2 Automaten und Logiken auf endlichen Wörtern

2.1 Monadische Logik zweiter Stufe auf endlichen Wörtern

Definition 5
Sei ein Alphabet $\Sigma = \{a_1, \ldots, a_n\}$ fest vorgegeben. Definiere eine *Signatur* $\tau = (<, P_{a_1}, \ldots, P_{a_n})$, wobei $<$ ein 2-stelliges Relationensymbol und die P_{a_i} jeweils einstellige Relationensymbole sind.

Ein Wort $w \in \Sigma^*$ kann als endliche τ-Struktur betrachtet werden. Diese hat als Universum die Menge $\{0, \ldots, |w| - 1\}$ der Positionen in w, $<$ wird als übliche totale Ordnung auf diesem Universum interpretiert, und P_{a_i} wird interpretiert durch die Menge aller Positionen, in denen das Symbol a_i steht.

Im folgenden werden wir jedoch nur noch nicht-leere Wörter betrachten. Dies macht jedoch keinen prinzipiellen Unterschied, da wir an einer logischen Charakterisierung regulärer Sprachen interessiert sind, und $L \subseteq \Sigma^*$ regulär ist, gdw. $L \setminus \{\epsilon\}$ regulär ist.

Beispiel 2
Sei $\Sigma = \{a, b, c\}$ und $w = aababaa$. Dies induziert die τ-Struktur mit Universum $\{0, 1, 2, 3, 4, 5, 6\}$, wobei

- $0 < 1 < 2 < 3 < 4 < 5 < 6$,
- $P_a = \{0, 1, 3, 5, 6\}$, $P_b = \{2, 4\}$, $P_c = \emptyset$.

Definition 6
Seien zwei abzählbar unendliche Mengen von *erststufigen Variablen* $V_1 = \{x, y, \ldots\}$ und *zweitstufigen Variablen* $V_2 = \{X, Y, \ldots\}$ gegeben. Formeln der *Monadischen Logik zweiter Stufe* (MSO[$<$]) über V_1, V_2 und einem Alphabet Σ sind gegeben durch folgende Grammatik.

$$\varphi \quad ::= \quad x = y \mid x < y \mid P_a(x) \mid X(x) \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists x.\varphi \mid \exists X.\varphi$$

wobei $a \in \Sigma$, $x, y \in V_1$ und $X \in V_2$.

Wir benutzen die üblichen Abkürzungen $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi := \neg\varphi \vee \psi$, $\forall x.\varphi := \neg\exists x.\neg\varphi$, $\forall X.\varphi := \neg\exists X.\neg\varphi$, $x \leq y := x = y \vee x < y$, etc.

Ein Vorkommen einer Variablen x (oder X) heißt *gebunden*, wenn im Syntaxbaum über ihm der Operator $\exists x$ vorkommt. Ansonsten heißt dieses Vorkommen *frei*. Wir schreiben auch $\varphi(X_1, \ldots, X_n, x_1, \ldots, x_m)$ um anzudeuten, dass die freien Variablen in φ

zu der Menge $\{X_1, \ldots, X_n, x_1, \ldots, x_m\}$ gehören. Eine Formel ohne freie Variablen wird auch *Satz* genannt.

Erststufige Variablen aus V_1 stehen für Positionen in einem Wort, zweitstufige Variablen aus v_2 für Mengen von Positionen in einem Wort. *Erststufige Logik* besteht aus allen MSO[<]-Formeln, in denen kein zweitstufiger Quantor verwendet wird.

Definition 7
Wir definieren die Semantik einer MSO[<]-Formel als eine Relation $\models$ zwischen Wörtern und Formeln, $w \models \varphi$ ist dabei zu lesen als "das Wort w erfüllt die Formel φ". Diese lässt sich leicht induktiv definieren. Dabei treten jedoch Formeln mit freien Variablen auf. Um diesen eine Bedeutung zukommen zu lassen, verwenden wir eine Abbildung I, die erststufigen Variablen Positionen und zweitstufige Variablen Mengen von Positionen zuordnet.

$$
\begin{array}{rll}
w \models_I x = y & \text{gdw.} & I(x) = I(y) \\
w \models_I x < y & \text{gdw.} & I(x) < I(y) \\
w \models_I P_a(x) & \text{gdw.} & w(I(x)) = a \\
w \models_I X(x) & \text{gdw.} & I(x) \in I(X) \\
w \models_I \varphi \vee \psi & \text{gdw.} & w \models_I \varphi \text{ oder } w \models_I \psi \\
w \models_I \neg\varphi & \text{gdw.} & w \not\models_I \varphi \\
w \models_I \exists x.\varphi & \text{gdw.} & \text{es gibt ein } i \in \{0, \ldots, |w|-1\} w \models_{I[x \mapsto i]} \varphi \\
w \models_I \exists X.\varphi & \text{gdw.} & \text{es gibt ein } M \subseteq \{0, \ldots, |w|-1\} w \models_{I[X \mapsto M]} \varphi
\end{array}
$$

wobei $I[x \mapsto i]$ diejenige Abbildung ist, die x auf i abbildet und sich wie I auf allen anderen Argumenten verhält.

Zwei Formeln sind Defäquivalent, $\varphi \equiv \psi$, falls für alle $w \in \Sigma^+$ und alle I gilt: $w \models_I \varphi$ gdw. $w \models_I \psi$.

Seinen $\mathcal{L}_1$ und $\mathcal{L}_2$ zwei Logiken, die über derselben Klasse von Strukturen interpretiert werden. $\mathcal{L}_2$ heißt *ausdrucksstärker* als $\mathcal{L}_1$, geschrieben $\mathcal{L}_1 \leq \mathcal{L}_2$, falls es für alle $\varphi \in \mathcal{L}_1$ ein $\psi \in \mathcal{L}_2$ gibt, so dass $\varphi \equiv \psi$ gilt. Beide Logiken sind *gleich ausdrucksstark*, $\mathcal{L}_1 \equiv \mathcal{L}_2$, falls $\mathcal{L}_1 \leq \mathcal{L}_2$ und $\mathcal{L}_2 \leq \mathcal{L}_1$ gilt. Offensichtlich ist z.B. FO[<] $\leq$ MSO[<].

