a}^n\ \mathrm{b}^n \mid n \geq 0\} = \{\varepsilon, \mathrm{ab},\ \mathrm{aabb},\ \mathrm{aabbb}\ , \ldots\}$
- $\{a^n b^n c^n \mid n \geq 0\} = \{\varepsilon, \mathrm{abc}, \mathrm{aabbcc},\ \mathrm{aaabbbccc}\ , \ldots\}$
- ...

Regulärer Ausdruck $\longrightarrow$ Automat

Allgemeine Regeln:



Beispiel:



Regulärer Ausdruck ⟵ Automat

Verallgemeinerter endlicher Automat

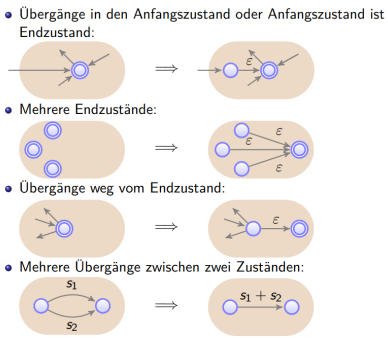
$\mathcal{A} = \langle Q, \Sigma, \delta, i, f \rangle$

Q	endliche Zustandsmenge
Σ	Eingabealphabet
$\delta : (Q - \{f\}) \times (Q - \{i\}) \mapsto R$	Übergangsfunktion wobei
R = Menge der regulären Ausdrücke über Σ	
$i \in Q$	Anfangszustand
$f \in Q, f \neq i$	Endzustand

Regeln

1. keine Kanten zurück zum Anfangszustand
2. nur ein Endzustand der kein Anfangszustand ist
3. Keine Kanten weg vom Endzustand
4. Nur eine einzige Kante zwischen 2 Zuständen
5. Übergänge beschriftet mit regulären Ausdrücken

Probleme lösen

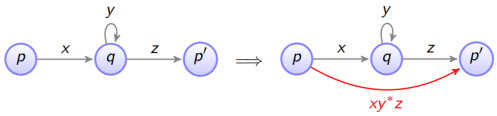


Algorithmus

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e234665b-bdd3-4fa2-a7f3-e87aa2a08245/fmod6-einzelschritte.pdf

Für jeden Zustand $q \in Q - \{i, f\}$:

1. Füge zwischen allen Nachbarn p, p' von q neue Übergänge hinzu mit regulären Ausdrücken beschriftet
2. Entferne q und alle Kanten von und nach q aus dem Automaten.



(x, y und z bezeichnen reguläre Ausdrücke.)

Kontextfreie Grammatiken

Motivation

Nicht reguläre Sprachen lassen sich nicht durch reguläre Ausdrücke ausdrücken.

Beispiele: Wohlgeformte Klammerausdrücke:

- $\{(), (()), ()(), ((())), (())(), ()()(), \ldots\}$
- $\{\mathrm{a}^n\ \mathrm{b}^n \mid n \geq 0\} = \{\varepsilon, \mathrm{ab},\ \mathrm{aabb},\ \mathrm{aabbb}\ , \ldots\}$

Formale Sprache ist kontextfrei, wenn sie von kontextfreier Grammatik generiert werden kann.

Reguläre Sprache lässt sich durch kontextfreie Sprache ausdrücken aber nicht umgekehrt.

Dinge die nicht ausgedrückt werden können:

Formale Sprachen

$\{a^n b^n c^n \mid n \geq 0\} = \{\varepsilon, \mathrm{abc}, \mathrm{aabbcc},\ \mathrm{aaabbbccc}\ , \ldots\}$ ist nicht kontextfrei

$\{ww \mid w \in \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}^*\}$ ist nicht kontextfrei

Natürliche Sprachen

Kontext-Sensitive Sätze, Reime in natürlicher Sprache

Kontextfreie Grammatik

$G = \langle V, T, P, S \rangle$

- V Non-Terminale (Variablen)
- T Terminal-Symbole
- $P \subseteq V \times (V \cup T)^*$ Produktionen
  Man schreibt $A \to w$ statt $(A, w) \in P$
- $S \in V$ Startsymbol

Ableitbarkeit $\Rightarrow$

Die Ableitungsregel beeinflusst nicht das Resultat.

Allgemein

Angenommen $x, y \in (V \cup T)^*$:

- $xAy \Rightarrow xwy$ falls $A \to w \in P$
- $u \overset{*}{\Rightarrow} v$ falls
  - $u = v$ oder
  - $u \Rightarrow u' \wedge u' \overset{*}{\Rightarrow} v$ für ein $u' \in (V \cup T)^*$

Links-Ableitbarkeit

Variable ganz links zuerst ersetzt.

$xAy \Rightarrow_L xwy$ wenn $A \to w \in P, x \in T^*$

Rechts-Ableitbarkeit

Variable ganz rechts zuerst ersetzt.

$xAy \Rightarrow_R xwy$ wenn $A \to w \in P, y \in T^*$

Parallel-Ableitbarkeit

$x_0 A_1 x_1 \cdots A_n x_n \Rightarrow_P x_0 w_1 x_1 \cdots w_n x_n$

wenn $A_1 \to w_1, \ldots, A_n \to w_n \in P$

und $x_0, \ldots, x_n \in T^*$

Kontextfreie Sprache $\mathcal{L}(G)$

Von Grammatik G generierte Sprache

Quasi Menge aller mit der Grammatik ableitbaren Wörter.

$\mathcal{L}(G) = \{w \in T^* \mid S \overset{*}{\Rightarrow} w\}$

Alle Wörter aus Terminal-Verknüpfungen bei denen ein Startsymbol nach Ableitung zu einem gültigen Wort führt.

Beispiel:

$G = \langle \{S\}, \{\mathrm{a}, \mathrm{b}\}, \{S \to \varepsilon \mid \mathrm{a}S\mathrm{b}\}, S \rangle$

$\mathcal{L}(G) = \{\mathrm{a}^n\,\mathrm{b}^n \mid n \geq 0\} = \{\varepsilon,\ \mathrm{ab},\ \mathrm{aabb},\ \mathrm{aaabbb},\ \ldots\}$

zB $S \overset{*}{\Rightarrow} \mathrm{aabb}$ möglich weil

$S \Rightarrow \mathrm{a}S\,\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow \mathrm{aa}\varepsilon\mathrm{bb} = \mathrm{aabb}$

1) Backus-Naur-Form BNF

V in Spitzen klammern $\langle \ldots \rangle$

T unter Anführungszeichen “...”

$::=$ Für Zuweisungen

Reele Numerale

$\langle \text{Real} \rangle ::= \langle \text{Digit} \rangle \langle \text{Digits} \rangle \text{“.”} \langle \text{Digits} \rangle \langle \text{Scale} \rangle$

$\langle \text{Digit} \rangle ::= \text{“0”} \mid \ldots \mid \text{“9”}$

$\langle \text{Digits} \rangle ::= \varepsilon \mid \langle \text{Digit} \rangle \langle \text{Digits} \rangle$

$\langle \text{Scale} \rangle ::= \varepsilon \mid \text{“E”} \langle \text{Sign} \rangle \langle \text{Digit} \rangle \langle \text{Digits} \rangle$

$\langle \text{Sign} \rangle ::= \varepsilon \mid \text{“+”} \mid \text{“-”}$

2) Erweiterte Backus-Naur-Form EBNF

Vorteile der BNF:

Keine Angabe von Terminalen und Nonterminalen sowie Anfangsterminalen mehr notwendig weil es sich von der Notation selbst ablesen lässt.

