Friedrich–Alexander–Universität Erlangen–Nürnberg
Institut für Mathematische Maschinen und Datenverarbeitung
Lehrstuhl für Künstliche Intelligenz (Informatik VIII)

Implementierung von Automaten-Algorithmen mit Hilfe von binären Entscheidungsdiagrammen

–Diplomarbeit–

vorgelegt von

Carsten Schneider

1. Oktober 1997

Betreuer: **Prof. Dr. Volker Strehl**

Erklärung :

Ich versichere, daß ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und daß die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 1. Oktober 1997 ....

Carsten Schneider

Inhaltsverzeichnis

1 Einleitung

Das zentrale Ziel der Diplomarbeit richtet sich darauf, alle möglichen Eingaben eines Automaten zu bestimmen, sodaß dieser eine bestimmte Ausgabe berechnet.

Sei im folgenden $\mathbb{B}$ die zweielementige boolesche Algebra. Die Darstellung der Automaten ist so gewählt worden, daß die Zustände des Automaten in dem Zustandsraum $\mathbb{B}^l$ kodiert sind und das Eingabealphabet mit $\mathbb{B}^m$ und das Ausgabealphabet mit $\mathbb{B}^n$ bestimmt werden ($l, m \in \mathbb{N}_0, n \in \mathbb{N}$).

Mit einem Automat wird nun genauer ein synchroner Schaltkreis mit Ausgabeberechtigung bezeichnet. Dieser kann mit einem Quadrupel

$$\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$$

beschrieben werden mit den booleschen Funktionen

- $\delta : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}^l$ zur Berechnung des *Folgezustands*,
- λ zur Berechnung der *Hilfsausgabe*:

$$\lambda : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \to & \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & (\lambda_0(\boldsymbol{X}, \boldsymbol{Q}), \ldots, \lambda_{n-1}(\boldsymbol{X}, \boldsymbol{Q})) \end{cases}$$

  wobei $\lambda_i : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}$ die i-te Komponentenfunktion von λ bezeichnet
- α zur Berechnung der *Ausgabeberechtigung*

$$\alpha : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \to & \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & (\alpha_0(\boldsymbol{X}, \boldsymbol{Q}), \ldots, \alpha_{n-1}(\boldsymbol{X}, \boldsymbol{Q})) \end{cases}$$

  wobei $\alpha_i : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}$ die i-te Komponentenfunktion von α bezeichnet
- und dem *Startzustand* $\boldsymbol{s} \in \mathbb{B}^l$

wobei $l \in \{\infty\} \cup \mathbb{N}_0$, $m \in \mathbb{N}_0$ und $n \in \mathbb{N}$.

$\mathcal{S}$ heißt *endlicher* synchroner Schaltkreis mit Ausgabeberechtigung, wenn der Zustandsraum endlich ist. Seien

$$\begin{array}{rcl} \xi_i & = & \langle x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(t)}, \ldots \rangle \in \mathbb{B}^\omega \text{ für } 0 \leq i \leq m-1 \\ \upsilon_i & = & \langle y_i^{(0)}, y_i^{(1)}, \ldots, y_i^{(t)}, \ldots \rangle \in (\mathbb{B} \cup \{\epsilon\})^\omega \text{ für } 0 \leq i \leq n-1 \end{array}$$

$\mathcal{S}$ *berechnet* zu der Eingabefolge $\boldsymbol{\xi} = \langle \xi_0, \ldots \xi_{m-1} \rangle$ die Ausgabenfolge $\boldsymbol{\upsilon} = \langle \upsilon_0, \ldots \upsilon_{n-1} \rangle$, wenn es eine unendliche Zustandsfolge $\langle q^{(0)}, q^{(1)}, \ldots, q^{(t)}, \ldots \rangle \in (\mathbb{B}^l)^\omega$ mit dem Startzustand $q^{(0)} = \boldsymbol{s}$ gibt, sodaß im t-ten Berechnungsschritt zur Eingabe $\boldsymbol{x}^{(t)} = \langle x_0^{(t)}, x_1^{(t)} \ldots, x_{m-1}^{(t)} \rangle$ die Ausgabe $\boldsymbol{y}^{(t)} = (y_0^{(t)}, y_1^{(t)} \ldots, y_{n-1}^{(t)})$ berechnet wird mit

$$q^{(t+1)} = \delta(\boldsymbol{x}^{(t)}, q^{(t)})$$

und

$$y_i^{(t)} = \begin{cases} \lambda_i(\boldsymbol{x}^{(t)}, q^{(t)}), & \text{wenn} \quad \alpha_i(\boldsymbol{x}^{(t)}, q^{(t)}) = 1 \\ \epsilon, & \text{wenn} \quad \alpha_i(\boldsymbol{x}^{(t)}, q^{(t)}) = 0 \end{cases}$$

für alle $0 \leq i < n$ und $t \geq 0$.

In ausführlicher Form werden synchrone Schaltkreise mit Ausgabeberechtigung in Abschnitt 3.2 behandelt. Für seine Definition kann die Abbildung auf Seite 37 hilfreich sein.

Eine bemerkenswert elegante Methode, alle möglichen Eingabefolgen L für eine bestimmte Ausgabefolge anzugeben, besteht darin, diese mit einem Automaten $\mathcal{S}_L$ darzustellen. Zu einer beliebigen Eingabefolge berechnet dieser Automat $\mathcal{S}_L$ eine Folge aus der gesuchten Lösung L. Werden nun von $\mathcal{S}_L$ mit allen Eingabefolgen alle Folgen aus L berechnet, so wird die Lösungsmenge L in kompakter Form mit $\mathcal{S}_L$ kodiert.

Eine wesentlicher Konstruktionsschritt für die Berechnung von $\mathcal{S}$ besteht darin, die Ausgabefunktion von $\mathcal{S}$ zu modifizieren. Eine häufig auftretende Konsequenz dabei ist, daß durch die Konstruktion die Ausgabefunktion stark vereinfacht wird. So werden viele Zustände erzeugt, die untereinander äquivalent sind, d.h. das Ausgabeverhalten ist - von diesen Zuständen ausgehend - für jede Eingabefolge identisch. Wird aus dem modifizierten Automat nun der Minimalautomat konstruiert, so wird in vielen Fällen die Anzahl der verwendeten Zustände stark eingeschränkt, und es entsteht ein sehr einfacher Automat.

Für die eben beschriebenen Konstruktionen muß zum einen gewährleistet sein, daß die Automaten in einer kompakten Form beschrieben werden können und zum anderen flexible und effiziente Operationen zur Verfügung stehen, um mit einem Automaten arbeiten und diesen verändern zu können.

Werden die booleschen Funktionen mit Hilfe von geordneten binären Entscheidungsdiagrammen (OBDDs) dargestellt, so gelingt in vielen Fällen eine kompakte Darstellung. Für den Datentyp OBDD stehen sehr flexible Operationen zur Verfügung, die boolesche Operationen für die kodierten booleschen Funktionen realisieren. Da die Berechnungskomplexität der OBDD-Operationen von der Größe der OBDDs abhängt, sind diese flexiblen Operationen für kompakt dargestellte boolesche Funktionen sehr effizient. Die Definition der OBDDs, ihre Eigenschaften und ihre typische Implementierung werden in Kapitel 2 behandelt.

In den folgenden Konstruktionen ist unter anderem die komprimierte Beschreibung von Zustandsmengen und deren Bearbeitung grundlegend. Da eine Menge $M \subseteq \mathbb{B}^l$ mit Hilfe ihrer charakteristischen Funktion $\chi_m : \mathbb{B}^l \to \mathbb{B}$ mit

$$q \in M \Leftrightarrow \chi_M(q) = 1$$

beschrieben werden kann, können Mengen mit Hilfe von OBDDs charakterisiert werden. Indem die Automaten und ihre Zustandsmengen mit einem gemeinsamen Datentyp OBDD dargestellt werden können, gelingen elegante Berechnungen, die Automaten und ihre Zustände mit effizienten flexiblen booleschen Operationen verknüpfen. Wie die Automaten mit Hilfe von OBDDs dargestellt werden, wird genauer in Kapitel 4 vorgestellt.

Lösung von online- und offline Gleichungen

Im folgenden wird dargestellt, wie synchrone Schaltkreise mit Ausgabeberechtigung durch online und offline Funktionen beschrieben werden können. Eine genauere Darstellung wird in Abschnitt 3.3 vorgelegt.

Zu einer Eingabefolge $\xi \in (\mathbb{B}^\omega)^m$ eines synchronen Schaltkreises mit Ausgabeberechtigung wird nun nur noch die Ausgabe $\upsilon = (\upsilon_0, \ldots, \upsilon_{n-1}) \in (\mathbb{B}^\omega)^n$ mit der Ausgabenkomponente

$$\upsilon_i = \langle \upsilon_i^{(0)}, \upsilon_i^{(1)} \ldots, \upsilon_i^{(k)} \ldots \rangle \in \mathbb{B}^\omega$$

betrachtet, wobei zum k-ten Mal die Ausgabeberechtigung α_i eine Ausgabe $\upsilon_i^{(k)}$ zuläßt.

Soll nur dieses Ausgabeverhalten betrachtet werden, so kann die Beschreibung des Ausgabeverhaltens mit Zuständen und der Übergangsfunktion δ sehr aufwendig sein. Mit Hilfe von 2-adischen

Funktionen gelingt es dagegen geschickt, das Ausgabeverhalten eines synchronen Schaltkreises mit Ausgabeberechtigung sehr elegant darzustellen.

Unendliche Eingabe- und Ausgabefolgen $\xi, \upsilon \in \mathbb{B}^\omega$ können mit Hilfe der ganzen 2-adischen Zahlen

$$_2\mathbb{Z} = \{\sum_{i\geq 0} b^{(t)}2^t \mid b^{(t)} \in \mathbb{B}\}$$

repräsentiert werden.

Ist garantiert, daß ein synchroner Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ zu jeder Eingabe $\xi \in {_2\mathbb{Z}^m}$ eine Ausgabe $\upsilon \in {_2\mathbb{Z}^n}$ ausgibt, so kann das Ausgabeverhalten mit einer 2-adischen Funktion

$$f : {_2\mathbb{Z}^m} \to {_2\mathbb{Z}^n}$$

beschrieben werden.

Seien $\xi_i = \sum_{t\geq 0} x_i^{(t)}2^t$ für $(0 \leq i < m)$ ganze 2-adische Zahlen und $\xi^{[t]}$ die Folge der ersten $t+1$ Koeffizienten von ξ_i. Eine Funktion $f : {_2\mathbb{Z}^m} \to {_2\mathbb{Z}^n}$ heißt

- *online*, wenn es boolesche Funktionen $f^{(t)} : (\mathbb{B}^{t+1})^m \to \mathbb{B}^n$ gibt mit

$$f(\xi_0, \ldots, \xi_m) = \sum_{t\geq 0} f^{(t)}(\xi_0^{[t]}, \ldots, \xi_{m-1}^{[t]})2^t$$

- *offline*, wenn es boolesche Funktionen $f^{(t)} : (\mathbb{B}^{s^{(t)}+1})^m \to \mathbb{B}^n$ zu einer streng monoton steigenden Folge $s^{(0)} < s^{(1)} < s^{(2)} < \ldots$ in $\mathbb{N}_0$ gibt mit

$$f(\xi_0, \ldots, \xi_m) = \sum_{t\geq 0} f^{(t)}(\xi_0^{[s^{(t)}]}, \ldots, \xi_{m-1}^{[s^{(t)}]})2^t$$

Durch diese Einschränkung von 2-adischen Funktionen gelingt folgende Korrespondenz:

Ist $f : {_2\mathbb{Z}^m} \to {_2\mathbb{Z}^n}$ eine Funktion über den ganzen 2-adischen Zahlen, dann gilt

- f ist online genau dann, wenn die Funktion f von einem synchronen Schaltkreis mit m Eingaben und n Ausgaben berechenbar ist.
- f ist offline genau dann, wenn die Funktion f von einem synchronen Schaltkreis mit Ausgabeberechtigung berechenbar ist, der m Eingaben und n Ausgaben besitzt.

Wie oben bereits hervorgehoben, stellt sich die interessante Frage, welche Eingaben zu einem endlichen synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ möglich sind, damit $\mathcal{S}$ eine bestimmte Ausgabe berechnet. Dieses Problem wird in den Kapiteln 6 und 7 gelöst.

Berechnet ein synchroner Schaltkreis mit Ausgabeberechtigung die offline Funktion

$$f : {_2\mathbb{Z}^m} \to {_2\mathbb{Z}^n}$$

so werden insbesondere alle Eingaben $\boldsymbol{\xi} \in {_2\mathbb{Z}^m}$ gesucht, sodaß

$$f(\boldsymbol{\xi}) = \mathbf{0}^\omega$$

erfüllt wird, d.h. es muß die Nullstellenmenge

$$L = \{\boldsymbol{\xi} \in {_2\mathbb{Z}^m} \mid f(\boldsymbol{\xi}) = \mathbf{0}^\omega\}$$

berechnet werden.

Dies gelingt mit einem synchronen Schaltkreis $\mathcal{S}_L$, der die online Funktion $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ berechnet. σ muß dabei so beschaffen sein, daß

$$L = \{\sigma(\boldsymbol{\zeta}) \mid \zeta \in {}_2\mathbb{Z}^m\}$$

erfüllt wird. σ wird in diesem Zusammenhang eine *Lösung* von der online Gleichung

$$f(\boldsymbol{\xi}) = \mathbf{0}^\omega$$

genannt. Mit Hilfe von booleschen Operationen können die Übergangsfunktion, Ausgabeberechtigung und Hilfsausgabe von $\mathcal{S}$ derart modifiziert werden, daß es gelingt,

1. einen synchronen Schaltkreis $\tilde{\mathcal{S}}$ zu berechnen, dessen online Funktion $\tilde{f} : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ dieselbe Nullstellenmenge besitzt wie die offline Funktion f (siehe Kapitel 6.2.2).
2. zu einem synchronen Schaltkreis $\tilde{\mathcal{S}}$ mit der online Funktion $\tilde{f}$ einen synchronen Schaltkreis $\tilde{\mathcal{S}}_L$ zu berechnen, dessen online Funktion σ eine Lösung für die Gleichung $\tilde{f} = 0$ und damit für die Gleichung $f = 0$ ist (siehe Kapitel 6.2.1).

Lösung von parameterbehafteten online Gleichungen

Sei $\mathcal{S}$ ein synchroner Schaltkreis, der die online Funktion

$$f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$$

berechnet. Gesucht ist zu einem Parameter $\boldsymbol{\pi} \in {}_2\mathbb{Z}^p$ die Menge aller Lösungen $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ zu der *parameterbehafteten Gleichung*

$$f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega$$

d.h. die Lösungsmenge

$$L_{\boldsymbol{\pi}} = \{\boldsymbol{\xi} \in {}_2\mathbb{Z}^m \mid f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega\}$$

Mit der obigen Lösung $\sigma : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m$ kann jedoch nur die Lösungsmenge für alle Parameter berechnet werden

$$L = \bigcup_{\boldsymbol{\pi} \in {}_2\mathbb{Z}^p} L_{\boldsymbol{\pi}}$$

Will man gezielt zu einem Parameter $\boldsymbol{\pi}$ die Lösungsmenge $L_{\boldsymbol{\pi}}$ berechnen, so muß eine parameterbehaftete Lösung $\sigma : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ gesucht werden, sodaß gilt

$$L_{\boldsymbol{\pi}} = \{\sigma(\boldsymbol{\pi}, \boldsymbol{\zeta}) \mid \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m\}$$

Dieses σ ist in der Regel aber nicht mehr online, ja sogar nicht einmal offline, d.h. σ kann nicht mehr durch einen synchronen Schaltkreis mit Ausgabeberechtigung berechnet werden. Besitzt f jedoch gewisse Eigenschaften, die in Bemerkung 6.41 vorgestellt werden, so kann eine online Funktion σ konstruiert werden, mit der eine Teilmenge von der Nullstellenmenge $L_{\boldsymbol{\pi}}$ zu dem Parameter π berechnet wird.

Unter bestimmten Umständen kann also ein synchroner Schaltkreis $\mathcal{S}_L$ konstruiert werden, dessen parameterbehaftete online Funktion einen eingeschränkten Teil der Nullstellen beschreibt. Wie $\mathcal{S}_L$ konstruiert werden kann, wird in den Abschnitten 6.4 und 6.5 vorgestellt.

Lösung von Ungleichungen

Eine interessante Frage ist, welche Eingaben möglich sind, sodaß ein synchroner Schaltkreis $\mathcal{S}$ nicht die Nullfolge ausgibt. Ist $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ die zugehörige online Funktion, dann wird die Lösungsmenge L zu der Ungleichung

$$f(\boldsymbol{\xi}) \neq 0^\omega$$

gesucht.

Soll die Lösungsmenge von einem endlichen synchronen Schaltkreis $\mathcal{S}_L$ berechnet werden, so muß dieser für alle Eingaben alle Lösungen berechnen, mit denen $\mathcal{S}$ nicht die Nullfolge ausgibt.

Es gibt nun zwei Typen von Eingabefolgen ξ, mit denen $\mathcal{S}$ eine Nullfolge berechnet. Gelangt der synchrone Schaltkreis $\mathcal{S}$ während der Berechnung der Nullfolge zur Eingabe ξ in einen Zustand, von dem aus nur noch der boolesche Wert 0 berechnet werden kann, so ist die Eingabefolge ξ eine universelle Nullstelle von $\mathcal{S}$.

Gerät $\mathcal{S}$ jedoch in jedem Berechnungsschritt in einen Zustand, von dem aus mit einer entsprechenden endlichen Eingabefolge eine 1 ausgegeben werden kann, so wird diese Eingabefolge als existentielle Nullstelle bezeichnet.

Es kann nun gelingen, einen synchronen Schaltkreis $\mathcal{S}'_L$ zu konstruieren, der alle Folgen bis auf die universellen Nullstellen berechnet. Soll $\mathcal{S}'_L$ jedoch alle Nicht-Nullstellen berechnen, so berechnet er zusätzlich alle existentiellen Nullstellen mit. Denn existiert eine existentielle Nullstelle, dann gibt es zu jeder Anfangsfolge dieser Nullstelle nach Definition eine Nicht-Nullstelle mit der gleichen Anfangsfolge. Da alle Nicht-Nullstellen berechnet werden können, kann jedes Anfangsstück der Länge $k \in \mathbb{N}_0$ von der existentiellen Nullstelle und damit auch die existentielle Nullstelle selbst berechnet werden. Die Lösungsmenge einer Ungleichung kann also im allgemeinen nicht mit Hilfe eines synchronen Schaltkreises berechnet werden.

Umgekehrt kann von jedem erreichbaren Zustand des Automaten $\mathcal{S}'_L$ jedoch eine 1 ausgegeben werden. Es kann nun ein synchroner Schaltkreis $\mathcal{S}_L$ konstruiert werden, der sich wie $\mathcal{S}'_L$ verhält, d.h. er berechnet alle Nicht-Nullstellen und existentiellen Nullstellen. Mit einer zusätzlichen Eingabe besteht nun die Möglichkeit, den Automaten $\mathcal{S}_L$ derart zu aktivieren, daß dieser eine Ausgabefolge berechnet, mit der eine 1 ausgegeben werden muß.

Berechnet $\mathcal{S}$ die online Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$, so kann ein synchroner Schaltkreis $\mathcal{S}_L$ konstruiert werden, der die online Funktion $\tau : {}_2\mathbb{Z} \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ berechnet und die gesuchte Lösungsmenge L folgendermaßen beschreibt:

$$L = \{\tau(\pi, \boldsymbol{\xi}) \mid \forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m \forall \pi \in {}_2\mathbb{Z} \setminus \{0^\omega\}\}$$

Diese Konstruktion wird ausführlich in Kapitel 8 behandelt.

Berechnung des Minimalautomaten

Mit den obigen Konstruktionen werden synchrone Schaltkreise $\mathcal{S}$ so modifiziert, daß das Ausgabeverhalten von Zuständen vereinfacht wird. So werden viele Zustände erzeugt, die untereinander äquivalent sind. Zwei Zustände $q_0, q_1 \in \mathbb{B}^l$ von $\mathcal{S}$ heißen dabei genau dann äquivalent, kurz

$$q_0 \sim q_1$$

wenn für jede Eingabefolge von q_0 und q_1 ausgehend dieselbe Ausgabefolge berechnet wird.

Wegen ihrer Reflexivität, Symmetrie und Transitivität ist $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$ eine Äquivalenzrelation.

Schaltkreise können nun derart minimalisiert werden, daß alle äquivalenten Zustände zu einem Repräsentanten zusammengefaßt werden können. Damit kann dann die Anzahl der erreichbaren Zustände erfolgreich reduziert werden. Eventuell vereinfachen sich dann auch die booleschen Funktionen, sodaß der synchrone Schaltkreis vereinfacht realisiert werden kann.

Ist R die Menge der erreichbaren Zustände von $\mathcal{S}$, so heißt $\mathcal{S}$ Minimalautomat bzgl. seiner erreichbaren Zustände, kurz Minimalautomat von $\mathcal{S}$, wenn gilt

$$\forall q_0, q_1 \in R : q_0 \sim q_1 \Rightarrow q_0 = q_1$$

Es gibt nun drei Berechnungsschritte für die Minimalisierung von $\mathcal{S}$, die in Kapitel 9 behandelt werden.

1. Die Äquivalenzrelation $\sim$ auf der Menge der erreichbaren Zustände wird berechnet.
2. Für jede Äquivalenzklasse wird ein Repräsentant ausgewählt.
3. Die Zustandsfunktion wird derart modifiziert, daß sie nicht den Folgezustand $q \in \mathbb{B}^l$, sondern den Repräsentanten aus seiner Äquivalenzklasse berechnet.

Diese Berechnungen können mit Hilfe von booleschen Operationen durchgeführt werden, die auf die Übergangsfunktion, Hilfsausgabefunktion und Ausgabeberechtigung sowie auf die Zustandsmengen angewendet werden.

Reduzieren der OBDD-Größe der Zustandsfunktionen, Ausgabefunktionen und Ausgabeberechtigung

Mit Hilfe von OBDDs können sehr effizient die erreichbaren Zustände R eines synchronen Schaltkreises mit Ausgabeberechtigung $\mathcal{S}$ berechnet werden.

Ein synchroner Schaltkreis darf bzgl. seiner booleschen Funktionen verändert werden, solange die Übergangsfunktion die erreichbaren Zustände korrekt auf die Folgezustände abbildet und die Hilfsausgabefunktion und Ausgabeberechtigung für alle Eingaben in den erreichbaren Zuständen die korrekte Ausgabe berechnet. Für die Eingaben $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ kann das Verhalten der booleschen Funktionen also beliebig verändert werden.

Die booleschen Funktionen von $\mathcal{S}$ können demnach in dem Bereich $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ als unspezifizierte Funktionen angesehen werden. Wenn die booleschen Funktionen mit OBDDs dargestellt werden, können die OBDDs für Eingaben $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ derart verändert werden, daß deren Graph mit einer kleineren Anzahl von Entscheidungsknoten repräsentiert werden kann.

Die OBDDs können dabei so modifiziert werden, daß nicht einzelne OBDDs möglichst wenige Knoten besitzen, sondern daß die gesamte Knotenanzahl aller OBDDs möglichst gering wird. Dies wird erreicht, indem die einzelnen OBDDs so modifiziert werden, daß möglichst viele Knoten von den verschiedenen OBDDs zu einem Knoten zusammengefaßt werden können. Dieser Algorithmus wird in Kapitel 10 erläutert.

Dieses Gesamt-OBDD mit mehreren Wurzelknoten kann nun in boolesche Formeln übersetzt werden, in denen möglichst viele Teilformeln gemeinsam verwendet werden. Aus den konstruierten Automaten können so eventuell kompakte boolesche Formeln berechnet werden, die dann direkt mit Hardwarebausteinen realisiert werden können.

Ausblick auf das weitere Procedere

Im nächsten Kapitel werden binäre Entscheidungsdiagramme und ihre Operationen vorgestellt. Anschließend werden im Kapitel 3 synchrone Schaltkreise mit Ausgabeberechtigung eingeführt und ihre Beziehung zwischen online und offline Funktionen aufgezeigt. Zusätzlich werden wichtige Operationen für online Funktionen definiert. Im Kapitel 4 wird nun aufgezeigt, wie synchrone Schaltkreise mit Ausgabeberechtigung mit Hilfe von OBDDs beschrieben werden. Im Kapitel 5 werden dann einfache Automatenkonstruktionen vorgestellt, deren Umsetzung als Vorbild für die weiteren Konstruktionen dienen. Im Kapitel 6 wird nun dargelegt, wie online Gleichungen und parameterbehaftete online Gleichungen gelöst werden können. Mit Kapitel 7 wird dann das Problem, offline Gleichungen zu lösen, auf den online Fall reduziert. Anschließend wird im Kapitel 8 das Lösung von Ungleichungen umgesetzt. Für diese Konstruktionen kann es hilfreich sein, den Minimalautomaten zu erzeugen (Kapitel 9) oder die OBDDs eines synchronen Schaltkreises bzgl. seiner erreichbaren Zustände zu reduzieren (Kapitel 10). In Kapitel 11 wird schließlich eine Implementierung dieser Algorithmen und Konstruktionen vorgestellt.

2 Geordnete binäre Entscheidungsdiagramme

Automaten werden im folgenden durch synchrone Schaltkreise beschrieben. Ein wesentliches Merkmal von diesen ist, daß ihr Verhalten durch boolesche Funktionen beschrieben wird. Konstruktionen mit synchronen Schaltkreisen effizient durchzuführen, hängt deshalb davon ab, wie kompakt boolesche Funktionen dargestellt werden und wie effizient typische boolesche Operationen realisiert werden können.

Randal E. Bryant stellt in [Bry85] geordnete binäre Entscheidungsdiagramme (OBDDs) als Datenstruktur für boolesche Funktionen vor. Reduzierte binäre Entscheidungsdiagramme ermöglichen in vielen Fällen, boolesche Funktionen mit geringem Speicheraufwand darzustellen und boolesche Operationen effizient durchzuführen.

Wegen dieser Eigenschaften werden die booleschen Funktionen von synchronen Schaltkreisen durch OBDDs beschrieben und die Automatenkonstruktion mit Hilfe von OBDD-Operationen umgesetzt.

In diesem Kapitel werden nun OBDDs und ihre Eigenschaften vorgestellt. Um ihre Effizienz zu verdeutlichen und das Verständnis für die Umsetzung ihrer Operationen zu verstehen, wird anschließend skizziert, wie die Darstellung von OBDDs und ihre Operationen nach [BRB90] implementiert werden. Das später verwendete OBDD-Paket [Lon93] setzt diese Ideen um.

2.1 Einführung

Geordnete binäre Entscheidungsdiagramme dienen als Datenstruktur für boolesche Funktionen. Der Zusammenhang der später folgenden Definition von OBDDs und der Darstellung von booleschen Funktionen soll nun verständlich gemacht werden, indem verschiedene Möglichkeiten vorgestellt werden, wie boolesche Funktionen dargestellt werden können.

Zuerst werden entsprechend [HR68] die 2-elementige boolesche Algebra und boolesche Funktionen eingeführt.

Definition 2.1. Die *2-elementige boolesche Algebra* $\mathbb{B}$ wird definiert durch die Menge

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

und die Operationen

$$\vee : \mathbb{B} \to \mathbb{B},\ \wedge : \mathbb{B} \to \mathbb{B} \text{ und } \neg : \mathbb{B} \to \mathbb{B}$$

mit

$$0 \vee 0 = 0,\ 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1,$$

$$0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0$$

$$\neg 0 = 1,\ \neg 1 = 0$$

Notation 2.2. In diesem Kapitel wird die Variablenmenge $\{x_0, \ldots, x_{n-1}\}$ immer mit X bezeichnet und der Vektor $\langle x_0, \ldots, x_{n-1}\rangle$ mit $\boldsymbol{X}$.

Definition 2.3. Eine *boolesche Funktion* f ist eine Abbildung

$$f : \mathbb{B}^n \to \mathbb{B}$$

Ein *boolescher Funktionsvektor* ist eine Abbildung

$$f : \left\{ \begin{array}{ccc} \mathbb{B}^n & \to & \mathbb{B}^m \\ \boldsymbol{X} & \mapsto & (f_0(\boldsymbol{X}), \ldots, f_{n-1}(\boldsymbol{X})) \end{array} \right.$$

wobei $f_i : \mathbb{B}^n \to \mathbb{B}$ die i-te boolesche Komponentenfunktionen von f bezeichnet.

Bemerkung 2.4. Im folgenden besitzen alle booleschen Funktionen die Stelligkeit n.

Mit Hilfe von booleschen Operationen können aus vorliegenden booleschen Funktionen neue gewonnen werden, wenn diese verknüpft und modifiziert werden. Als Beispiel wird die Operation Restriktion eingeführt.

Definition 2.5. Sei

$$f : \begin{cases} \mathbb{B}^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & f(\boldsymbol{X}) \end{cases}$$

eine boolesche Funktion. Dann wird die boolesche Operationen *Restriktion* $f|_{x_i=b}$ bzgl. $b \in \mathbb{B}$ definiert mit

$$f|_{x_i=b} : \begin{cases} \mathbb{B}^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & f(x_0, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_{n-1}) \end{cases}$$

Weitere wichtige Beispiele werden in Bemerkung 2.9 und Abschnitt 2.6.3 vorgestellt.

Die einfachste Möglichkeit, boolesche Funktionen darzustellen, sind Wertetabellen. Da eine n-stellige Funktion jedoch 2^n Einträge benötigt, ist diese Darstellungsart für viele Anwendungen unbrauchbar. In vielen Fällen werden deshalb boolesche Funktionen durch boolesche Formeln repräsentiert.

2.1.1 Boolesche Formeln

Boolesche Formeln werden induktiv definiert:

Definition 2.6.

1. Die Zeichen $0, 1$ sind boolesche Formeln.
2. Die Variablen $X = \{x_0, \ldots, x_{n-1}\}$ sind boolesche Formeln.
3. Sind F, G boolesche Formeln, dann auch $F \vee G$, $F \wedge G$ und $\neg F$.

Eine boolesche Formel $F(\boldsymbol{X})$ kodiert genau eine boolesche Funktion $f : \mathbb{B}^n \rightarrow \mathbb{B}$.