Ein Wort w mit solch einer Abbildung I ist eine *Interpretation* für eine Formel. Eine Interpretation, die eine Formel erfüllt, wird auch als *Modell* einer solchen bezeichnet. Oft bietet es sich an, die Interpretation explizit aufzulistet anstatt sie in einer Funktion zu verstecken. Sei $\varphi(X_1, \ldots, X_n, x_1, \ldots, x_m)$ eine MSO[<]-Formel mit freien zweitstufigen Variablen $X_1, \ldots, X_n$ und freien erststufigen Variablen $x_1, \ldots, x_m$. Eine Interpretationsfunktion I für solche eine Formel kann in natürlicher Weise auch als Liste $M_1, \ldots, M_n, i_1, \ldots, i_m$ dargestellt werden. Wir schreiben also auch

$$w, M_1, \ldots, M_n, i_1, \ldots, i_m \models \varphi(X_1, \ldots, X_n, x_1, \ldots, x_m)$$

statt $w \models_I \varphi$, wobei $I(X_j) := M_j$ für $j = 1, \ldots, n$ und $I(x_j) = i_j$ für $j = 1, \ldots, m$.

Beachte: Die Semantik eines Satzes φ ist unabhängig von der Interpretationsfunktion I (Koinzidenz). Daher schreiben wir in solch einem Fall auch einfach $w \models \varphi$.

Beispiel 3
Sei $\Sigma = \{a, b\}$. Betrachte die folgenden drei MSO[<]-Formeln

$$\begin{aligned} \varphi_1(X_1, x_1) &:= P_a(x_1) \wedge \forall y.\big(x_1 < y \wedge X_1(y) \rightarrow P_b(y)\big) \\ \varphi_2(x_1, x_2) &:= \exists z.x_1 < z \wedge z < x_2 \wedge P_a(z) \\ \varphi_3 &:= \exists x.\exists y.P_a(x) \wedge P_b(y) \wedge x < y \wedge \neg\exists z.x < z \wedge z < y \end{aligned}$$

Sei nun $w = abbab$. Es gelten

$$\begin{aligned} w, \{2,3,4\}, 2 &\not\models \varphi_1(X_1, x_1) \\ w, 2, 4 &\models \varphi_2(x_1, x_2) \\ w &\models \varphi_3 \end{aligned}$$

Lemma 2
Die folgenden Formeln lassen sich in MSO[<] definieren.

a) $min(x), max(x)$ mit der Bedutung, dass x der Wortanfang bzw. das -ende ist,

b) $succ(x, y)$ mit der Bedeutung, dass y unmittelbar auf x folgt,

c) $pred(x, y)$ mit der Bedeutung, dass y unmittelbar vor x vorkommt.

Beweis Übung. ■

Die folgende Definition schlägt den Bogen zwischen der Logik MSO[<] und der Theorie formaler Sprachen.

Definition 8
Sei φ ein MSO[<]-Satz. Definiere $L(\varphi) := \{w \in \Sigma^+ \mid w \models \varphi\}$.
Eine Sprache $L \subseteq \Sigma^+$ heißt *MSO[<]-definierbar*, falls es einen MSO[<]-Satz φ gibt mit $L = L(\varphi)$. FO[<]-Definierbarkeit wird analog erklärt.

Beispiel 4
1. $L(\Sigma^* ab \Sigma^*)$ ist MSO[<]-definierbar, siehe φ_3 aus Beispiel 3.
2. $L = \{w \mid |w| \text{ is ungerade }\}$ ist MSO[<]-definierbar:
$$\exists X.X(min) \wedge \big(\forall x.\forall y.succ(x, y) \rightarrow (X(x) \leftrightarrow \neg X(y))\big) \wedge X(max)$$
wobei $\psi(min) := \exists z.min(z) \wedge \psi(z)$, etc.
3. $L = \{w \in \{a, b, c\}^+ \mid \text{in } w \text{ folgen auf jedes } a \text{ nur } a\text{'s bis irgendwann ein } b \text{ auftritt }\}$:
$$\forall x.P_a(x) \rightarrow \exists y.P_b(y) \wedge x < y \wedge \forall z.x < z \wedge z < y \rightarrow P_a(z)$$

Sei MSO[*succ*] definiert wie MSO[<] mit dem Unterschied, dass statt atomaren Formeln der Form $x < y$ nur atomare Formeln der Form $succ(x, y)$ benutzt werden dürfen.

Lemma 3
MSO[succ] ≡ MSO[<].

Beweis Richtung $\leq$ ist eine direkte Konsequenz aus Lemma 2. Für die Richtung $\geq$ reicht es aus zu zeigen, dass es eine MSO[*succ*]-Formel $\varphi(x, y)$ gibt, so dass $\varphi(x, y) \equiv x < y$ gilt.

$$\varphi(x,y) \;:=\; \exists X.\neg X(x) \wedge \big(\exists z.succ(x,z) \wedge X(z)\big) \wedge \big(\forall z.\forall v.X(z) \wedge succ(z,v) \rightarrow X(v)\big) \wedge X(y)$$

Da die Übersetzungen zwischen den beiden Logiken die Formeln auch nur höchstens linear vergrößern, unterscheiden wir im folgenden nicht mehr explizit zwischen MSO[<] und MSO[*succ*] und schreiben auch nur MSO. Beachte, dass Lemma 3 sich nicht auf FO[<] und FO[*succ*] übertragen lässt.