Man gibt nur noch die Produktionen an.

Vorteile der EBNF (verglichen zur BNF):

Leichtere Syntax ohne `(<, >, ::=)` indem zwischen Terminalen und Nonterminalen mit `""` differenziert wird.

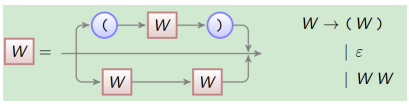
Abkürzungen aus regulären Ausdrücken übernommen wie `(), [], {}, ?`.

Regular Right Part Grammar (RRPG)

Man benutzt nur `""` und die folgenden Klammern: Vermeidung von Rekursionen und Leerwörtern

(w_i)	1 Mal w_i
$\{w_i\}$	≥0 Mal w_i
$[w_i]$	0-1 Mal w_i

3) (Rekursive) Syntaxdiagramme



Beispiele

Syntax aussagenlogischer Formeln

$$G = \langle \{Fm, Op, \text{Var}\}, T, P, Fm \rangle$$

$$T = \{\top, \bot, \neg, (,), \wedge, \uparrow, \vee, \downarrow, \equiv, \not\equiv, \supset, \subset\} \cup \mathcal{V}$$

$$P = \{Fm \to \text{Var} \,|\top|\, \bot \,|\neg Fm|(FmOpFm),$$
$$Op \to \wedge|\uparrow|\vee|\downarrow|\equiv|\not\equiv|\supset|\subset$$
$$\text{Var} \to \text{A}|\text{B}|\text{C}\,|\cdots\}$$

Beispiel: $((A \uparrow B) \uparrow (A \uparrow B))$

$$Fm \Rightarrow_P P(FmOpFm)$$
$$\Rightarrow_P P((FmOpFm) \uparrow (FmOpFm))$$
$$\Rightarrow_P P((\text{Var} \uparrow \text{Var}) \uparrow (\text{Var} \uparrow Var))$$
$$\Rightarrow_P P((A \uparrow B) \uparrow (A \uparrow B))$$

Reele Numerale

$$G = \langle V, T, P, \text{ Real } \rangle$$

$$V = \{ \text{ Real, Scale, Sign, Digits, Digit } \}$$

$$T = \{0, \ldots, 9, ., \text{E}, +, -\}$$

$$P = \{$$
$$Real \to Digit\ Digits.Digits\ Scale$$
$$Scale \to \varepsilon \mid E\ Sign\ Digit\ Digits$$
$$\text{Sign} \to \varepsilon| + | -$$
$$Digits \to \varepsilon \mid Digit\ Digits$$
$$Digit \to 0|\cdots|9$$
$$\}$$

Wohlgeformte Klammerausdrücke

Sie sind die Stärke der kontextfreien Grammatik die sie vor allem von regulären Sprachen unterscheidet.

$$G = \langle \{W\}, \quad \{(,)\}, \quad \{W \to \varepsilon|(W)|WW\}, \quad W \rangle$$

$$\mathcal{L}(G) = \{\varepsilon, (), (()), ()(), ((())), (())(), ()(()), ()()(), \ldots\}$$

Beispiel:

$$W \Rightarrow_P PWW$$
$$\Rightarrow_P P(W)(W)$$
$$\Rightarrow_P P()(WW)$$
$$\Rightarrow_P P()((W)(W))$$
$$\Rightarrow_P P()(()())$$

Definition, Automatenarten

Sichtweisen

Akzeptor	Automat liest Symbole und akzeptiert nur gültige
Generator	Automat schreibt Symbole die als Zeichenkette zum Entzust. führen

Automat

Endlisch: automaton/automata, finite state machine, Finite Automaton FA, DFA (Deterministic), NFA (Non-Determinstic)

- endliche Anzahl an Zuständen
  - Anfangszustand (nur 1)
  - Endzustände (optional)
- endliche Symbolfolgen als Eingabe
- Übergänge zwischen Zuständen
- Eingaben / Ausgaben / Aktionen
  - während Zustandsübergang oder im Zustand selbst
  - Eingaben XOR Ausgaben
- kann mit / ohne ε-Übergängen sein

Transducer

- mit Ein- und Ausgang
- kann auch unendliche Symbolfolgen verarbeiten

Unterarten

1. Deterministisch DEA
   - momentaner Zustand $\times$ nächste Eingabe $\mapsto$ Folgezustand (eindeutig)

   Verglichen zu NEAs
   - Effizientere Abarbeitung, kein Backtracking
2. Nichtdeterministisch NEA
   - momentaner Zustand $\times$ nächste Eingabe $\mapsto$ { Folgezustände }

   Verglichen zu DEAs
   - Flexibler
   - Leerwort erlaubt
   - Weniger Zustände
   - leichter zu konstruieren
3. Mealy-Automat (Transducer)
   - Deterministisch
   - keine Endzustände
   - Ausgabe mit Eingabe verknüpft
   - Die Ausgabe erfolgt bei Zustandswechsel
4. Moore-Automat (Transducer)
   - Deterministisch
   - keine Endzustände

Ausgabe mit Zustand verknüpft

Die Ausgabe erfolgt durch momentanen Zustand

5. Büchi-Automat

   Wie endlicher Automat aber unendliche Symbolfolgen

6. ...

   zb Verallgemeinerter endlicher Automat, Muller-Automat, Rabin-Automat, Baumautomaten, ...

Allgemein gilt:

1. DEAs und NEAs besitzen dieselbe Ausdrucksstärke.

   Jede DEA ist ein NEA.

   Jede NEA lässt sich Determinisieren.

2. Moore und Mealy besitzen dieselbe Ausdrucksstärke.

   beide schwächer als Transducer.

   Moore haben in der Regel mehr Zustände als Mealy.

3. Nichtdeterministische Büchli sind ausdrucksstärker als deterministische

Endliche Automaten

Definition

📌 **Definition, Automatenarten**

Deterministische endliche Automaten DEA

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$

Q *all possible states (finite)*

$\sum$ *input alphabet* (kein ε erlaubt)

$\delta : Q \times \Sigma \mapsto Q$ *transition function*

Totale Funktion: Funktion, die für *jeden* Wert aus Definitonsmenge ein Abbild hat. Deshalb deterministisch.

$q_0 \in Q$ *initial state*

$F \subseteq Q$ *final states*

Erweiterte Übergangsfunktion (auch für Leerwort)

$\delta^* : Q \times \Sigma^* \mapsto Q$

$\forall q \in Q, s \in \Sigma, w \in \Sigma^* : \quad \delta^*(q, \varepsilon) = q, \quad \delta^*(q, sw) = \delta^*(\delta(q, s), w)$

Akzeptierte / Generierte Sprache

Nur Wörter die von *initial state* zu einem *final state* führen.