Definition 2.7. Sei $F(\boldsymbol{X})$ eine boolesche Formel. Die boolesche Funktion $f : \mathbb{B}^n \rightarrow \mathbb{B}$ wird durch F erzeugt, indem

- $x_0, \ldots, x_{n-1}$ als Variablen in $\mathbb{B}$,
- $\wedge, \vee, \neg$ als die Funktionen definiert in (2.1)
- Symbole $0, 1$ als die Konstantenfunktionen

$$0 : \begin{cases} B^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & 0 \end{cases}, \quad 1 : \begin{cases} B^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & 1 \end{cases}$$

interpretiert werden.

Bekannterweise können boolesche Formeln mit den booleschen Operationen $\wedge, \vee, \neg$ alle booleschen Funktionen darstellen.

Definition 2.8. Die booleschen Operationen $\neg, \wedge, \vee$ werden syntaktisch mit Hilfe von booleschen Formeln definiert. Seien $F(\boldsymbol{X}), G(\boldsymbol{X})$ und $H(\boldsymbol{X})$ boolesche Formeln zu den booleschen Funktionen f, g und h. Dann entstehen mit $\neg f$, $f \wedge g$ und $f \vee g$ boolesche Funktionen, die von den booleschen Formeln $\neg F$, $F \wedge G$ und $F \vee G$ repräsentiert werden. Entsprechend wird die boolesche Operation $\mathrm{ite}(f, g, h)$ auf den booleschen Funktionen f, g, h syntaktisch mit der booleschen Formel

$$\mathrm{ite}(F, G, H) = (F \wedge G) \vee (\neg F \wedge H)$$

definiert.

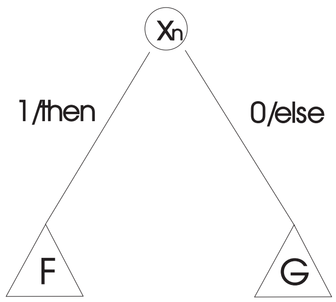
Besonders wichtig für diese Arbeit wird die "If-Then-Else" (*ite*) Operation sein.

Bemerkung 2.9. Folgendes ist für die *ite*-Operation hervorzuheben:

- Mit einer *ite*-Operation und der Negation $\neg$ können alle binären booleschen Operationen syntaktisch mit Hilfe von booleschen Formeln F und G ausgedrückt werden, wie z.B.

$$\begin{aligned} F \wedge G &= \mathrm{ite}(F, G, 0) & \mathrm{nor}(F, G) &= \mathrm{ite}(F, 0, \neg G) \\ \mathrm{nand}(F, G) &= \mathrm{ite}(F, \neg G, 1) & F \equiv G &= \mathrm{ite}(F, G, \neg G) \\ F \setminus G &= \mathrm{ite}(F, \neg G, 0) & F \oplus G &= \mathrm{ite}(F, \neg G, G) \\ F \vee G &= \mathrm{ite}(F, 1, G) & &\vdots \end{aligned}$$

- Betrachtet man den Sonderfall $H(x_0, \ldots, x_{n-1}, x_n) = \mathrm{ite}(x_n, F(\boldsymbol{X}), G(\boldsymbol{X}))$, wobei x_n eine Variable ist mit $x_n \notin X$, so stellt diese Operation einen *Entscheidungsknoten* bzgl. der Variablen x_n dar. Wenn x_n den Wert 1 besitzt, dann entspricht die boolesche Formel H der booleschen Formel F, ansonsten der booleschen Formel G. Dies kann graphisch folgendermaßen veranschaulicht werden:



Für die Definition von binären Entscheidungsdiagrammen hat die *ite*-Operation daher eine besondere Bedeutung.

2.1.2 Geordnete binäre Entscheidungsbäume

Die Wertetabelle einer booleschen Funktion kann mit Hilfe eines geordneten binären Entscheidungsbaums dargestellt werden.

Definition 2.10. Sei $X = \{x_0, \ldots, x_{n-1}\}$ eine Variablenmenge mit folgender totaler Ordnung der Variablen:

$$x_0 < x_1 < \ldots < x_{n-1}$$

Ein *geordneter binärer Entscheidungsbaum* mit Ordnung auf den Variablen X ist ein vollständiger binärer Baum, der aus zwei Knotentypen besteht:

- Blätter beschriftet mit Elementen aus $\mathbb{B} = \{0, 1\}$
- internen Knoten, die den *ite*-Operator realisieren. Diese
  - sind in der Baumtiefe k mit der Variablen x_k beschriftet
  - und besitzen zwei ausgehende Kanten, beschriftet mit 1/then und 0/else.

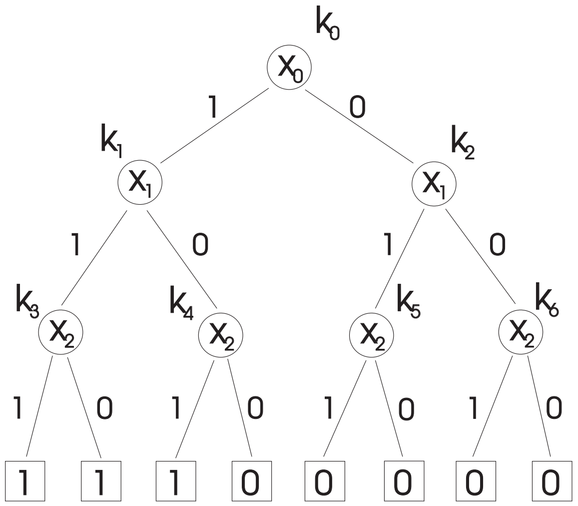
Beispiel 2.11. Zu der booleschen Formel

$$F(x_0, x_1, x_2) = x_0 \wedge (x_1 \vee x_2)$$

kann der geordnete binäre Entscheidungsbaum mit der Variablenordnung

$$x_0 < x_1 < x_2$$

folgendermaßen dargestellt werden.



Zu einer Bewertung der Variablen X erhält man den Wert der booleschen Funktion f, dargestellt durch F, indem der Pfad des Baumes durchlaufen wird. In dem Knoten der Tiefe k wird dabei bzgl. der Bewertung der Variable x_k entschieden, ob dem Pfad weiter entlang der then- oder else-Kante gefolgt wird.

Man sieht dabei leicht, daß auf viele Knoten verzichtet werden kann, ohne die Beschreibung der Wertetabelle zu verfälschen. So können z.B. Knoten k_5 und k_6 zusammengefaßt werden.

Natürlich geht bei diesem Prozeß die strenge Baumstruktur verloren. Es ist deshalb naheliegend, eine flexiblere Struktur zuzulassen: geordnete binäre Entscheidungsdiagramme.

2.2 Geordnete binäre Entscheidungsdiagramme (OBDD)

Definition 2.12. Sei $X = \{x_0, \dots, x_{n-1}\}$ eine Variablenmenge mit folgender totaler Ordnung der Variablen:

$$x_0 < x_1 < \dots < x_{n-1}$$

Ein *geordnetes binäres Entscheidungsdiagramm* (OBDD) auf den Variablen X ist ein gerichteter azyklischer Graph, der aus zwei Knotentypen besteht:

- terminale Knoten, beschriftet mit Elementen aus $\mathbb{B} = \{0, 1\}$
- interne Knoten, die den *ite*-Operator realisieren. Diese
  - sind mit Variablen aus X beschriftet
  - und besitzen zwei ausgehende Kanten, beschriftet mit 1/then und 0/else.

Ein OBDD besitzt genau einen *Wurzelknoten*, d.h. einen Knoten, der keine eingehenden Kanten hat, und erfüllt folgende Regel:
Wenn ein interner Knoten, beschriftet mit Variable $x \in X$, erreichbar ist von einem anderen Knoten, beschriftet mit Variable $y \in X$, dann gilt:

$$y < x.$$

Für den Graphentyp OBDD werden nun folgende Funktionen definiert.

Definition 2.13. Für die terminalen Knoten T und nichtterminalen Knoten N eines OBDDs mit Variablen X werden folgende Funktionen definiert:

$$\mathrm{IF} : \begin{cases} N & \rightarrow & X \\ v & \mapsto & \text{Variable, die Knoten } v \text{ beschriftet} \end{cases}$$

$$\mathrm{THEN} : \begin{cases} N & \rightarrow & N \cup T \\ v & \mapsto & \text{Knoten, der an der then-Kante hängt} \end{cases}$$

$$\mathrm{ELSE} : \begin{cases} N & \rightarrow & N \cup T \\ v & \mapsto & \text{Knoten, der an der else-Kante hängt} \end{cases}$$

Zu einem OBDD kann nun eine boolesche Formel definiert werden, d.h. mit Hilfe eines OBDDs kann eine boolesche Funktion dargestellt werden.

Definition 2.14. Zu einem binären Entscheidungsdiagramm F wird folgende boolesche Formel $\mathcal{F}_F$ rekursiv definiert.

- Sei F ein terminaler Knoten. Dann gilt:

$$\mathcal{F}_F = \begin{cases} 1, & \text{wenn} \quad \text{F beschriftet ist mit 1} \\ 0, & \text{wenn} \quad \text{F beschriftet ist mit 0} \end{cases}$$

- Sei F ein nichtterminaler Knoten. Dann gilt:

$$\begin{aligned} \mathcal{F}_F &= \mathrm{ite}(\mathrm{IF}(F), \mathcal{F}_{\mathrm{THEN}(F)}, \mathcal{F}_{\mathrm{ELSE}(F)}) \\ &= (\mathrm{IF}(F) \wedge \mathcal{F}_{\mathrm{THEN}(F)}) \vee (\neg \mathrm{IF}(F) \wedge \mathcal{F}_{\mathrm{ELSE}(F)}) \end{aligned}$$

Bemerkung 2.15. In einem OBDD-Diagramm kann jeder Knoten selbst als Wurzelknoten und der zugehörige Untergraph als ein OBDD angesehen werden. Dieses stellt nun ebenfalls eine boolesche Formel dar, die von den Variablen abhängt, die die Unterknoten beschriften.

Aus praktischen Gründen können mehrere OBDDs mit gemeinsamen Knoten dargestellt werden. So können Pfade von den Wurzelknoten der verschiedenen OBDDs ausgehend in gemeinsame Knoten führen. Sollen mehrere boolesche Funktionen mit Hilfe von OBDDs dargestellt werden, so können dadurch Speicherressourcen eingespart werden, indem identische Untergraphen von OBDDs zu einem Untergraphen zusammengefaßt werden. Dieses Konzept ist in [MIY90] mit "Shared Binary Decision Diagram" vorgestellt.

Beispiel 2.16. Ein OBDD wird mit seinem Wurzelknoten eindeutig bestimmt. Im folgenden wird ein OBDD deshalb mit seinem Wurzelknoten bezeichnet.

In Beispiel 2.11 stellt das OBDD k_0 die boolesche Formel $x_0 \wedge (x_1 \vee x_2)$, das OBDD k_1 die Formel $x_1 \vee x_2$, die OBDDs k_3, k_4 die Formel x_2 und die OBDDs k_2, k_5, k_6 die Formel 0 dar.

Bemerkung 2.17. Ein Knoten F in einem OBDD setzt den Spezialfall einer *ite*-Operation um (siehe auch Bemerkung 2.9.(2)). Ist $T = \text{THEN}(F)$ der Knoten, der an der then-Kante von F hängt, und $E = \text{ELSE}(F)$ entsprechend der Knoten, der an der else-Kante hängt, so beschreiben diese OBDDs boolesche Formeln $\mathcal{T}$ und $\mathcal{E}$, die von der Variable x nicht mehr abhängen. Der Knoten F, der mit der Variable $x = \text{IF}(F)$ beschriftet ist, repräsentiert dann die boolesche Formel

$$\text{ite}(x, \mathcal{T}, \mathcal{E})$$

Im folgenden wird deshalb ein OBDD-Knoten auch mit

$$\text{ITE}(x, T, E)$$

notiert.

Notation 2.18. Die Funktionen THEN und ELSE setzen für OBDDs einen Spezialfall der Restriktion aus Definition 2.5 um. Sind t und e die booleschen Funktionen zu THEN(F) und ELSE(F), so gilt

$$f|_{\text{IF}(F)=1} = t, \quad f|_{\text{IF}(F)=0} = e$$

Deshalb werden im folgenden die Notationen

$$F|_{x=1} = \text{THEN}(F), \quad F|_{x=1} = \text{ELSE}(F)$$

für die Variable $x = \text{IF}(F)$ eingeführt.

2.3 Reduzierte geordnete binäre Entscheidungsdiagramme (ROBDD)

Solange die beschriebene boolesche Funktion eines OBDDs nicht verändert wird, ist es zulässig, das OBDD zu modifizieren.

Durch folgende Überlegungen können OBDDs reduziert werden.

Lemma 2.19. *Seien f, g boolesche Funktionen. Dann gilt:*

$$\text{ite}(f, g, g) = g$$

Daraus ergeben sich nun zwei einfache *Reduktionsregeln*:

Regel 1 (Elimination)
Tritt in einem OBDD A ein Knoten F mit Kodierung ITE(y, E, E) auf, so darf Knoten F gelöscht werden, indem alle Kanten, die zu F führen, umgeleitet werden zu dem Knoten E.

Regel 2 (Verschmelzen)
Gibt es zwei verschiedene nichtterminale Knoten $V \neq L$ in einem OBDD A mit derselben Kodierung ITE(x, T, E), so dürfen die Knoten zusammengefaßt werden, indem Knoten V als Repräsentant ausgewählt wird und alle Kanten, die zu dem Knoten L führen, umgeleitet werden zu Knoten V.

Nach [Weg94] kann fast jede boolesche Funktion, dargestellt durch einen geordneten binären Entscheidungsbaum, mit der Eliminations-Regel reduziert werden, während die Verschmelzungsregel nur eine exponentiell kleine Anzahl von booleschen Funktionen reduzieren kann.

Mit folgender Regel werden die terminalen Knoten auf zwei Knoten reduziert.

Regel 0 Wähle jeweils einen terminalen Knoten mit der Beschriftung 0 und 1 und tausche in allen Knoten des OBDDs die Kodierung aus, sodaß terminale then-Knoten oder else-Knoten die ausgewählten 0 oder 1 Knoten sind.

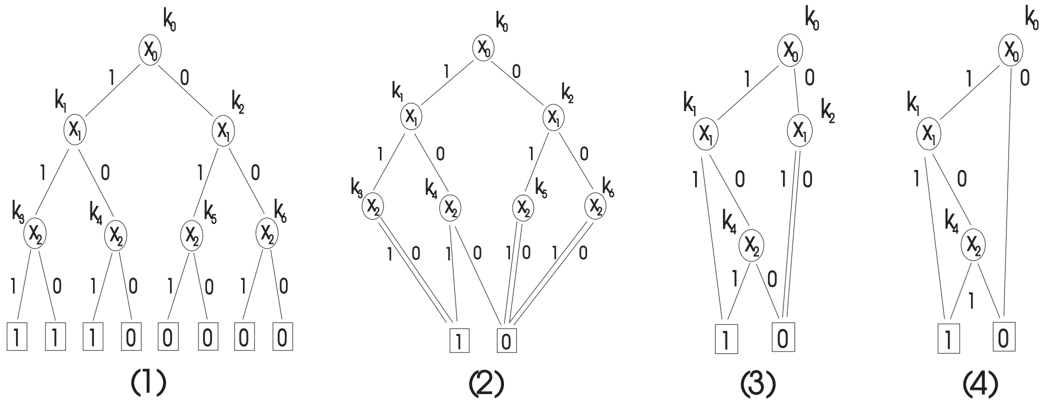
Wendet man diese Vereinfachungen, von den terminalen Knoten beginnend, rekursiv an, so erhält man folgende Eigenschaften für OBDDs:

1. Ein Knoten hat keine zwei gleichen Nachfolger.
2. Es gibt keine zwei Knoten A, B, sodaß die Untergraphen mit den Wurzeln A und B isomorph sind.

Definition 2.20. OBDDs mit diesen zwei Eigenschaften heißen *reduzierte geordnete binäre Entscheidungsdiagramme* (ROBDD).

Folgendes Beispiel verdeutlicht die rekursive Anwendung der Regeln (0)-(2), um ein ROBDD zu erhalten.

Beispiel 2.21. Abbildung (1) zeigt das OBDD aus Beispiel 2.11.



Mit Hilfe von Regel (0) gelangt man zu Abbildung (2). Mit Regel (1) können Knoten k_3, k_5 und k_6 eliminiert werden und man erhält Abbildung (3). Mit nochmaliger Anwendung von Regel (2) wird Knoten k_4 herausgeschnitten und man erhält in Abbildung (4) "das" reduzierte OBDD.

2.4 Eigenschaften von ROBDDs

In diesem Abschnitt werden grundlegende Eigenschaften von ROBDDs vorgestellt. Dadurch soll aufgezeigt werden, wie mit ROBDDs sinnvoll umzugehen ist.

Die Eigenschaften von reduzierten OBDDs sind hierbei so vorteilhaft, daß die reduzierte Darstellung die grundsätzliche Speicherform von OBDDs in Implementierungen ist.

Weiter wird sich herausstellen, daß der effiziente Einsatz von ROBDDs entscheidend von der Variablenordnung abhängt. Die Knotenanzahl eines ROBDDs wird von der Variablenordnung vorherbestimmt. Deshalb können nur mit einer geschickten Wahl der Variablenordung boolesche Funktionen kompakt durch ROBDDs dargestellt werden. Das Auffinden von brauchbaren Variablenordnungen ist deshalb die Vorbedingung, um mit ROBDDs arbeiten zu können. Die dabei auftretenden Schwierigkeiten müssen folglich vermieden werden, um den sinnvollen Einsatz von ROBDDs zu ermöglichen.

2.4.1 Minimalität und Eindeutigkeit von ROBDDs

Bryant zeigt in [Bry86], daß zu einer gegebenen Variablenordnung ein reduziertes OBDD von einer booleschen Funktion bis auf Isomorphie eindeutig ist. Liegen also zwei reduzierte ROBDDs vor, so kann durch einen Isomorphietest zwischen zwei Graphen festgestellt werden, ob sie dieselbe boolesche Funktion darstellen. Diese bemerkenswerte Eigenschaft wird nochmals in folgendem Satz festgehalten.

Satz 2.22 (Eindeutigkeit). *Zu einer festen Variablenordnung ist das reduzierte geordnete binäre Entscheidungsdiagramm von einer booleschen Funktion bis auf Isomorphie der Graphen eindeutig.*

Wird ein OBDD durch Regeln (0)-(2) in sein reduziertes OBDD transformiert, so gelingt dies, indem Knoten aus dem Graphen eliminiert werden. Die Knotenanzahl des OBDDs wird dadurch verringert. Liegt nun das reduzierte OBDD vor, so gibt es keine Möglichkeit, dieses in der Knotenanzahl weiter zu verkleinern. Gelänge dies doch, so könnte es durch Regeln (0)-(2) wieder in ein ROBDD übergeführt werden, jedoch mit weniger Knoten als das ursprüngliche ROBDD. Dies liegt dann im Widerspruch, daß zwei reduzierte OBDD derselben booleschen Funktion isomorph sind und damit die gleiche Knotenanzahl besitzen. Es gilt also folgende Konsequenz:

Korollar 2.23 (Minimalität). *Das reduzierte* OBDD *einer booleschen Funktion ist bzgl. seiner Variablenordnung minimal.*

Bemerkung 2.24. Jeder Knoten eines ROBDDs kodiert einen *ITE*-Operator mit bis auf Isomorphie eindeutigen then- und else-Knoten. Ein Knoten A wird deshalb eindeutig mit seiner Variablenbeschriftung $\mathrm{IF}(A)$ und seinem then-Knoten $\mathrm{THEN}(A)$ und else-Knoten $\mathrm{ELSE}(A)$ eindeutig bezeichnet, d.h. er wird mit der Kodierung

$$\mathrm{ITE}(\mathrm{IF}(A), \mathrm{THEN}(A), \mathrm{ELSE}(A))$$

eindeutig beschrieben. Die zwei terminalen Knoten werden im folgenden mit **1** und **0** bezeichnet.

In Beispiel 2.11 kann das reduzierte OBDD k_0 deshalb auch mit $\mathrm{ITE}(x_0, k_1, k_2)$ bezeichnet werden.

2.4.2 Die Variablenordnung von ROBDDs

In diesem Abschnitt soll verdeutlicht werden, daß die Variablenordnung eines ROBDDs bestimmt, wie kompakt boolesche Funktionen dargestellt werden können. Leider ist die Suche nach einer Ordnung, die das ROBDD minimiert, NP-vollständig. Es muß also mit heuristischen Methoden versucht werden, für eine konkrete Anwendung praktikable Ordnungen zu finden. Die Suche nach einer geschickten Ordnung liegt damit hauptsächlich in der Verantwortung des Anwenders.

In diesem Abschnitt soll weiter verdeutlicht werden, daß ROBDDs für bestimmte Beispiele nicht anwendbar sind, denn es gibt boolesche Funktionen, bei denen die Größe von ROBDDs unabhängig von der Variablenordnung exponentiell zur Variablenanzahl ist.

Die Abhängigkeit der OBDD-Größe von der Variablenordnung

Die Größe eines ROBDDs hängt grundlegend von der Variablenordnung ab. Dies soll an folgendem Beispiel verdeutlicht werden.

Beispiel 2.25. Dargestellt werden soll eine boolesche Funktion f, die entscheidet, ob ein Wort $w = \langle w_0, \ldots, w_{n-1} \rangle \in \mathbb{B}^n$ größer ist als Wort $\tilde{w} = \langle \tilde{w}_0, \ldots, \tilde{w}_{n-1} \rangle \in \mathbb{B}^n$ bzgl. der strikten lexikographischen Ordnung

$$w > \tilde{w} \Leftrightarrow (\exists i (w_0 = \tilde{w}_0 \wedge \ldots \wedge w_{i-1} = \tilde{w}_{i-1} \wedge w_i = 1 \wedge \tilde{w}_i = 0))$$

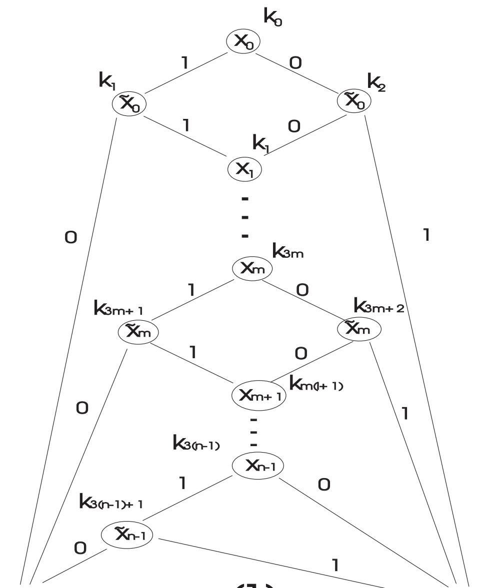
Die boolesche Funktion f ist definiert durch

$$f : \begin{cases} \mathbb{B}^{2n} & \rightarrow \quad \mathbb{B} \\ (w, \tilde{w}) & \mapsto \quad \begin{cases} 1, & \text{wenn} \quad w > \tilde{w} \\ 0, & \text{sonst} \end{cases} \end{cases}$$

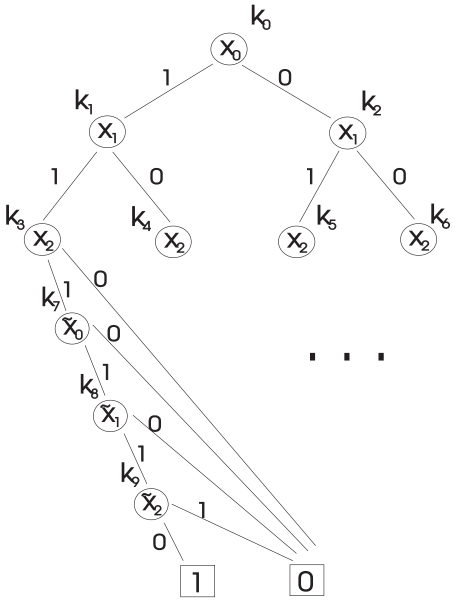
Das reduzierte OBDD $F(x_0, \ldots, x_{n-1}, \tilde{x}_0, \ldots, \tilde{x}_{n-1})$ für die boolesche Funktion f mit der Variablenordnung

$$x_0 < \tilde{x}_0 < \ldots < x_{n-1} < \tilde{x}_{n-1}$$

wird in Abbildung (1) dargestellt.



(1)



(2)

Dabei ist sofort einsichtig, daß die Knotengröße linear wächst zur Variablenanzahl:

$$\text{Knotenanzahl} = 3 \cdot n - 1$$

Wählt man dagegen die Variablenordnung

$$x_0 < \ldots x_{n-1} < \tilde{x}_0 < \ldots < \tilde{x}_{n-1},$$

so ist das Wachstum der Knoten exponentiell in Bezug auf die Variablenanzahl n. Im Fall $n = 3$ ist der Anfang eines ROBDDs mit dieser Variablenordnung in Abbildung (2) skizziert. Das ROBDD bläht sich bis zur Tiefe n zu einem vollständigen binären Baum auf. Jeder dieser Knoten der Tiefe n trägt nun die Information, wie die Variablen $x_0, \ldots, x_{n-1}$ belegt worden sind. Da bei jeder Belegung der ersten n Variablen sich das Verhalten der booleschen Funktion verändert, können diese Knoten nicht zusammengefaßt werden. Das Wachstum insgesamt ist also exponentiell:

$$\text{Knotenanzahl} \geq 2^n - 1$$

Die erste Ordnung vermeidet dieses exponentielle Wachstum. Zum einen werden dort Variablen $x_i, \tilde{x}_i$, die funktional zusammengehören, auch in der Variablenordnung benachbart angeordnet, zum anderen werden die Variablenpaare $(x_i, \tilde{x}_i)$ in der Reihenfolge angeordnet, in der so bald wie möglich eine Entscheidung bzgl. der lexikograpischen Ordnung ausgegeben werden kann. Diese Variablen bestimmen also stärker das Verhalten der booleschen Funktion. Variablen sollten also auch in ihrer Wichtigkeit bzgl. des Ausgabeverhaltens der booleschen Funktion angeordnet werden.

Folgende zwei Regeln können deshalb aufgeführt werden, wie Variablen für die Darstellung eines ROBDD geordnet werden sollen:

1. Variablen, die inhaltlich zusammengehören, sollten in der Variablenordnung benachbart sein.
2. Variablen, die maßgeblich das Verhalten der booleschen Funktion entscheiden, sollten den Anfang einer Variablenordnung bilden.

Das Berechnen einer optimalen Ordnung ist NP-vollständig

Mit dem vorherigen Abschnitt ist es natürlich naheliegend, für eine boolesche Funktion eine Variablenordnung zu finden, sodaß die Knotenanzahl des reduzierten OBDD minimal wird. Einer der effizientesten veröffentlichten Algorithmen in [FS90] besitzt eine Laufzeit von $O(n^2 3^n)$ bezüglich der Variablenanzahl n. Es ist also kein Algorithmus mit polynomialer Laufzeit bekannt.

Tatsächlich ist dieses Problem NP-vollständig, denn in [BW96] wird gezeigt, daß bereits folgendes Problem NP-vollständig ist:

Eingabe: OBDD G für eine boolesche Funktion g, $s \in \mathbb{N}$

Ausgabe:
- 1, wenn es eine Variablenordnung gibt, mit der g durch ein OBDD mit maximal s Knoten dargestellt werden kann
- 0 sonst

Für in der Praxis übliche boolesche Funktionen können daher nur heuristische Algorithmen zur Suche von geeigneten Variablenordnungen benutzt werden. Zwei Beispiele sind in Abschnitt 2.6.5 skizziert.

Boolesche Funktionen mit stets exponentiellem Wachstum

Oftmals ist die Suche nach geeigneten Variablenordnungen vergeblich. Denn es gibt viele boolesche Funktionen, die unabhängig von der Variablenordnung nur mit exponentiell vielen Knoten in Bezug auf die Variablenanzahl dargestellt werden können.

So benötigen nach [LL92] fast alle booleschen Funktionen sogar mit einer optimalen Variablenordnung mindestens[1]

$$\frac{2^n}{2n}$$

Knoten.

Hierbei ist nach [Weg94] zu berücksichtigen, daß dieses Ergebnis alle booleschen Funktionen einbezieht. In der Praxis finden dagegen hauptsächlich boolesche Funktionen eine Anwendung, die sehr strukturiert und relativ einfach aufgebaut sind. Mit diesen Merkmalen gelingt es dagegen oft, eine Ordnung zu finden, die eine Darstellung durch ROBDDs zuläßt.

Dennoch gibt es in der Praxis wichtige boolesche Funktionen, die nur mit exponentiell vielen Knoten darstellbar sind. Z.B wird in [Bry86] gezeigt, daß die boolesche Funktion für die Multiplikation von den ganzen Zahlen für jede Variablenordnung exponentiell viele Knoten besitzt. Insgesamt muß man bei der Anwendung von ROBDDs darauf vorbereitet sein, daß bestimmte boolesche Funktionen nicht praktikabel dargestellt werden können.

2.5 Negation eines ROBDDs

Sei F ein ROBDD, das die boolesche Funktion f darstellt. Die boolesche Funktion $\neg f$ kann dann von dem ROBDD NOT(F) beschrieben werden, indem die terminalen Knoten **1** und **0** vertauscht werden.

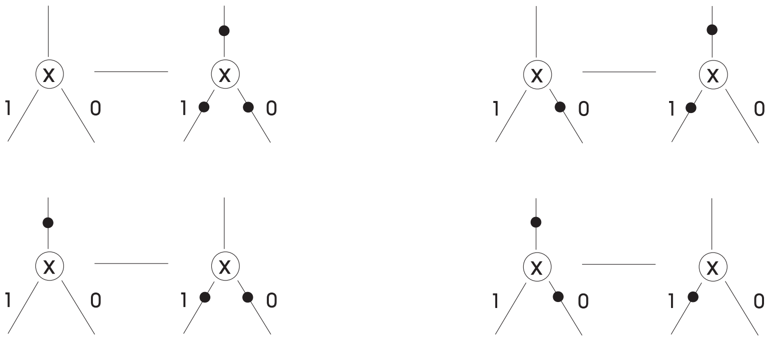
Dieses Vertauschen der terminalen Knoten in F muß nicht explizit durchgeführt werden, sondern es muß lediglich protokolliert werden, in welcher Weise die Beschriftung der terminalen Knoten interpretiert werden soll. Dies gelingt, indem die Kanten eines ROBDDs mit den Elementen aus $\mathbb{B}$ beschriftet werden. Wenn eine Kante nun mit 1 beschriftet ist, müssen die terminalen Knoten vertauscht interpretiert werden, ansonsten liegt das herkömmliche ROBDD vor. Mit dieser Ergänzung wird ein ROBDD nicht durch seinen Wurzelknoten eindeutig bestimmt, sondern es wird zusätzlich eine hinführende Kante zu dem Wurzelknoten benötigt. Je nachdem, wie diese beschriftet ist, wird das ROBDD von dem Wurzelkonten ausgehend interpretiert. Dieses Konzept wird in [MIY90] als ROBDDs mit komplementierbaren Kanten eingeführt.