Definition 9
Die Logik MSO_0 über einem Alphabet Σ und einer Menge $V = \{X, Y, \ldots\}$ von zweitstufigen Variablen ist gegeben durch die folgende Grammatik.

$$\varphi \;::=\; X \subseteq Y \mid Sing(X) \mid Succ(X,Y) \mid X < Y \mid X \subseteq P_a \mid \varphi \vee \varphi \mid \neg\varphi \mid \exists X.\varphi$$

wobei $X, Y \in V$, $a \in \Sigma$. Wie bei MSO definieren wir die üblichen Abkürzungen $\wedge$, $\forall$, etc.

Die Semantik von MSO_0 wird durch eine Übersetzung nach MSO eindeutig festgelegt.

$$\begin{aligned}
Sing(X) &:= \exists x.X(x) \wedge \forall x.\forall y.X(x) \wedge X(y) \rightarrow x = y \\
X \subseteq Y &:= \forall x.X(x) \rightarrow Y(x) \\
Succ(X,Y) &:= Sing(X) \wedge Sing(Y) \wedge \exists x.\exists y.X(x) \wedge Y(y) \wedge succ(x,y) \\
X < Y &:= Sing(X) \wedge Sing(Y) \wedge \exists x.\exists y.X(x) \wedge Y(y) \wedge x < y \\
X \subseteq P_a &:= \forall x.X(x) \rightarrow P_a(x)
\end{aligned}$$

Die übrigen Fälle übersetzen sich in trivialer Weise.

Lemma 4
$MSO_0 \equiv MSO.$

Beweis Die Richtung $\leq$ ist aufgrund der Semantik von MSO_0 bereits gegeben. Es bleibt noch zu zeigen, dass sich jede MSO-Formel äquivalent in eine MSO_0-Formel übersetzen läßt. Um dies induktiv durchführen zu können, muss dies wiederum auch für Formeln mit freien Variablen funktionieren. Da es in MSO_0 jedoch nur noch zweitstufige Variablen gibt, vereinbaren wir folgende Spezifikation für die Übersetzung einer MSO-Formel φ

in eine MSO_0-Formel φ^*. Für alle $w \in \Sigma^+$, alle $M_i \subseteq \{0, \ldots, |w|-1\}$ und alle $i \in \{0, \ldots, |w|-1\}$ soll gelten:

$$\begin{aligned} w, M_1, \ldots, M_n, i_1, \ldots, i_m &\models \varphi(X_1, \ldots, X_n, y_1, \ldots, y_m) \quad \text{gdw.} \\ w, M_1, \ldots, M_n, \{i_1\}, \ldots, \{i_m\} &\models \varphi^*(X_1, \ldots, X_n, Y_1, \ldots, Y_m) \end{aligned}$$

Die atomaren Fälle sind:

$$\begin{aligned} (y_1 < y_2)^* &:= Y_1 < Y_2 \\ (P_a(y))^* &:= \mathit{Sing}(Y) \wedge Y \subseteq P_a \\ (\mathit{succ}(y_1, y_2))^* &:= \mathit{Succ}(Y_1, Y_2) \\ (X(y))^* &:= \mathit{Sing}(Y) \wedge Y \subseteq X \end{aligned}$$

Die übrigen Fälle sind offensichtlich. ■

Im folgenden werden wir MSO_0-Formeln φ rekursiv in NFAs $\mathcal{A}_\varphi$ übersetzen, so dass $L(\mathcal{A}_\varphi) = L(\varphi)$ gilt. Dabei treten jedoch in Zwischenschritten Formeln mit freien Variablen auf, deren Modelle Wörter mit Interpretationsfunktion sind. Diese lassen sich jedoch als Wörter über einem erweiterten Alphabet auffassen.

Sei $\varphi(X_1, \ldots, X_n)$ eine Formel mit lediglich freien zweitstufigen Variablen $X_1, \ldots, X_n$. Eine Interpretation $w, M_1, \ldots, M_n$ induziert in natürlicher Weise ein Wort über dem Alphabet $\Sigma \times \{0,1\}^n$. Sei $w = a_1 \ldots a_m$. Das induzierte Wort ist dann

$$\begin{pmatrix} a_1 \\ c_{11} \\ \vdots \\ c_{1n} \end{pmatrix} \begin{pmatrix} a_2 \\ c_{21} \\ \vdots \\ c_{2n} \end{pmatrix} \ldots \begin{pmatrix} a_m \\ c_{m1} \\ \vdots \\ c_{mn} \end{pmatrix} \qquad \text{wobei } c_{ij} = \begin{cases} 1, & \text{falls } i \in M_j \\ 0, & \text{sonst} \end{cases}$$

Beispiel 5
Sei $\Sigma = \{a, b\}$, $w = abbab$, $M_1 = \emptyset$, $M_2 = \{1, 3, 4\}$ und $M_3 = \{0, 1\}$. Dies induziert das Wort

$$\begin{pmatrix} a \\ 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} b \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} a \\ 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} b \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Definition 10
Sei $w \in (\Sigma \times \{0,1\}^n)^+$ für ein $n \geq 1$. Das Wort $\hat{w}$ entstehe aus w durch Streichung der n-ten Spur in w. Außerdem sei $\hat{L} := \{\hat{w} \mid w \in L\}$ für ein beliebiges $L \subseteq (\Sigma \times \{0,1\}^n)^+$.

Lemma 5
Für alle $L \subseteq (\Sigma \times \{0,1\}^n)^+$, $n \geq 1$ und alle $\varphi(X_1, \ldots, X_n) \in \mathit{MSO}$ gilt: wenn $L = L(\varphi(X_1, \ldots, X_n))$ dann ist $\hat{L} = L(\exists X_n.\varphi(X_1, \ldots, X_n))$.

Beweis Übung. ■

Beachte, dass sich Lemma 5 natürlich auch auf Formeln der Form $\exists X_i.\varphi(X_1, \ldots, X_n)$ für beliebiges $i \in \{1, \ldots, n\}$ durch eventuelles Umnummerieren der freien Variablen erweitern läßt.