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \in F\}$

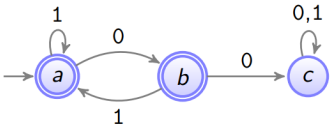
Falle / Fehlerzustand

Wird meistens nicht eingezeichnet aber existiert in Tabelle.

Alle Fehlverhalten → Falle, dann kein Weg heraus.

Beispiel:

▼ 00-freie Binärstrings



c ist die Falle

$Q = \{a, b, c\}$ Zustandsmenge

$\Sigma = \{0, 1\}$ Eingabealphabet

$q_0 = a$ Anfagszustand

$F = \{a, b\}$ Endzustand

$\delta : Q \times \Sigma \mapsto Q$ definiert durch:

δ	0	1
a	b	a
b	c	a
c	c	c

Beispielsweise: $101 \in \mathcal{L}(\mathcal{A})$

Nichtdeterministische endliche Automaten NEA

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$

Alles gleich mit DEA nur ε erlaubt:

$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ Übergangsrelation

$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \mapsto 2^Q$ Übergangsfunktion (bildet auf Element aus Menge ab)

Erweiterte Übergangsfunktion (auch für Leerwort)

δ^* ist die kleinste Menge mit den Eigenschaften

$\forall q \in Q : (q, \varepsilon, q) \in \delta^*$

$\exists (q_1, w, q_2) \in \delta^*, (q_2, s, q_3) \in \delta : \quad (q_1, ws, q_3) \in \delta^*$ (Transivität)

Akzeptierte / Generierte Sprache

Nur Wörter die von *initial state* zu einem *final state* führen.

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, w, q_f) \in \delta^* \text{ für ein } q_f \in F\}$ (Es kann mehrere Endzustände geben)

Beispiel:

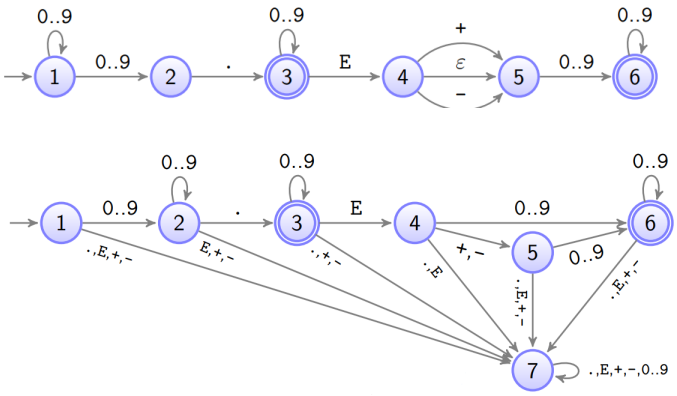
▼ Reelle Numerale mit Exponential-Teil

gültige Beispiele: `3.14`, `0.314E1`, `314.E-2`,...

Vorausgesetzt:

1. Min 1 Ziffer vor Punkt
2. Exponentialteil optional (mit E)
   - Vorzeichen optional
   - mindestens eine Ziffer nach E

Endlicher (nicht deterministischer Automat) mit ϵ als Leerwort.



$\mathcal{A} = \langle \{1, \ldots, 7\}, \{0, \ldots, 9, ., \mathrm{E}, +, -\}, \delta, 1, \{3, 6\} \rangle$

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid (1, w, 3) \in \delta^* \text{ oder } (1, w, 6) \in \delta^*\}$

δ	0	$\cdots$	9	.	E	+	−	ε
1	$\{1,2\}$	$\cdots$	$\{1,2\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
2	$\{\}$	$\cdots$	$\{\}$	$\{3\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
3	$\{3\}$	$\cdots$	$\{3\}$	$\{\}$	$\{4\}$	$\{\}$	$\{\}$	$\{\}$
4	$\{\}$	$\cdots$	$\{\}$	$\{\}$	$\{\}$	$\{5\}$	$\{5\}$	$\{5\}$
5	$\{5,6\}$	$\cdots$	$\{5,6\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
6	$\{\}$	$\cdots$	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$

Determinisierung (NEA $\longrightarrow$ DEA)

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9c9df8d2-c83b-4742-996d-b094dd2e1903/determinisierung.pdf

Gegeben: NEA

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ (A steht für Automat)

$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$

Gesucht: DEA

$\widehat{\mathcal{A}} = \left\langle \widehat{Q}, \Sigma, \hat{\delta}, \widehat{q_0}, \widehat{F} \right\rangle$

$\hat{\delta} : \widehat{Q} \times \Sigma \mapsto \widehat{Q}$

Unter der Bedingung dass $\widehat{\mathcal{A}}$ und $\mathcal{A}$ äquivalent sind (dieselbe Sprache akzeptieren = sich gleich verhalten) gilt: $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\widehat{\mathcal{A}})$

Definition von DEA $\widehat{\mathcal{A}}$

$\widehat{Q} = 2^Q$ (Potenzmenge von Q - Alle Teilmengen / Zustands-Wörter)

$\widehat{q_0} = \{q_0\}$ (Klammern stehen nicht für Menge)

$$\widehat{F} = \begin{cases} \{\hat{q} \in \widehat{Q} \mid \hat{q} \cap F \neq \emptyset\} \cup \{\widehat{q_0}\} & \text{falls } \varepsilon \in \mathcal{L}(\mathcal{A}) \\ \{\hat{q} \in \widehat{Q} \mid \hat{q} \cap F \neq \emptyset\} & \text{sonst} \end{cases}$$

Unsere finalen Zustände sind alle Zustands-Wörter / Mengen die bei einem Durchschnitt mit dem Zustand F nicht leer sind → Die den Buchstaben F enthalten.

Falls Leerwörter erlaubt sind muss der Anfangszustand auch enthalten sein.

$\forall \hat{q} \in Q, s \in \sum$:

$\hat{\delta}(\hat{q}, s) = \{q' \in Q \mid \exists q \in \hat{q}, \text{ sodass } (q, s, q') \in \delta^*\}$

bzw

$\hat{\delta}(\hat{q}, s) = \bigcup_{q \in \hat{q}} \{q' \in Q \mid (q, s, q') \in \delta^*\} = \bigcup_{q \in \hat{q}} \delta^*(q, s)$

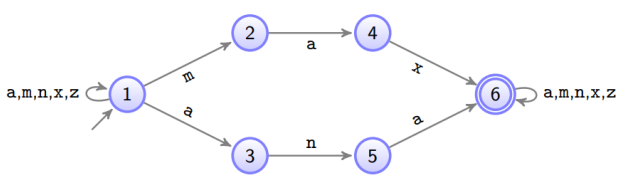
Die Übergangsfunktion:

$Q \times s \in \sum \mapsto \{\dots\}$ (Wort - keine Menge)

Bildet jeden Zustand mit Symbol auf einem Wort, dass alle Folgezustände enthält

Beispiel:

▼ Suche nach "max" und "ana"



Dadurch, dass es keine ϵ gibt ist Tabelle von δ gleich mit δ^*

Anfangszustand ist 1, Endzustand ist 6.