Wird die Definition von ROBDDs um komplementierbare Kanten erweitert, so können boolesche Formeln und ihre Negation durch dasselbe ROBDD dargestellt werden.

Zum einen kann dadurch die Negation in konstanter Zeit realisiert werden. Zum anderen ist eine weitere Reduktion von ROBDDs möglich. Wenn in einem ROBDD zwei Knoten F, F' existieren, die bis auf Negation dieselben booleschen Funktionen darstellen, so kann der Teilgraph F' eliminiert werden. Dabei werden alle Kanten, die zu dem Knoten F' hinführen, zu F umgeleitet und entsprechend die Kantenbeschriftung aus $\mathbb{B}$ vertauscht.

[1] In [Weg94] ist eine noch präzisere, aber auch sehr viel komplexere Schranke angegeben.

Mit dem Konzept der komplementierbaren Kanten gibt es nun verschiedene Darstellungsarten:



Um die Eindeutigkeit von ROBDDs mit komplementären Kanten zu erhalten, wird in [BRB90] folgende Konvention vorgeschlagen.

Regel 3 Die then-Kante ist nicht komplementiert.

Somit wird zur Darstellung von ROBDDs immer die linke Möglichkeit ausgewählt.

2.6 Implementierung eines ROBDD-Pakets

Die eben vorgestellten Eigenschaften ermöglichen eine sehr effiziente und elegante Implementierung für ROBDDs. Der folgende Abschnitt legt nun dar, wie diese Eigenschaften in einer Implementierung ausgenutzt werden können.

Da diese Arbeit auf ROBDDs aufbaut, soll hier die Effizienz herausgestellt werden und ein Grundverständnis für den Aufbau der Algorithmen nahegebracht werden. Das Zusammenwirken von der Implementierung eines ROBDD-Paketes und deren Umsetzung in Anwendungen, wie z.B. Automatenkonstruktionen, ist dabei von entscheidender Bedeutung.

Viele veröffentlichte Implementierungen basieren dabei auf der Arbeit [BRB90], die hier nun inhaltlicht vorgestellt wird.

2.6.1 Speichern der Knoten

Einer der wichtigsten Eigenschaften von OBDDs für eine Implementierung ist die Eindeutigkeit von ROBDDs. Jeder Knoten eines ROBDDs (ohne komplementierbare Kanten) stellt genau eine boolesche Formel mit den Variablen, die die Unterknoten beschriften, und damit eine boolesche Funktion dar.

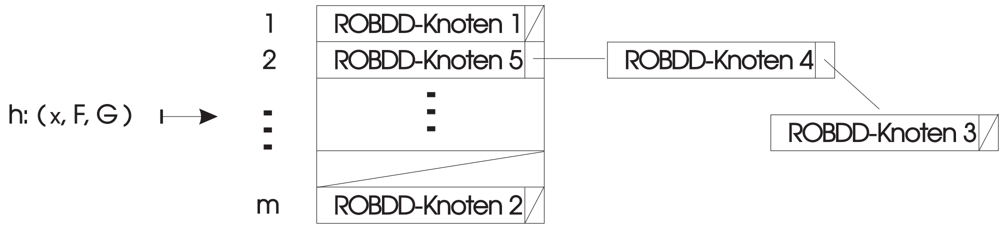
Besitzt nun ein Knoten K in einem ROBDD die Variablenbeschriftung x und die Nachfolgerknoten F an der then- und G an der else-Kante, so wird nach Bemerkung 2.24 der Knoten K eindeutig bezeichnet mit $\mathrm{ITE}(x, F, G)$ oder kurz

$$(x, F, G)$$

Unique-Table

Das Tripel bildet somit einen Schlüssel, um einen Knoten eindeutig zu bezeichnen. Das Zugreifen auf einen Knoten und damit auf ein ROBDD wird nun durch ein erweiterbares Hashverfahren mit

Kollisionsbehandlung realisiert.



Eine Hash-Funktion h bildet dabei das Tripel (x, F, G) auf eine Zeile der Hash-Tabelle ab. In dieser Zeile wird dann der Knoten mit einer entsprechenden Speicheradresse mit folgenden Informationen eingelagert:

- Variable
- Speicheradressen der then- und else-Knoten
- Speicheradresse des nachfolgenden Knoten in der Kollisionskette
- weitere Verwaltungsdaten (z.B. Anzahl der Referenzen)

Da die Hashfunktion nicht injektiv ist, können Knoten auf die gleiche Zeile der Tabelle abgebildet werden. Tritt eine solche Kollision auf, so kann mit einer verketteten Liste auf diese Knoten zugegriffen werden. Ist die Kollisionsrate zu hoch, so wird dynamisch die Zeilenanzahl der Hashtabelle erweitert. Die Hashtabelle mit ihren Kollisionsketten wird in [BRB90] als Unique-Table bezeichnet.

Für die Unique-Table steht die Operation

$$\text{find\_or\_add\_unique\_table}(x, F, G)$$

zur Verfügung, um

- einen Knoten ITE(x, F, G) in die Unique-Table einzulagern, wenn der Knoten sich nicht in der Tabelle befindet und
- die Speicheradresse des Knotens ITE(x, F, G) zu erhalten, der in der Tabelle eingelagert ist.

Negation NOT

Nach Abschnitt 2.5 muß für die Negation NOT(F) eines ROBDDs F lediglich eine komplementierbare Kante eingeführt werden. Das Komplement einer Kante wird nun berücksichtigt, indem ein Bit in der Speicheradresse eines Knotens reserviert wird. Ist das Komplement-Bit gesetzt, so wird der Knoten und das zugehörige ROBDD als negiert interpretiert.

Eindeutigkeit

Da die Regeln (1)-(2) aus Abschnitt 2.3 und Regel (3) aus Abschnitt 2.5 bei einem Aufruf

$$\text{find\_or\_add\_unique\_table}(x, F, G)$$

konsequent angewendet werden, wird jedes OBDD reduziert in der Unique-Table gespeichert. Denn geht man davon aus, daß bisher alle Knoten reduziert und eindeutig in der Unique-Table gespeichert sind, so wird

Regel 1 eingehalten, indem im Fall $F = G$ die Adresse von Knoten F zurückgegeben wird,

Regel 2 automatisch erfüllt, da zwei verschiedene Knoten mit der gleichen *ITE*-Realisierung nicht eingelagert werden können und

Regel 3 entsprochen, wenn Knoten F negiert ist: der Knoten

$$K = \text{find\_or\_add\_unique\_table}(x, \text{NOT}(F), \text{NOT}(G))$$

wird eingelagert und Knoten $\text{NOT}(K)$ zurückgegeben.

Da alle OBDD-Operationen nur mit der Operation find_or_add_unique_table auf OBDD-Knoten zugreifen und diese einlagern können, erzeugen alle OBDD-Operationen nur reduzierte OBDDs. Im folgenden wird also angenommen, daß alle OBDDs in reduzierter Form vorliegen.

Beschreiben nun zwei ROBDDs dieselbe boolesche Funktion, so werden diese durch denselben Knoten in der Unique-Table gespeichert. Zwei ROBDDs stellen deshalb genau dann dieselbe boolesche Funktion dar, wenn die Adressen der ROBDD-Knoten übereinstimmen.

2.6.2 Die *ITE*-Operation

Werden boolesche Funktionen f, g und h von ROBDDs F, G und H dargestellt, so wird mit der Operation $\text{ITE}(F, G, H)$ ein ROBDD erzeugt, das die boolesche Funktion $\text{ite}(f, g, h)$ repräsentiert.

Die zentrale Operation für ROBDDs ist nun dieser *ITE* Operator. Die Operation ist deshalb besonders geeignet, da mit einer *ITE*-Operation und eventuell mehreren NOT-Operationen alle binären booleschen Operationen für ROBDDs umgesetzt werden können (siehe Bemerkung 2.9). Da außerdem jeder Knoten selbst eine *ITE* Operation darstellt, können mit dieser Operation viele andere ROBDD-Operationen realisiert werden.

Rekursive Implementierung

Die Operation $\text{ITE}(F, G, H)$ für ROBDDs läßt sich folgendermaßen rekursiv definieren.

- Sei x die oberste Variable von F, G und H bzgl. ihrer gemeinsamen Variablenordnung, dann gilt

$$\text{ITE}(F, G, H) = \text{ITE}(x, \text{ITE}(F|_{x=1}, G|_{x=1}, H|_{x=1}), \text{ITE}(F|_{x=0}, G|_{x=0}, H|_{x=0}))$$

- Der terminale Fall für die Rekursion ist:

$$\text{ITE}(\mathbf{1}, F, G) = \text{ITE}(\mathbf{0}, G, F) = \text{ITE}(F, \mathbf{1}, \mathbf{0}) = F \text{ und } \text{ITE}(F, \mathbf{0}, \mathbf{1}) = \neg F$$

Cache für Zwischenergebnisse

Ein Merkmal von reduzierten OBDDs besteht darin, daß isomorphe Teilgraphen zu einem Knoten zusammengefaßt werden. Durchläuft der obige Algorithmus nun von dem Wurzelknoten ausgehend alle Pfade bis zu dem **1** und **0** Knoten, so werden diese zusammengefaßten Teilgraphen mehrmals durchlaufen.

Mit Hilfe eines Caches können diese unnötigen Rekursionsaufrufe reduziert werden. Nach Durchlaufen einer Rekursion werden die Knoten $(\text{Ergebnis}, F, G, H)$ mit der Beziehung $\text{Ergebnis} = \text{ITE}(F, G, H)$ mit Hilfe einer Hash-Funktion

$$\text{insert\_in\_ite\_cache}(\text{Ergebnis}, F, G, H)$$

in einen Cache eingelagert und der vorherige Eintrag gelöscht. Zu Beginn jedes Rekursionsaufrufes wird nun abgefragt, ob die Berechnung des Knotens Ergebnis zu (F, G, H) sich in dem Cache befindet. Die Funktion

$$\text{lookup\_in\_ite\_cache}(\&\text{Ergebnis}, F, G, H)$$

gibt bei Erfolg den Wert 1 zurück und schreibt in die Variable Ergebnis die zugehörige Adresse des Knotens $\text{ITE}(F, G, H)$.

Durch dieses Verfahren werden im Normalfall viele Rekursionsschritte eingespart. Geht eine Berechnung im Cache verloren, so ist die Wahrscheinlichkeit sehr hoch, daß die rekursive Berechnung für darunterliegende Knoten noch existiert. Erst wenn die Kapazität des Caches überlastet wird, kann die Wirkungsweise verloren gehen.

Für ROBDDs F, G und H können für den Knoten $\text{ITE}(F, G, H)$ viele Darstellungen existieren, wie z.B.

$$\text{ITE}(F, F, G) = \text{ITE}(F, \mathbf{1}, G) = \text{ITE}(G, \mathbf{1}, G) = \text{ITE}(G, G, F)$$

Aus diesen Klassen mit dem gleichen ROBDD-Knoten wird nach bestimmten Regeln jeweils ein Tripel als Repräsentant ausgewählt. Mit der Funktion

$$\text{ite\_representant}(\&F, \&G, \&H)$$

werden die Knoten zu der eindeutig ausgewählten Darstellung modifiziert. Dadurch kann die Cache-Trefferrate erhöht werden, da weniger verschiedene Einträge erfolgen.

Insgesamt ergibt sich nun der *ITE*-Algorithmus 2.27.

Bemerkung 2.26. Die Laufzeit des *ITE*-Algorithmus 2.6.2 besitzt im schlechtesten Fall die Komplexität $O(|F| \cdot |G| \cdot |H|)$. Denn zur Berechnung von $\text{ITE}(F, G, H)$ wird der Algorithmus maximal $|F| \cdot |G| \cdot |H|$-mal aufgerufen, da es höchstens so viele verschiedene Kombinationen der Knoten gibt. Die Effizienz des Algorithmus hängt also davon ab, wie klein die ROBDDs mit einer geschickten Variablenordnung dargestellt werden können.

Algorithmus 2.27 (*ITE*-Algorithmus).

// Input: ROBDDs F, G, H, die boolesche Funktionen f, g, h kodieren
// Output: ROBDD, das die boolesche Funktion ite(f, g, h) kodiert
bdd ITE(F, G, H)
{
    if(Terminaler Fall)
        return(Ergebnis);
    ite_representant(&F, &G, &H);
    if(lookup_in_ite_cache(&Ergebnis, F, G, H))
        return(Ergebnis);
    x = top_variable(F, G, H);
    Then = ITE$(F|_{x=1}, G|_{x=1}, H|_{x=1})$;
    Else = ITE$(F|_{x=0}, G|_{x=0}, H|_{x=0})$;
    if(Then == Else)
        return(Then);
    Ergebnis = find_or_add_unique_table(x, Then, Else);
    insert_in_ite_cache(Ergebnis, F, G, H);
    return(Ergebnis);

}

2.6.3 Weitere Operationen

In diesem Abschnitt werden weitere wichtige boolesche Operationen für ROBDDs dargestellt, die in dieser Arbeit verwendet werden. Da der grundsätzliche Aufbau dem *ITE*-Algorithmus entspricht, werden nur die Grundideen vorgestellt, wie die booleschen Operationen für ROBDDs implementiert werden.

Restriktion

Sei $f(\boldsymbol{X})$ eine boolesche Funktion. Dann wird die boolesche Operation Restriktion $f|_{x_i=b}$ mit $b \in \mathbb{B}$ und $x_i \in X$ definiert mit

$$f|_{x_i=b} : \begin{cases} \mathbb{B}^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & f(x_0, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_{n-1}) \end{cases}$$

Die boolesche Operation Restriktion kann für das zugehörige ROBDD F realisiert werden, indem alle Kanten, die zu den Entscheidungsknoten mit der Variable x führen, zu dem Knoten an der then-Kante im Fall $b = 1$ und an der else-Kante im Fall $b = 0$ umgelenkt werden. Die ROBDD-Operation wird mit RESTRICT(F, x, b) bezeichnet.

Bemerkung 2.28. Die maximale Laufzeit des Algorithmus RESTRICT besitzt die Komplexität $O(|F|)$, denn im ungünstigsten Fall müssen alle Knoten des ROBDDs durchlaufen werden.

Komposition

Seien $f(\boldsymbol{X})$ und $g(\boldsymbol{X})$ boolesche Funktionen und $x_i \in X$. Dann wird die boolesche Operation Komposition definiert durch

$$f|_{x_i=g} : \begin{cases} \mathbb{B}^n & \rightarrow & \mathbb{B} \\ \boldsymbol{X} & \mapsto & f(x_0, \ldots, x_{i-1}, g(\boldsymbol{X}), x_{i+1}, \ldots, x_{n-1}) \end{cases}$$

Die boolesche Funktion $f|_{x_i=g}$ kann auch mit Hilfe der booleschen Operationen $\wedge$ und Restriktion beschrieben werden:

$$f|_{x_i=g} = g \wedge f|_{x_i=1} \vee \neg g \wedge f|_{x_i=0}$$

Ein ROBDD-Algorithmus mit dieser Methode würde im ungünstigsten Fall eine Laufzeit mit Komplexität $O(|F|^2 \cdot |G|^2)$ besitzen.

Dies kann nach [Bry86] verbessert werden, indem die Komposition durch die *ite*-Operation formuliert wird.

$$f|_{x_i=g} = \text{ite}(g, f|_{x_i=1}, f|_{x_i=0})$$

Wegen Bemerkung 2.26 folgt dann

Bemerkung 2.29. Die Laufzeit der Komposition ITE$(G, F|_{x_i=1}, F|_{x_i=0})$ für ROBDDs F, G besitzt im ungünstigsten Fall die Komplexität $O(|F|^2 \cdot |G|)$.

Quantifizierung

Ist $f(\boldsymbol{X})$ eine boolesche Funktion, so wird die existentielle Quantifizierung $\exists_x f(\boldsymbol{X})$ von f bzgl. einer Variablen $x_i \in X$ definiert durch

$$\exists_{x_i} f = f|_{x_i=1} \vee f|_{x_i=0}$$

Die existentielle Quantifizierung $\exists_V f$ bzgl. der Variablenmenge $V = \{x_{i_0}, \ldots, x_{i_k}\} \subseteq X$ wird nun durch iterative Anwendung von $\exists_{x_{i_j}} f$ erklärt

$$\exists_V f = \exists_{x_{i_0}} \left(\exists_{x_{i_1}} \cdots \left(\exists_{x_{i_k}} f \right) \ldots \right)$$

Die existentielle Quantifizierung $\exists_V f$ einer booleschen Funktion f kann für das ROBDD F zur booleschen Funktion f in einem Algorithmus EXISTS(F, V) formuliert werden. Die Knoten von F werden von diesem rekursiv durchlaufen. Wird ein Knoten K mit Variablenbeschriftung $x \in V$ erreicht, so werden die Restriktionen $K|_{x=1}$ und $K|_{x=0}$ mit den THEN und ELSE Funktionen umgesetzt und die Rekursion für tiefer liegende Variablen aus V fortgesetzt. Die Ergebnisse der then- und else-Knoten werden anschließend mit der OR-Operation verknüpft und so die x Variable aus dem ROBDD existentiell quantifiziert.

Algorithmus 2.30 (Existentielle Quantifizierung).

```
// Input: ROBDD F, das die boolesche Funktion f kodiert, Variablenmenge V
// Output: ROBDD, das die boolesche Funktion ∃_V f kodiert
bdd EXISTS(F, V)
{
  (1)   if(F == 1 || F == 0)
  (2)         return(F);
  (3)   x = top_variable(F);
  (4)   if(Min(V) > x)
  (5)         return(F);
  (6)   if(lookup_in_exists_cache(&Ergebnis, F, V))
  (7)         return(Ergebnis);
  (8)   Then = EXISTS(F|x=1, V);
  (9)   Else = EXISTS(F|x=0, V);
 (10)   if(x ∈ V)
 (11)         Ergebnis = OR(Then, Else);
        else
 (12)         Ergebnis = find_or_add_unique_table(x, Then, Else);
 (13)   insert_in_exists_cache(Ergebnis, F, V);
 (14)   return(Ergebnis);
}
```

Die universelle Quantifizierung von f bzgl. der Variablenmenge V wird analog definiert durch

$$\forall_V f = \forall_{x_{i_0}} \left(\forall_{x_{i_1}} \cdots \left(\forall_{x_{i_k}} f \right) \ldots \right)$$

wobei

$$\forall_{x_i} f = f|_{x_i=1} \wedge f|_{x_i=0}$$

Der zugehörige Algorithmus FORALL(F, V) für ROBDDs kann nun entweder umgesetzt werden, indem in Zeile 11 von Algorithmus 2.30 die OR-Operation mit einer AND-Operation vertauscht wird, oder er kann dargestellt werden durch die Operation NOT und EXISTS wegen

$$\forall_V f = \neg \exists (\neg f)$$

Relationales Produkt

Das relationale Produkt für boolesche Funktionen $f(\boldsymbol{X}), g(\boldsymbol{X})$ bzgl. der Variablenmenge V wird definiert durch

$$\exists_V \, [f \wedge g]$$

Wird diese Operation für ROBDDs F, G realisiert, indem zuerst die AND-Operation ausgeführt und anschließend mit EXISTS quantifiziert wird, so wird das ROBDD in vielen Fällen mit der AND-Verknüpfung sehr groß und schrumpft mit der Quantifizierung wieder zusammen.

Diese Operation wird für die Automatenkonstruktionen eine herausragende Rolle spielen. Ein effizienter Algorithmus ist deshalb entscheidend für das Anwenden der Konstruktionen in dieser Arbeit.

Der Algorithmus besitzt den Aufbau von Algorithmus 2.30 der existentiellen Quantifizierung. Zusätzlich wird in dem rekursiven Durchlauf die ROBDD-Verknüpfung AND von F und G in dem terminalen Fall der Rekursion durchgeführt. Indem die Operationen AND und EXISTS in einen Algorithmus zusammengefaßt werden, gelingt es in vielen Fällen, diese Operation praktikabel durchzuführen.

Algorithmus 2.31 (Relationales Produkt).

```
// Input: ROBDDs F, G, die die booleschen Funktionen f, g kodieren, Variablenmenge V
// Output: ROBDD, das die boolesche Funktion ∃_V(f ∧ g) kodiert
bdd RELPROD(F, V)
{
        if(F == 0 || G == 0)
                return(0);
        if(F == 1 && G == 1)
                return(1);
        x = top_variable(F, G);
        if(Min(V) > x)
                return(F);
        if(lookup_in_relprod_cache(&Ergebnis, F, G, V))
                return(Ergebnis);
        Then = RELPROD(F|x=1, G|x=1, V);
        Else = RELPROD(F|x=0, G|x=0, V);
        if(x ∈ V)
                Ergebnis = OR(Then, Else);
        else
                Ergebnis = find_or_add_unique_table(x, Then, Else);
        insert_in_relprod_cache(Ergebnis, F, G, V);
        return(Ergebnis);

}
```

Bemerkung 2.32. Im ungünstigsten Fall verhält sich der Algorithmus folgendermaßen

1. Berechne $H = \text{AND}(F, G) = \text{ITE}(F, G, \mathbf{0})$ - Zeitkomplexität $O(|F||G|)$, d.h. maximale ROBDD-Größe $|H| = |F||G|$
2. Berechne EXISTS(H, V) - Zeitkomplexität $O(|H|^{2^{|V|}})$

d.h. die worst case Laufzeit-Komplexität beträgt $O((|F||G|)^{2^{|V|}})$.

Diese Laufzeit ist erschreckend hoch. In der Praxis tritt dieser Fall nach [BCL+94] üblicherweise nur dann auf, wenn die Knotenanzahl des berechneten ROBDDs im Bezug auf die Größe von F und G exponentiell wächst. Dann muß jedoch jeder ROBDD-Algorithmus eine exponentielle Laufzeit besitzen.

2.6.4 Garbage Collecting

Werden auf ROBDDs die obigen Algorithmen angewendet, so entstehen viele Knoten, die nur temporär benötigt werden und anschließend als "tote" Knoten den Speicherplatz belegen. So werden in Algorithmus 2.31 Knoten "Then" und "Else" erzeugt, die nach Zeile (13) nicht mehr verwendet werden.

Übersteigt die Knotenanzahl eine gewisse Grenze, so werden mit Garbage Collecting die nicht verwendeten Knoten aus der Unique-Table entfernt. Für dieses Verfahren wird das Konzept der Referenz-Zähler eingesetzt. Ist ein Referenz-Zähler ungleich 0, so ist das ROBDD mit diesem Wurzelknoten reserviert. Der Wurzelknoten und alle Knoten, die von dem Wurzelknoten erreichbar sind, werden vom Garbage Collecting nicht gelöscht. Es gibt dabei zwei Arten von Referenz-Zählern.

- Interne/Temporäre Referenz-Zähler
  Verwendet einer der obigen Algorithmen ein ROBDD mit Wurzelknoten K, so wird der interne Referenz-Zähler inkrementiert, um sicherzustellen, daß K nicht gelöscht werden kann. Wird der Knoten K von dem Algorithmus nicht mehr benötigt, so wird der temporäre Referenz-Zähler wieder dekrementiert.
- Externe Referenz-Zähler
  Liefert eine Operation dem Anwender ein ROBDD zurück, so wird der temporäre Referenz-Zähler dekrementiert und der externe Referenz-Zähler inkrementiert. Dieses ROBDD ist nun solange im Speicher resident, bis der Anwender dieses mit dem Befehl FREE freigibt. Dies wird realisiert, indem der externe Referenz-Zähler dekrementiert wird.

Wenn die Knotenanzahl während einer Operation die obere Grenze überschreitet, so wird das Verfahren Garbage Collecting aktiviert. Dieses entfernt dann alle Knoten, die von den internen und externen Referenz-Zählern nicht gesichert sind.

Werden nicht ausreichend viele Knoten entfernt, so kann die zuletzt aufgerufene Operation nicht beendet werden. In diesem Fall wird die Operation terminiert und an den Anwender der Wert 0 zurückgegeben. Alle temporären Knoten, auf die der Anwender keinen Zugriff hat, werden dann entfernt. Der Anwender hat nun die Möglichkeit, ROBDDs mit FREE freizugeben und die Operation neu zu starten.

Aus diesem Grund sind im folgenden alle Algorithmen so aufgebaut, daß ROBDDs so kurz wie möglich reserviert werden. Der Abbruch einer Operation kann damit eventuell vermieden werden.

2.6.5 Dynamische Algorithmen zur Neuordnung von Variablen

Wie in Abschnitt 2.4.2 bereits erwähnt, besitzt einer der effizientesten Algorithmen zur Berechnung einer optimalen Variablenordnung die Komplexität $O(\frac{2^n}{2n})$. Deshalb werden intensiv heuristische Verfahren eingesetzt, um die Knotenanzahl eines ROBDDs möglichst gering zu halten.

Bekannte heuristische Algorithmen [YBSV93], [Rud93] basieren auf Fenster-Strategien. Die Idee besteht darin, eine gewisse Anzahl von Variablen, die in der Ordnung benachbart sind, auszuwählen. So wird ein Ausschnitt/Fenster der ROBDDs bestimmt, in dem nur die Knoten betrachtet werden, die mit diesen ausgewählten Variablen beschriftet sind. Innerhalb des Fensters können nun je zwei Variablen simultan substituiert werden. Anschließend werden die Variablen umbenannt. So kann die Ordnung mit Transpositionen permutiert werden. Folgende Strategien mit unterschiedlicher Geschwindigkeit und Wirkungsgrad haben sich hierbei in der Implementierung [Lon93] bewährt.

1. "reorder_sift"
   Jeweils eine Variable wird durch alle Variablen "durchgeschoben", indem sie mit der aktuell benachbarten Variable vertauscht wird. Die Variable wird anschließend an die Stelle in der Variablenordnung positioniert, mit der die geringste Knotenanzahl entsteht. Diese Strategie bewirkt sehr gute Ergebnisse und ist sehr schnell, da der benachbarte Variablenaustausch sehr effizient durchgeführt werden kann.

2. "reorder_stable_window3"
   Jeweils drei benachbarte Variablen werden solange ausgetauscht, bis die Knotenanzahl "stabil" bleibt. Dieses Verfahren ist schneller, nähert sich jedoch im Vergleich zu dem ersten Verfahren nicht so gut an eine optimale Ordnung an.

Diese Strategien werden dynamisch aktiviert. Wird in einer Operation die Knotenanzahl überschritten, so wird die Operation zurückgesetzt (siehe Kapitel 2.6.4) und ein Verfahren zum Neuordnen der Variablen gestartet. Anschließend wird die abgebrochene Operation wiederholt.

In der Implementierung [Lon93] werden die Fenster vom Anwender durch Blöcke definiert. Hier kann eine Hierarchie von Blöcken aufgebaut werden. Nur in der tiefsten Ebene werden direkt Variablen permutiert. In höheren Ebenen werden die Blöcke in ihrer Ebene vertauscht, ohne die Struktur in den tieferen Ebenen zu verändern. So werden zusammengehörige Variablenblöcke (siehe Kapitel 2.4.2) nicht getrennt.

2.7 Boolesche Funktionen, ROBDDs und die Algorithmen für Automatenkonstruktionen

Im nächsten Kapitel werden Automaten mit Hilfe von booleschen Funktionen definiert.

Automatenkonstruktionen können nun gelingen, indem die booleschen Funktionen der Automaten mit Hilfe von booleschen Operationen modifiziert werden.

Die booleschen Funktionen dieser Automaten werden dabei mit Hilfe von ROBDDs dargestellt. Durch diese Festlegung auf eine Darstellungsform der booleschen Funktionen können die Algorithmen derart optimiert werden, daß möglichst kleine ROBDDs erzeugt werden.

In den folgenden Algorithmen wird deshalb grundsätzlich angenommen, daß die booleschen Funktionen mit Hilfe von ROBDDs dargestellt werden.

Der Zugriff auf ein OBDD A wird durch einen Zeiger auf den Wurzelknoten ermöglicht. Wenn im folgenden von einem OBDD A gesprochen wird, so wird dies in einem Algorithmus umgesetzt, indem in Variable A der Zeiger des Wurzelknotens von dem OBDD A gespeichert wird.

Zuweisungen wie $B = A$ bewirken im folgenden, daß der OBDD-Zeiger B mit dem OBDD-Zeiger A überschrieben wird. Mit Zeiger B kann dann auf dasselbe OBDD wie mit dem Zeiger A mit Hilfe der OBDD-Operationen zugegriffen werden.

Gibt eine OBDD-Operation ein OBDD A zurück, so belegt dieses solange Speicherressourcen, bis der Anwender das OBDD A mit FREE(A) wieder freigibt (siehe Abschnitt 2.6.4). Im folgenden wird daher darauf geachtet, daß nicht mehr benötigte OBDDs sofort wieder freigegeben werden.

Folgende Notation wird hierfür eingeführt. Wenn einer Variable A^- in einem Operationsbefehl ein $^-$ angefügt ist, dann wird nach der Ausführung der Operation das ursprüngliche OBDD A implizit freigegeben, d.h. der externe Referenzzähler wird dekrementiert.

Will der Anwender umgekehrt eine Kopie des OBDDs A erstellen, so wird dies in einer Algorithmuszeile mit A^+ verdeutlicht. Dies bedeutet dann, daß vor der Ausführung der Zeile der externe Referenzzähler des Wurzelknotens A inkrementiert wird.

Die Zeichen $^+$ und $^-$ dienen daher nur zur Verwaltung des Speichers der OBDDs. Zum Verständnis der Algorithmen selbst können diese überlesen werden, wenn die Konstruktionen der Automaten im Mittelpunkt stehen.

Die grundsätzliche Idee für eine Automatenkonstruktion wird im folgenden auf der semantischen Ebene der booleschen Funktionen und booleschen Operationen vorgestellt. Dadurch wird hervorgehoben, daß die prinzipielle Idee der Algorithmen nicht an eine bestimmte Darstellungsform wie z.B. den OBDDs gebunden ist.