Satz 5 (Büchi-Elgot, '60)
Eine Sprache $L \subseteq \Sigma^+$ ist regulär gdw. sie MSO-definierbar ist.

BEWEIS "$\Leftarrow$" Angenommen L ist MSO-definierbar, d.h. es gibt einen MSO-Satz φ, sodass $L(\varphi) = L$ gilt. Laut Lemma 4 existiert dann auch ein MSO_0-Satz φ^* mit $L(\varphi^*) = L$.

Wir zeigen, dass sich zu jeder MSO_0-Formel $\varphi(X_1, \ldots, X_n)$ ein NFA $\mathcal{A}_\varphi$ konstruieren läßt, so dass $L(\varphi) = L(\mathcal{A}_\varphi)$ über dem Alphabet $\Sigma' := \Sigma \times \{0,1\}^n$ gilt. Es sollte klar sein, dass dies die zu beweisende Aussage impliziert.

Die Induktionsanfänge sind:

- $\varphi = X_i \subseteq X_j$. Definiere $\mathcal{A}_\varphi := (\{q\}, \Sigma', q, \delta, \{q\})$, wobei

$$\delta(q, (a, b_1, \ldots, b_n)) := \begin{cases} \{q\} & , \text{ falls } b_i \leq b_j, \\ \emptyset & , \text{ sonst} \end{cases}$$

- $\varphi = \mathit{Sing}(X_i)$: Definiere $\mathcal{A}_\varphi := (\{q_0, q_1\}, \Sigma', q_0, \delta, \{q_1\})$, wobei

$$\delta(q_0, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_0\} & , \text{ falls } b_i = 0 \\ \{q_1\} & , \text{ sonst} \end{cases}$$
$$\delta(q_1, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_1\} & , \text{ falls } b_i = 0 \\ \emptyset & , \text{ sonst} \end{cases}$$

- $\varphi = \mathit{Succ}(X_i, X_j)$: Definiere $\mathcal{A}_\varphi := (\{q_0, q_1, q_2\}, \Sigma', q_0, \delta, \{q_2\})$, wobei

$$\delta(q_0, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_0\} & , \text{ falls } b_i = 0 \text{ und } b_j = 0 \\ \{q_1\} & , \text{ falls } b_i = 1 \text{ und } b_j = 0 \\ \emptyset & , \text{ sonst} \end{cases}$$
$$\delta(q_1, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_2\} & , \text{ falls } b_i = 0 \text{ und } b_j = 1 \\ \emptyset & , \text{ sonst} \end{cases}$$
$$\delta(q_1, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_2\} & , \text{ falls } b_i = 0 \text{ und } b_j = 0 \\ \emptyset & , \text{ sonst} \end{cases}$$

- $\varphi = X_i < X_j$: Definiere $\mathcal{A}_\varphi := (\{q_0, q_1, q_2\}, \Sigma', q_0, \delta, \{q_2\})$, wobei

$$\delta(q_0, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_0\} & , \text{ falls } b_i = 0 \text{ und } b_j = 0 \\ \{q_1\} & , \text{ falls } b_i = 1 \text{ und } b_j = 0 \\ \emptyset & , \text{ sonst} \end{cases}$$

$$\delta(q_1, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_1\} & , \text{ falls } b_i = 0 \text{ und } b_j = 0 \\ \{q_2\} & , \text{ falls } b_i = 0 \text{ und } b_j = 1 \\ \emptyset & , \text{ sonst} \end{cases}$$

$$\delta(q_1, (a, b_1, \ldots, b_n)) := \begin{cases} \{q_2\} & , \text{ falls } b_i = 0 \text{ und } b_j = 0 \\ \emptyset & , \text{ sonst} \end{cases}$$

- $\varphi = X_i \subseteq P_a$: $\mathcal{A}_\varphi := (\{q\}, \Sigma', q, \delta, \{q\})$, wobei

$$\delta(q, (c, b_1, \ldots, b_n)) := \begin{cases} \{q\} & , \text{ falls } b_i \neq 1 \text{ oder } c = a \\ \emptyset & , \text{ sonst} \end{cases}$$

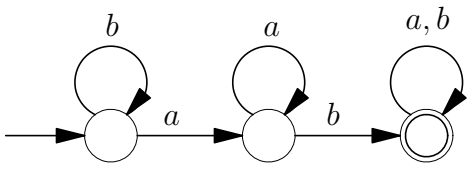
Induktionsschritte:

- $\varphi = \psi_1 \vee \psi_2$. Durch disjunkte Vereinigung der beiden Automaten $\mathcal{A}_{\psi_1}$ und $\mathcal{A}_{\psi_2}$ läßt sich leicht der Automat $\mathcal{A}_\varphi$ bauen.
- $\varphi = \neg\psi$. Per Potenzmengenkonstruktion kann man $\mathcal{A}_\psi$ in einen DFA umbauen und diesen dann komplementieren.
- $\varphi = \exists X_i.\psi(X_1, \ldots, X_n)$. Durch Streichen der i-ten Komponenten in jeder Transition in $\mathcal{A}_\psi$ erhält man laut Lemma 5 und der Induktionshypothese einen NFA $\mathcal{A}_\varphi$ mit $L(\mathcal{A}_\varphi) = L(\exists X_i.\psi)$.

"$\Rightarrow$" Sei L regulär. Laut Satz 2 existiert dann ein NFA $\mathcal{A}$ mit $L(\mathcal{A}) = L$. Wir konstruieren nun einen MSO-Satz $\varphi_\mathcal{A}$, der die Existenz eines akzeptierenden Laufs von $\mathcal{A}$ auf einem vorgelegten Wort w beschreibt.

Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, und o.B.d.A. nehmen wir an, dass $Q = \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$ und $q_0 = 0$ gilt. Beachte: Ein Lauf von $\mathcal{A}$ auf einem $w \in \Sigma^+$ läßt sich als Wort in $\{0,1\}^n$ (Indikatorfunktion) modellieren. Dabei gibt der j-te Eintrag in der i-ten Spur an, ob sich der Automat *nach* Lesen des j-ten Symbols im Zustand i befindet. Betrachte dazu folgenden NFA $\mathcal{A}$.