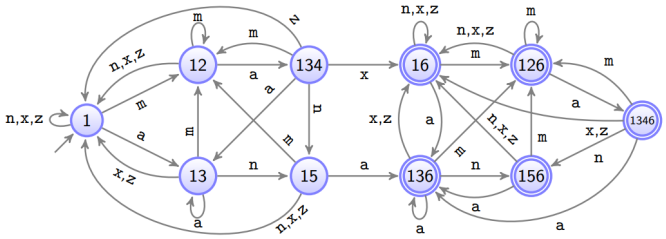
δ^*	a	m	n	x	z
1	$\{1,3\}$	$\{1,2\}$	$\{1\}$	$\{1\}$	$\{1\}$
2	$\{4\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
3	$\{\}$	$\{\}$	$\{5\}$	$\{\}$	$\{\}$
4	$\{\}$	$\{\}$	$\{\}$	$\{6\}$	$\{\}$
5	$\{6\}$	$\{\}$	$\{\}$	$\{\}$	$\{\}$
6	$\{6\}$	$\{6\}$	$\{6\}$	$\{6\}$	$\{6\}$

Wir definieren eine $\hat{\delta}$

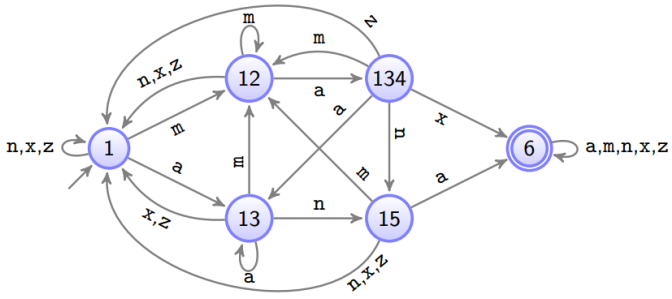
Anfangszustand ist 1, Endzustand ist jeder Block der mit 1 anfängt und mit 6 endet.

Wichtig: die Tabelleneinträge sind keine Mengen sondern Wörter obwohl wir {} benutzen.

$\hat{\delta}$	a	m	n	x	z
$\{1\}$	$\{1,3\}$	$\{1,2\}$	$\{1\}$	$\{1\}$	$\{1\}$
$\{1,2\}$	$\{1,3,4\}$	$\{1,2\}$	$\{1\}$	$\{1\}$	$\{1\}$
$\{1,3\}$	$\{1,3\}$	$\{1,2\}$	$\{1,5\}$	$\{1\}$	$\{1\}$
$\{1,3,4\}$	$\{1,3\}$	$\{1,2\}$	$\{1,5\}$	$\{1,6\}$	$\{1\}$
$\{1,5\}$	$\{1,3,6\}$	$\{1,2\}$	$\{1\}$	$\{1\}$	$\{1\}$
$\{1,6\}$	$\{1,3,6\}$	$\{1,2,6\}$	$\{1,6\}$	$\{1,6\}$	$\{1,6\}$
$\{1,3,6\}$	$\{1,3,6\}$	$\{1,2,6\}$	$\{1,5,6\}$	$\{1,6\}$	$\{1,6\}$
$\{1,2,6\}$	$\{1,3,4,6\}$	$\{1,2,6\}$	$\{1,6\}$	$\{1,6\}$	$\{1,6\}$
$\{1,5,6\}$	$\{1,3,6\}$	$\{1,2,6\}$	$\{1,6\}$	$\{1,6\}$	$\{1,6\}$
$\{1,3,4,6\}$	$\{1,3,6\}$	$\{1,2,6\}$	$\{1,5,6\}$	$\{1,6\}$	$\{1,6\}$



Da von den Endzuständen keine Kanten zurückführen und man somit den Endzustand nicht verlässt: kürzen möglich.



Endlicher Transducer

Automat beschrieben durch $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, I, F \rangle$

- Eingabe *und* Ausgabe
- Menge an Anfangszuständen

Γ *output alphabet*

$\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \times Q$ Übergangsrelation

$I \subseteq Q$ *initial states*

Erweiterte Übergangsrelation (auch für Leerwort)

$\delta^* \subseteq Q \times \Sigma^* \times \Gamma^* \times Q$

δ^* ist die kleinste Menge bei der

$\forall q \in Q : (q, \varepsilon, \varepsilon, q) \in \delta^*$

$(q_1, w, w', q_2) \in \delta^*, (q_2, s, s', q_3) \in \delta \implies (q_1, ws, w's', q_3) \in \delta^*$

Übersetzungsrelation

Automat generiert nicht mehr Sprache sondern gibt uns eine Übersetzungsrelation (zu jedem Eingabewort, ein Ausgabewort)

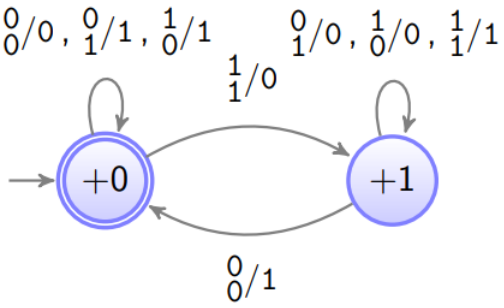
$[\mathcal{A}] = \{(w, w') \in \Sigma^* \times \Gamma^* \mid \forall i \in I, f \in F : (i, w, w', f) \in \delta^*\}$

$[\mathcal{A}] \subseteq \Sigma^* \times \Gamma^*$

Beispiele:

▼ Binäraddition von R nach L (deterministisch)

$$\begin{array}{cccccc|l} 0 & 0 & 1 & 0 & 1 & 1 & = 11_{10} \\ 0 & 0 & 0 & 1 & 0 & 1 & = 5_{10} \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & = 16_{10} \end{array}$$



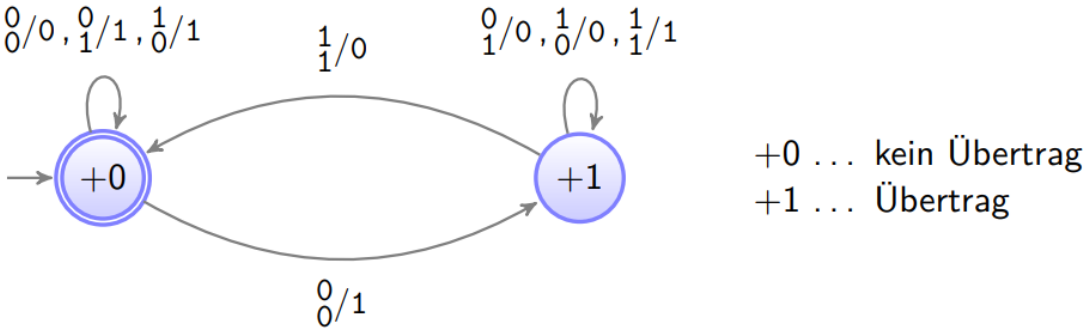
$\mathcal{A} = \langle \{+0, +1\}, \{\binom{0}{0}, \binom{0}{1}, \binom{1}{0}, \binom{1}{1}\}, \{0,1\}, \delta, \{+0\}, \{+0\} \rangle$, wobei