Außerdem ist die Notation der booleschen Operationen

$$f \wedge g, f \vee g, f|_{x=b}, \exists_V f, \forall_V g, \exists_V [f \wedge g], \ldots$$

leichter zu lesen als die Notation der zugehörigen OBDD-Operationen

$$\mathrm{AND}(f, g), \mathrm{OR}(f, g), \mathrm{RESTRICT}(F, x, b), \mathrm{EXISTS}(f, V), \mathrm{FORALL}(f, V), \mathrm{RELPROD}(f, g, V), \ldots$$

Diese semantische Notation wird nun auf die vorzustellenden Algorithmen übertragen. Die Darstellung der booleschen Funktionen durch OBDDs wird im folgenden nur dann berücksichtigt, wenn die Speicherverwaltung der OBDDs mit den Symbolen $^+$ und $^-$ berücksichtigt wird oder spezielle Operationen für die Datenstruktur OBDD in den Algorithmen verwendet werden.

3 Automaten

In diesem Abschnitt werden verschiedene Möglichkeiten beschrieben, wie Automaten dargestellt werden können.

3.1 Sequentielle Maschinen

Sequentielle Maschinen besitzen folgenden Aufbau.

Definition 3.1. Eine *sequentielle Maschine* mit m Eingaben und n Ausgaben ist ein Sechstupel

$$M = (Q, X, Y, \delta, \lambda, s)$$

mit

- Q ist eine endliche nichtleere Menge von Zuständen.
- X ist ein endliches nichtleeres Eingabealphabet.
- Y ist ein endliches nichtleeres Ausgabealphabet.
- $\delta : X^m \times Q \to Q$ heißt Transitionsfunktion.
- $\lambda : X^m \times Q \to (Y \cup \{\epsilon\})^n$ heißt Ausgabefunktion, wobei ϵ das leere Wort darstellt.
- $s \in Q$ ist ein Startzustand.

Ist die Ausgabefunktion eingeschränkt auf $\lambda : X^m \times Q \to Y^n$, so heißt die sequentielle Maschine M auch *Mealy Maschine* mit m Eingaben und n Ausgaben.

Seien

$$\begin{aligned} \xi_i &= \langle x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(t)}, \ldots \rangle \in X^\omega \text{ für } 0 \le i \le m-1 \\ \upsilon_i &= \langle y_i^{(0)}, y_i^{(1)}, \ldots, y_i^{(t)}, \ldots \rangle \in (Y \cup \{\epsilon\})^\omega \text{ für } 0 \le i \le n-1 \end{aligned}$$

M berechnet zu der Eingabefolge $\boldsymbol{\xi} = \langle \xi_0, \ldots \xi_{m-1} \rangle$ die Ausgabefolge $\boldsymbol{\upsilon} = \langle \upsilon_0, \ldots \upsilon_{n-1} \rangle$, wenn es eine unendliche Zustandsfolge $\langle q^{(0)}, q^{(1)}, \ldots, q^{(t)}, \ldots \rangle \in Q^\omega$ mit dem Startzustand $q^{(0)} = s$ gibt, sodaß im t-ten Berechnungsschritt zur Eingabe $\boldsymbol{x}^{(t)} = \langle x_0^{(t)}, x_1^{(t)} \ldots, x_{m-1}^{(t)} \rangle$ die Ausgabe $\boldsymbol{y}^{(t)} = (y_0^{(t)}, y_1^{(t)} \ldots, y_{n-1}^{(t)})$ berechnet wird mit

$$\begin{aligned} \delta(\boldsymbol{x}^{(t)}, q^{(t)}) &= q^{(t+1)} \\ \lambda(\boldsymbol{x}^{(t)}, q^{(t)}) &= \boldsymbol{y}^{(t)} \end{aligned}$$

3.2 Synchrone Schaltkreise mit Ausgabeberechtigung

Sequentielle Maschinen sind in dem Sinn abstrakt, da sie unabhängig bzgl. einer konkreten Realisierung formuliert sind. Mit synchronen Schaltkreisen mit Ausgabeberechtigung können sequentielle Maschinen sehr hardwarenah beschrieben werden. So wird die Definition von sequentiellen Maschinen folgendermaßen konkretisiert:

- Die Ein-, Ausgabealphabete werden eingeschränkt auf $\mathbb{B}$.

- Die Zustände werden kodiert in $\mathbb{B}^l$.
- Die Übergangsfunktion δ und Ausgabefunktion λ werden entsprechend der binären Kodierung der Eingabe-, Ausgabewörter und Zustände dargestellt durch boolesche Funktionsvektoren.
- Die Ausgabe des leeren Wortes ϵ wird realisiert durch eine boolesche Funktion, genannt Ausgabeberechtigungsfunktion. Diese bestimmt, ob die Berechnung von λ ausgegeben oder gesperrt wird.

Definition 3.2. Sei $l \in \mathbb{N}_0 \cup \{\infty\}$, $m \in \mathbb{N}_0$ und $n \in \mathbb{N}$.
Seien folgende Variablenmengen und Vektoren definiert:

- Eingabevariablen $X = \{X_0, \ldots, X_{m-1}\}$ mit $\boldsymbol{X} = \langle X_0, \ldots, X_{m-1}\rangle$
- Zustandsvariablen $Q = \{Q_0, \ldots, Q_{l-1}\}$ mit $\boldsymbol{Q} = \langle Q_0, \ldots, Q_{l-1}\rangle$.
- Folgezustandsvariablen $Q' = \{Q'_0, \ldots, Q'_{l-1}\}$ mit $\boldsymbol{Q}' = \langle Q'_0, \ldots, Q'_{l-1}\rangle$.
- Hilfsausgabevariablen $H = \{H_0, \ldots, H_{n-1}\}$ mit $\boldsymbol{H} = \langle H_0, \ldots, H_{n-1}\rangle$
- Ausgabeberechtigungsvariablen $A = \{A_0, \ldots, A_{n-1}\}$ mit $\boldsymbol{A} = \langle A_0, \ldots, A_{n-1}\rangle$.

Ein *synchroner Schaltkreis mit Ausgabeberechtigung* $\mathcal{S}$, der m *Eingaben*, n *Ausgaben* und den *Zustandsraum* $\mathbb{B}^l$ besitzt

- kurz: ein Automat $\mathcal{S}$ vom Typ (l, m, n) -

ist ein Quadrupel $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ mit den booleschen Funktionsvektoren

- δ zur Berechnung des Folgezustands:

$$\delta : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow \quad \mathbb{B}^l \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto \quad \boldsymbol{Q}' \end{cases}$$

  wobei $Q'_i = \delta_i(\boldsymbol{X}, \boldsymbol{Q})$ und $\delta_i : \mathbb{B}^m \times \mathbb{B}^l \rightarrow \mathbb{B}$ die i-te Komponentenfunktion von δ bezeichnet. δ heißt *Übergangsfunktion*.

- λ zur Berechnung der Hilfsausgabe:

$$\lambda : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow \quad \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto \quad \boldsymbol{H} \end{cases}$$

  wobei $H_i = \lambda_i(\boldsymbol{X}, \boldsymbol{Q})$ und $\lambda_i : \mathbb{B}^m \times \mathbb{B}^l \rightarrow \mathbb{B}$ die i-te Komponentenfunktion von λ bezeichnet. λ heißt *Hilfsausgabefunktion*.

- α zur Berechnung der Ausgabeberechtigung:

$$\alpha : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow \quad \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto \quad \boldsymbol{A} \end{cases}$$

  wobei $A_i = \alpha_i(\boldsymbol{X}, \boldsymbol{Q})$ und $\alpha_i : \mathbb{B}^m \times \mathbb{B}^l \rightarrow \mathbb{B}$ die i-te Komponentenfunktion von α bezeichnet. α heißt *Ausgabeberechtigungsfunktion*.

- und dem *Startzustand* $\boldsymbol{s} = (s_0, \ldots, s_{l-1}) \in \mathbb{B}^l$.

Ein synchroner Schaltkreis mit Ausgabeberechtigung heißt endlich, wenn der Zustandsraum $\mathbb{B}^l$ endlich ist, d.h. $l \in \mathbb{N}_0$.

Notation 3.3. Sind $\boldsymbol{Y} = \langle Y_0, \ldots, Y_k \rangle$ und $\boldsymbol{Z} = \langle Z_0, \ldots, Z_l \rangle$ Vektoren von Variablen, so wird $\langle \boldsymbol{Y}, \boldsymbol{Z} \rangle$ für

$$\langle Y_0, \ldots, Y_k, Z_0, \ldots, Z_l \rangle$$

und $\langle \boldsymbol{Y}, Z_0, \ldots, Z_r \rangle$ für

$$\langle Y_0, \ldots, Y_k, Z_0, \ldots, Z_r \rangle$$

notiert.

Seien $\boldsymbol{s} = (s_0, \ldots, s_{l-1}) \in \mathbb{B}^l$, $\boldsymbol{t} = (t_0, \ldots, t_{q-1}) \in \mathbb{B}^q$. Die Konkatenation

$$(s_0, \ldots, s_{l-1}, t_0, \ldots, t_{q-1})$$

der Wörter $\boldsymbol{s}$ und $\boldsymbol{t}$ wird mit $\boldsymbol{s}.\boldsymbol{t}$ notiert.

Bemerkung 3.4. Nur in diesem Kapitel wird der Fall der unendlichen synchronen Schaltkreise mit Ausgabeberechtigung berücksichtigt. In allen weiteren Kapiteln werden nur endliche synchrone Schaltkreise mit Ausgabeberechtigung behandelt. Deshalb wird im folgenden auf den Terminus "endlich" verzichtet. Wenn der unendliche Fall betrachtet wird, so wird im folgenden darauf hingewiesen.

Da ein endlicher synchroner Schaltkreis mit Ausgabeberechtigung eine Konkretisierung einer sequentiellen Maschine darstellt und umgekehrt ein endlicher synchroner Schaltkreis mit Ausgabeberechtigung von einer sequentiellen Maschine beschrieben werden kann, können beide Konzepte den gleichen Automatentyp darstellen.

Die Funktionsweise eines synchronen Schaltkreises wird definiert durch

Definition 3.5 (Fortsetzung 1). Seien

$$\begin{aligned} \xi_i &= \langle x_i^{(0)}, x_i^{(1)}, \ldots, x_i^{(t)}, \ldots \rangle \in \mathbb{B}^\omega \text{ für } 0 \leq i \leq m-1 \\ \upsilon_i &= \langle y_i^{(0)}, y_i^{(1)}, \ldots, y_i^{(t)}, \ldots \rangle \in (\mathbb{B} \cup \{\epsilon\})^\omega \text{ für } 0 \leq i \leq n-1 \end{aligned}$$

$\mathcal{S}$ berechnet zu der Eingabefolge $\boldsymbol{\xi} = \langle \xi_0, \ldots \xi_{m-1} \rangle$ die Ausgabenfolge $\boldsymbol{\upsilon} = \langle \upsilon_0, \ldots \upsilon_{n-1} \rangle$, wenn es eine unendliche Zustandsfolge $\langle q^{(0)}, q^{(1)}, \ldots, q^{(t)}, \ldots \rangle \in (\mathbb{B}^l)^\omega$ mit dem Startzustand $q^{(0)} = \boldsymbol{s}$ gibt, sodaß im t-ten Berechnungsschritt zur Eingabe $\boldsymbol{x}^{(t)} = \langle x_0^{(t)}, x_1^{(t)} \ldots, x_{m-1}^{(t)} \rangle$ die Ausgabe $\boldsymbol{y}^{(t)} = (y_0^{(t)}, y_1^{(t)} \ldots, y_{n-1}^{(t)})$ berechnet wird mit

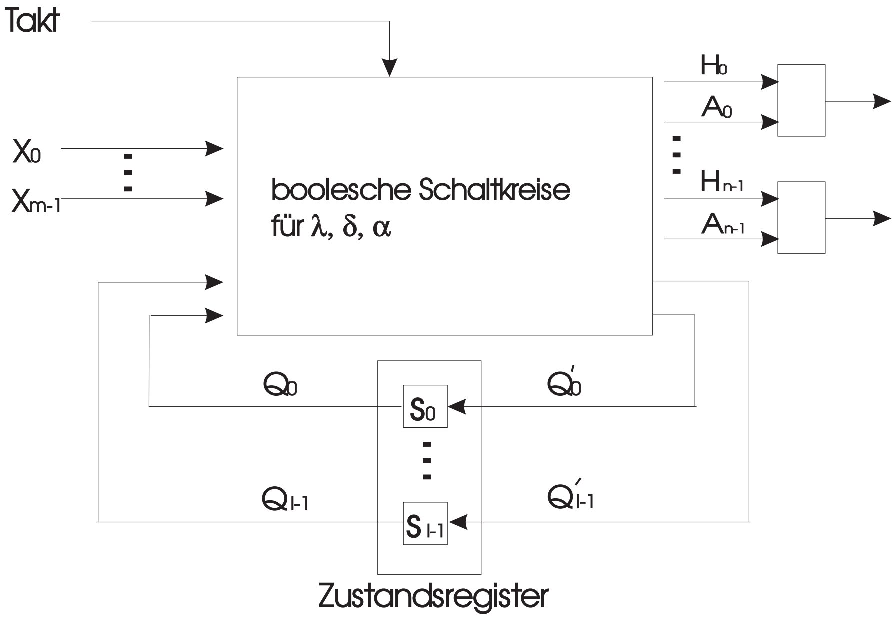
$$q^{(t+1)} = \delta(\boldsymbol{x}^{(t)}, q^{(t)})$$

und

$$y_i^{(t)} = \begin{cases} \lambda_i(\boldsymbol{x}^{(t)}, q^{(t)}), & \text{wenn} \quad \alpha_i(\boldsymbol{x}^{(t)}, q^{(t)}) = 1 \\ \epsilon, & \text{wenn} \quad \alpha_i(\boldsymbol{x}^{(t)}, q^{(t)}) = 0 \end{cases} \tag{1}$$

für alle $0 \leq i < n$ und $t \geq 0$.

Die obige Beschreibung eines synchronen Schaltkreises kann schematisch dargestellt werden mit:



Der Zustand eines synchronen Schaltkreises wird bestimmt durch den Registerinhalt, dargestellt durch Quadrate. Im Anfangszustand sind die Register mit dem Startzustand $\boldsymbol{s}$ initialisiert. Nach einem Taktimpuls lesen die booleschen Schaltkreise δ, λ, α

- die Eingabebewertung der Eingabevariablen[2] $\boldsymbol{X}$ und
- die Zustandsbewertung aus den aktuellen Zustandsvariablen $\boldsymbol{Q}$

ein und liefern ihre Ergebnisse in den

- Hilfsausgabevariablen $\boldsymbol{H}$,
- Ausgabeberechtigungsvariablen $\boldsymbol{A}$ und
- Folgezustandsvariablen $\boldsymbol{Q}'$

zurück. Aus den Bewertungen der $\boldsymbol{A}$ und $\boldsymbol{H}$ werden dann entsprechend (1) die Ausgaben ausgegeben. Abschließend wird die Bewertung der Folgezustandsvariablen in den Zustandsregistern für den nächsten Berechnungsschritt eingelagert.

Definition 3.6 (Fortsetzung 2). Ein synchroner Schaltkreis mit Ausgabeberechtigung heißt

- *synchroner Schaltkreis mit gemeinsamer Ausgabeberechtigung*, wenn

$$\alpha_0 = \alpha_1 = \ldots = \alpha_{n-1}$$

  gilt. Die Ausgabeberechtigung kann dann dargestellt werden durch

$$\alpha : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \to & \mathbb{B} \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & \alpha_0(\boldsymbol{X}, \boldsymbol{Q}) \end{cases}$$

[2] Die Variablen können als Bezeichnung der Leitungen in einem Schaltkreis interpretiert werden.

und die Ausgabeberechnung wird vereinfacht zu

$$\boldsymbol{y}^{(t)} = (y_0^{(t)}, \ldots, y_{n-1}^{(t)}) = \begin{cases} \lambda(\boldsymbol{x}^{(t)}, q^{(t)}), & \text{wenn} \quad \alpha(\boldsymbol{x}^{(t)}, q^{(t)}) = 1 \\ (\epsilon \ldots, \epsilon), & \text{wenn} \quad \alpha(\boldsymbol{x}^{(t)}, q^{(t)}) = 0 \end{cases}$$

- *synchroner Schaltkreis* oder *Mealy Automat*, wenn die Ausgabeberechtigung die konstante 1-Funktion ist. Die Ausgabeberechnung vereinfacht sich dann zu

  $$\boldsymbol{y}^{(t)} = \lambda(\boldsymbol{x}^{(t)}, q^{(t)})$$

  λ heißt in diesem Fall *Ausgabefunktion*.

3.3 Online und offline Funktionen

Soll zu einer Eingabefolge $\boldsymbol{\xi} \in (\mathbb{B}^\omega)^m$ nur das Ausgabeverhalten $\boldsymbol{\upsilon} \in (\mathbb{B}^\omega)^n$ eines Mealy Automaten vom Typ (l, m, n) betrachtet werden, so kann die Beschreibung der Zustände und ihre Übergangsfunktion sehr aufwendig sein (siehe Definition 3.5). Im folgenden werden online Funktionen eingeführt, mit deren Hilfe das Ausgabeverhalten eines synchronen Schaltkreises sehr elegant dargestellt werden kann.

Im folgenden wird außerdem zu einem synchronen Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n) nicht das Ausgabeverhalten im t-ten Berechnungsschritt betrachtet, sondern die *i-te Ausgabe*

$$a^{(i)} = (y_0^{(s_0^{(i)})}, \dots, y_{n-1}^{(s_{n-1}^{(i)})})$$

wobei die Ausgabeberechtigung α_j im Berechnungsschritt $s_j^{(i)}$ das i-te Mal eine 1 berechnet, d.h. $\mathcal{S}$ berechnet zum i-ten Mal in der Ausgabekomponente j die Ausgabe $y_{n-1}^{(s_j^{(i)})} \in \mathbb{B}$. Soll das Ausgabeverhalten der i-ten Ausgabe betrachtet werden, so kann dieses in vielen Fällen mit Hilfe von offline Funktionen beschrieben werden.

Unendliche Folgen $\boldsymbol{\xi} \in \mathbb{B}^\omega$ können mit Hilfe der ganzen 2-adischen Zahlen beschrieben werden.

Definition 3.7. Eine *ganze* 2-*adische Zahl* ist eine formale Potenzreihe

$$\sum_{t \geq 0} b^{(t)} 2^t$$

mit den Koeffizienten $b^{(t)} \in \mathbb{B}$. Die Menge der ganzen 2-adischen Zahlen wird mit ${}_2\mathbb{Z}$ bezeichnet. $\sum_{t \geq 0} 2^t$ wird im folgenden auch mit 1^ω und $\sum_{t \geq 0} 0 \cdot 2^t$ auch mit 0^ω bezeichnet.

Das Ausgabeverhalten der i-ten Ausgabe eines synchronen Schaltkreises mit Ausgabeberechtigung $\mathcal{S}$ kann nun mit Hilfe einer Funktion

$$f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$$

über den ganzen 2-adischen Zahlen dargestellt werden. Sei $\boldsymbol{X} = \langle X_0, \dots, X_{m-1} \rangle$ mit

$$X_j = \langle x_i^{(0)}, x_i^{(1)}, \dots \rangle \in \mathbb{B}^\omega$$

eine Eingabefolge und $\boldsymbol{\xi} = \langle \xi_0, \dots, \xi_{m-1} \rangle$ mit

$$\xi_j = \sum_{t \geq 0} x_i^{(t)} 2^t \in {}_2\mathbb{Z}$$

ein zugehöriger Vektor von 2-adische Zahlen. Dann berechnet $\mathcal{S}$ die Funktion f, wenn die i-te Ausgabe $a^{(i)}$ von $\mathcal{S}$ für jede Eingabefolge $\boldsymbol{X} \in (\mathbb{B}^\omega)^m$ dem Vektor

$$\boldsymbol{z^{(i)}} = \langle z_0^{(i)}, \dots, z_{n-1}^{(i)} \rangle$$

von $f(\boldsymbol{\xi}) = \langle \zeta_0, \dots, \zeta_{n-1} \rangle$ mit

$$\zeta_j = \sum_{t \geq 0} z_i^{(t)} 2^t \in {}_2\mathbb{Z}^n$$

entspricht. $\mathcal{S}$ muß also für jede Eingabefolge und für jede Ausgabekomponente unendlich oft den booleschen Wert 0 oder 1 ausgeben können. Es gilt damit folgendes Lemma.

Lemma 3.8. *Ist $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n), dann gilt:*

$\mathcal{S}$ berechnet eine Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ über den ganzen 2-adischen Zahlen genau dann, wenn für jedes Eingabetupel $\boldsymbol{\xi} \in (\mathbb{B}^\omega)^m$ jede Ausgabeberechtigungsfunktion α_i für $(0 \leq i < n)$ in den Berechnungsschritten $t \geq 0$ unendlich oft den booleschen Wert 1 berechnet.

Mit Hilfe folgender Metrik können 2-adische Zahlen bzgl. ihrer Koeffizienten verglichen werden.

Definition 3.9. Sei $\xi \in {}_2\mathbb{Z}$ mit $\xi = 2^k \sum_{t \geq 0} \upsilon^{(t)} 2^t$, $k \in \mathbb{N}_0$ und $\upsilon_0 = 1$. Dann wird k als die *Ordnung* von ξ bezeichnet: $\mathrm{ord}(\xi) = k$. Weiter gilt $\mathrm{ord}(0) := \infty$.

Seien $\boldsymbol{\xi} = (\xi_0, \ldots, \xi_{m-1}) \in {}_2\mathbb{Z}^m$ und $\boldsymbol{\upsilon} = (\upsilon_0, \ldots, \upsilon_{m-1}) \in {}_2\mathbb{Z}^m$. Mit ord wird nun die *Metrik* $\mu_m : {}_2\mathbb{Z}^m \times {}_2\mathbb{Z}^m \to \mathbb{Q}$ definiert mit

$$\mu_m(\boldsymbol{\xi}, \boldsymbol{\upsilon}) := \max\{\mu_1(\xi_i, \upsilon_i) \mid 0 \leq i < m\}$$

wobei

$$\mu_1(\xi_i, \upsilon_i) = \begin{cases} 2^{-\mathrm{ord}(\xi_i, \upsilon_i)}, & \text{wenn} \quad \xi_i \neq \upsilon_i \\ 0, & \text{wenn} \quad \xi_i = \upsilon_i \end{cases}$$

$\mu_m(\boldsymbol{\xi}, \boldsymbol{\upsilon})$ wird auch mit ${}_2|\boldsymbol{\xi} - \boldsymbol{\upsilon}|$ notiert.

Zwei 2-adische Vektoren $\boldsymbol{\xi}, \boldsymbol{\upsilon} \in {}_2\mathbb{Z}^m$ besitzen also den Abstand 2^{-t}, wenn sich $\boldsymbol{\xi}$ und $\boldsymbol{\upsilon}$ erst nach t Koeffiziententupeln unterscheiden, und den Abstand 0, wenn $\boldsymbol{\xi} = \boldsymbol{\upsilon}$ gilt.

Eine Anfangsfolge von einer 2-adischen Zahl wird mit folgender Notation dargestellt.

Notation 3.10. Ist $\xi = \sum_{t \geq 0} x^{(t)} 2^t$ eine ganze 2-adische Zahl, so wird folgende Notation für die Folge der ersten $k+1$ Koeffizienten eingeführt

$$\xi^{[k]} = \langle x^{(0)}, x^{(1)}, \ldots, x^{(k)} \rangle \in \mathbb{B}^{k+1}$$

Ist $\boldsymbol{\xi} = \langle \xi_0, \ldots, \xi_{m-1} \rangle \in {}_2\mathbb{Z}^m$ so wird $\boldsymbol{\xi}^{[k]}$ für

$$\langle \xi_0^{[k]}, \ldots \xi_{m-1}^{[k]} \rangle \in \mathbb{B}^{m(k+1)}$$

notiert.

Eine 2-adische Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ wird nun derart eingeschränkt, daß für die t-te Ausgabe eine boolesche Funktion $f^{(t)} : \mathbb{B}^{m(s^{(t+1)})} \to \mathbb{B}^n$ existieren muß, die mit den ersten $s^{(t+1)}$ Eingaben die Ausgabe berechnet.

Definition 3.11. Sei $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$. Eine Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ heißt

- *online*, wenn es boolesche Funktionsvektoren $f^{(t)} : (\mathbb{B}^{t+1})^m \to \mathbb{B}^n$ gibt mit

$$f(\boldsymbol{\xi}) = \sum_{t \geq 0} f^{(t)}(\boldsymbol{\xi}^{[t]}) 2^t$$

- *offline*, wenn es boolesche Funktionsvektoren $f^{(t)} : (\mathbb{B}^{s^{(t)}+1})^m \to \mathbb{B}^n$ zu einer streng monoton steigenden Folge $s^{(0)} < s^{(1)} < s^{(2)} < \ldots$ in $\mathbb{N}_0$ gibt mit

$$f(\boldsymbol{\xi}) = \sum_{t \geq 0} f^{(t)}(\boldsymbol{\xi}^{[s^{(t)}]}) 2^t$$

Die Menge der online Funktionen $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ wird mit $\mathrm{ON}_{m,n}$ und die Menge der offline Funktionen $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ mit $\mathrm{OFF}_{m,n}$ bezeichnet.

Bemerkung 3.12. Mit einer online Funktion f wird tatsächlich eine 2-adische Funktion f mit $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ definiert. Denn ist $\boldsymbol{b}^{(t)} = \langle b_0^{(t)}, \ldots, b_{n-1}^{(t)} \rangle = f^{(t)}(\boldsymbol{\xi}^{[t]}) \in \mathbb{B}^n$ für ein $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$, so gilt

$$\sum_{t \geq 0} \boldsymbol{b}^{(t)} 2^t = \langle \sum_{t \geq 0} b_0^{(t)} 2^t, \ldots, \sum_{t \geq 0} b_{n-1}^{(t)} 2^t \rangle \in {}_2\mathbb{Z}^n$$

Mit einer ähnlichen Betrachtung kann verdeutlicht werden, daß eine offline Funktion wirklich eine 2-adische Funktion ist.

Mit Hilfe dieser Beschränkung der 2-adischen Funktionen wird eine Korrespondenz zwischen den online Funktionen und den synchronen Schaltkreisen sowie den offline Funktionen und den synchronen Schaltkreisen mit Ausgabeberechtigung ermöglicht.

Satz 3.13. *Ist $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ eine Funktion über den ganzen 2-adischen Zahlen, dann gilt*

- *f ist online genau dann, wenn die Funktion f von einem synchronen Schaltkreis vom Typ (l, m, n) berechenbar ist.*
- *f ist offline genau dann, wenn die Funktion f von einem synchronen Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n) berechenbar ist.*

Die Korrespondenz von online Funktionen und synchronen Schaltkreisen wird in [Vui94] gezeigt und die Beziehung von offline Funktionen und synchronen Schaltkreisen mit Ausgabeberechtigung wird in [Stu96] vorgestellt.

Die Konstruktionen der vorliegenden Arbeit beschränken sich auf endliche synchrone Schaltkreise mit Ausgabeberechtigung. Im folgenden werden daher nur online und offline Funktionen zur Beschreibung von endlichen synchronen Schaltkreisen mit Ausgabeberechtigung benötigt.

Definition 3.14. Eine Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ heißt *endlich online*, wenn sie von einem endlichen synchronen Schaltkreis berechenbar ist und *endlich offline*, wenn sie von einem endlichen synchronen Schaltkreis mit Ausgabeberechtigung berechenbar ist.

Die Menge der endlichen online Funktionen $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ wird mit $\mathrm{eON}_{m,n}$ und die Menge der endlichen offline Funktionen $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ mit $\mathrm{eOFF}_{m,n}$ bezeichnet.

Folgende wichtige Konstruktionen für online Funktionen werden zum Abschluß dieses Kapitels vorgestellt.

Definition 3.15. Seien $f, g, h \in \mathrm{ON}_{m,1}$ mit $f(\boldsymbol{\xi}) = \sum_{t\geq 0} f^{(t)}(\boldsymbol{\xi}^{[t]})2^t$, $g(\boldsymbol{\xi}) = \sum_{t\geq 0} g^{(t)}(\boldsymbol{\xi}^{[t]})2^t$, $h(\boldsymbol{\xi}) = \sum_{t\geq 0} h^{(t)}(\boldsymbol{\xi}^{[t]})2^t$.

Mit $a_0, \ldots, a_{m-1} \in \mathbb{B}^k$, $k \geq 0$, wird der $(a_0, \ldots, a_{m-1})$-*Prädiktor* zu f definiert als die online Funktion

$$f_{\langle a_0,\ldots,a_{m-1}\rangle}(\xi_0, \ldots, \xi_{m-1}) = \sum_{t\geq 0} f^{(t+k)}(\xi_0^{[t]}, a_0, \ldots, \xi_{m-1}^{[t]}, a_{m-1})2^t$$

Mit den booleschen Operationen $ite, \wedge, \vee$ werden folgende Operationen für online Funktionen definiert.

$$\mathrm{ite}(f, g, h) : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t\geq 0} \mathrm{ite}(f^{(t)}(\boldsymbol{\xi}^{[t]}), g^{(t)}(\boldsymbol{\xi}^{[t]}), h^{(t)}(\boldsymbol{\xi}^{[t]}))2^t \end{cases}$$

$$f \wedge g : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t\geq 0} (f^{(t)}(\boldsymbol{\xi}^{[t]}) \wedge g^{(t)}(\boldsymbol{\xi}^{[t]}))2^t \end{cases}$$

$$f \vee g : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t\geq 0} (f^{(t)}(\boldsymbol{\xi}^{[t]}) \vee g^{(t)}(\boldsymbol{\xi}^{[t]}))2^t \end{cases}$$

Der Rechtsshift von $\boldsymbol{\xi}$ kann mit folgender online Funktion definiert werden

$$R : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z}^m \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t\geq 0} \boldsymbol{\xi}^{(t)} 2^{t+1} \end{cases}$$

Mit diesem Rechtsshift kann nun die später wichtige Operation mem(f) definiert werden.

$$\text{mem(f)} : \begin{cases} {}_2\mathbb{Z}^m & \rightarrow \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \text{ite}(f(\boldsymbol{\xi}), 1^\omega, R(\text{mem}(\boldsymbol{\xi}))) \end{cases}$$

Wenn das t-te Ausgabebit von $f(\boldsymbol{\xi})$ eine 1 ist, so werden alle weiteren Ausgaben von mem(f($\boldsymbol{\xi}$)) eine 1 sein. Solange aber f keine 1 berechnet, wird mem(f($\boldsymbol{\xi}$)) eine 0 ausgeben.