Sei $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, und o.B.d.A. nehmen wir an, dass $Q = \{0, \ldots, n\}$ für ein $n \in \mathbb{N}$ und $q_0 = 0$ gilt. Beachte: Ein Lauf von $\mathcal{A}$ auf einem $w \in \Sigma^+$ läßt sich als Wort in $\{0,1\}^n$ modellieren. Dabei gibt der j-te Eintrag in der i-ten Spur an, ob sich der Automat *nach* Lesen des j-ten Symbols im Zustand i befindet. Betrachte dazu folgenden NFA $\mathcal{A}$.



Ein akzeptierender Lauf auf $w = bbbabb$ ist z.B.

	b	b	b	a	b	b	$\leftarrow$	w
1	1	1	1	0	0	0	$\leftarrow$	$\mathcal{A}$ ist in Zustand 1
0	0	0	0	1	0	0	$\leftarrow$	$\mathcal{A}$ ist in Zustand 2
0	0	0	0	0	1	1	$\leftarrow$	$\mathcal{A}$ ist in Zustand 3

Beachte: Ein Lauf auf einem Wort der Länge $|w|$ ist ein Wort der Länge $|w| + 1$. Dieses läßt sich zwar generell durch existenzielle Quantifizierung erraten und dann in MSO als korrekt verifizieren (Start im Anfangszustand, Ende in einem Endzustand, Schritte nur entlang der Transitionstabelle des Automaten, etc.). Man hat jedoch sozusagen nur den Platz $|w|$ zur Verfügung. Daher lassen wir im allgemeinen $\varphi_{\mathcal{A}}$ nicht ausdrücken, dass dieser Lauf in dem Startzustand, sondern in einem Nachfolger dessen beginnt, der über das erste Symbol des Wortes erreicht wird.

$$\begin{aligned}\varphi_{\mathcal{A}} \;:=\; \exists X_1 \ldots \exists X_n. \Big(\forall x. \bigvee_{i=0}^{n} X_i(x) \wedge \bigwedge_{j \neq i} \neg X_j(x) \Big) \;\wedge \\ \Big(\bigwedge_{a \in \Sigma} P_a(min) \rightarrow \bigvee_{i \in \delta(0,a)} X_i(min) \Big) \;\wedge\; \bigvee_{i \in F} X_i(max) \;\wedge \\ \forall x. \forall y. succ(x,y) \rightarrow \Big(\bigwedge_{(i,a) \in Q \times \Sigma} X_i(x) \wedge P_a(y) \;\rightarrow \bigvee_{j \in \delta(i,a)} X_j(y) \Big)\end{aligned}$$

$\varphi_{\mathcal{A}}$ besagt: Es gibt einen als Wort über $\{0,1\}^n$ kodierten Lauf. In diesem ist $\mathcal{A}$ zu jedem Zeitpunkt in genau einem Zustand. Der Lauf beginnt in einem a-Nachfolger des Anfangszustands, falls a der erste Buchstabe des zugrundeliegenden Wortes ist. Er endet in einem Endzustand, und in jedem Schritt befolgt er die Transitionsrelation. ■

Definition 11
Sei EMSO die Menge aller MSO-Formeln der Form $\exists X_1 \ldots \exists X_n.\psi$, so dass ψ eine FO-Formel ist. Das Fragment AMSO von MSO wird analog mit $\forall$ statt $\exists$ definiert.

Korollar 6
EMSO $\equiv$ *MSO* $\equiv$ *AMSO.*

Beweis "EMSO $\equiv$ MSO" Die Richtung $\leq$ ist klar. Für die andere Richtung nehme eine MSO-Formel φ und übersetze diese in einen NFA $\mathcal{A}_\varphi$ laut Satz 5. Übersetze diesen wiederum zurück in eine MSO-Formel φ'. Es gilt $\varphi' \equiv \varphi$ und $\varphi' \in$ EMSO.

"AMSO $\equiv$ MSO" Übung. ■

Korollar 7
Erfüllbarkeit und Äquivalenz von MSO-Formeln über endlichen Wortmodellen ist entscheidbar.

Beweis (1) Sei φ eine MSO-Formel. Diese läßt sich laut Satz 5 effektiv in einen NFA $\mathcal{A}_\varphi$ übersetzen. Dann gilt: φ ist erfüllbar gdw. $L(\mathcal{A}_\varphi) \neq \emptyset$. Letzteres ist entscheidbar.

(2) Zwei MSO-Formeln φ und ψ sind äquivalent gdw. $\models \varphi \leftrightarrow \psi$ gdw. $\neg(\varphi \leftrightarrow \psi)$ unerfüllbar ist. Damit ist Äquivalenz auf Erfüllbarkeit reduziert worden. ∎

Zuletzt analysieren wir noch die worst-case-Komplexität der Übersetzung einer MSO-Formel in einen NFA. Seien $\mathcal{A}_1$ und $\mathcal{A}_2$ zwei NFAs mit jeweils höchstens n Zuständen. Dann hat ein NFA

- für die Vereinigung höchstens $2n + 1$,
- für das Komplement höchstens 2^n,
- für die existenzielle Quantifizierung höchstens n

Zustände. Damit kann eine Formel φ mit k logischen Operatoren zu einem NFA $\mathcal{A}$ mit $2^c_{O(k)}$ vielen Zuständen für ein $c \in \mathbb{N}$ führen, wobei

$$2^c_0 := c \qquad\qquad 2^c_{k+1} := 2^{2^c_k}$$

2.2 MONA: eine Implementierung der wMSO

In diesem Abschnitt lernen wir die praktisch einsetzbare Implementierung *MONA* (www.brics.dk/mona) der wMSO kennen. MONA wurde am BRICS, Aarhus, von Nils Klarlund, Anders Møller und Michael Schwartzbach entwickelt und u.a., von David Basin, ETH Zürich, in Industrieprojekten benutzt.