δ	$\binom{0}{0}$	$\binom{0}{1}$	$\binom{1}{0}$	$\binom{1}{1}$
$+0$	$\{(0,+0)\}$	$\{(1,+0)\}$	$\{(1,+0)\}$	$\{(0,+1)\}$
$+1$	$\{(1,+0)\}$	$\{(0,+1)\}$	$\{(0,+1)\}$	$\{(1,+1)\}$

$$[\mathcal{A}] = \{ (\varepsilon, \varepsilon), (\tbinom{0}{0}, 0), (\tbinom{0}{1}, 1), (\tbinom{1}{0}, 1), (\tbinom{0\,0}{0\,0}, 00), (\tbinom{0\,0}{0\,1}, 01), \ldots, (\tbinom{1\,0}{1\,0}, 01), \ldots, (\tbinom{1\,0\,1\,0}{1\,0\,1\,0}, 0101), \ldots \}$$

▼ Binäraddition von L nach R (nicht deterministisch)

$$\begin{array}{cccccc|l} 0 & 0 & 1 & 0 & 1 & 1 & = 11_{10} \\ 0 & 0 & 0 & 1 & 0 & 1 & = 5_{10} \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & = 16_{10} \end{array}$$



Mealy-Automat

Art von Transducer, deterministisch

- Ausgabe mit Eingabe verknüpft
- Nur ein Anfangszustand $I = \{q_0\}$
- Alle Zustände sind Endzustände $F = Q$
- Keine ε Übergänge

Automat beschrieben durch $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, \gamma, q_0 \rangle$

Γ *output alphabet*

$\gamma : Q \times \Sigma \mapsto \Gamma$ *output function* / Ausgabefunktion

- Bildet Relations-Tupel → Zu jeder Eingabe und jedem Zustand eine Ausgabe $(q, s, \gamma(q, s), \delta(q, s))$

Erweiterte Ausgabefunktion (auch für Leerwort)

$\gamma^* : Q \times \Sigma^* \mapsto \Gamma^*$

$\forall q \in Q, s \in \Sigma, w \in \Sigma^* :$

- $\gamma^*(q, \varepsilon) = \varepsilon$
- $\gamma^*(q, sw) = \gamma(q, s) \cdot \gamma^*(\delta(q, s), w)$

Übersetzungsfunktion

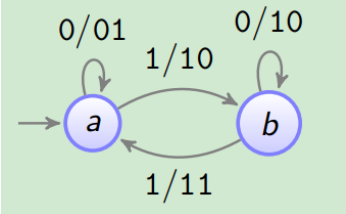
$[\mathcal{A}](w) = \gamma^*(q_0, w)$

$[\mathcal{A}] : \Sigma^* \mapsto \Gamma^*$

Beispiel:

▼ (1:2)-(0,1)-RLL Encoder

$\mathcal{A} = \langle \{a, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$



δ	0	1
a	a	b
b	b	a

Zustand

γ	0	1
a	01	10
b	10	11

Ausgabe

$w:$	ε	0	1	00	10	01	11	000	100	$\cdots$
$[\mathcal{A}](w):$	ε	01	10	0101	1010	0110	1011	010101	101010	$\cdots$

Moore-Automat

Art von Transducer (technisch gesehen aber nicht), deterministisch

- Ausgabe mit Zustand verknüpft
- Nur ein Anfangszustand $I = \{q_0\}$
- Alle Zustände sind Endzustände $F = Q$
- Keine ε Übergänge

Automat beschrieben durch $\mathcal{A} = \langle Q, \Sigma, \Gamma, \delta, \gamma, q_0 \rangle$

Γ *output alphabet*

$\gamma : Q \mapsto \Gamma$ *output function* (bei Mealy: $\gamma : Q \times \Sigma \mapsto \Gamma$)

Erweiterte Ausgabefunktion (auch für Leerwort)

$\gamma^* : Q \times \Sigma^* \mapsto \Gamma^*$

$\forall q \in Q, s \in \Sigma, w \in \Sigma^* :$

- $\gamma^*(q, \varepsilon) = \varepsilon$
- $\gamma^*(q, sw) = \gamma(q) \cdot \gamma^*(\delta(q, s), w)$
- $\gamma^*(q, sw) = \gamma(q, s) \cdot \gamma^*(\delta(q, s), w)$ - bei Mealy

Übersetzungsfunktion (gleich wie Mealy)

$[\mathcal{A}](w) = \gamma^*(q_0, w)$

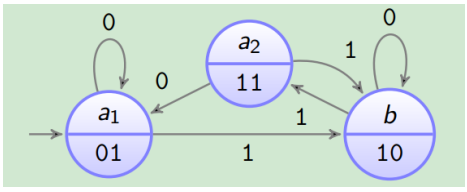
$[\mathcal{A}] : \Sigma^* \mapsto \Gamma^*$

Beispiel:

▼ (1:2)-(0,1)-RLL Encoder

$\mathcal{A} = \langle \{a, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$ bei Mealy

$\mathcal{A} = \langle \{a_1, a_2, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$



δ	0	1
a_1	a_1	b
a_2	a_1	b
b	b	a_2

γ	
a_1	01
a_2	11
b	10

Büchi-Automaten

Deterministisch

Automat beschrieben durch $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ (alles gleich wie in DEAs definiert)

Akzeptanz von Wörtern

Wort $s_1 s_2 s_3 \cdots \in \sum^{\omega}$ wird akzeptiert wenn es Zustände $q_0, q_1, q_2, q_3, \cdots \in Q$ gibt, sodass

- $q_0 \in Q$ ist Startzustand
- $\forall i \in \mathbb{N} : \delta\left(q_{i-1}, s_i\right) = q_i$
- $\lim_{i \to \infty}$ sodass irgendwann $q_i \in F$ (Endzustand ist ein "guter Zustand", kein Deadlock)

Akzeptierte/Generierte Sprache

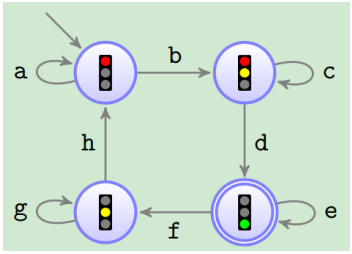
$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^{\omega} \mid w \text{ wird von } \mathcal{A} \text{ akzeptiert } \}$

Beispiel:

Anwendung: in Programmen die liveliness property feststellen - Programm darf nicht absterben

▼ Faire Verkehrsampel

Es werden die wörter aus $\{\mathtt{a}, \ldots, \mathtt{h}\}^{\omega}$ akzeptiert bei denen es immer wieder grün wird.