Sind die online Funktionen f, g, h endlich, so sind auch die online Funktionen ite(f, g, h), $f \wedge g$, $f \vee g$ und mem(f) endliche online Funktionen. Wegen Bemerkung 3.16 kann der Prädiktor $f_{\langle a_0,\ldots,a_{m-1}\rangle}$ von einem endlichen synchronen Schaltkreis berechnet werden und ist daher ebenfalls eine endliche online Funktion.

In einem späteren Kapitel wird eine Iterationsvorschrift

$$\begin{aligned} f[0] &= \text{mem(f)} \\ f[j+1] &= \bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} f[j]_{\langle b_0,\ldots,b_{m-1}\rangle} \end{aligned}$$

für eine endliche online Funktion f mit Hilfe dieser Prädiktoren definiert. Mit folgender wichtigen Eigenschaft kann später gezeigt werden, daß diese Iteration terminiert.

Lemma 3.16. *Ist f eine endliche online Funktion, dann ist die Anzahl der verschiedenen Prädiktoren endlich.*

Beweis: Berechnet ein synchroner Schaltkreis $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ vom Typ (l, m, n) die online Funktion f, so werden mit allen $\tilde{\mathcal{S}} = (\delta, \tilde{\lambda}, \boldsymbol{s})$ mit $\tilde{\lambda} : \mathbb{B}^m \rightarrow \mathbb{B}^n$ alle Prädiktoren von f berechnet. Die Anzahl der unterschiedlichen Prädiktoren von f ist deshalb durch die Anzahl der möglichen booleschen Ausgabefunktionen $\tilde{\lambda}$ beschränkt und damit endlich.

Der Prädiktor $f = f_{\langle\epsilon,\ldots,\epsilon\rangle}$ wird von $\mathcal{S}$ berechnet. Sei nun bereits gezeigt, daß alle Prädiktoren $f_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle}$ der Länge $i \geq 0$ mit $a_j \in \mathbb{B}^i$ durch synchrone Schaltkreise $\mathcal{S}' = (\delta, \lambda', \boldsymbol{s})$ mit der Ausgabefunktion $\lambda' : \mathbb{B}^m \rightarrow \mathbb{B}^n$ berechnet werden. Ein Prädiktor $f_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle\langle b_0,\ldots,b_{m-1}\rangle}$ mit $b_j \in \mathbb{B}$ und $a_j \in \mathbb{B}^i$ kann dann durch den synchronen Schaltkreis $\tilde{\mathcal{S}} = (\delta, \tilde{\lambda}, \boldsymbol{s})$ mit der Ausgabefunktion

$$\tilde{\lambda} = \left(\lambda'|_{X_0=b_0,\ldots,X_{m-1}=b_{m-1}}\right)|_{Q_0=\delta_0,\ldots,Q_{l-1}=\delta_{l-1}}$$

berechnet werden, da es nach Induktionsvoraussetzung ein $\mathcal{S}' = (\delta, \lambda', \boldsymbol{s})$ mit $\lambda' : \mathbb{B}^m \rightarrow \mathbb{B}^n$ gibt, der den Prädiktor $f_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle}$ berechnet. $\square$

4 Grundlagen der Automatenkonstruktionen mit OBDDs

Wie in der Einleitung festgestellt, muß bei den Automatenkonstruktionen gewährleistet sein, daß die Automaten zum einen in einer kompakten Form beschrieben werden können und zum anderen flexible und effiziente Operationen zur Verfügung stehen, um mit den beschriebenen Automaten arbeiten und diese verändern zu können.

In Kapitel 3.2 sind verschiedene äquivalente Formalismen zur Beschreibung von deterministischen Automaten vorgestellt worden. Synchrone Schaltkreise mit Ausgabeberechtigung betonen von diesen Beschreibungsarten die hardwarenahe Betrachtungsweise, indem sie mit Hilfe von booleschen Funktionen und Zustandsregistern definiert werden.

Werden die booleschen Funktionen mit Hilfe von binären Entscheidungsdiagrammen (OBDDs) dargestellt, so gelingt in vielen Fällen eine kompakte Beschreibung, die effiziente und flexible Operationen zur Modifikation der beschriebenen Automaten ermöglicht.

Da die Beschreibung eines Automaten nur dann praktikabel ist, wenn die benötigten Operationen effizient anwendbar sind, muß diese immer zusammen mit den Operationen betrachtet werden.

In den folgenden Konstruktionen ist die kompakte Beschreibung von Zustandsmengen und deren Bearbeitung grundlegend. Im folgenden wird deshalb zuerst vorgestellt, wie Mengen effizient mit OBDDs dargestellt und bearbeitet werden. Anschließend wird vorgestellt, wie die booleschen Funktionen eines synchronen Schaltkreises mit OBDDs umgesetzt werden.

4.1 Darstellung von Mengen durch OBDDs

In [CBM89] bringt Coudert die Idee auf, Menge mit Hilfe von OBDDs darzustellen.

Eine Menge A kann durch ihre charakteristische Funktion $\chi_A : A \to \mathbb{B}$ mit

$$x \in A \Leftrightarrow \chi_A(x) = 1$$

dargestellt werden.

Ist die Menge A eine Teilmenge von $\mathbb{B}^l$, so ist die charakteristische Funktion χ_A eine boolesche Funktion $\chi_A : \mathbb{B}^l \to \mathbb{B}$.

Deshalb können die Zustände $A \subseteq \mathbb{B}^l$ eines synchronen Schaltkreises $\mathcal{S}$ mit den Zustandsvariablen $\boldsymbol{Q}$ durch ein OBDD $A(\boldsymbol{Q})$ beschrieben werden:

$$(b_0, \ldots, b_{l-1}) \in A \Leftrightarrow A(b_0, \ldots, b_{l-1}) = 1$$

Wenn A nun viele Zustände besitzt, die in mehreren Zustandbits identisch sind, kann das OBDD $A(\boldsymbol{Q})$ in reduzierter Form sehr kompakt dargestellt werden.

Mit Hilfe der booleschen Operationen können wichtige Mengenoperationen und Abfragen umgesetzt werden.

Lemma 4.1. *Seien $A, B, C \subseteq \mathbb{B}^l$ Mengen und χ_A, χ_B, χ_C die zugehörigen charakteristischen booleschen Funktionen. Besitzen χ_A und χ_B dieselben Variablen $\boldsymbol{Q}$, dann gilt*

- $C = A \cup B \Leftrightarrow \chi_C = \chi_A \vee \chi_B$
- $C = A \cap B \Leftrightarrow \chi_C = \chi_A \wedge \chi_B$
- $C = A \setminus B \Leftrightarrow \chi_C = \chi_A \wedge \neg\chi_B$

- $A \subseteq B \Leftrightarrow \chi_A = \chi_A \wedge \chi_B$

Besitzt χ_A *die Variablen* $\boldsymbol{Q}$ *und* χ_B *die Variablen* $\boldsymbol{Q}'$*, dann gilt*

$$C = A \times B \Leftrightarrow \chi_C = \chi_A \wedge \chi_B$$

Da die OBDD-Operationen von der Größe der OBDDs abhängig sind, können deshalb bei kompakten OBDDs sehr effizient Mengenoperationen durchgeführt werden.

Im folgenden wird nicht mehr streng zwischen einer Menge A und der charakteristischen booleschen Funktion χ_A unterschieden. Wenn eine Unterscheidung zum Verständnis notwendig ist, wird auf die jeweilige Interpretation hingewiesen.

4.2 Darstellung eines synchronen Schaltkreises durch OBDDs

Die booleschen Funktionen eines synchronen Schaltkreises $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ vom Typ (l, m, n) werden durch OBDDs mit den Eingabevariablen $\boldsymbol{X}$ und Zustandsvariablen $\boldsymbol{Q}$ dargestellt. Normalerweise besitzen die Eingabevariablen einen sehr großen Einfluß auf das Ausgabeverhalten von δ, λ und α. Deshalb kommen die Eingabevariablen X in der Variablenordnung der OBDDs vor den Zustandsvariablen Q:

$$\forall x \in X \ \ \forall q \in Q : x < q$$

Variablenordnungen mit dieser Eigenschaft vermeiden nach Abschnitt 2.4.2 in vielen Fällen, daß die OBDDs unnötig groß werden.

Der Startzustand $\boldsymbol{s}$ wird mit einem OBDD $S(\boldsymbol{Q})$ beschrieben, das die Menge $S = \{\boldsymbol{s}\}$ dargestellt. Wird der synchrone Schaltkreis $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ mit Hilfe von OBDD*s* darstellt, so wird dieser mit $\mathcal{S} = (\delta, \lambda, \alpha, S)$ notiert.

4.3 Darstellung der Übergangsfunktion durch die Transitionsrelation

Für viele Konstruktionen mit synchronen Schaltkreisen ist es sinnvoll, die Übergangsfunktion

$$\delta : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow & \mathbb{B}^l \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & \boldsymbol{Q}' \end{cases}$$

mit $Q'_i = \delta_i(\boldsymbol{X}, \boldsymbol{Q})$ und der i-ten Komponentenfunktion $\delta_i : \mathbb{B}^m \times \mathbb{B}^l \rightarrow \mathbb{B}$ durch eine Relation darzustellen.

Zur Motivation wird die Berechnung der Folgezustände bzw. der möglichen Vorgängerzustände eines synchronen Schaltkreises $\mathcal{S}$ ausgehend von einer Zustandsmenge $M(\boldsymbol{Q})$ vorgestellt.

Folgende zwei Mengen werden definiert.

Definition 4.2. Sei $f : \mathbb{B}^m \rightarrow \mathbb{B}^n$ eine boolesche Funktion. Das *Bild* von f bzgl. der Menge $M \subseteq \mathbb{B}^m$ ist

$$f(M) = \{y \in \mathbb{B}^n \mid \exists x \in M : f(x) = y\}$$

Das *inverse Bild* von f bzgl. $M \subseteq \mathbb{B}^n$ ist

$$f^{-1}(M) = \{x \in \mathbb{B}^m \mid \exists y \in M : f(x) = y\}$$

Die Menge der Folgezustände $N \in \mathbb{B}^l$ ausgehend von $M \in \mathbb{B}^l$ ist dann das Bild von $\delta(\mathbb{B}^m \times M)$, d.h.

$$N = \{q' \in \mathbb{B}^l \mid \exists x \in \mathbb{B}^m \exists q \in M : \delta(x,q) = q'\}$$

Gelingt es nun, δ durch eine charakteristische Funktion $T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})$ mit

$$\forall x \in \mathbb{B}^m \forall q, q' \in \mathbb{B}^l (T(x,q,q') = 1 \Leftrightarrow \delta(x,q) = q')$$

darzustellen, so kann N als charakteristische Funktion

$$N(\boldsymbol{Q'}) = \exists_{X \cup Q} \left[M(\boldsymbol{Q}) \wedge T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})\right] \tag{2}$$

mit der booleschen Operation $\exists$ aus Abschnitt 2.6.3 berechnet werden.

Werden $M(\boldsymbol{Q})$ und $T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})$ mit OBDDs kompakt dargestellt, so kann mit Hilfe des relationalen Produktes aus Abschnitt 2.6.3 sehr effizient die Menge der Folgezustände von $M(\boldsymbol{Q})$ berechnet werden. Analog können die möglichen Vorgängerzustände von $M(\boldsymbol{Q'})$ durch

$$N(\boldsymbol{Q}) = \exists_{\boldsymbol{X} \cup \boldsymbol{Q'}} \left[M(\boldsymbol{Q'}) \wedge T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})\right] \tag{3}$$

berechnet werden.

Definition 4.3. Die boolesche Funktion $T : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}$ mit

$$\forall x \in \mathbb{B}^m \forall q, q' \in \mathbb{B}^l (T(x,q,q') = 1 \Leftrightarrow \delta(x,q) = q')$$

heißt *Transitionsrelation* von dem synchronen Schaltkreis $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$.

4.3.1 Berechnung der Transitionsrelation

Folgendermaßen kann nun die boolesche Funktion $T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})$ aus δ gewonnen werden.

Die Relation $T_i(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})$ mit

$$\forall x \in \mathbb{B}^m \forall q, q' \in \mathbb{B}^l (T_i(x,q,q') = 1 \Leftrightarrow \delta_i(x,q) = q') \tag{4}$$

für $(0 \leq i < l)$ wird bestimmt durch

$$T_i = (\delta_i(\boldsymbol{X},\boldsymbol{Q}) \equiv \boldsymbol{Q}'_i)$$

wobei die boolesche Operation $\equiv$ in Bemerkung 2.9 beschrieben ist.

Die Relation $T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'})$ wird dann durch

$$T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'}) = \bigwedge_{0 \leq i < l} T_i(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q'}) \tag{5}$$

berechnet.

Bemerkung 4.4. Wird die Transitionsrelation durch ein OBDD dargestellt, so müssen zusätzlich zu X und Q die Variablen Q' in der Variablenordnung eingeordnet werden. Da in der Regel eine starke funktionale Abhängigkeit zwischen der Zustandsvariablen Q_i und der Folgezustandsvariablen Q'_i besteht, werden die Variablen Q_i und Q'_i in der Variablenordnung der OBDDs als Nachbarn aufgestellt (siehe Abschnitt 2.4.2). Dadurch kann die Knotenanzahl von der Transitionsrelation T in vielen Fällen relativ niedrig gehalten werden.

Mit (2) wird $N(\boldsymbol{Q}')$ anstelle $N(\boldsymbol{Q})$ berechnet. Deshalb müssen nach dem relationalen Produkt die Variablen Q_i in Q_i' substituiert werden, kurz

$$N(\boldsymbol{Q}) := N(\boldsymbol{Q}' \leftarrow \boldsymbol{Q})$$

Ebenso muß für Berechnung (3) die Zustandsmenge $M(\boldsymbol{Q})$ durch $M(\boldsymbol{Q}')$ ausgedrückt werden.

Wenn die Variablen Q_i und Q_i' in der Variablenordnung benachbart sind, so kann die Variablensubstitution $N(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$ eines OBDDs mit linearer Zeitkomplexität zur Knotenanzahl von $N(\boldsymbol{Q}')$ durchgeführt werden. Außerdem verändert sich die Größe des OBDDs $N(\boldsymbol{Q})$ zu $N(\boldsymbol{Q}')$ nicht.

Folgendes Lemma legitimiert die Berechnung (4) der T_i.

Lemma 4.5. *Die* OBDD*s T_i können nur um einen konstanten Faktor größer werden als die ursprünglichen* OBDD*s δ_i.*

Beweis: Die Operation $\delta_i \equiv Q_i'$ kann ausgedrückt werden durch $\mathrm{ITE}(Q_i', \delta_i, \neg\delta_i)$.

Der *ITE*-Algorithmus durchläuft rekursiv das OBDD T_i. Wird ein Knoten A erreicht mit einer Variablenbeschriftung V, sodaß $Q_i' > V$ gilt, dann wird Knoten A ersetzt durch $\mathrm{ITE}(Q_i', A, \neg A)$ und anschließend der Rekursionsaufruf beendet. Da A und $\neg A$ nach Abschnitt 2.5 mit demselben Knoten dargestellt werden können, wird nur ein Knoten hinzugefügt. Im Extremfall verdoppelt sich also die Knotenanzahl. □

Für Berechnung (5) kann dagegen die Knotenanzahl sehr groß werden: im Extremfall kann T

$$\prod_{0 \le i < l} |T_i|$$

Knoten besitzen.

4.3.2 Partitionierung der Transitionsrelation

In dieser Arbeit wird das OBDD $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$ in der Regel nur mit dem relationalen Produkt der Form

$$\exists_V \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')\right] \tag{6}$$

mit $V \in \{Q, Q \cup X\}$ oder

$$\exists_V \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')\right] \tag{7}$$

mit $V \in \{Q', Q' \cup X\}$ benötigt. Im folgenden wird nun nach [JED91] gezeigt, wie das relationale Produkt mit einer partitionierten Transitionsrelation umgesetzt werden kann.

Dabei wird die einfache Tatsache ausgenutzt, daß boolesche Funktionen T_i von einer Quantifizierung ausgeklammert werden können, wenn diese nicht von den quantifizierten Variablen abhängen.

Definition 4.6. Sei $f(A_0, \ldots, A_k)$ eine boolesche Funktion in den Variablen $A = \{A_0, \ldots, A_k\}$. Dann wird mit

$$\mathrm{SUPP}(f) = \{A_i \in A \mid f|_{A_i=1} \neq f|_{A_i=0}\}$$

die Menge der Variablen von f definiert, von denen f echt abhängt.

Lemma 4.7. *Seien $T_0(A), T_1(A), M(A)$ boolesche Funktionen mit den Mengen $X = \mathrm{SUPP}(T_0)$ und $Y = \mathrm{SUPP}(T_1)$. Dann gilt für $D = Y \setminus X$*

$$\exists_{X \cup Y} \left[T_0 \wedge T_1 \wedge M\right] = \exists_{X \setminus D} \left[T_0 \wedge \exists_D \left[T_1 \wedge M\right]\right]$$

Ist das OBDD einer Transitionsrelation bzgl. der vorhandenen Speicherressourcen zu groß, so kann nun die Transitionsrelation T in der zerlegten Form $T_0, \ldots, T_{l-1}$ mit OBDDs dargestellt werden. Mit diesen relativ kleinen OBDDs kann dann die Operation (7) für $V = Q'$ mit

$$\exists_{\{Q'_0\}} \Big[T_0(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge \exists_{\{Q'_1\}} \Big[T_1(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \ldots \exists_{\{Q'_{l-1}\}} \left[T_{l-1}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \right] \ldots \Big] \Big] \tag{8}$$

und für $V = Q' \cup X$ mit

$$\exists_{\{Q'_0\} \cup X} \Big[T_0(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge \exists_{\{Q'_1\}} \Big[T_1(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \ldots \exists_{\{Q'_{l-1}\}} \left[T_{l-1}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \right] \Big] \ldots \Big] \tag{9}$$

berechnet werden.

Für das relationale Produkt (6) gelingt die Partitionierung der Transitionsrelation nicht immer, da ein T_i in der Regel von mehreren Variablen aus Q abhängt. Mit einer ungeschickten Reihenfolge der T_i kann dann keine Variable aus einer Quantifizierung ausgeklammert werden, sodaß die Berechnung von $T = \bigwedge_{0 \leq i < l} T_i$ notwendig ist.

Ein synchroner Schaltkreis setzt sich oft aus mehreren Komponenten zusammen, sodaß der Einfluß einer Zustandsvariable auf wenige T_i beschränkt ist. Mit einer geschickten Reihenfolge der T_i kann daher eine Partitionierung der Transitionsrelation gelingen, sodaß Variablen ausgeklammert werden können.

Sei $\rho \in \mathcal{S}_l$ eine Permutation auf der Menge $\{0, 1, \ldots, l-1\}$, die die Reihenfolge der T_i für $(0 \leq i < l)$ festsetzt und sei $\mathrm{SUPP}_i = \mathrm{SUPP}(T_i) \wedge V$ die Menge der Variablen aus V, von denen T_i abhängt.

Dann ist V_i mit

$$V_i = \mathrm{SUPP}_{\rho(i)} - \bigcup_{i+1 \leq k < l} \mathrm{SUPP}_{\rho(k)}$$

die Menge der Variablen, die in $\mathrm{SUPP}_{\rho(i)}$, aber nicht in den Mengen $\mathrm{SUPP}_{\rho(k)}$ für alle $k > i$ enthalten sind. Außerdem bilden die V_i eine Partition auf V, d.h.

$$\forall i \neq j : V_i \cap V_j = \emptyset$$

und

$$\bigcup_{0 \leq i < l} V_i = V$$

Mit diesen Mengen V_i kann nun die Transitionsrelation aus (6) schrittweise berechnet werden durch

$$\begin{aligned} M_0 &:= \exists_{V_0} \left[T_{\rho(0)} \wedge M \right] \\ M_1 &:= \exists_{V_1} \left[T_{\rho(1)} \wedge M_0) \right] \\ &\vdots \\ M_{l-1} &:= \exists_{V_{l-1}} \left[T_{\rho(l-1)} \wedge M_{l-2}) \right] \end{aligned} \tag{10}$$

Wenn $V_i = \emptyset$ gilt, wird das zugehörige $T_{\rho(i)}$ mit dem benachbarten $T_{\rho(i-1)}$ oder $T_{\rho(i+1)}$ mit der Variablenmenge V_{i-1} oder V_{i+1} zusammengefaßt.

Gelingt nun die Partitionierung der Transitionsrelation derart, daß die OBDDs von der zerlegten Transitionsrelation relativ klein sind, so kann die Operation (6) durchgeführt werden.

Solange die Knotenanzahl der OBDDs nicht zu groß wird, sollten dabei benachbarte $T_{\rho(i)}$ zusammengefaßt werden, da die Berechnung des relationalen Produktes an sich sehr effizient implementiert werden kann. Zu kleine $T_{\rho(i)}$ und Quantifizierungen mit nur einer Variablen zerstören dann die Effizienz des Algorithmus.

Auf jedenfall sollten diejenigen $T_{\rho(i)}$ zusammengefaßt werden, deren Variablen Q_i funktional sehr stark voneinander abhängig sind. Werden diese $T_{\rho(i)}, \ldots, T_{\rho(i+r)}$ getrennt, so können sich zwischenzeitlich die OBDDs $M_{\rho(i)}, \ldots, M_{\rho(i+r)}$ unnötig vergrößern. Dieselbe Überlegung trifft auf die Partitionierung der Transitionsrelation (8)/(9) zu.

Damit der Benutzer eine möglichst große Flexibilität erhält, kann dieser die Übergangsfunktionen bzw. die Zustandsvariablen in einer Partition nach Zusammengehörigkeit sortieren. Anstelle einer Permutation ρ, die die Reihenfolge der Variablen festsetzt, erhalten die einzelnen Klassen der Partition eine Reihenfolge. Die Klassen der Partition werden nun nach der obigen Berechnung für eine partitionierte Transitionsrelation abgearbeitet, wobei jedoch die Variablen einer Klasse nicht getrennt werden dürfen.

In den folgenden Kapiteln wird die Operation des relationalen Produktes eine entscheidende Rolle spielen. Eine geschickte Partitionierung der Transitionsrelation entscheidet deshalb maßgeblich die Effizienz der folgenden Automatenkonstruktionen. Der Anwender muß also bei der Wahl der Partitionierung die lokalen Verhältnisse des synchronen Schaltkreises ausnutzen und auf eine sinnvolle Größe der OBDDs achten.

Es wird nun angenommen, daß die Transitionsrelation jeweils für die Operationen (6) und (7) entsprechend partitioniert sind. Da die vollständige Transitionsrelation T nicht benötigt wird, kann ihre Berechnung vermieden werden. Die Schreibweisen (6) und (7) dienen im folgenden nur als Abkürzung für die Berechnungen (10) und (8),(9).

5 Einfache Automatenkonstruktionen mit OBDDs

Die folgenden Konstruktionen und Analysen von Automaten zeigen auf, wie OBDDs effizient für Automaten eingesetzt werden können. Die Erreichbarkeitsanalysen der Automatenzustände sind in der Literatur ausführlich behandelt worden. Die hierfür entwickelten Strategien werden in dieser Arbeit entsprechend für Automatenkonstruktionen angewendet.

Mit Hilfe der Berechnung der erreichbaren Zustände und der Konstruktion des Produktautomaten sowie der Komposition von Automaten können dann Korrektheitstests für konstruierte Automaten durchgeführt werden. Zu den später vorgestellten Automatenkonstruktionen werden in diesem Kapitel Algorithmen angegeben, mit denen die Korrektheit der konstruierten Automaten getestet werden kann. Die Komposition wird später außerdem für das Lösen von Ungleichungen angewendet.

5.1 Erreichbarkeitsanalysen

5.1.1 Berechnen der erreichbaren Zustände

[CBM89] ist eine der ersten Arbeiten, in denen OBDDs eingesetzt werden, um die erreichbaren Zustände eines synchronen Schaltkreises $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ zu berechnen.

Sei im folgenden $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$ die Transitionsrelation von $\mathcal{S}$.

Mit folgendem Algorithmus[3] werden alle Zustände berechnet, die erreicht werden können, wenn der synchrone Schaltkreis mit dem Startzustand $S(\boldsymbol{Q})$ initialisiert wird.

Algorithmus 5.1 (einfacher Erreichbarkeits-Algorithmus).
// Input: Automat mit Startzustand $S(\boldsymbol{Q})$ und Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$
// Output: die erreichbaren Zustände
bdd `erreicht`()
{
(1) `erreicht_akt`$(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
(2) **do**
{
(3) `erreicht`$(\boldsymbol{Q}) =$ `erreicht_akt`$(\boldsymbol{Q})$;
(4) `erreicht_akt`$(\boldsymbol{Q}') = \exists_{X \cup Q}\, [T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge$ `erreicht_akt`$(\boldsymbol{Q})]$;
(5) `erreicht_akt`$(\boldsymbol{Q}) =$ `erreicht_akt`$^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$;
(6) `erreicht_akt`$(\boldsymbol{Q}) =$ `erreicht_akt`$^-(\boldsymbol{Q}) \vee$ `erreicht`$(\boldsymbol{Q})$;
}
(7) **while**(`erreicht_akt` $!=$ `erreicht`$^-$)
(8) **return**(`erreicht_akt`);
}

Die Zeilen (4) und (5) werden in Abschnitt 4.3.1 beschrieben.

Im i-ten Berechnungsschritt werden die Zustände `erreicht`$(\boldsymbol{Q})$ berechnet, die ausgehend von dem Startzustand $S(\boldsymbol{Q})$ in maximal i Übergängen erreicht werden können.

Das OBDD `erreicht`$(\boldsymbol{Q})$ wächst meistens während der Iteration stark an. Es besteht dann die Gefahr, daß das relationale Produkt mit OBDDs nicht mehr effizient ausgeführt werden kann. Aus

[3] Abschnitt 2.7 begründet die grundsätzliche Notation in dem Algorithmus. Insbesondere die Symbole $^+$ und $^-$ werden erklärt.

diesem Grund wird in [CBM89] vorgeschlagen, anstelle der bisher erreichten Zustände $\texttt{erreicht}(\boldsymbol{Q})$ nur die im letzten Schritt neu berechneten Zustände $\texttt{neu}(\boldsymbol{Q})$ für das relationale Produkt zu verwenden. Es ergibt sich dann folgender Algorithmus.

(1) $\texttt{erreicht}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
(2) $\texttt{neu}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
(3) **do**
{
(4) $\texttt{neu}(\boldsymbol{Q}') = \exists_{X \cup Q} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge \texttt{neu}(\boldsymbol{Q})\right]$;
(5) $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$;
(6) $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q}) \wedge \neg\texttt{erreicht}(\boldsymbol{Q})$;
(7) $\texttt{erreicht}(\boldsymbol{Q}) = \texttt{erreicht}^-(\boldsymbol{Q}) \vee \texttt{neu}(\boldsymbol{Q})$;
}
(8) **while**(neu!=**0**)
(9) **return**(erreicht);

In $\texttt{neu}(\boldsymbol{Q})$ werden dann alle Zustände berechnet, die genau nach i Zustandsübergängen das erste Mal erreicht werden können. Da nach endlich vielen Schritten die Menge $\texttt{erreicht}(\boldsymbol{Q})$ berechnet ist, können keine neuen Zustände berechnet werden, d.h. es gilt $\texttt{neu} = \mathbf{0}$ und der Algorithmus terminiert.

In vielen Fällen kann mit diesem veränderten Algorithmus das relationale Produkt berechnet werden. Oft gibt es jedoch eine noch bessere Wahl als das OBDD $\texttt{erreicht\_akt}(\boldsymbol{Q})$ oder $\texttt{neu}(\boldsymbol{Q})$. In [JSL+90] wird folgendes heuristisches Verfahren vorgeschlagen.

Seien E_i die Zustände, die von dem Startzustand S beginnend in maximal i Schritten erreichbar sind. Um im Schritt $i+1$ die Menge E_{i+1} zu berechnen, kann folgendermaßen vorgegangen werden.

1. Wähle ein OBDD e_{i+1} in dem Bereich

$$E_i \setminus E_{i-1} \subseteq e_i \subseteq E_i$$

   das möglichst wenige Knoten besitzt.

2. Berechne $\texttt{neu\_alt}_i(\boldsymbol{Q}') = \exists_{X \cup Q} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge e_i(\boldsymbol{Q})\right]$
3. Dann gilt $E_{i+1}(\boldsymbol{Q}) = E_i(\boldsymbol{Q}) \vee \texttt{neu\_alt}_i(\boldsymbol{Q})$

Die beiden obigen Algorithmusvorschläge haben die Schranke $e_i = E_i$ für $\texttt{erreicht\_akt}(\boldsymbol{Q})$ und $e_i = E_i \setminus E_{i-1}$ für $\texttt{neu}(\boldsymbol{Q})$ zur Bildberechnung gewählt.

In Schritt 1 wird dabei eine boolesche Funktion e_{i+1} gesucht, die sich bei den Eingaben aus der Menge $\neg E_{i-1}(\boldsymbol{Q})$ wie E_i verhält und sich für die Eingaben aus der Menge $E_{i-1}(\boldsymbol{Q})$ beliebig verhalten kann. Mit diesem Freiheitsgrad wird nun heuristisch ein OBDD e_i berechnet, das möglichst klein ist.

Folgender Algorithmus wird in [CM90] als Restrikt-Operation für OBDDs eingeführt.

Algorithmus 5.2 (Reduktions-Operation).