Die Logik wMSO wird von MONA leicht anders interpretiert, als in der Vorlesung angegeben: erststufige Variablen werden immer über den gesamten natürlichen Zahlen $\mathbb{N}$ interpretiert, Prädikatvariablen werden als endliche Teilmengen von $\mathbb{N}$ interpretiert. Hat man eine Formel mit freien Prädikatvariablen $P_1, \ldots, P_n$, so definiert ein Wort w über $\Sigma = 2^{\{1,\ldots,n\}}$ wie folgt eine Belegung der Prädikatvariablen: $P_i = \{t \mid t < |w| \wedge i \in w_t\}$.

Das Werkzeug MONA

- liest eine wMSO Formel,
- konstruiert den dazugehörigen Automaten,
- entscheidet mit dessen Hilfe, ob die Formel gültig, unerfüllbar, oder keines von beiden ist.
  - Falls erfüllbar, so wird eine erfüllende Interpretation minimaler Größse angegeben.
  - Falls nicht gültig, so wird eine falsifizierende Interpretation minimaler Größe angegeben ("Gegenbeispiel").
- Falls gewünscht, wird auch der Automat als Zustandstabelle angegeben.

2.2.1 Einfaches Beispiel

Sei `even.mona` eine Datei des folgenden Inhalts:

```
var2 A;
var1 maxi;
maxi = 9;
0 notin A &
all1 i: i < maxi => (i+1 in A <=> i in A)
```

Mit dem Befehl `mona even.mona` erhält man folgende erfüllende Interpretation:

$$A \hat{=} 0101010101$$
$$A = \{1, 3, 5, 7, 9\}$$
$$maxi = 9$$

Mit der Option `-w` erhält man auch den Automaten.

2.2.2 Binärzähler

Als nächstes wollen wir mit vier Mengenvariablen bis 16 zählen:

```
var2 A, B, C, D;
var1 maxi;

0 notin A & 0 notin B & 0 notin C & 0 notin D  &
maxi in A & maxi in B & maxi in C & maxi in D  &
all1 i: i < maxi =>
    (i+1 in A <=> i notin A)
  & ((i+1 in B <=> i notin B) <=> (i+1 notin A & i in A))
  & ((i+1 in C <=> i notin C) <=> (i+1 notin B & i in B))
  & ((i+1 in D <=> i notin D) <=> (i+1 notin C & i in C)))
;
```

An der Stelle 0 ist $A = B = C = D = 0$, an der Stelle *maxi* ist $A = B = C = D = 1$. Dazwischen schreiten A, B, C, D wie ein Binärzähler fort. Somit muss *maxi* mindestens 15 sein.

Wir können also mit k Mengenvariablen ein Modell der Größe 2^k erzwingen. Durch Verfeinerung dieser Methode kann man noch viel größere Modelle erzwingen, aber dazu später mehr.

2.2.3 Spezifikation der Addition von Binärzahlen beliebiger Größe

Als "praktisches" Beispiel wollen wir jetzt einen Addierer verifizieren. Wir geben uns eine Wortlänge $ vor:

```
var1 $;
```

Hier ist $ ein ganz normaler Bezeichner.

Die folgenden beiden Direktiven relativieren freie und gebundene Variablen auf den Bereich 0..$:

```
defaultwhere1(p) = p <= $;
defaultwhere2(P) = P sub {0,...,$};
```

Ab jetzt bedeutet also zum Beispiel `all1 i: ...` in Wirklichkeit `all1 i: i <= $ => ...`. Wir definieren folgende Hilfsprädikate:

```
pred at_least_two(var0 A,var0 B,var0 C) =
       (A & B) | (A & C) | (B & C);
pred mod_two(var0 A,var0 B,var0 C,var0 @d) = (A <=> B <=> C <=> @d);
```

Das erste drückt aus, dass mindestens zwei von drei Boole'schen Variablen wahr sind. Die zweite drückt aus, dass *@d* die Summe modulo 2 von A, B, C ist.

Das nun folgende Prädikat drückt aus, dass *Result* die Summe von X und Y ist. Dabei werden *Result*, X, Y jeweils als Binärzahlen aufgefasst. *Cin* ist ein initialer Übertrag; *Cout* ist der resultierende Übertrag:

```
pred add(var2 X, var2 Y, var2 Result, var0 Cin, var0 Cout) =
ex2 C: (0 in C <=> Cin)
      & (all1 p:
           mod_two(p in X, p in Y, p in C, p in Result)
      & (p < $ =>  ((p+1 in C)
                  <=> at_least_two(p in X, p in Y, p in C))))
      & (Cout <=> at_least_two($ in X, $ in Y, $ in C));
```

Die existentiell quantifizierte Variable C bezeichnet die Übertragsbits. Generell bietet sich existentielle Quantifizierung für die Modellierung interner Leitungen an.

Verhältnis der wMSO zu QBF: wMSO erweitert quantifizierte Boole'sche Formeln um eine weitere "unendliche" Dimension: eine zweitstufige Variable bezeichnet gleich einen ganzen Vektor Boole'scher Variablen beliebiger Länge. Im Beispiel nutzen wir diese zusätzliche Dimension, um parametrisierte Schaltkreise zu modellieren (Addieren von $-bits statt nur 3 oder 5 bits). Alternativ oder manchmal sogar gleichzeitig lässt sich die unendliche Dimension zur Modellierung der Zeit einsetzen.