Nicht-Deterministisch

Automat beschrieben durch $\mathcal{A} = \langle Q, \Sigma, \delta, I, F \rangle$

$I \subseteq Q$ Anfangszustände

$\delta \subseteq Q \times \Sigma \times Q$ Übergangsrelation

Akzeptanz von Wörtern

Wort $s_1 s_2 s_3 \cdots \in \Sigma^{\omega}$ wird akzeptiert wenn es Zustände $q_0, q_1, q_2, q_3, \cdots \in Q$ gibt, sodass

$q_0 \in Q,\ q_0 \in I$

$\forall i \in \mathbb{N} : \delta\left(q_{i-1}, s_i\right) = q_i$

$\lim_{i \to \infty}$ sodass irgendwann $q_i \in F$

Akzeptierte/Generierte Sprache

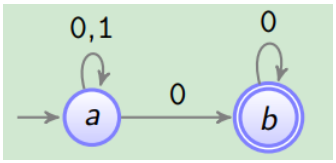
$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^{\omega} \mid w \text{ wird von } \mathcal{A} \text{ akzeptiert } \}$

Beispiel:

▼ Nur endlich viele 1er

Ein Büchi-Automat, der genau jene ω-Wörter über {0, 1} akzeptiert, die nur eine endliche Anzahl an 1ern enthalten.

(wird nur von deterministischen akzeptiert)



Modellierung

📌 Modellierung durch endlichen Automaten

Modellierung durch endlichen Automaten

Vorgang:

1. Zustände? → Q und I, F
2. Aktionen/Eingaben? → die Zustandsübergänge auslösen / dabei stattfinden
3. Tabelle erstellen, alle Fälle abdecken

Beispiel: Simpsons Flussbeispiel

Dieses Beispiel wäre als Automat leichter zu modellieren.

$$\left.\begin{array}{ll} M & \dots \text{ Maggie} \\ K & \dots \text{ Knecht Ruprecht} \\ G & \dots \text{ Gift} \\ H & \dots \text{ Homer} \end{array}\right\} \text{System Koponenten}$$

System-Zustand:

$$\frac{\text{Dinge hier}}{\text{Dinge dort}}$$

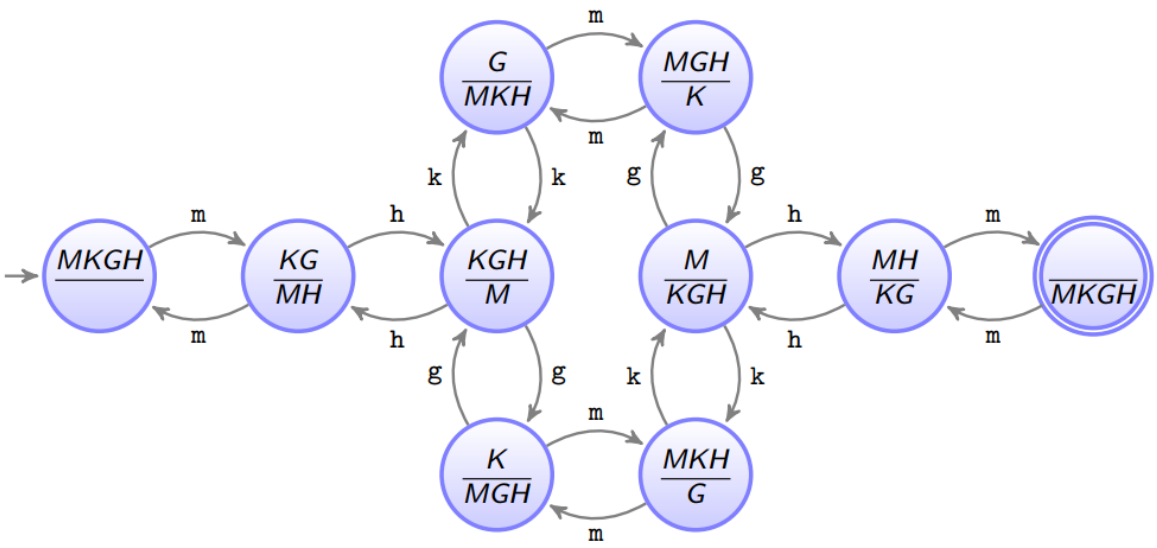
Endzustand: $\frac{\text{-}}{\text{MKGH}}$ Anfangszustand: $\frac{\text{MKGH}}{\text{-}}$

Verbotene Zustände:

$$\frac{GH}{MK}, \frac{MK}{GH}, \frac{KH}{MG}, \frac{MG}{KH}, \frac{H}{MKG}, \frac{MKG}{H}$$

Zustandsübergänge: $\mathrm{h}, \mathrm{m}, \mathrm{k}, \mathrm{g}$

Dadurch Automat ohne verbotenen Zuständen:

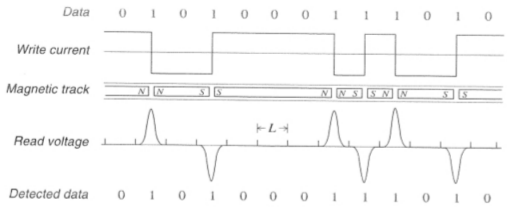


Alle gültigen Wörter (die zum Ziel führen) sind die Sprache des Automaten.

Mögliche Lösungen: { mhkmghm, mmmhhhgmkhm, ..., mhkmgkmgkmghm, ... }

Beispiel: RLL (run-length limited) Codes

Daten werden auf Festplatten, CDs und DVDs binär kodiert.



Probleme: Abstand zwischen den 1en...

- zu groß - zu viele 0en - Synchronisation geht verloren
- zu klein - zu wenige 0en - ununterscheidbar

(m : n)-(d, k)-RLL-Codes

Wir wollen die Anzahl der 0en beeinflussen.

Ziel: minimiere n/m, maximiere d

Lösung: Nütze die „natürlichen“ 0er-Strecken und 1er im Datenstrom.

m Datenbits $\rightarrow$ n Codebits

Zwischen zwei 1 sollen:

min d, max k

0 vorkommen.

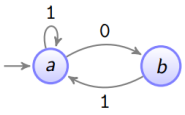
(1 : 2)-(0, 1)-RLL-Code

1 eingabe-bit → 2 ausgabe-bits

Es dürfen nicht zwei 0 hintereinander vorkommen.

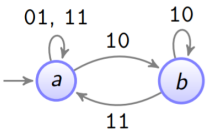
Wir machen einen Automaten der das erzwingt. (Ausgaben)

- Zustand **a** hat 2 augehende Kanten.
- Zustand **b** hat 1 augehende Kante. → problematisch bei 3 input bits



Wir "Potenzieren" den Automaten (mehr Kanten)

- Zustand **a** hat 3 augehende Kanten.
- Zustand **b** hat 2 augehende Kanten.



Jetzt entscheidet man willkürlich welches der 3 Kanten man für 0 und welches man für 1 wählt:

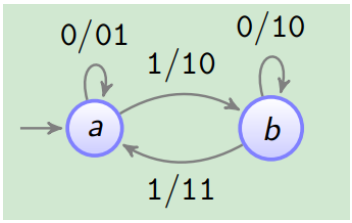
$10010 \Rightarrow 10\ 10\ 10\ 11\ 01$ oder $11\ 01\ 01\ 11\ 01$ oder...

Bedingung bleibt immer erhalten.