// Input: OBDDs f, $c \neq \mathbf{0}$
// Output: $f_{\text{reduziert}}$
bdd REDUCE(bdd f, bdd c)
{
(1) **if**($c == \mathbf{1}$ || $f == \mathbf{1}$ || $f == \mathbf{0}$)
(2)   **return**(f);
(3) **if**(lookup_in_reduce_cache(&Ergebnis, f, c))
(4)   **return**(Ergebnis);
(5) v = top_variable(f, c);
(6) **if**($c|_{v=1} == \mathbf{0}$)
(7)   **return**(REDUCE($f|_{v=0}, c|_{v=0}$));
(8) **if**($c|_{v=0} == \mathbf{0}$)
(9)   **return**(REDUCE($f|_{v=1}, c|_{v=1}$));
(10) **if**($f|_{v=1} == f|_{v=0}$)
(11)   **return**(REDUCE(f, ITE($c|_{v=1}, \mathbf{1}, c|_{v=0}$)));
(12) temp0 = REDUCE($f|_{v=1}, c|_{v=1}$);
(13) temp1 = REDUCE($f|_{v=0}, c|_{v=0}$);
(14) Ergebnis = find_or_add_unique_table(v, temp0, temp1);
(15) insert_in_reduce_cache(Ergebnis, f, c);
(16) **return**(Ergebnis);
}

Dieser Algorithmus besitzt folgende wichtige Eigenschaft

Lemma 5.3. *Seien F und C OBDDs in den Variablen $X = \{X_0, \ldots, X_{m-1}\}$ zu den booleschen Funktionen f, c. Dann gilt für die boolesche Funktion g von dem OBDD $G = \text{REDUCE}(f, c)$*

$$\forall b \in \mathbb{B}^m : c(b) = 1 \Rightarrow f(b) = g(b)$$

In vielen Fällen wird das OBDD REDUCE(f, c) im Vergleich zu f verkleinert, da mit den Schritten (7) und (9) Knoten "herausgeschnitten" werden. In Zeile (11) wird für die weitere Rekursion das OBDD OR($c|_{v=1}, c|_{v=0}$) in den meisten Fällen verkleinert. Damit können dann neben (7) und (9) zusätzlich Knoten entfernt werden. Manchmal tritt mit dieser OR-Verknüpfung aber auch der umgekehrte Effekt auf.

Mit $e_i = \text{REDUCE}(E_i, \neg E_{i-1})$ kann nun oft ein OBDD berechnet werden, das wesentlich kleiner ist als die OBDDs E_i oder $E_i \setminus E_{i-1}$. In manchen Fällen jedoch kann mit $E_i \setminus E_{i-1}$ ein besseres Ergebnis erzielt werden. Mit einer Abfrage bzgl. der Größe erhält man dann die bessere Wahl. Mit diesen Überlegungen ergibt sich nun Algorithmus 5.4, bei dem die OBDDs der booleschen Funktionen möglichst klein gehalten werden.

Sollen alle möglichen Zustände berechnet werden, die in eine Zustandsmenge $Z(\boldsymbol{Q})$ führen, d.h. wird das inverse Bild von der Übergangsfunktion bzgl. der Zustandsmenge $Z(\boldsymbol{Q})$ gesucht, so können alle Überlegungen von Algorithmus 5.4 mit folgender Änderung übernommen werden.

- Initialisiere `erreicht` und `neu` mit $Z(\boldsymbol{Q})$ in den Zeilen (1) und (2).
- Berechne das inverse Bild in einem Iterationsschritt, d.h. die Zeilen (4) und (5) werden ausgetauscht mit

  $\texttt{neu}^-(\boldsymbol{Q}') = \texttt{neu}(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$;
  $\texttt{neu}(\boldsymbol{Q}) = \exists_{X \cup Q'} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge \texttt{neu}^-(\boldsymbol{Q}')\right]$;

Algorithmus 5.4 (Erreichbarkeits-Algorithmus mit OBDD-Reduktion).

// Input: Automat mit Startzustand $S(\boldsymbol{Q})$ und Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$
// Output: die erreichbaren Zustände
bdd `erreicht()`
{
(1) $\texttt{erreicht}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
(2) $\texttt{neu}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
(3) **do**
{
(4) $\texttt{neu}(\boldsymbol{Q}') = \exists_{X \cup Q} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge \texttt{neu}^-(\boldsymbol{Q})\right]$;
(5) $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$;
(6) $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q}) \wedge \neg\texttt{erreicht}(\boldsymbol{Q})$;
(7) $\texttt{erreicht\_akt}(\boldsymbol{Q}) = \texttt{erreicht}(\boldsymbol{Q}) \vee \texttt{neu}(\boldsymbol{Q})$;
(8) $\texttt{neu\_alternative} = \text{REDUCE}(\texttt{erreicht\_akt}, (\neg\texttt{erreicht}^-)^-)$;
(9) **if**$(|\texttt{neu}| > |\texttt{neu\_alternative}|)$
(10) $\texttt{neu}^- = \texttt{neu\_alternative}$;
(11) **else** FREE(`neu_alternative`);
(12) $\texttt{erreicht} = \texttt{erreicht\_akt}$;
}
(13) **while**(neu!=**0**)
(14) **return**(`erreicht`);
}

5.1.2 Zustandseigenschaften

Mit Hilfe der obigen Algorithmen können die erreichbaren Zustände eines synchronen Schaltkreises berechnet werden. Über diese Zustände können dann z.B. Aussagen über ihr temporales Verhalten gemacht werden (siehe [CM90], [BCL+94]).

Besonders wichtig für Korrektheitsanalysen ist das Testen des Ausgabeverhaltens eines synchronen Schaltkreises. Es stellt insbesondere die Frage, ob ein synchroner Schaltkreis zu jeder Eingabe immer eine 1 oder immer eine 0 ausgibt. Im folgenden sei $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ (l, m, n) und mit den Hilfsausgabefunktionen $(0 \leq i < n)$ $\lambda_i : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}$ und zugehörigen Ausgabeberechtigungen $\alpha_i : \mathbb{B}^m \times \mathbb{B}^l \to \mathbb{B}$

Werden mit der booleschen Funktion $\texttt{erreicht}(\boldsymbol{Q})$ die erreichbaren Zustände von $\mathcal{S}$ beschrieben, so werden mit

$$A_i(\boldsymbol{X}, \boldsymbol{Q}) = \texttt{erreicht}(X) \wedge \alpha_i(\boldsymbol{X}, \boldsymbol{Q})$$

die Zustände dargestellt, die erreichbar sind und zu der Eingabe $\boldsymbol{X}$ eine Ausgabeberechtigung für die i-te Ausgabekomponente besitzen.

Mit

$$\text{ite}(A_i(\boldsymbol{X}, \boldsymbol{Q}), \lambda_i(\boldsymbol{X}, \boldsymbol{Q}), 1) == 1$$

wird dann getestet, ob die Ausgabefunktion λ_i für alle ausgabeberechtigten Zustände der Ausgabekomponente i unabhängig von der Eingabe den booleschen Wert 1 ausgibt. Ist dies der Fall, so kann $\mathcal{S}$ in der Ausgabekomponente i keine 0 ausgeben.

Analog kann mit

$$\text{ite}(A_i(\boldsymbol{X}, \boldsymbol{Q}), \neg(\lambda_i(\boldsymbol{X}, \boldsymbol{Q})), 1) == 1$$

getestet werden, ob in allen erreichbaren Zuständen für alle Eingaben keine eine 1 ausgegeben wird.

In Algorithmus 5.4 können nach Zeile (6) die neu berechneten erreichbaren Zustände mit

for$(i = 0; i < n; i++)$ {

  $A(\boldsymbol{X}, \boldsymbol{Q}) = \texttt{neu} \wedge \alpha_i)$;

  if$(\text{ite}(A, \lambda_i, 1)! = 1)$

    "Automat kann eine 1 ausgeben"

}

getestet werden. Wenn das Ausgabeverhalten nicht immer 1 ist, kann vorzeitig die Berechnung der erreichbaren Zustände beendet werden.

Insgesamt ergibt sich folgender Algorithmus

Algorithmus 5.5 (Testet Ausgabeverhalten).
// Input: Automat $\mathcal{S} = (\delta, \lambda, \alpha, S(\boldsymbol{Q}))$ und Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$, $\texttt{Wert} \in \mathbb{B}$
// Output: 1, wenn Ausgabe immer `Wert` entspricht, ansonsten 0
int Ist_Ausgabe_gleich(int `Wert`)
{
  $\texttt{erreicht}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
  $\texttt{neu}(\boldsymbol{Q}) = S^+(\boldsymbol{Q})$;
  do
  {
    for$(i = 0; i < n; i++)$ {
      $A(\boldsymbol{X}, \boldsymbol{Q}) = \texttt{neu} \wedge \alpha_i$;
      if$(\texttt{Wert} == 1)$
        $\texttt{Teste} = \text{ite}(A^-, \lambda_i, 1)$;
      else $\texttt{Teste} = \text{ite}(A^-, (\neg\lambda_i)^-, 1)$;
      if$(\texttt{Teste}^-! = 1)$
        {FREE(`neu`); FREE(`erreicht`); **return**(0); }
    }
    $\texttt{neu}(\boldsymbol{Q}') = \exists_{X \cup Q} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge \texttt{neu}^-(\boldsymbol{Q})\right]$;
    $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}')$;
    $\texttt{neu}(\boldsymbol{Q}) = \texttt{neu}^-(\boldsymbol{Q}) \wedge \neg\texttt{erreicht}(\boldsymbol{Q})$;
    $\texttt{erreicht\_akt}(\boldsymbol{Q}) = \texttt{erreicht}(\boldsymbol{Q}) \vee \texttt{neu}(\boldsymbol{Q})$;
    $\texttt{neu\_alternative} = \text{REDUCE}(\texttt{erreicht\_akt}, (\neg\texttt{erreicht}^-)^-)$;
    if$(|\texttt{neu}| > |\texttt{neu\_alternative}|)$
      $\texttt{neu}^- = \texttt{neu\_alternative}$;
    else FREE(`neu_alternative`);
    $\texttt{erreicht} = \texttt{erreicht\_akt}$;
  }
  while(neu!=**0**)
  FREE(`erreicht`);
  return(1);

}

5.2 Produktautomat

In der Literatur werden zwei Mealy Automaten mit gleichem Eingabealphabet und Ausgabealphabet äquivalent genannt, wenn sie zu jeder Eingabefolge dieselbe Ausgabefolge berechnen. Dieses Konzept wird nun erweitert für synchrone Schaltkreise mit Ausgabeberechtigung.

Definition 5.6. Zwei synchrone Schaltkreise mit Ausgabeberechtigung $\mathcal{S}^{(1)}$ vom Typ (l_1, m, n) und $\mathcal{S}^{(2)}$ vom Typ (l_2, m, n) heißen äquivalent, kurz

$$\mathcal{S}^{(1)} \simeq \mathcal{S}^{(2)}$$

wenn sie zu jeder Eingabefolge dieselbe Ausgabefolge berechnen.

Zwei so beschaffene synchrone Schaltkreise mit Ausgabeberechtigung

$$\mathcal{S}^{(1)} = (\lambda^{(1)}, \delta^{(1)}, \alpha^{(1)}, \boldsymbol{s}^{(1)}), \mathcal{S}^{(2)} = (\lambda^{(2)}, \delta^{(2)}, \alpha^{(2)}, \boldsymbol{s}^{(2)})$$

sind genau dann äquivalent, wenn sich beide für jede gemeinsame Eingabefolge nach $t \geq 0$ Berechnungsschritten in zwei Zuständen $q_1 \in \mathbb{B}^{l_1}, q_2 \in \mathbb{B}^{l_2}$ befinden, die das gleiche Ausgabeverhalten besitzen, d.h. es gilt

$$\forall x \in \mathbb{B}^m : \alpha^{(1)}(x, q_1) = \alpha^{(2)}(x, q_2)$$

und

$$\forall i (0 \leq i < n)\ \forall x \in \mathbb{B}^m \left(\alpha_i^{(*)}(x, q_1) = 1 \Rightarrow \lambda_i^{(1)}(x, q_1) = \lambda_i^{(2)}(x, q_2) \right)$$

$\alpha_i^{(*)}$ bedeutet dabei, daß entweder $\alpha_i^{(1)}$ oder $\alpha_i^{(2)}$ verwendet werden können. Werden diese zwei Eigenschaften mit Hilfe von OBDDs dargestellt, so kann immer das $\alpha_i^{(*)}$ gewählt werden, das mit der geringsten Knotenanzahl dargestellt werden kann.

Diese zwei Bedingungen lassen sich in der Ausgabefunktion eines Produktautomaten von $\mathcal{S}^{(1)}$ und $\mathcal{S}^{(2)}$ kodieren.

Konstruktion 5.7. Seien $\mathcal{S}^{(1)} = (\lambda^{(1)}, \delta^{(1)}, \alpha^{(1)}, \boldsymbol{s}^{(1)})$ und $\mathcal{S}^{(2)} = (\lambda^{(2)}, \delta^{(2)}, \alpha^{(2)}, \boldsymbol{s}^{(2)})$ synchrone Schaltkreise mit Ausgabeberechtigung vom Typ (l_1, m, n) und (l_2, m, n) mit den gemeinsamen Eingabevariablen $\boldsymbol{X}$ und den verschiedenen[4] Zustandsvariablen $\boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}$ und Folgezustandsvariablen $\boldsymbol{Q'}^{(1)}, \boldsymbol{Q'}^{(2)}$.

Der Produktautomat $\mathcal{S} = \mathcal{S}^{(1)} * \mathcal{S}^{(2)}$ ist ein synchroner Schaltkreis $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ vom Typ $(l_1 + l_2, m, 1)$, der definiert wird durch

- die Übergangsfunktion

$$\delta : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} & \rightarrow & \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} \\ (\boldsymbol{X}, \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}) & \mapsto & (\boldsymbol{Q'}^{(1)}, \boldsymbol{Q'}^{(2)}) \end{cases}$$

  mit $\boldsymbol{Q'}^{(1)} = \delta^{(1)}(\boldsymbol{X}, \boldsymbol{Q}^{(1)})$ und $\boldsymbol{Q'}^{(2)} = \delta^{(2)}(\boldsymbol{X}, \boldsymbol{Q}^{(2)})$

- die Ausgabefunktion

$$\lambda : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} & \rightarrow & \mathbb{B} \\ (X, \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}) & \mapsto & \boldsymbol{H} \end{cases}$$

  mit

$$\boldsymbol{H} = \left(\bigwedge_{0 \leq i < n_1} \mathrm{ite}(\alpha_i^{(*)}, \lambda_i^{(1)} \equiv \lambda_i^{(2)}, 1) \right) \wedge \left(\bigwedge_{0 \leq i < n_1} \left(\alpha_i^{(1)} \equiv \alpha_i^{(2)} \right) \right)$$

  wobei $* \in \{0, 1\}$.

[4] OBdA. können die Variablen umbenannt werden.

- die Ausgabeberechtigung $\alpha = 1$
- und dem Startzustand $\boldsymbol{s} = \boldsymbol{s}^{(1)}.\boldsymbol{s}^{(2)}$.

Mit Hilfe des Produktautomaten und der Überprüfung des Ausgabeverhaltens des Produktautomaten kann nun die Äquivalenz von $\mathcal{S}^{(1)}$ und $\mathcal{S}^{(2)}$ nachgeprüft werden.

Lemma 5.8. *Seien $\mathcal{S}^{(1)}$ und $\mathcal{S}^{(2)}$ zwei synchrone Schaltkreise mit Ausgabeberechtigung vom Typ (l_1, m, n) und (l_2, m, n). Dann gilt:*

$\mathcal{S}^{(1)}$ und $\mathcal{S}^{(2)}$ sind äquivalent genau dann, wenn das Ausgabeverhalten von dem Produktautomaten $\mathcal{S}^{(1)} * \mathcal{S}^{(2)}$ immer 1 ist.

Sind die zwei synchronen Schaltkreise mit Ausgabeberechtigung durch OBDDs beschrieben und sind die Zustandsvariablen $\boldsymbol{Q}^{(1)}$,$\boldsymbol{Q'}^{(1)}$ von $\mathcal{S}^{(1)}$ und $\boldsymbol{Q}^{(2)}$,$\boldsymbol{Q'}^{(2)}$ von $\mathcal{S}^{(2)}$ verschieden, so kann aus $\mathcal{S}^{(1)}$ und $\mathcal{S}^{(2)}$ nach Abschnitt 4.2 und Konstruktion 5.7 der Produktautomat konstruiert werden.

Oft wird ein Automat $\mathcal{S}^{(1)}$ zu einem Automat $\mathcal{S}^{(2)}$ modifiziert und anschließend getestet, ob diese noch äquivalent sind. In vielen Fällen ist deshalb bei der gleichen Eingabefolge das Verhalten der Zustandsfunktionen und Ausgabefunktionen ähnlich. Da deshalb eine funktionale Beziehung zwischen den OBDDs besteht, werden üblicherweise die Zustandsvariablen von $Q_i^{(1)}, Q'^{(1)}_i, Q_i^{(2)}$ und $Q'^{(2)}_i$ in der Variablenordnung benachbart angeordnet, um die resultierenden OBDDs des Produktautomaten und berechnete Zustandsmengen des Produktautomaten möglichst klein zu halten.

Da die Zustandsvariablen $Q^{(1)}$ und $Q^{(2)}$ verschieden sind, liegt automatisch eine Partitionierung der Transitionsrelation vor. Ein relationales Produkt von $\mathcal{S} = \mathcal{S}^{(1)} * \mathcal{S}^{(2)}$

$$\exists_{Q \cup X} \left[T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'})\right]$$

wird dann z.B. zerlegt, indem zuerst mit der partitionierten Transitionsrelation von $\mathcal{S}^{(1)}$

$$M'(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) = \exists_{Q^{(1)}} \left[T^{(1)}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge M(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'})\right]$$

und anschließend mit der partitionierten Transitionsrelation von $\mathcal{S}^{(2)}$

$$\exists_{Q^{(2)} \cup X} \left[T^{(2)}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'}) \wedge M'(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q'})\right]$$

das relationale Produkt berechnet wird.

5.3 Komposition

In Kapitel 7 werden zu einem synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}^{(2)}$ alle Eingabefolgen gesucht, sodaß $\mathcal{S}^{(2)}$ nicht den Wert 1 berechnen kann. Als Lösung wird dafür ein synchroner Schaltkreis $\mathcal{S}^{(1)}$ konstruiert, der für alle Eingabefolgen alle Lösungen für $\mathcal{S}^{(2)}$ berechnet.

Will man testen, ob die Lösungen von $\mathcal{S}^{(1)}$ korrekt sind, d.h. ob $\mathcal{S}^{(2)}$ keine 1 ausgibt, wenn alle Ausgabefolgen von $\mathcal{S}^{(1)}$ in $\mathcal{S}^{(2)}$ eingegeben werden, so bietet sich die Komposition $\mathcal{S} = \mathcal{S}^{(1)} \rightarrow \mathcal{S}^{(2)}$ als Testautomat an.

In der Komposition $\mathcal{S}^{(1)} \rightarrow \mathcal{S}^{(2)}$ werden die Ausgaben von $\mathcal{S}^{(1)}$ als Eingaben von $\mathcal{S}^{(2)}$ weitergeleitet und in demselben Berechnungsschritt die entsprechenden Ausgaben von $\mathcal{S}^{(2)}$ berechnet.

Wenn nun der Automat $\mathcal{S}^{(1)} \rightarrow \mathcal{S}^{(2)}$ für alle Eingaben eine 0 ausgibt, dann ist Automat $\mathcal{S}_1$ eine korrekte Lösung für die Nullstellenberechnung von Automat $\mathcal{S}^{(1)}$.

In der Komposition wird die Ausgabe von $\mathcal{S}^{(1)}$ nicht in Registern zwischengespeichert, sondern direkt weitergegeben. Die folgende Konstruktion beschreibt eine mögliche Realisierung.

Konstruktion 5.9. Sei $\mathcal{S}^{(1)} = (\delta^{(1)}, \lambda^{(1)}, \boldsymbol{s}^{(1)})$ ein Mealy Automat vom Typ (l_1, m, k) mit den Eingabevariablen $\boldsymbol{X}$ und $\mathcal{S}^{(2)} = (\delta^{(2)}, \lambda^{(2)}, \alpha^{(2)}, \boldsymbol{s}^{(2)})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l_2, k, n) mit den Eingabevariablen $\boldsymbol{X}^{(2)}$, Hilfsausgabevariablen $\boldsymbol{H}$ und Ausgabeberechtigungsvariablen $\boldsymbol{A}$.

Die Zustandsvariablen $\boldsymbol{Q}^{(1)}$ und $\boldsymbol{Q'}^{(1)}$ von $\mathcal{S}^{(1)}$ seien verschieden zu den Zustandsvariablen $\boldsymbol{Q}^{(2)}$ und $\boldsymbol{Q'}^{(2)}$ von $\mathcal{S}^{(2)}$.

Die Komposition $\mathcal{S} = \mathcal{S}^{(1)} \to \mathcal{S}^{(2)}$ ist ein synchroner Schaltkreis mit Ausgabeberechtigung $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ vom Typ $(l_1 + l_2, m, n)$ mit den Eingabevariablen $\boldsymbol{X}$, der Ausgabevariable H und Ausgabeberechtigungsvariable A, der definiert wird durch

- die Übergangsfunktion

$$\delta : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} & \to & \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} \\ (\boldsymbol{X}, \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}) & \mapsto & (\boldsymbol{Q'}^{(1)}, \boldsymbol{Q'}^{(2)}) \end{cases}$$

  mit

$$\begin{aligned} Q'^{(1)}_i &= \delta^{(1)}_i \quad \forall i(0 \leq i < l_1) \\ Q'^{(2)}_i &= \delta^{(2)}_i|_{X^{(2)}_j = \lambda^{(1)}_j (0 \leq j < k)} \quad \forall i(0 \leq i < l_2) \end{aligned}$$

- die Hilfsausgabefunktion

$$\lambda : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} & \to & \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}) & \mapsto & \boldsymbol{H} \end{cases}$$

  mit

$$H_i = \lambda^{(2)}_i|_{X^{(2)}_j = \lambda^{(1)}_j (0 \leq j < k)} \quad \forall i(0 \leq i < n)$$

- die Ausgabeberechtigung

$$\alpha : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l_1} \times \mathbb{B}^{l_2} & \to & \mathbb{B}^n \\ (\boldsymbol{X}, \boldsymbol{Q}^{(1)}, \boldsymbol{Q}^{(2)}) & \mapsto & \boldsymbol{A} \end{cases}$$

  mit

$$A_i = \alpha^{(2)}_i|_{X^{(2)}_j = \lambda^{(1)}_j (0 \leq j < k)} \quad \forall i(0 \leq i < n)$$

- und den Startzustand $\boldsymbol{s} = \boldsymbol{s}^{(1)}.\boldsymbol{s}^{(2)}$.

Bemerkung 5.10. Bedingungen für die Realisierung der Komposition 5.9 $\mathcal{S} = \mathcal{S}^{(1)} \to \mathcal{S}^{(2)}$ sind:

- Die Anzahl der Ausgaben von $\mathcal{S}^{(1)}$=Anzahl der Eingaben von $\mathcal{S}^{(2)}$.
- $\mathcal{S}^{(1)}$ ist ein Mealy Automat, d.h die Ausgabeberechtigung liefert immer 1. Nur so ist gewährleistet, daß in jedem Berechnungsschritt an $\mathcal{S}^{(2)}$ eine Eingabe weitergeleitet wird. Mit dieser Einschränkung kann jedoch der oben beschriebene Korrektheitstest für Kapitel 6 durchgeführt werden.

Die Eingabevariablen von $\mathcal{S}^{(2)}$ werden mit der Ausgabefunktion von $\mathcal{S}^{(1)}$ substituiert. Da Eingabevariablen meistens einen sehr großen Einfluß auf ihre boolesche Funktion besitzen, erlangen durch diese Substitution die Variablen der Ausgabefunktion von $\mathcal{S}^{(1)}$, d.h. die Zustandsvariablen und Eingabevariablen von $\mathcal{S}^{(1)}$, einen dominierenden Einfluß auf die booleschen Funktionen von $\mathcal{S}^{(2)}$. Deshalb werden die Zustandsvariablen und Eingabevariablen von $\mathcal{S}^{(1)}$ in der Komposition

$\mathcal{S}^{(1)} \to \mathcal{S}^{(2)}$ wie Eingabevariablen behandelt, d.h. sie werden in der Variablenordnung zuerst eingeordnet. Mit dieser Strategie bleiben die OBDDs von $\mathcal{S}^{(2)}$ möglichst klein (siehe Abschnitt 2.4.2).

Durch die Substitution der OBDDs von $\mathcal{S}_1$ in $\mathcal{S}_2$ geht jedoch die lokale Abhängigkeit der Zustandsvariablen und Eingabevariablen von $\mathcal{S}^{(1)}$ verloren. Aus der Partitionierung der ursprünglichen Transitionsrelationen läßt sich ohne Analyse der synchronen Schaltkreise keine sinnvolle Regel ableiten, wie die Transitionsrelation von $\mathcal{S}$ strukturiert sein soll.

Im folgenden werden alle Variablen von $\mathcal{S}^{(1)}$ in einem Partitionsblock (siehe Abschnitt 4.3.2) zusammengefaßt. Da in $\mathcal{S}^{(1)}$ keine Zustandsvariablen aus $\mathcal{S}^{(2)}$ vorkommen, bleibt die Partitionierung der Variablen von $\mathcal{S}^{(2)}$ erhalten. Mit dieser Aufteilung der Variablen wird nun die partitionierte Transitionsrelation von $\mathcal{S}$ berechnet.

Bemerkung 5.11. Die Komposition $\mathcal{S} = \mathcal{S}^{(1)} \to \mathcal{S}^{(2)}$ gelingt mit OBDDs nur, wenn die Hilfsausgabefunktion von $\mathcal{S}^{(1)}$ und die OBDDs von $\mathcal{S}^{(2)}$ nicht zu groß sind. Ansonsten werden durch die Substitution der OBDDs in Konstruktion 5.9 die OBDDs der Zustandsfunktionen von $\mathcal{S}$ zu groß. Mit diesen Zustandsfunktionen kann dann in vielen Fällen nicht mehr die Transitionsrelation berechnet werden.

6 Lösung von online Gleichungen

Eine wichtige Frage, die an einem synchronen Schaltkreis $\mathcal{S}$ zu stellen ist, lautet, welche Eingaben möglich sind, damit $\mathcal{S}$ eine bestimmte Ausgabe berechnet. Wie in der Einleitung beschrieben, können die gesuchten Eingaben L mit einem synchronen Schaltkreis $\mathcal{S}_L$ dargestellt werden. Zu einer beliebigen Eingabe berechnet dieser synchrone Schaltkreis $\mathcal{S}_L$ eine Folge aus der gewünschten Lösung L. Werden mit allen Eingaben alle Folgen aus L berechnet, so wird L in kompakter Form durch $\mathcal{S}_L$ kodiert.

Dieses Problem, aus einem synchronen Schaltkreis $\mathcal{S}$ einen synchronen Schaltkreis $\mathcal{S}_L$ zu konstruieren, kann mit Hilfe von endlichen online Funktionen geschickt gelöst werden.

Im folgenden Kapitel gelten die Notationen aus 3.10.

Da das Ausgabeverhalten eines Mealy Automaten $\mathcal{S}$ vom Typ (l, m, n) äquivalent durch eine online Funktion $\tilde{f} : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ beschrieben werden kann, darf das Problem folgendermaßen umformuliert werden.

Gesucht werden alle Eingaben $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$, sodaß für eine online Funktion

$$\tilde{f} : \begin{cases} {}_2\mathbb{Z}^m & \to & {}_2\mathbb{Z}^n \\ \boldsymbol{\xi} & \mapsto & \sum_{t\geq 0}(\tilde{f}_0^{(t)}(\boldsymbol{\xi}^{[t]}), \ldots, \tilde{f}_{n-1}^{(t)}(\boldsymbol{\xi}^{[t]}))2^t \end{cases}$$

die Gleichung

$$\tilde{f}(\boldsymbol{\xi}) = \boldsymbol{v}$$

mit $\boldsymbol{v} = \sum_{t\geq 0}(y_0^{(t)}, \ldots, y_{n-1}^{(t)})2^t \in {}_2\mathbb{Z}^n$ erfüllt wird.

Mit der online Funktion

$$f : \begin{cases} {}_2\mathbb{Z}^m & \to & {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto & \sum_{t\geq 0} f^{(t)}(\boldsymbol{\xi}^{[t]})2^t \end{cases}$$

mit

$$f^{(t)} = \bigvee_{0\leq i<n} f_i^{(t)}$$

wobei

$$f_i^{(t)} = \begin{cases} \tilde{f}_i^{(t)}, & \text{wenn} \quad y_i^{(t)} = 0 \\ \neg\tilde{f}_i^{(t)}, & \text{wenn} \quad y_i^{(t)} = 1 \end{cases}$$

wird also die Nullstellenmenge

$$L = \{\boldsymbol{\xi} \in {}_2\mathbb{Z}^m \mid f(\boldsymbol{\xi}) = 0^\omega\}$$

gesucht.

Findet man nun eine online Funktion $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ mit

$$L = \{\sigma(\boldsymbol{\zeta}) \mid \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m\}$$

so wird die gesuchte Nullstellenmenge von einer online Funktion beschrieben. σ ist in diesem Zusammenhang eine Lösung zu der online Gleichung $f = 0^\omega$.

Definition 6.1. Sei $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine online Funktion. Eine Funktion $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ heißt *parametrisierte Lösung* von der Gleichung

$$f(\boldsymbol{\xi}) = 0^\omega$$

(kurz: parametrisierte Lösung von f), wenn gilt

$$\forall \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m (f(\sigma(\boldsymbol{\zeta})) = 0^\omega)$$

Eine parametrisierte Lösung σ von f heißt *allgemein*, wenn zusätzlich gilt

$$\forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m (f(\boldsymbol{\xi}) = 0^\omega \Rightarrow \exists \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m (\boldsymbol{\xi} = \sigma(\boldsymbol{\zeta})))$$

Eine parametrisierte allgemeine Lösung σ von f heißt *reproduktiv*, wenn gilt

$$\forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m (f(\boldsymbol{\xi}) = 0^\omega \Rightarrow \sigma(\boldsymbol{\xi}) = \boldsymbol{\xi})$$

In diesem Kapitel wird nun vorgestellt, wie eine reproduktive allgemeine parametrisierte Lösung zu einer online Gleichung gewonnen werden kann.

Da mit dem Lösen von einstelligen online Gleichungen ($m = 1$) die Grundidee ohne einen aufwendigen Formalismus vorgestellt werden kann, wird in einer Einführung dieser Fall detailliert ausgeführt. Für den mehrstelligen Fall ($m > 1$) gelingt dann eine Erweiterung, die den Formalismus stark erhöht, jedoch die Grundidee aus dem 1-stelligen Fall nicht verändert. Diese Einführung soll also insbesondere eine Art Lesehilfe für den mehrstelligen Fall darstellen.