2.2.4 Synchronaddierer

Als nächstes definieren wir einen "effizienteren" Addierer, in dem der Übertrag separat vorab berechnet wird. Dadurch muss nicht auf das Ergebnis des jeweils vorherigen Ergebnisbits gewartet werden.

```
pred sync_add(var2 X, var2 Y, var2 Result, var0 Cin, Cout) =
   ex2 C:
      (all1 i:i in C <=> i=0 & Cin |
           ex1 j: j<i & (all1 k:j<k&k<i=> k in X | k in Y) &
                at_least_two(j in X, j in Y, j=0&Cin)) &
      (all1 p: mod_two(p in X, p in Y, p in C, p in Result))      &
      (Cout <=> at_least_two($ in X, $ in Y, $ in C));
```

Das Übertragsbit ist an Position i gesetzt, wenn entweder $i = 0$ und *Cin* gesetzt ist, oder aber an einer früheren Position ein Übertrag aufgetreten ist und zwischen i und j nicht wieder verschluckt wurde.

Wir können nun formulieren, dass beide Versionen äquivalent sind:

```
all2 X,Y,Z: all0 Cin, Cout:
  add(X,Y,Z,Cin,Cout) <=> sync_add(X,Y,Z,Cin,Cout);
```

Das ist in der Tat der Fall, wie die folgende Mona-Ausgabe zeigt.

```
Automaton has 3 states and 3 BDD-nodes
ANALYSIS
Formula is valid
```

Als Gegenprobe ändern wir $j < i$ zu $j \leq i$ um: die Formel wird unerfüllbar. Wir können auch konkret Ergebnisse ausrechnen:

```
var2 X, Y, Z;
var0 Cout;
$ = 5;
X = {2,3}; # X=12
Y = {0,2,3,5}; # Y=35
sync_add(X,Y,Z,false,Cout);
```

Ein (das) erfüllende(s) Modell setzt Z auf $\{0, 3, 4, 5\}$ und $Cout = \mathit{false}$.

2.3 Komplexität des Entscheidungsproblems für wMSO

Sei 2_n definiert durch $2_0 = 1$ und $2_{n+1} = 2^{2_n}$. Wir haben bereits einen Algorithmus kennengelernt, der zu einer gegebenen wMSO Formel der Länge n einen Automaten der Größe $2_{O(n)}$ bildet und daraus abliest, ob die Formel erfüllbar ist. Somit liegt das Erfüllbarkeitsproblem wMSOSAT für die wMSO in der Komplexitätsklasse $\mathit{DTIME}(2_{O(n)})$. Wir wollen jetzt zeigen, dass wMSOSAT sogar vollständig für diese Klasse ist. Das bedeutet, dass jedes Problem $P \in \mathit{DTIME}(2_{O(n)})$ polynomial auf wMSOSAT reduziert werden kann. Sei also T eine Turingmaschine, die solch ein Problem in Zeit 2_{cn} entscheidet. Zu gegebenem Input w werden wir in polynomialer Zeit eine wMSO-Formel ϕ_w konstruieren, derart dass ϕ_w erfüllbar ist, gdw. T die Eingabe w akzeptiert, also w in P ist.

Maschine T akzeptiert w, genau dann wenn es ein Wort

$$\tilde{w} = w_0 \# w_1 \# w_2 \# w_3 \# w_4 \# w_5 \# w_6 \# w_7 \# \ldots \# w_N$$

gibt, wobei $N = 2_{c|w|}$, die Raute (#) ein besonderes (Trenn-)Symbol ist, und die w_i, $i = 0, \ldots, N$ globale Konfigurationen (Zustand, Kopfposition und Bandinhalt) kodieren, in einer Weise, dass:

- w_0 die initiale Konfiguration von T mit Eingabe w ist,

- w_{i+1} die Folgekonfiguration von w_i gemäß der Übergangstafel von T ist (i.Z. manchmal $w_i \vdash_T w_{i+1}$)
- w_N eine akzeptierende Konfiguration ist.

Durch Wahl einer geeigneten Kodierung und ggf. Erhöhen von c können wir annehmen, dass alle w_i die Länge N haben.

Die genannten Eigenschaften an ein solch ein Wort $\tilde{w}$ lassen sich sehr einfach in wMSO spezifizieren, sofern es nur gelingt, ein Prädikat $dist(x, y)$ zu definieren, derart dass $dist(x, y) \iff y - x = N$. Indem wir die w_i noch weiter künstlich ausstopfen, sehen wir, dass schon $dist(x, y) \iff y - x = N'$ wobei $N' \geq N$ genügen würde.

Ein solches wollen wir jetzt definieren. Natürlich muss die Größe der Formel $dist(x, y)$ polynomiell in n sein, sonst entstünde keine polynomielle Reduktion. Wir dürfen also insbesondere nicht einfach schreiben

$$dist(x, y) \iff y = x + 1 + 1 + 1 + 1 + 1 + \cdots + 1 \ (N + 1 \text{ Summanden})$$

Sei die Funktion F definiert durch $F(0) = 1, F(n + 1) = F(n)2^{F(n)}$.

Satz 8
Zu gegebenem $n \in \mathbb{N}$ lässt sich in polynomieller Zeit eine Formel $dist_n(x, y)$ berechnen, derart dass $dist_n(x, y) \iff y - x = F(n)$. Insbesondere ist die Länge von $dist_n(x, y)$ polynomiell in n.

Beweis Man definiert die Formeln rekursiv über n wie folgt.

Wir setzen $dist_0(x, y) : \iff y = x + 1$. Wegen $F(0) = 1$ leistet dies das Verlangte.

Sei jetzt $dist_n(x, y)$ bereits konstruiert. Um $dist_{n+1}(x, y)$ zu definieren, verlangen wir die Existenz eines Wortes, das zwischen x und y alle Binärzahlen der Länge $dist_n(x, y)$ der Reihe nach hintereinandergeschrieben enthält. Gelingt es, das zu erzwingen, so gilt gerade $y - x = F(n + 1)$ wie verlangt.