Vorherige Beispiele

▼ (1:2)-(0,1)-RLL Encoder

$\mathcal{A} = \langle \{a, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$



δ	0	1
a	a	b
b	b	a

Zustand

γ	0	1
a	01	10
b	10	11

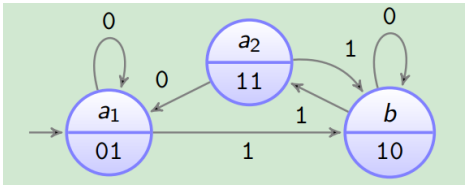
Ausgabe

$w:$	ε	0	1	00	10	01	11	000	100	$\cdots$
$[\mathcal{A}](w):$	ε	01	10	0101	1010	0110	1011	010101	101010	$\cdots$

▼ (1:2)-(0,1)-RLL Encoder

$\mathcal{A} = \langle \{a, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$ bei Mealy

$\mathcal{A} = \langle \{a_1, a_2, b\}, \{0, 1\}, \{01, 10, 11\}, \delta, \gamma, a \rangle$



δ	0	1
a_1	a_1	b
a_2	a_1	b
b	b	a_2

γ	
a_1	01
a_2	11
b	10

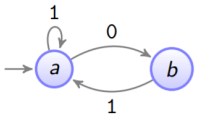
(2 : 3)-(0, 1)-RLL-Code

2 eingabe-bits → 3 ausgabe-bits

Es dürfen nicht zwei 0 hintereinander vorkommen.

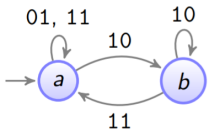
Wir machen einen Automaten der das erzwingt. (Ausgaben)

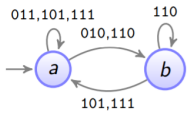
- Zustand **a** hat 2 augehende Kanten. → problematisch bei 4 input bots
- Zustand **b** hat 1 augehende Kante. → problematisch bei 4 input bits



Wir "Potenzieren" den Automaten mehrmals (mehr Kanten)

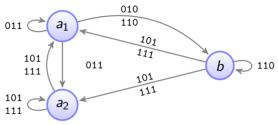
- Zustand **a** hat 2 augehende Kanten. → immer noch problematisch
- Zustand **b** hat 2 augehende Kanten. → immer noch problematisch



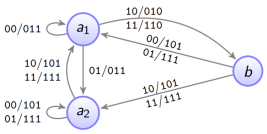


Wir "splitten" Zustand **a** auf damit **b** mehr Kanten hat

- Zustand a_1 hat 3 augehende Kanten.
- Zustand a_2 hat 2 augehende Kanten.
- Zustand **b** hat 3 augehende Kanten.



Mealy Automat:



Petri-Netze

Motivation

Als Aktivitätsdiagramme (activity diagrams) in UML.

Modellierung von nebenläufigen (concurrent/parallel) Systemen.

- Automat mit mehreren aktiven Stellen

Endliche Automaten ungeeignet:

- Anzahl der Zustände steigt exponentiell
- Nur jeweils ein aktiver Zustand

Definition

Marken / Tokens	Aktive Stelle, dürfen nur gleichzeitig weiter
Stellen / Places	Plätze für Marken
Transitionen	"feuern" wenn *alle* Eingangszustände Marken besitzen. (Synchronisation)
	Marken werden dann konsumiert und neu erzeugt.
	Es darf eine beliebige Transition gewählt werden.

Formale Definition

$N = \langle S,\ T,\ {}^\bullet(),\ ()^\bullet,\ m_0 \rangle$

- S Stellen-Menge
- T Transitions-Menge
- ${}^\bullet() : T \mapsto M$ Vorbedingungen (von Transition)
- $()^\bullet : T \mapsto M$ Nachbedingungen (von Transition)

M ... Markierungs-Zähler-Menge

Alle Funktionen die Marken von jeder Stelle zählen: $S \mapsto \mathbb{N}$

Teilmengen: (sind Mengen obwohl klein geschrieben)

- $m_0 \in M$ Anzahl der Marken an *jeder Stelle* zu Beginn
- ${}^\bullet t \in M$ Anzahl der entfernten Marken von *jeder Stelle* durch Transition t
- $t^\bullet \in M$ Anzahl der hinzugefügten Marken von *jeder Stelle* durch Transition t

Beliebige $m, m', m'' \in M$:

$\forall s \in S$:

Ordnung	$m \leq m'$ falls $m(s) \leq m'(s)$
Addition	$m \oplus m' = m''$ falls $m(s) + m'(s) = m''(s)$
Subtraktion	$m \ominus m' = m''$ falls $m(s) - m'(s) = m''(s)$

Schalten und Erreichbarkeit

- t Transition
- m Anzahl der Marken an *jeder Stelle* die mit Transition verbunden ist

t ist für m aktiviert wenn ${}^\bullet t \leq m$

Anzahl der entfernten Marken von *jeder Stelle* s durch Transition t $\leq$ Anzahl der Marken in jeder Stelle aus m

${}^\bullet t(S_1) \leq m(S_1)$

${}^\bullet t(S_2) \leq m(S_2)$

$\dots$

${}^\bullet t(S_n) \leq m(S_n)$

wenn an jeder Stelle S_i genug Marken vorhanden sind, um die Transition zu schalten.

Wenn t schaltet / feuert nachdem es von m aktiviert wurde, dann

$m' = m \ominus {}^\bullet t \oplus t^\bullet$

Werden die Tokens die ${}^\bullet t$ verlangt hat von m abgezogen und die Tokens die $t^\bullet$ generiert dem m hinzugefügt.

Das machen die Funktionen:

${}^\bullet() : T \mapsto M$ Vorbedingungen (von Transition)

$()^\bullet : T \mapsto M$ Nachbedingungen (von Transition)

Symbolische Abkürzung dafür:

$m[t\rangle m'$

Eine Markierung m_n (Funktion nur für einer einzigen Stelle S_i) ist "erreichbar im Netz" wenn,

es eine Folge von Transitionen $t_1, \dots, t_n$ gibt sodass $m_0 \, [t_1\rangle \, m_1 \dots m_{n-1} \, [t_n\rangle \, m_n$

(wobei m_0 die Anfangsmarkierung ist).

Graphische Notation

für ${}^\bullet t, t^\bullet$ zwischen s und t

Kein Pfeil	${}^\bullet t(s) = 0$ (bzw. $t^\bullet(s) = 0$)
Ein Pfeil	${}^\bullet t(s) = 1$ (bzw. $t^\bullet(s) = 1$)
Ein Pfeil mit Beschriftung	${}^\bullet t(s) = n > 1$, (bzw. $t^\bullet(s) = n > 1$)

${}^\bullet t(s), t^\bullet(s)$ wird auch als *Gewicht* bezeichnet.