Anschließend wird in Abschnitt 6.2 der mehrstellige Fall behandelt. In Abschnitt 6.3 wird dann vorgestellt, wie ein synchroner Schaltkreis konstruiert werden kann, der die Lösung σ von einer online Gleichung $f = 0^\omega$ berechnet. Im Anschnitt 6.4 wird danach das Lösen von parameterbehafteten Gleichungen und die Umsetzung für synchrone Schaltkreise in Abschnitt 6.5 behandelt.

6.1 Einführung - einstellige online Gleichungen

Gesucht ist eine reproduktive allgemeine parametrisierte Lösung

$$\sigma : \begin{cases} {}_2\mathbb{Z} & \rightarrow & {}_2\mathbb{Z} \\ \zeta & \mapsto & \sum_{t \geq 0} \sigma^{(t)}(z^{[t]})2^t \end{cases}$$

für die online Gleichung $f(\xi) = \sum_{t \geq 0} f^{(t)}(x^{[t]})2^t = 0^\omega$.

Mit folgenden einfachen Gedanken wird σ im Berechnungsschritt $t + 1$ zur Eingabe $\zeta \in {}_2\mathbb{Z}$ umgesetzt:

- $\sigma^{(t+1)}$ betracht die Ausgabe $a = f^{(t+1)}(\sigma^{(0)}(\zeta^{[0]}), \ldots, \sigma^{(t)}(\zeta^{[t]}), z^{(t+1)})$
- Wenn $a = 0$ ist, so gibt $\sigma^{(t+1)}$ den Wert $z^{(t+1)}$
- sonst gibt $\sigma^{(t+1)}$ den Wert $\neg z^{(t+1)}$ aus.

$\sigma^{(t+1)}$ betrachtet also nur das Ausgabeverhalten von $f^{(t+1)}$, um eine Lösung $\sigma^{(t)}(\zeta^{[t+1]})$ zu ermitteln, sodaß gilt

$$f^{(t+1)}(\sigma^{(0)}(\zeta^{[0]}), \ldots, \sigma^{(t+1)}(\zeta^{[t+1]})) = 0$$

Diese Berechnung ist korrekt, wenn f folgende Eigenschaften besitzt:

- Es gibt eine Lösung $\xi = \sum_{t \geq 0} x^{(t)}2^t$ mit $f(\xi) = 0^\omega$

- Ist $x^{[t]} = \langle x^{(0)}, x^{(1)}, \ldots, x^{(t)} \rangle$ eine Anfangslösung mit

$$f^{(0)}(x^{[0]}) = \cdots = f^{(t)}(x^{[t]}) = 0$$

so kann diese Folge fortgesetzt werden zu einer Folge $\xi \in {}_2\mathbb{Z}$ mit $f(\xi) = 0^\omega$.

Eine online Funktion f mit diesen Eigenschaften heißt *erweiterbar*. Diese Eigenschaften können folgendermaßen charakterisiert[5] werden.

Definition 6.2. Eine online Funktion$f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ heißt *erweiterbar*, wenn gilt

$$\forall t \in \mathbb{N}_0 \forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m ({}_2|f(\xi)| \leq 2^{-t} \Rightarrow \exists \boldsymbol{v} \in {}_2\mathbb{Z}^m ({}_2|\boldsymbol{v} - \boldsymbol{\xi}| \leq 2^{-t} \wedge f(\boldsymbol{v}) = 0^\omega))$$

Daß mit dieser Strategie tatsächlich eine reproduktive allgemeine parametrisierte Lösung realisiert wird, kann mit den folgenden Überlegungen bewiesen werden.

Die Konstruktionsidee besteht nun darin, σ mit Hilfe von reproduktiven allgemeinen parametrisierten Lösungen von den booleschen Gleichungen

$$f^{(t)}(x^{(0)}, \ldots, x^{(t)}) = 0 \quad t \geq 0$$

zu bestimmen.

Definition 6.3. Sei $f : \mathbb{B}^m \to \mathbb{B}$ ein boolescher Funktionsvektor. Eine Funktion $\sigma : \mathbb{B}^m \to \mathbb{B}^m$ heißt *parametrisierte Lösung* von der Gleichung

$$f(x) = 0$$

(kurz: parametrisierte Lösung von f), wenn gilt

$$\forall u \in \mathbb{B}^m (f(\sigma(u)) = 0)$$

Eine parametrisierte Lösung σ von f heißt *allgemein*, wenn zusätzlich gilt

$$\forall x \in \mathbb{B}^m (f(x) = 0 \Rightarrow \exists u \in \mathbb{B}^m (x = \sigma(u)))$$

Eine parametrisierte Lösung σ von f heißt *reproduktiv*, wenn gilt

$$\forall x \in \mathbb{B}^m (f(x) = 0 \Rightarrow \sigma(x) = x)$$

In [HR68][Seite 26:Theorem 2,3] gibt S. Rudeanu für eine reproduktive allgemeine parametrisierte Lösung zu einer booleschen Gleichung folgende Lösung an.

Lemma 6.4. *Ist $f : \mathbb{B} \to \mathbb{B}$ eine boolesche Funktion, dann gilt*

1. *$f(x) = 0$ ist lösbar genau dann, wenn gilt*

$$f(0) \wedge f(1) = 0$$

2. *Ist f lösbar, so ist*

$$\sigma : \begin{cases} \mathbb{B} & \to \quad \mathbb{B}^m \\ z & \mapsto \quad \mathrm{ite}(z, \neg f(1), f(0)) \end{cases}$$

eine reproduktive allgemeine parametrisierte Lösung von f

[5] Die erste Bedingung wird insbesondere mit $t = 0$ abgedeckt, da ${}_2|f(\xi)| \leq 2^1$ für alle $\xi \in {}_2\mathbb{Z}$ erfüllt wird.

Mit diesem Hilfsmittel gelingt es nun, eine Lösung σ für die online Gleichung $f = 0$ anzugeben.

Sei die reproduktive allgemeine parametrisierte Lösung

$$\sigma^{[t]} : \left\{ \begin{array}{lll} \mathbb{B}^{t+1} & \to & \mathbb{B}^{t+1} \\ (z^{[t]}) & \mapsto & (\sigma^{(0)}(z^{[0]}), \ldots, \sigma^{(t)}(z^{[t]})) \end{array} \right.$$

zu dem booleschen Gleichungssystem

$$\begin{array}{rcl} f^{(0)}(x^{[0]}) & = & 0 \\ f^{(1)}(x^{[1]}) & = & 0 \\ & \vdots & \\ f^{(t)}(x^{[t]}) & = & 0 \end{array} \tag{11}$$

bzw. für die boolesche Gleichung

$$F^{(t)}(x^{[t]}) = \bigvee_{0 \le i \le t} f^{(i)}(x^{[i]}) = 0$$

gegeben.

Im Induktionsschritt muß $\sigma^{(t+1)}$ bestimmt werden, sodaß

$$\sigma^{[t+1]} : \left\{ \begin{array}{lll} \mathbb{B}^{t+2} & \to & \mathbb{B}^{t+2} \\ (z^{[t+1]}) & \mapsto & (\sigma^{(0)}(z^{[0]}), \ldots, \sigma^{(t+1)}(z^{[t+1]})) \end{array} \right.$$

eine reproduktive allgemeine parametrisierte Lösung für das erweiterte boolesche Gleichungssystem

$$\begin{array}{rcl} f^{(0)}(x^{[0]}) & = & 0 \\ f^{(1)}(x^{[1]}) & = & 0 \\ & \vdots & \\ f^{(t+1)}(x^{[t+1]}) & = & 0 \end{array} \tag{12}$$

bzw. für die boolesche Gleichung

$$F^{(t+1)}(x^{[t+1]}) = \bigvee_{0 \le i \le t+1} f^{(i)}(x^{[i]}) = 0$$

wird. Erfüllt $\sigma^{(t+1)}$ die Bedingungen

1. Für alle $b^{[t+1]} \in \mathbb{B}^{t+2}$ gilt
$$f^{(t+1)}(\sigma^{[t+1]}(b^{[t+1]})) = 0,$$
2. Für alle $b^{[t+1]} = (b^{(0)}, \ldots, b^{(t+1)}) \in \mathbb{B}^{t+2}$ mit $F^{(t+1)}(b^{[t+1]}) = 0$ gilt
$$\sigma^{(t+1)}(b^{[t+1]}) = b^{(t+1)}$$
d.h. es gilt
$$\sigma^{[t+1]}(b^{[t+1]}) = (\sigma^{[t]}(b^{[t]}), \sigma^{(t+1)}(b^{[t+1]})) \stackrel{I.V.,2.}{=} (b^{(0)}, \ldots, b^{(t)}, b^{(t+1)})$$

so ist $\sigma^{[t+1]}$ wegen 1. eine parametrisierte und wegen 2. eine reproduktive Lösung von $F^{(t+1)}$. Damit ist $\sigma^{[t+1]}$ insbesondere eine parametrisierte allgemeine Lösung, also insgesamt eine reproduktive allgemeine parametrisierte Lösung von $F^{(t+1)}$.

Diese zwei Bedingungen können erfüllt werden, wenn $\sigma^{(t+1)}$ eine reproduktive allgemeine parametrisierte Lösung mit dem Parameter $x^{(t+1)}$ für die boolesche Gleichung

$$\tilde{f}(z^{[t]}, x^{(t+1)}) = f^{(t+1)}(\sigma^{[t]}(z^{[t]}), x^{(t+1)}) = 0$$

ist.

Wann und wie kann nun eine reproduktive allgemeine parametrisierte Lösung zu $\tilde{f} = 0$ gewonnen werden?

Für eine boolesche Funktion

$$\tilde{f}|_{z^{(0)}=b^{(0)},\ldots,z^{(t)}=b^{(t)}}$$

mit $b^{[t]} = (b^{(0)}, \ldots, b^{(t)}) \in \mathbb{B}^{t+1}$ ist nach obigem Lemma

$$\tilde{\sigma}_{(b^{(0)},\ldots,b^{(t)})} : \left\{ \begin{array}{lcl} \mathbb{B} & \to & \mathbb{B} \\ z^{(t+1)} & \mapsto & \mathrm{ite}(z^{(t+1)}, \neg\tilde{f}|_{z^{(0)}=b^{(0)},\ldots,z^{(t)}=b^{(t)}}(1), \tilde{f}|_{z^{(0)}=b^{(0)},\ldots,z^{(t)}=b^{(t)}}(0)) \end{array} \right.$$

eine reproduktive allgemeine parametrisierte Lösung, wenn

$$\tilde{f}|_{z^{(0)}=b^{(0)},\ldots,z^{(t)}=b^{(t)}}(1) \wedge \tilde{f}|_{z^{(0)}=b^{(0)},\ldots,z^{(t)}=b^{(t)}}(0) = 0$$

gilt.

Gelingt dieser Schritt für alle $b^{[t]} \in \mathbb{B}^{t+1}$, d.h. gilt

$$f^{(t+1)}(\sigma^{[t]}(b^{[t]}), 1) \wedge f^{(t+1)}(\sigma^{[t]}(b^{[t]}), 0) = 0 \tag{13}$$

so können die booleschen Funktionen $\tilde{\sigma}_{b^{[t]}}$ zu der Lösung

$$\sigma^{(t+1)} : \left\{ \begin{array}{lcl} \mathbb{B}^{t+2} & \to & \mathbb{B} \\ z^{[t+1]} & \mapsto & \mathrm{ite}(z^{(t+1)}, \neg\tilde{f}(z^{[t]}, 1), \tilde{f}(z^{[t]}, 0)) \end{array} \right.$$

für $\tilde{f}$ zusammengefaßt werden.

Die obigen zwei Bedingungen können mit diesem $\sigma^{(t+1)}$ erfüllt werden, denn

zu 1: für alle $b^{[t+1]} \in \mathbb{B}^{t+2}$ gilt

$$f^{(t+1)}(\sigma^{[t+1]}(b^{[t+1]})) = \tilde{f}(b^{[t]}, \sigma^{(t+1)}(b^{[t+1]})) = 0$$

zu 2: für alle $b^{[t+1]} \in \mathbb{B}^{t+2}$ mit $F^{(t+1)}(b^{[t+1]}) = 0$ gilt

$$\begin{aligned} \sigma^{(t+1)}(b^{[t+1]}) &= \mathrm{ite}(b^{(t+1)}, \neg f^{(t+1)}(\sigma^{[t]}(b^{[t]}), 1), f^{(t+1)}(\sigma^{[t]}(b^{[t]}), 0)) \\ &= \mathrm{ite}(b^{(t+1)}, \neg f^{(t+1)}(b^{[t]}, 1), f^{(t+1)}(b^{[t]}, 0)) \\ &= b^{(t+1)} \end{aligned}$$

Für den Induktionsanfang ist wegen Lemma 6.4 $\sigma^{[0]} = (\sigma^{(0)})$ mit

$$\sigma^{(0)} : \left\{ \begin{array}{lcl} \mathbb{B} & \to & \mathbb{B} \\ z^{(0)} & \mapsto & \mathrm{ite}(z^{(0)}, \neg f(1), f(0)) \end{array} \right.$$

eine reproduktive allgemeine parametrisierte Lösung für

$$F^{(0)}(x^{(0)}) = f^{(0)}(x^{(0)})$$

wenn die Bedingung

$$f^{(0)}(0) \wedge f^{(0)}(1) = 0 \tag{14}$$

erfüllt ist.

Gelingt der Übergang mit Bedingung (13) von $\sigma^{[t]}$ auf $\sigma^{[t+1]}$ für alle $t \geq 0$ und kann $\sigma^{[0]}$ bestimmt werden mit Bedingung (14), so konvergiert die Folge $\langle \sigma^{[t]} \rangle_{t \geq 0}$ gegen die gesuchte online Funktion σ.

Insgesamt kann die reproduktive allgemeine parametrisierte Lösung σ von f berechnet werden mit

$$\sigma^{(0)} : \left\{ \begin{array}{lcl} \mathbb{B} & \rightarrow & \mathbb{B} \\ z^{(0)} & \mapsto & \mathrm{ite}(z^{(0)}, \neg f(1), f(0)) \end{array} \right.$$

$$\sigma^{(t+1)} : \left\{ \begin{array}{lcl} \mathbb{B}^{t+2} & \rightarrow & \mathbb{B} \\ z^{[t+1]} & \mapsto & \mathrm{ite}(z^{(t+1)}, \neg f(\sigma^{[t]}(z^{[t]}), 1), f(\sigma^{[t]}(z^{[t]}), 0)) \end{array} \right.$$

und σ kann dargestellt werden durch

$$\sigma : \left\{ \begin{array}{lcl} {}_2\mathbb{Z} & \rightarrow & {}_2\mathbb{Z} \\ \zeta & \mapsto & \mathrm{ite}(\zeta, \neg(f^{(0)}(1) + 2f_{\langle 1 \rangle}(\sigma(\zeta))), f^{(0)}(0) + 2f_{\langle 0 \rangle}(\sigma(\zeta))) \end{array} \right.$$

Die Bedingungen (13) und (14) werden von f erfüllt, wenn f erweiterbar ist.

Denn ist f erweiterbar, so gibt es eine Lösung $\xi \in {}_2\mathbb{Z}$ mit $f(\xi) = 0^\omega$. Es gilt deshalb Bedingung (14) und es kann $\sigma^{(0)}$ berechnet werden. Sei nun bereits gezeigt, daß Bedingung (13) für alle $i \leq t$ erfüllt ist. Damit gibt es eine reproduktive allgemeine parametrisierte Lösung $\sigma^{[t]}$ zu der booleschen Funktion $F^{(t)}$. Es gilt damit $f^{(t)}(\sigma^{[t]}(b^{[t]})) = 0$ für alle $b^{[t]} \in \mathbb{B}^{t+1}$. Da f erweiterbar ist, kann jede Folge $\sigma^{[t]}(b^{[t]}) \in \mathbb{B}^{t+1}$ zu einer Folge ξ mit $f(\xi) = 0^\omega$ fortgesetzt werden, d.h. es gilt insbesondere Bedingung (13).

Insgesamt gilt also folgender Satz

Satz 6.5. *Ist $f : {}_2\mathbb{Z} \rightarrow {}_2\mathbb{Z}$ eine erweiterbare online Funktion, so ist*

$$\sigma : \left\{ \begin{array}{lcl} {}_2\mathbb{Z} & \rightarrow & {}_2\mathbb{Z} \\ \zeta & \mapsto & \mathrm{ite}(\zeta, \neg(f^{(0)}(1) + 2f_{\langle 1 \rangle}(\sigma(\zeta))), f^{(0)}(0) + 2f_{\langle 0 \rangle}(\sigma(\zeta))) \end{array} \right.$$

eine reproduktive allgemeine parametrisierte Lösung von f. Ist f eine endliche online Funktion, dann ist σ auch eine endliche online Funktion.

Wie kann eine online Funktion nun zu einer erweiterbaren online Funktion umgewandelt werden?

Die gelingt, wenn das Ausgabeverhalten von f so modifiziert wird, daß es keine Anfangsfolge $x^{[t]} \in \mathbb{B}^{t+1}$ mit

$$f^{(0)}(x^{[0]}) = \cdots = f^{(t)}(x^{[t]}) = 0$$

gibt, sodaß gilt

$$f^{(t+1)}(x^{[t]}, 1) \wedge f^{(t+1)}(x^{[t]}, 0) = 1$$

Diese "problematische" Folge kann dann nicht zu einer Lösung ξ mit $f(\xi)$ fortgesetzt werden.

Mit folgender Iterationsvorschrift

$$\begin{array}{rcl} f[0] & := & \mathrm{mem(f)} \\ f[j+1] & := & f[j]_{\langle 1 \rangle} \wedge f[j]_{\langle 0 \rangle} \quad \forall j \geq 0 \end{array}$$

wird in [Stu96] die Existenz eines Fixpunktes $f[*]$ nachgewiesen. Ist die online Funktion endlich, so gelingt es sogar, einen Fixpunkt $f[M]$ zu berechnen mit $f[M] = f[M+j]$ für $j \geq 0$. In diesem sind dann alle problematischen Eingabefolgen für σ eliminiert worden.

- Im Schritt 0 bleibt die Ausgabe immer 1, wenn f das erste Mal eine 1 ausgibt.
- Im Schritt $j+1$ wird das Ausgabeverhalten von $f[j]$ folgendermaßen modifiziert. Wenn für eine Folge $x^{[t]} \in \mathbb{B}^{t+1}$ mit
$$f^{(0)}(x^{[0]}) = \cdots = f^{(t)}(x^{[t]}) = 0$$
die Gleichung
$$f[j]^{(t+1)}(x^{[t]}, 1) \wedge f[j]^{(t+1)}(x^{[t]}, 0) = 1$$
gilt, so wird $f[j+1]$ mit $f[j]_{\langle 1\rangle} \wedge f[j]_{\langle 0\rangle}$ berechnet, sodaß gilt
$$f[j+1]^{(t)}(x^{[t]}) = 1$$
Mit jedem Iterationsschritt werden damit die Längen der problematischen Eingabefolgen von $f[j+1]$ im Vergleich zu $f[j]$ um 1 verkürzt.

Nach endlich vielen Schritten können alle problematischen Eingabefolgen eliminiert werden, sodaß folgendes gilt:

- $f[M] = 1$: Es gibt keine Lösung $\xi \in {}_2\mathbb{Z}$ mit $f(\xi) = 0^\omega$.
- $f[M] \neq 1$: $f[M]$ ist erweiterbar.

Für den mehrstelligen Fall $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z} \in \mathrm{eON}_{m,1}$ wird für eine entsprechend erweiterte Fixpunktberechnung die Terminierung und Korrektheit im nächsten Abschnitt formal nachgewiesen.

6.2 Mehrstellige online Gleichungen

6.2.1 Lösung von erweiterbaren online Gleichungen

Gesucht wird eine reproduktive allgemeine parametrisierte Lösung
$$\sigma(\boldsymbol{\zeta}) = \sum_{t\geq 0} \sigma^{(t)}(\boldsymbol{\zeta}^{[t]})2^t$$
für die online Gleichung
$$f(\boldsymbol{\xi}) = \sum_{t\geq 0} f^{(t)}(\boldsymbol{\xi}^{[t]})2^t = 0^\omega$$

Satz 6.5 aus dem vorherigen Abschnitt gibt für den speziellen Fall $\mathrm{ON}_{1,1}$ einen Lösungsvorschlag an. Mit der Anwendung von Lemma 6.4 gelingt es dort, aus reproduktiven allgemeinen parametrisierten Lösungen für einstellige boolesche Funktionen eine Lösung für einstellige online Gleichungen zu konstruieren. Rudeanu gibt nun für boolesche Funktionen $f : \mathbb{B}^m \to \mathbb{B}$ die Erweiterung von Satz 6.5 in [HR68][Seite 34: Therorem 9] an.

Hierfür wird folgende Schreibweise eingeführt.

Notation 6.6. Sei $f : \mathbb{B}^m \to \mathbb{B}$ eine boolesche Funktion mit den Variablen $\{x_0, \ldots, x_{m-1}\}$. Dann wird für $(0 \leq i < m)$ mit $V_i = \{x_{m-i}, \ldots, x_{m-1}\}$ folgende Notation eingeführt
$$\begin{aligned} f_{\{0\}}(x_0, \ldots, x_{m-1}) &= f \\ f_{\{i\}}(x_0, \ldots, x_{m-1-i}) &= \forall_{V_i} f(x_0, \ldots, x_{m-1}) \end{aligned}$$

Lemma 6.7. *Sei $f : \mathbb{B}^m \to \mathbb{B}$ eine boolesche Funktion mit den Variablen $V = \{x_0, \dots, x_{m-1}\}$. Dann gilt*

1. *$f(x) = 0$ ist lösbar genau dann, wenn gilt*
$$\forall_V\, f(x) = 0$$

2. *Ist f lösbar, so ist*
$$\sigma : \begin{cases} \mathbb{B}^m & \to \quad \mathbb{B}^m \\ (z_0, \dots, z_{m-1}) & \mapsto \quad (\tau_0(z_0), \tau_1(z_0, z_1), \dots, \tau_{m-1}(z_0, \dots, z_{m-1})) \end{cases}$$
mit
$$\begin{aligned} \tau_0(z_0) &= \mathrm{ite}(z_0, \neg f_{\{m-1\}}(1), f_{\{m-1\}}(0)) \\ \tau_1(z_0, z_1) &= \mathrm{ite}(z_1, \neg f_{\{m-2\}}(\tau_0(z_0), 1), f_{\{m-2\}}(\tau_0(z_0), 0)) \\ &\vdots \\ \tau_i(z_0, \dots, z_i) &= \mathrm{ite}(z_i, \neg\ f_{\{m-1-i\}}(\tau_0(z_0), \dots \tau_{i-1}(z_0, \dots, z_{i-1}), 1), \\ &\qquad\qquad f_{\{m-1-i\}}(\tau_0(z_0), \dots \tau_{i-1}(z_0 \dots, z_{i-1}), 0)) \\ &\vdots \\ \tau_{m-1}(z_0, \dots, z_{m-1}) &= \mathrm{ite}(z_{m-1}, \neg\ f_{\{0\}}(\tau_0(z_0), \dots \tau_{m-2}(z_0, \dots, z_{m-2}), 1), \\ &\qquad\qquad f_{\{0\}}(\tau_0(z_0), \dots \tau_{m-2}(z_0, \dots, z_{m-2}), 0)) \end{aligned}$$

eine reproduktive allgemeine parametrisierte Lösung von f.

Mit diesem Lemma kann nun Satz 6.5 erweitert werden für online Funktionen $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$. Der Beweis für den einstelligen Fall kann vollständig übernommen werden. Zum Verständnis sollte deshalb wegen der besseren Lesbarkeit der einstellige Fall betrachtet werden. Der zugehörige Beweis für den mehrstelligen Fall wird vor allem deshalb hier skizziert, um die Anwendung folgender Schreibweise nachzuvollziehen.

Notation 6.8. Sei
$$f : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t \geq 0} f^{(t)}(\boldsymbol{\xi}^{[t]}) 2^t \end{cases}$$

Dann wird für $(0 \leq i < m)$ mit $V_{(i,t)} = \{x^{(t)}_{m-i}, \dots, x^{(t)}_{m-1}\}$ folgende Notation eingeführt
$$\begin{aligned} f_{\{0\}}(\boldsymbol{\xi}) &= f \\ f_{\{i\}}(\boldsymbol{\xi}) &= \sum_{t \geq 0} \left(\forall_{V_{(i,t)}} f^{(t)}(\xi_0^{[t-1]}, x_0^{(t)}, \dots, \xi_{m-1}^{[t-1]}, x_{m-1}^{(t)}) \right) 2^t \end{aligned}$$

Sind $s_0, \dots, s_{m-1}$ online Funktionen mit
$$s_i : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t \geq 0} s_i^{(t)}(\boldsymbol{\xi}^{[t]}) 2^t \end{cases}$$

so gilt folgende Notation
$$f_{[s_0, s_1, \dots, s_k]}(\boldsymbol{\xi}) = \sum_{t \geq 0} f^{(t+1)}(\xi_0^{[t]}, s_0^{(t)}, \dots, \xi_k^{[t]}, s_k^{(t)}, \xi_{k+1}^{[t]}, x_{k+1}^{(t)}, \dots, \xi_{m-1}^{[t]}, x_{m-1}^{(t)}) 2^t$$

Satz 6.9. *Ist $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine erweiterbare online Funktion, so ist*

$$\sigma : \left\{ \begin{array}{lll} {}_2\mathbb{Z}^m & \to & {}_2\mathbb{Z}^m \\ \boldsymbol{\zeta} & \mapsto & (\tau_0(\boldsymbol{\zeta}), \ldots, \tau_{m-1}(\boldsymbol{\zeta})) \end{array} \right.$$

mit

$$\begin{array}{rcl} \tau_0 & = & \mathrm{ite}(\zeta_0, \neg(\; f^{(0)}_{\{m-1\}}(1) + 2f_{\{m-1\}((1^\omega))}(\tau_0, \ldots, \tau_{m-1})), \\ & & \qquad\quad f^{(0)}_{\{m-1\}}(0) + 2f_{\{m-1\}((0^\omega))}(\tau_0, \ldots, \tau_{m-1})) \\ & \vdots & \\ \tau_i & = & \mathrm{ite}(\zeta_i, \neg(\; f^{(0)}_{\{m-1-i\}}(\tau^{(0)}_0, \ldots, \tau^{(0)}_{i-1}, 1) + 2f_{\{m-1\}((\tau_0,\ldots,\tau_{i-1},1^\omega))}(\tau_0, \ldots, \tau_{m-1})), \\ & & \qquad\quad f^{(0)}_{\{m-1-i\}}(\tau^{(0)}_0, \ldots, \tau^{(0)}_{i-1}, 0) + 2f_{\{m-1\}((\tau_0,\ldots,\tau_{i-1},0^\omega))}(\tau_0, \ldots, \tau_{m-1})) \\ & \vdots & \\ \tau_{m-1} & = & \mathrm{ite}(\zeta_{m-1}, \neg(\; f^{(0)}_{\{0\}}(\tau^{(0)}_0, \ldots, \tau^{(0)}_{m-1}, 1) + 2f_{\{0\}((\tau_0,\ldots,\tau_{m-2},1^\omega))}(\tau_0, \ldots, \tau_{m-1})), \\ & & \qquad\quad f^{(0)}_{\{0\}}(\tau^{(0)}_0, \ldots, \tau^{(0)}_{m-1}, 0) + 2f_{\{0\}((\tau_0,\ldots,\tau_{m-2},0^\omega))}(\tau_0, \ldots, \tau_{m-1})) \end{array}$$

eine reproduktive allgemeine parametrisierte Lösung von f.

Beweis: Sei im folgenden $V^{(t)} = \{x^{(t)}_0, \ldots, x^{(t)}_{m-1}\}$.

Gesucht wird die reproduktive allgemeine parametrisierte Lösung

$$\sigma(\boldsymbol{\zeta}) = \sum_{t \geq 0} \sigma^{(t)}(\boldsymbol{\zeta}^{[t]})2^t$$

zu der online Gleichung $f(\boldsymbol{\xi}) = 0^\omega$.

Für den Induktionsanfang erhält man durch Anwendung von Lemma 6.7 die Berechnungsvorschrift

$$\sigma^{(0)} : \left\{ \begin{array}{lll} \mathbb{B}^m & \to & \mathbb{B}^m \\ (z^{(0)}_0, \ldots, z^{(0)}_{m-1}) & \mapsto & (\tau^{(0)}_0(z^{(0)}_0), \tau^{(0)}_1(z^{(0)}_0, z^{(0)}_1), \ldots, \tau^{(0)}_{m-1}(z^{(0)}_0, \ldots, z^{(0)}_{m-1})) \end{array} \right.$$

wobei die $\tau^{(0)}_i$ für $(0 \leq i < m)$ bestimmt sind mit

$$\begin{array}{rl} \tau^{(0)}_i(z^{(0)}_0, \ldots, z^{(0)}_i) = \mathrm{ite}(z^{(0)}_i, \neg & f^{(0)}_{\{m-1-i\}}(\tau^{(0)}_0(z^{(0)}_0), \ldots, \tau^{(0)}_{i-1}(z^{(0)}_0, \ldots, z^{(0)}_{i-1}), 1), \\ & f^{(0)}_{\{m-1-i\}}(\tau^{(0)}_0(z^{(0)}_0), \ldots, \tau^{(0)}_{i-1}(z^{(0)}_0 \ldots, z^{(0)}_{i-1}), 0)) \end{array}$$

wenn die Bedingung

$$\forall_{V^{(0)}} f^{(0)}(\boldsymbol{\xi}^{[0]}) = 0 \tag{15}$$

erfüllt ist.