Hier ist die Lösung in MONA Notation. Man beachte, dass man *dist* nicht uniform in n definieren kann, es gibt also keine wMSO-Formel $\phi(n, x, y)$ in drei Variablen derart, dass $\phi(n, x, y) \iff dist_n(x, y)$ wäre, aber das wird ja zum Glück auch nicht benötigt.

```
distnpluseins(x,y) =
   ex2 B, C:
   # Abstand mindestens distn.
        (ex1 a,b: distn(a,b) & x<=a & b<=y) &
   # a, b zeigen der Reihe nach auf alle Positionen im Abstand distn
     all1 a, b: distn(a,b) & x<=a & b<=y =>
   # Initialisierung des ersten Blocks: B=10...0, C=00...0
     (a=x => a in B & a notin C &
           all1 c:a<c&c<b => c notin B & c notin C) &
   # B-Besetzung in alle Bloecke kopieren.
     (a in B <=> b in B) &
   # Verlangen, dass bei y wieder ein Block anfaengt, also (b-a) | (y-x)
     (b=y => b in B) &
   # Besetzung des letzten Blocks zu C=11....1
```

```
  (b=y => all1 c: (a<=c & c<b) => c in C) &
# Das erste C-bit jeden Blocks wechselt von Block zu Block
  (b in B  => (a in C <=> b notin C)) &
# Die folgenden C-bits jeden Blocks wechseln, wenn das
#  vorhergehende Bit von Eins auf Null geht.
  (a+1 notin B  =>
    ((b+1 notin C <=> a+1 in C) <=> (b notin C & a in C)));
```

Zu Testzwecken kann man definieren:

```
pred distn(var1 x, y) = y=x+5;
var1 x, y;
distnpluseins(x,y);
```

und erhält als Antwort: $x = 0, y = 160$.

Indem man B, C zu freien Variablen macht, kann man sich deren Wertverlauf mit MONA ansehen. Man kann auch einzelne Konjunkte weglassen, um das Funktionsprinzip von *distnpluseins* zu studieren.

Wir fassen zusammen:

Satz 9 (Meyer, Stockmeyer)
Das Problem wMSOSAT ist vollständig für die Klasse $DTIME(2_{O(n)})$.

Zu dieser Komplexitätsklasse bemerken wir, dass auf dieser Höhe die Unterscheidungen zwischen Platz und Zeit, sowie Determinismus und Nichtdeterminismus verwischen; es ist:

$$DTIME(2_{O(n)}) = NTIME(2_{O(n)}) = NSPACE(2_{O(n)}) = DSPACE(2_{O(n)}).$$

Stockmeyer bewies in seiner Dissertation sogar die folgende sehr konkrete Version:

Satz 10 (Stockmeyer)
Jeder Boole'sche Schaltkreis, der die Erfüllbarkeit von wMSO-Formeln der Länge 613 entscheidet, hat mindestens 10^{128} *Verbindungsleitungen.*

In der Praxis funktioniert MONA auch mit längeren Formeln ganz gut. Das liegt daran, dass viele Formeln, die aus der Praxis kommen, relativ einfach sind. Manchmal aber, speziell wenn man sich vertippt, kann MONA auch in eine de facto Endlosschleife geraten.

2.4 Presburger Arithmetik

Unter der Presburger Arithmetik versteht man die erststufige Logik über den natürlichen Zahlen und der Signatur $0, +, =, <$.

Hier sind drei Formeln der Presburger Arithmetik, die letzte verwendet Abkürzungen.

$\exists x.y = x + x$	"y ist gerade"
$\exists xr.r < 5 \wedge y = x + x + x + x + x + r$	"$y = r \bmod 5$"
$\exists z.\forall x > z.x = 0 \bmod 3 \Rightarrow \exists uv.x = 15u + 27v$	

Es gibt in der Presburger Arithmetik keine Multiplikation und man kann sie auch nicht irgendwie definieren; insbesondere gibt es keine Formel $\phi(x, y)$, derart dass $\phi(x) \iff y = x^2$ wäre.

Die Allgemeingültigkeit und die Erfüllbarkeit von Formeln der Presburger Arithmetik ist durch Übersetzung in wMSO entscheidbar:

Variablen der Presburger Arithmetik werden immer in Mengenvariablen übersetzt: die "Bedeutung" einer solchen Mengenvariablen ist durch die Binärkodierung gegeben. So wird 42 etwa durch $\{5, 3, 1\}$ kodiert.

Die Relation $x = y + z$ lässt sich nun durch eine wMSO Formel beschreiben, wobei man im Prinzip so vorgeht wie bei dem Addierwerk aus Abschnitt 2.2.3. Die Relation $x < y$ ersetzt man durch $\exists z.y = x + z + 1$. Somit wird es möglich, zu jeder Presburger Formel ϕ eine wMSO Formel $\hat{\phi}$ anzugeben, derart dass ϕ erfüllbar ist, gdw. $\hat{\phi}$ erfüllbar ist. Folglich ist die Presburger Arithmetik entscheidbar.

Es gibt andere Entscheidungsverfahren für die Presburger Arithmetik, die auf Quantorenelimination (Verallgemeinerung von Gauss-Elimination) beruhen. Diese liefern insbesondere ein $DTIME(2^{2^{2^{cn}}})$ Verfahren.

Dagegen hat das angegebene Verfahren anscheinend die schlechtere Komplexität $DTIME(2_{cn})$ (vgl. Satz 9). In der Praxis zeigt sich aber, dass Automaten für wMSO Formeln, die aus der Übersetzung von Presburger Formeln entstehen, relativ klein bleiben und das resultierende Entscheidungsverfahren mit den auf Quantorenelimination basierenden konkurrieren kann. Kürzlich wurde von Felix Klaedtke (Freiburg) eine theoretische Begründung geliefert:

Satz 11 (Klaedtke)
Der minimale deterministische Automat für eine gegebene wMSO-Formel $\hat{\phi}$, wobei ϕ eine Presburger-Formel ist, hat Grösse $2^{2^{O(n)}}$, wobei n die Länge von ϕ ist.