Modellierungs-Beispiele

Ampel

Dining Philosophers

Leser-Schreiber-Problem

Hermann-Lin

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/305ec5a8-abc7-4742-a51a-448ce9580d95/hermann_lin_marked_(1).pdf

Dining Philosophers

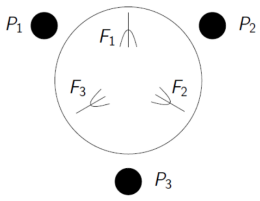
Philosoph sitzt am Tisch:

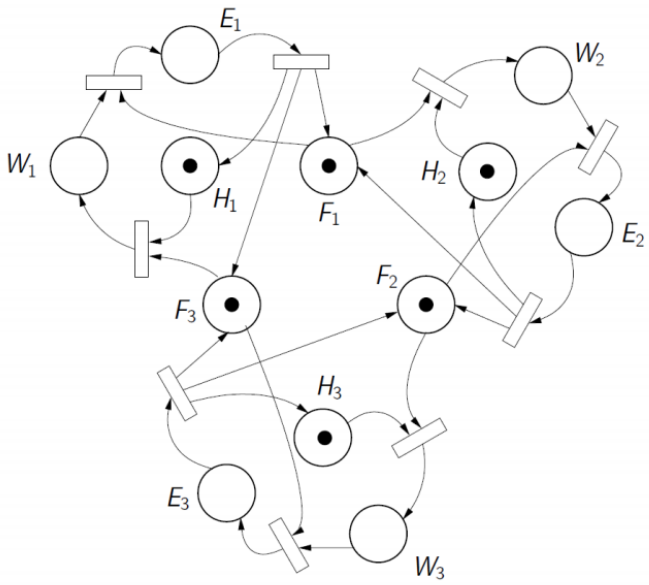
Wenn Philosoph hungrig ist dann

1. Rechte Gabel nehmen
2. Linke Gabel nehmen
3. Essen
4. Gabeln zurücklegen

Problem: Wenn Philosophen Greedy-Algorithmen sind die alle verfügbaren Ressourcen nehmen und warten dann entsteht ein Dead-Lock.

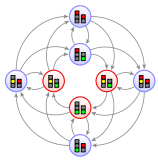
Es braucht einen Scheduler.

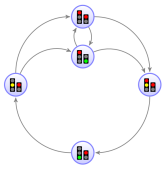




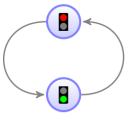
Ampel

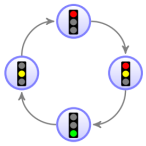
Als ein einziges System

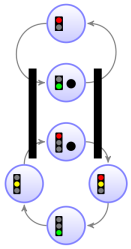




Als nebenläufiges System







Leser-Schreiber-Problem

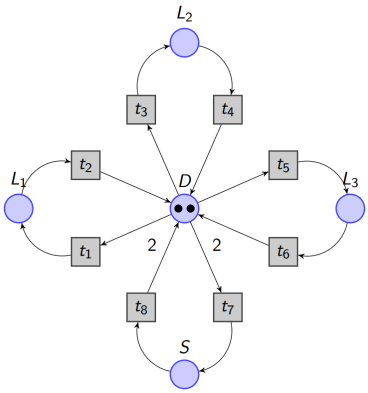
$n = 3$ Leseprozesse

$m = 1$ Schreibprozesse

Alle greifen auf dieselbe Datei zu.

Regeln:

- Zur gleichen Zeit dürfen mehrere Leserprozesse auf die Datei zugreifen.
- Schreiberprozess darf nur dann zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

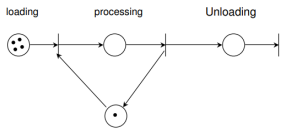


Hermann-Lin

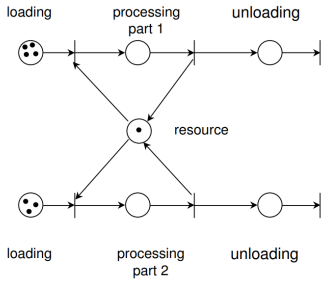
Basics

Synchronization and Resource Sharing

Only one process at a time

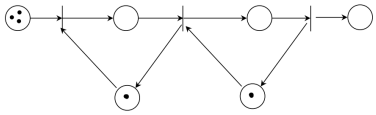


One worker for two machines.
The worker can work at one machine at a time.



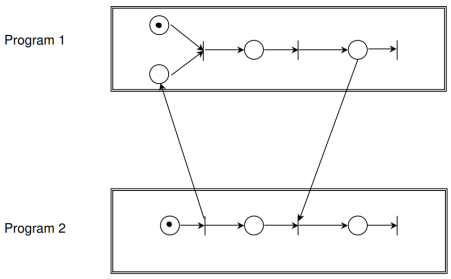
Buffer

Just like above, only one item processed at a time

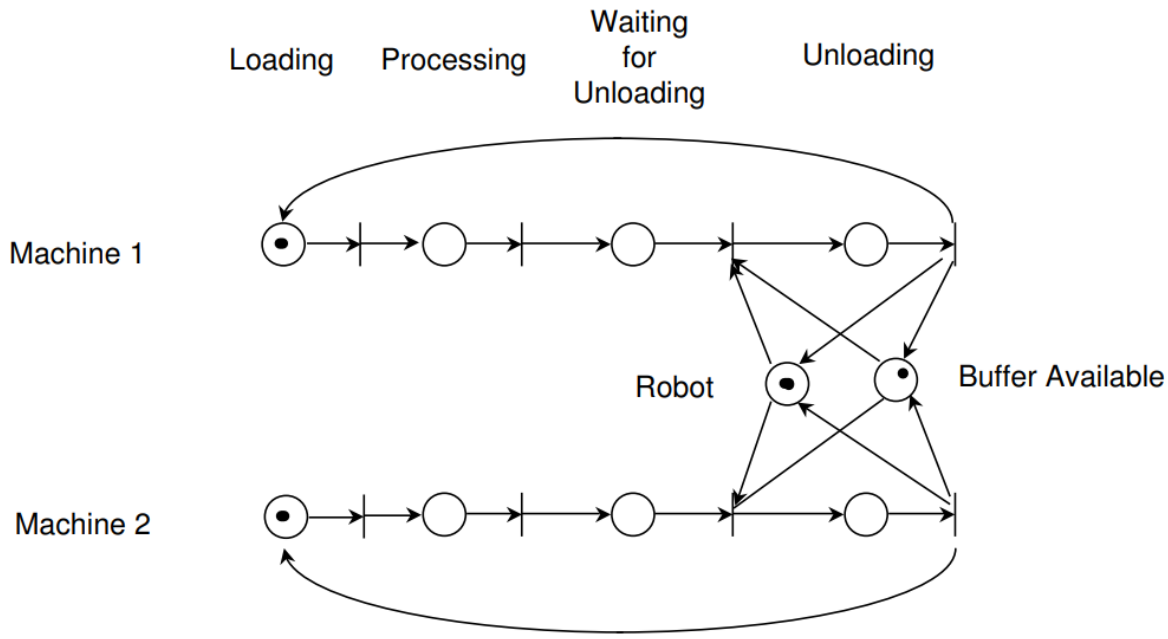


Communication (Sequential calling)

$Program2 \rightarrow Program1 \rightarrow Program2$



Model


Loading
Processing
Waiting for Unloading
Unloading
Machine 1
Robot
Buffer Available
Machine 2