Sei für den Induktionsschritt $t \to t+1$

$$\sigma^{[t]} : \left\{ \begin{array}{lll} \mathbb{B}^{m(t+1)} & \to & \mathbb{B}^{m(t+1)} \\ \boldsymbol{\zeta}^{[t]} & \mapsto & (\tau^{[t]}_0(\boldsymbol{\zeta}^{[t]}), \ldots, \tau^{[t]}_{m-1}(\boldsymbol{\zeta}^{[t]})) \end{array} \right.$$

mit

$$\tau^{[t]}_i = (\tau^{(0)}_i(\boldsymbol{\zeta}^{[0]}), \ldots, \tau^{(t)}_i(\boldsymbol{\zeta}^{[t]}))$$

eine reproduktive allgemeine parametrisierte Lösung zu der Gleichung

$$F^{(t)}(\boldsymbol{\xi}^{[t]}) = \bigvee_{0 \leq i \leq t} f^{(i)}(\boldsymbol{\xi}^{[i]}) = 0$$

Nach Lemma 6.7 erhält man dann die Lösung

$$\sigma^{(t+1)}(\boldsymbol{\zeta}^{[t+1]})) = (\tau_0^{(t+1)}(\boldsymbol{\zeta}^{[t+1]}), \ldots, \tau_{m-1}^{(t+1)}(\boldsymbol{\zeta}^{[t+1]}))$$

mit

$$\begin{aligned}
\tau_0^{(t+1)} &= \mathrm{ite}(z_0^{(t+1)}, \neg\ f_{\{m-1\}}^{(t+1)}(\tau_0^{[t]}, 1, \tau_1^{[t]}, \ldots, \tau_{m-1}^{[t]}),\\
&\phantom{= \mathrm{ite}(z_0^{(t+1)}, \neg\ } f_{\{m-1\}}^{(t+1)}(\tau_0^{[t]}, 0, \tau_1^{[t]}, \ldots, \tau_{m-1}^{[t]}))\\
&\vdots\\
\tau_i^{(t+1)} &= \mathrm{ite}(z_i^{(t+1)}, \neg\ f_{\{m-1-i\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{i-1}^{[t+1]}, \tau_i^{[t]}, 1, \tau_{i+1}^{[t]}, \ldots, \tau_{m-1}^{[t]}),\\
&\phantom{= \mathrm{ite}(z_i^{(t+1)}, \neg\ } f_{\{m-1-i\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{i-1}^{[t+1]}, \tau_i^{[t]}, 0, \tau_{i+1}^{[t]}, \ldots, \tau_{m-1}^{[t]}))\\
&\vdots\\
\tau_{m-1}^{(t+1)} &= \mathrm{ite}(z_{m-1}^{(t+1)}, \neg\ f_{\{0\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{m-2}^{[t+1]}, \tau_{m-1}^{[t]}, 1),\\
&\phantom{= \mathrm{ite}(z_{m-1}^{(t+1)}, \neg\ } f_{\{0\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{m-2}^{[t+1]}, \tau_{m-1}^{[t]}, 0)
\end{aligned}$$

für die boolesche Gleichung

$$f^{(t+1)}(\tau_0^{[t]}(\boldsymbol{\zeta}^{[t]}), x_0^{(t+1)}, \ldots, \tau_{m-1}^{[t]}(\boldsymbol{\zeta}^{[t]}), x_{m-1}^{(t+1)}) = 0$$

wenn die Bedingung

$$\forall_{V^{(t+1)}}\left(f^{(t+1)}(\tau_0^{[t]}(b^{[t]}), x_0^{(t+1)}, \ldots, \tau_{m-1}^{[t]}(b^{[t]}), x_{m-1}^{(t+1)})\right) = 0 \tag{16}$$

für alle $b^{[t]} = (b_0^{[t]}, \ldots, b_{m-1}^{[t]}) \in \mathbb{B}^{m(t+1)}$ erfüllt ist.

Es gelten die Eigenschaften:

1. Für alle $b^{[t+1]} = (b_0^{[t+1]}, \ldots, b_{m-1}^{[t+1]}) \in \mathbb{B}^{m(t+2)}$ gilt

$$f^{(t+1)}(\sigma^{[t+1]}(b^{[t+1]})) = 0$$

Dies ist nach Konstruktion erfüllt.

2. Für alle $b^{[t+1]} = (b_0^{[t+1]}, \ldots, b_{m-1}^{[t+1]}) \in \mathbb{B}^{m(t+2)}$ mit

$$F^{(t+1)}(b^{[t+1]}) = 0 \tag{17}$$

gilt

$$\sigma^{(t+1)}(b_0^{[t+1]}, \ldots, b_{m-1}^{[t+1]}) = (b_0^{(t+1)}, \ldots, b_{m-1}^{(t+1)})$$

Die Bedingung gilt bereits für alle $\tilde{t} \le t$ ($I.V.^{(1)}$). Im Schritt $t+1$ folgt dies mit Induktion über $0 \le i < m$. Im Schritt i gelte

$$\tau_j^{(t+1)}(b^{[t+1]}) = b_j^{(t+1)} \quad \forall j (0 \le j < i) \qquad\qquad (I.V.^{(2)})$$

Für $b_i^{(t+1)} = 1$ gilt $\tau_i^{(t+1)}(b^{[t+1]}) =$

$$\begin{aligned}
&\overset{I.V.^{(1)}}{=} \neg f_{\{m-1-i\}}^{(t+1)}(b_0^{[t]}, \tau_0^{(t+1)}(b^{[t+1]}), \ldots, b_{i-1}^{[t]}, \tau_{i-1}^{(t+1)}(b^{[t+1]}), b_i^{[t]}, 1, b_{i+1}^{[t]}, \ldots, b_{m-1}^{[t]})\\
&\overset{I.V.^{(2)}}{=} \neg f_{\{m-1-i\}}^{(t+1)}(b_0^{[t+1]}, \ldots, b_{i-1}^{[t+1]}, b_i^{[t]}, 1, b_{i+1}^{[t]}, \ldots, b_{m-1}^{[t]})\\
&\overset{6.6}{=} \neg \forall_{\{x_{i+1}, \ldots, x_{m-1}\}} f^{(t+1)}(b_0^{[t+1]}, \ldots, b_{i-1}^{[t+1]}, b_i^{[t]}, 1, b_{i+1}^{[t]}, x_i, \ldots, b_{m-1}^{[t]}, x_{m-1})\\
&\overset{(17)}{=} \neg 0 = 1
\end{aligned}$$

und für $b_i^{(t+1)} = 0$ analog $\tau_i^{(t+1)}(b^{[t+1]}) = 0$ und deshalb insgesamt $\tau_i^{(t+1)}(b^{[t+1]}) = b_i^{(t+1)}$.

Wegen dieser Eigenschaften ist $\sigma^{[t+1]}$ eine reproduktive allgemeine parametrisierte Lösung für $F^{(t+1)}$, wenn die Bedingungen (15) und (16) erfüllt sind. Werden Bedingung (15) und Bedingung (16) für alle $t \geq 0$ erfüllt, so gelingt die Konstruktion $\sigma^{(t)}$ für alle $t \geq 0$. Die zugehörigen booleschen Funktionen $\langle \tau_i^{(t)} \rangle_{t \geq 0}$ mit

$$\begin{aligned}\tau_i^{(0)}(\zeta_0^{[0]}, \ldots, \zeta_i^{[0]}) = \mathrm{ite}(z_i^{(0)}, \neg\ & f_{\{m-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 1), \\ & f_{\{m-1-i\}}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 0))\end{aligned}$$

und

$$\begin{aligned}&\tau_i^{(t+1)}(\zeta_0^{[t+1]}, \ldots, \zeta_i^{[t+1]}, \zeta_i^{[t]}, \ldots, \zeta_{m-1}^{[t]}) \\ &\qquad = \mathrm{ite}(z_i^{(t+1)}, \neg\ f_{\{m-1-i\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{i-1}^{[t+1]}, \tau_i^{[t]}, 1, \tau_{i+1}^{[t]}, \ldots, \tau_{m-1}^{[t]}), \\ &\qquad\qquad f_{\{m-1-i\}}^{(t+1)}(\tau_0^{[t+1]}, \ldots, \tau_{i-1}^{[t+1]}, \tau_i^{[t]}, 0, \tau_{i+1}^{[t]}, \ldots, \tau_{m-1}^{[t]}))\end{aligned}$$

können mit Hilfe der Notation 6.8 zu der online Funktion

$$\begin{aligned}\tau_i = \mathrm{ite}(\zeta_i, \neg(\ & f_{\{m-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 1) + 2f_{\{m-1\}((\tau_0, \ldots, \tau_{i-1}, 1^\omega))}(\tau_0, \ldots, \tau_{m-1})), \\ & f_{\{m-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 0) + 2f_{\{m-1\}((\tau_0, \ldots, \tau_{i-1}, 0^\omega))}(\tau_0, \ldots, \tau_{m-1}))\end{aligned}$$

zusammengefaßt werden und

$$\sigma : \begin{cases} {}_2\mathbb{Z}^m & \rightarrow \quad {}_2\mathbb{Z}^m \\ \boldsymbol{\zeta} & \mapsto \quad (\tau(\boldsymbol{\zeta}), \ldots, \tau_{m-1}(\boldsymbol{\zeta})) \end{cases}$$

ist eine reproduktive allgemeine parametrisierte Lösung von f.

Mit der gleichen Argumentation wie in Satz 6.5 erfüllt f die Bedingungen (15) und (16), wenn f eine erweiterbare online Funktion ist. □

6.2.2 Lösung von nicht erweiterbaren online Gleichungen

Für eine online Funktion $f : {}_2\mathbb{Z} \rightarrow {}_2\mathbb{Z} \in \mathrm{eON}_{1,1}$ ist die Fixpunktberechnung

$$\begin{aligned} f[0] &:= \mathrm{mem(f)} \\ f[j+1] &:= f[j]_{\langle 1 \rangle} \wedge f[j]_{\langle 0 \rangle} \quad \forall j \geq 0 \end{aligned}$$

beschrieben worden, um aus einer endlichen online Funktion f eine erweiterbare online Funktion mit der gleichen Nullstellenmenge zu berechnen.

Die Iterationsvorschrift kann für den mehrstelligen Fall nun folgendermaßen erweitert werden.

Definition 6.10. Sei $f : {}_2\mathbb{Z}^m \rightarrow {}_2\mathbb{Z}$ eine online Funktion und $f_{\langle b_0, \ldots, b_{m-1} \rangle}$ nach Definition 3.15 der Prädiktor zu $(b_0, \ldots, b_{m-1}) \in \mathbb{B}^m$. Dann wird folgende Iterationsvorschrift für f definiert

$$\begin{aligned} f[0] &:= \mathrm{mem(f)} \\ f[j+1] &:= \bigwedge_{b_0, \ldots, b_{m-1} \in \mathbb{B}} f[j]_{\langle b_0, \ldots, b_{m-1} \rangle} \end{aligned}$$

Für die späteren Betrachtungen wird folgende einfache Beobachtung benötigt.

Bemerkung 6.11. Die online Funktionen $f[j] : {}_2\mathbb{Z}^m \rightarrow {}_2\mathbb{Z}$ mit $j \geq 0$ zu der online Funktion $f : {}_2\mathbb{Z}^m \rightarrow {}_2\mathbb{Z}$ haben folgendes Ausgabeverhalten für ein $\alpha \in {}_2\mathbb{Z}^m$: entweder gilt $f[j](\alpha) = 0^\omega$ oder $f[j](\alpha) \in 0^*1^\omega$, d.h. es gilt

$$\mathrm{im}(f[j]) \subseteq 0^\omega \cup 0^*1^\omega$$

Terminierung

Wenn f eine endliche online Funktion ist, so ist $f[0]$ nach Definition 3.15 auch eine endliche online Funktion. Nach Bemerkung 3.16 gibt es deshalb nur endlich viele Prädiktoren

$$f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle}$$

mit $\alpha_0, \ldots, \alpha_{m-1} \in \mathbb{B}^k$ und $k \in \mathbb{N}_0$.

Im folgenden Lemma wird gezeigt, daß $f[j]$ mit Hilfe der Prädiktoren von $f[0]$ dargestellt werden kann. Da es nur endlich viele Prädiktoren von $f[0]$ gibt, kann mit dieser Feststellung die Terminierung der Fixpunktberechnung gezeigt werden, d.h. es gibt ein $M \in \mathbb{N}_0$ mit $f[M] = f[M+i]$ für jedes $i \geq 0$.

Lemma 6.12. *Für alle $j \in \mathbb{N}_0$ gilt:*

$$f[j] = \bigwedge_{\alpha_0,\ldots,\alpha_{m-1}\in\mathbb{B}^j} f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle} \tag{18}$$

Beweis: Die Behauptung wird durch Induktion über j gezeigt. Für den Induktionsanfang $j = 0$ ist nichts zu zeigen. Gilt Gleichung (18) für j, so folgt

$$\begin{aligned}
f[j+1] &= \bigwedge_{(b_0,\ldots,b_{m-1})\in\mathbb{B}^m} f[j]_{\langle b_0,\ldots,b_{m-1}\rangle} \\
&\overset{I.V.}{=} \bigwedge_{(b_0,\ldots,b_{m-1})\in\mathbb{B}^m} \left(\bigwedge_{(\alpha_0,\ldots,\alpha_{m-1})\in\mathbb{B}^j} f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle} \right)_{\langle b_0,\ldots,b_{m-1}\rangle} \\
&= \bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} \bigwedge_{\alpha_0,\ldots,\alpha_{m-1}\in\mathbb{B}^j} f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle\langle b_0,\ldots,b_{m-1}\rangle} \\
&= \bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} \bigwedge_{\alpha_0,\ldots,\alpha_{m-1}\in\mathbb{B}^j} f[0]_{\langle\alpha_0.b_0,\ldots,\alpha_{m-1}.b_{m-1}\rangle} \\
&= \bigwedge_{\beta_0,\ldots,\beta_{m-1}\in\mathbb{B}^{j+1}} f[0]_{\langle\beta_0,\ldots,\beta_{m-1}\rangle}
\end{aligned}$$

mit der Konkatenation $\alpha_i.b_i \in \mathbb{B}^{j+1}$ von $\alpha_i \in \mathbb{B}^j, b_i \in \mathbb{B}$. □

Folgende partielle Ordnung auf $\mathrm{ON}_{m,1}$ ermöglicht nun den Beweis für die Terminierung.

Definition 6.13. Seien $f(\boldsymbol{\xi}) = \sum f^{(t)}(\boldsymbol{\xi}^{[t]})2^t, g(\boldsymbol{\xi}) = \sum g^{(t)}(\boldsymbol{\xi}^{[t]})2^t \in \mathrm{ON}_{m,1}$.

Folgende *partielle Ordnung* $\leq$ wird auf $\mathrm{ON}_{m,1}$ definiert:

$$f \leq g \Leftrightarrow f^{(t)}(\boldsymbol{\xi}^{[t]}) \leq g^{(t)}(\boldsymbol{\xi}^{[t]}) \ \ \forall t \geq 0$$

Lemma 6.14. *Ist $f \in \mathrm{ON}_{m,1}$ und $k \in \mathbb{N}_0$, dann gilt*

$$\forall\alpha_0, \ldots, \alpha_{m-1} \in \mathbb{B}^k \forall b_0, \ldots, b_{m-1} \in \mathbb{B} : f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle} \leq f[0]_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle\langle b_0,\ldots,b_{m-1}\rangle}$$

Beweis: Sei $\alpha_0, \ldots, \alpha_{m-1} \in \mathbb{B}^k, k \in \mathbb{N}_0$. Da $\mathrm{im}(f[0]) \subseteq 0^\omega \cup 0^*1^\omega$ gilt, folgt aus

$$f[0]^{(t)}_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle}(\xi_0^{[t]}, \ldots, \xi_{m-1}^{[t]}) = f[0]^{(t+k)}(\xi_0^{[t]}, \alpha_0, \ldots, \xi_{m-1}^{[t]}, \alpha_{m-1}) = 1$$

die Gleichung

$$f[0]^{(t)}_{\langle\alpha_0,\ldots,\alpha_{m-1}\rangle\langle b_0,\ldots,b_{m-1}\rangle}(\boldsymbol{\xi}^{[t]}) = f[0]^{(t+k+1)}(\xi_0^{[t]}, \alpha_0, b_0, \ldots, \xi_{m-1}^{[t]}, \alpha_{m-1}, b_{m-1}) = 1$$

für alle $t \geq 0$ und alle $b_0, \ldots, b_{m-1}$ und damit die zu zeigende Behauptung. □

Lemma 6.15. *Für* $f \in \mathrm{eON}_{m,1}$ *gibt es ein* $M \in \mathbb{N}_0$ *mit* $f[M] = f[M+1]$.

Beweis:

Der entscheidende Schritt für diesen Beweis ist die Tatsache, daß die Anzahl der Prädiktoren von $f[0]$ endlich ist (siehe Bemerkung 3.16).

Daraus folgt, daß jede echt steigende Prädiktorkette

$$f[0] < f[0]_{\langle a_0\rangle} < f[0]_{\langle a_0\rangle\langle a_1\rangle} < \ldots < f[0]_{\langle a_0\rangle\ldots\langle a_e\rangle}$$

mit $a_0, \ldots, a_e \in \mathbb{B}^m$ nur endliche Länge besitzen kann.

Sei M die maximale Länge aller echt steigenden Prädiktorketten, $a_0, \ldots, a_{M-1} \in \mathbb{B}^m$ und $b \in \mathbb{B}^m$.

Wegen Lemma 6.14 gilt

$$f[0] \leq f[0]_{\langle a_0\rangle} \leq f[0]_{\langle a_0\rangle\langle a_1\rangle} \leq \ldots \leq f[0]_{\langle a_0\rangle\ldots\langle a_{M-1}\rangle} \leq f[0]_{\langle a_0\rangle\ldots\langle a_{M-1}\rangle\langle b\rangle}$$

Da M die maximale Länge der echt aufsteigenden Prädiktorketten ist, gibt es zwei Prädiktoren mit

$$f[0]_{\langle a_0\rangle\ldots\langle a_r\rangle} = f[0]_{\langle a_0\rangle\ldots\langle a_s\rangle}, \quad 0 \leq r < s < M$$

und daraus folgt $f[0]_{\langle a_0\rangle\ldots\langle a_r\rangle} = f[0]_{\langle a_0\rangle\ldots\langle a_r\rangle\langle a_{r+1}\rangle} = \ldots = f[0]_{\langle a_0\rangle\ldots\langle a_s\rangle}$. Jeder Prädiktor der Länge $M+1$ kann also durch einen Prädiktor der Länge M ausgedrückt werden:

$$f[0]_{\langle a_0\rangle\ldots\langle a_{M-1}\rangle\langle b\rangle} = f[0]_{\langle a_0\rangle\ldots\langle a_{r-1}\rangle\langle a_{r+1}\rangle\ldots\langle a_{M-1}\rangle\langle b\rangle}$$

Ist $P_f^j = \{f_{\langle a_0\rangle\ldots\langle a_{j-1}\rangle} \mid a_i \in \mathbb{B}(0 \leq i < j)\}$ die Menge der Prädiktoren von f der Länge j, so folgt also $P_f^M \supseteq P_f^{M+1}$. Dieselbe Argumentation gilt für alle $j \geq M$, sodaß gilt

$$P_f^M \supseteq P_f^{M+1} \supseteq P_f^{M+2} \supseteq \ldots$$

Da die Menge der Prädiktoren endlich ist, gibt es keine echte unendlich absteigende Kette, d.h. es gibt ein $j \geq M$ mit $P_f^j = P_f^{j+1}$. Mit Lemma 6.12 gilt damit $f[j] = f[j+1]$.

$\square$

Die Terminierung der Fixpunktberechnung ist deshalb garantiert.

Definition 6.16. Sei $f \in \mathrm{eON}_{m,1}$. Die online Funktion mit $f[M] = f[M+j]$ für alle $j \geq 0$ mit $M \in \mathbb{N}_0$ heißt *Fixpunkt* $f[*]$ von der Iterationsvorschrift 6.10.

Korrektheit

Gezeigt werden muß für den Fixpunkt $f[*]$, daß

- die Nullstellenmenge von f und $f[*]$ übereinstimmt und
- $f[*]$ eine erweiterbare online Funktion ist.

Lemma 6.17. *Sei* $f \in \mathrm{ON}_{m,1}$. *Dann gilt für alle* $j \in \mathbb{N}_0$ *und alle* $Vxi \in {}_2\mathbb{Z}^m$*:*

$$f(\boldsymbol{\xi}) = 0^\omega \Leftrightarrow f[j](\boldsymbol{\xi}) = 0^\omega \tag{19}$$

Beweis: Der Induktionsanfang für $j = 0$ ist klar:

$$\mathrm{mem}(\mathrm{f}(\boldsymbol{\xi})) = 0^\omega \Leftrightarrow \mathrm{f}(\boldsymbol{\xi}) = 0^\omega$$

Sei Aussage (19) für $j \geq 0$ erfüllt. Wegen

$$f[j+1]^{(t)}(\boldsymbol{\xi}^{[t]}) = \bigwedge_{(b_0,\ldots,b_{m-1})\in\mathbb{B}^m} f[j]^{(t+1)}(\xi_0^{[t]}, b_0, \ldots, \xi_{m-1}^{[t]}, b_{m-1})$$

gilt

$$\begin{aligned} &\forall t \geq 0 : (f[j+1]^{(t)}(\boldsymbol{\xi}^{[t]}) = 0 \\ &\quad \Leftrightarrow \exists (b_0, \ldots, b_{m-1}) \in \mathbb{B}^m f[j]^{(t+1)}(\xi_0^{[t]}, b_0, \ldots, \xi_{m-1}^{[t]}, b_{m-1}) = 0) \end{aligned} \tag{20}$$

Gilt $f[j+1](\boldsymbol{\xi}) = 0^\omega$ für $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$, so folgt wegen $\mathrm{im}(f[j]) \subseteq 0^\omega \cup 0^*1^\omega$ und (20)

$$f[j]^{(t)}(\boldsymbol{\xi}^{[t]}) = 0 \quad \forall t \geq 0 \tag{21}$$

Gilt umgekehrt(21), so folgt aus (20) die Gleichung $f[j+1]^{(t)}(\boldsymbol{\xi}^{[t]}) = 0$ für $t \geq 0$. Insgesamt folgt

$$f[j](\boldsymbol{\xi}) = 0^\omega \Leftrightarrow f[j+1](\boldsymbol{\xi}) = 0^\omega$$

und aus der Induktionshypothese folgt

$$f(\boldsymbol{\xi}) = 0^\omega \Leftrightarrow f[j+1](\boldsymbol{\xi}) = 0^\omega$$

□

Mit dem folgenden Lemma ist garantiert, daß der Fixpunkt $f[*]$ erweiterbar ist. Damit ist Bedingung 2. erfüllt und die Korrektheit der Iterationsvorschrift bewiesen.

Lemma 6.18. *Sei $f \in \mathrm{eON}_{m,1}$. Besitzt die Gleichung $f(\boldsymbol{\xi}) = 0^\omega$ eine Lösung, dann ist $f[*]$ erweiterbar.*

Beweis: Es muß folgende Eigenschaft für $f[*]$ mit

$$f[*] = \bigwedge_{(b_0,\ldots,b_{m-1})\in\mathbb{B}} f[*]_{\langle b_0,\ldots,b_{m-1}\rangle} \tag{22}$$

gezeigt werden

$$\forall t \in \mathbb{N}_0 \forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m ({}_2|f(\boldsymbol{\xi})| \leq 2^{-t} \Rightarrow \exists \boldsymbol{\upsilon} \in {}_2\mathbb{Z}^m ({}_2|\boldsymbol{\upsilon} - \boldsymbol{\xi}| \leq 2^{-t} \wedge f(\boldsymbol{\upsilon}) = 0^\omega)) \tag{23}$$

Nach Voraussetzung gibt es ein $\boldsymbol{\upsilon} \in {}_2\mathbb{Z}^m$ mit $f(\boldsymbol{\upsilon}) = 0^\omega$ und wegen Lemma 6.17 gilt $f[*](\boldsymbol{\upsilon}) = 0^\omega$. Daher ist (23) für $t = 0$ erfüllt. Sei nun $t > 0$ und $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ mit ${}_2|f[*](\boldsymbol{\xi})| \leq 2^{-t}$.
Wegen $f[*]^{(t-1)}(\boldsymbol{\xi}^{[t-1]}) = 0$ folgt aus (22) die Gleichung

$$\bigwedge_{(b_0,\ldots,b_{m-1})\in\mathbb{B}^m} f[*]^{(t)}(\xi_0^{[t-1]}, b_0, \ldots, \xi_{m-1}^{[t-1]}, b_{m-1}) = 0$$

Es gibt deshalb $b_0^{(t)}, \ldots, b_{m-1}^{(t)} \in \mathbb{B}$ mit $f[*]^{(t)}(\xi_0^{[t-1]}, b_0^{(t)}, \ldots, \xi_{m-1}^{[t-1]}, b_{m-1}^{(t)}) = 0$. Dieser Prozeß kann sukzessive fortgeführt werden, sodaß Lösungen $\upsilon_i^{[l]} = \langle x_i^{(0)}, \ldots, x_i^{(t-1)}, b_i^{(t)}, \ldots, b_i^{(l)} \rangle$ zu

$$f[*]^{(l)}(\boldsymbol{\upsilon}^{[l]}) = 0$$

für jedes $l \geq t$ konstruiert werden können. Es gibt also ein $\boldsymbol{\upsilon} \in {}_2\mathbb{Z}^m$ mit

$$f[*](\boldsymbol{\upsilon}) = 0^\omega$$

□

Mit dem Ergebnis aus dem vorherigen Abschnitt, daß aus einer erweiterbaren online Funktion eine reproduktive allgemeine parametrisierte online Funktion $\sigma \in \mathrm{eON}_{m,1}$ von f gewonnen werden kann, gilt folgender Satz

Satz 6.19. *Wenn $f \in \mathrm{eON}_{m,1}$ eine Lösung zu $f(\boldsymbol{\xi}) = 0^\omega$ besitzt, dann kann eine reproduktive allgemeine parametrisierte online Funktion $\sigma \in \mathrm{eON}_{m,1}$ zu f berechnet werden.*

Beweis: Da $f \in \mathrm{eON}_{m,1}$ eine Lösung zu $f(\boldsymbol{\xi}) = 0^\omega$ besitzt, kann nach Lemma 6.18 mit der Fixpunktberechnung 6.10 aus f eine erweiterbare online Funktion $f[*]$ berechnet werden. Da die Nullstellenmenge von $f[*]$ mit f übereinstimmt, wird mit $\sigma \in \mathrm{eON}_{m,1}$ aus Satz 6.9 dann eine reproduktive allgemeine parametrisierte online Funktion $\sigma \in \mathrm{eON}_{m,1}$ zu f berechnet. □

6.3 Berechnung der Nullstellen eines synchronen Schaltkreises

Sei $\mathcal{S}$ ein synchroner Schaltkreis mit m Eingaben und n Ausgaben und $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ die zugehörige online Funktion. Das Ziel ist es nun, alle Eingaben für $\mathcal{S}$ zu bestimmen, sodaß die berechnete Ausgabe von $\mathcal{S}$ die Nullfolge ergibt.

Wenn eine solche Eingabefolge existiert, dann kann ein synchroner Schaltkreis $\mathcal{S}_L$ berechnet werden, sodaß die zugehörige online Funktion eine reproduktive allgemeine parametrisierte Lösung von $f = 0^\omega$ ist.

Die Berechnung von $\mathcal{S}$ geschieht nach Satz 6.19 in zwei Phasen und einem Test.

1. Berechnen eines synchronen Schaltkreises $\mathcal{S}[*] = (\delta, \lambda, \boldsymbol{s})$ mit der zugehörigen online Funktion $f[*]$ aus Iteration 6.10.

2. Test, ob die online Funktion $f[*]$ von dem synchronen Schaltkreis $\mathcal{S}[*]$ erweiterbar ist.

   Wenn $f[*]^{(0)}(X^{(t)}) \neq 0$ gilt, dann existiert nach Konstruktion von $f[*]$ eine Eingabefolge $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ mit $f[*](\boldsymbol{\xi}) = 0^\omega$. Da $f[*]$ genau die gleiche Nullstellenmenge wie f besitzt, hat $f = 0^\omega$ also eine Lösung und damit ist nach Lemma 6.18 $f[*]$ erweiterbar. Ansonsten gibt es für $f = 0^\omega$ keine Lösung und die Berechnung kann abgebrochen werden.

   $f[*]$ ist deshalb genau dann erweiterbar, wenn der synchrone Schaltkreis in seinem Startzustand $\boldsymbol{s}$ zu einer Eingabe aus $\mathbb{B}^m$ eine 0 ausgeben kann. Dies ist genau dann der Fall, wenn die boolesche Funktion
   $$\forall_X \lambda(X, \boldsymbol{s}) \tag{24}$$
   die 0-Funktion ist.

   Gilt $\mathcal{S}[*] \neq 1^\omega$, so heißt $\mathcal{S}[*]$ im folgenden ein erweiterbarer synchroner Schaltkreis.

3. Konstruieren eines synchronen Schaltkreises $\mathcal{S}_L$, der eine reproduktive allgemeine parametrisierte Lösung von $f[*]$ berechnet. $\mathcal{S}_L$ wird deshalb als reproduktive allgemeine parametrisierte Lösung von $\mathcal{S}[*]$ bezeichnet.

6.3.1 Konstruktion eines erweiterbaren synchronen Schaltkreises

Im folgenden sei $\mathcal{S}$ ein synchroner Schaltkreis mit der online Funktion f, die eine Lösung $f = 0^\omega$ besitzt.

Aus f kann dann eine erweiterbare online Funktion $\tilde{f}$ konstruiert werden, die dieselbe Nullstellenmenge wie f besitzt. Ein synchroner Schaltkreis $\tilde{\mathcal{S}}$, der $\tilde{f}$ berechnet, wird im folgenden konstruiert.

$\tilde{f}$ kann dabei als der Fixpunkt $f[*]$ von der Iterationsvorschrift

$$\begin{aligned} f[0] &:= \mathrm{mem(f)} \\ f[j+1] &:= \bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} f[j]_{\langle b_0,\ldots,b_{m-1}\rangle} \end{aligned}$$

aus Definition 6.10 bestimmt werden. Wird diese Iterationsvorschrift entsprechend auf den synchronen Schaltkreis $\mathcal{S}$ angewendet, so wird ein synchroner Schaltkreis $\tilde{\mathcal{S}} = \mathcal{S}[*]$ zu der online Funktion $\tilde{f} = f[*]$ berechnet.

Ein synchroner Schaltkreis $\mathcal{S}[0]$ zu der online Funktion $f[0] = \mathrm{mem(f)}$ kann folgendermaßen konstruiert werden.

Konstruktion 6.20. Sei $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ $(l, m, 1)$. Seien $\boldsymbol{X}$ die Eingabevariablen, $\boldsymbol{Q}$ die Zustandsvariablen, $\boldsymbol{Q}'$ die Folgezustandsvariablen und H eine Ausgabevariable.

Sei Q_H eine zusätzliche Zustandsvariable und Q'_H die zugehörige Folgezustandsvariable. Der Automat $\mathcal{S}[0] = (\tilde{\delta}, \tilde{\lambda}, \tilde{s})$ vom Typ $(l+1, m, 1)$ besitzt die Eingabevariablen X, Ausgabevariable H, Zustandsvariablen $\tilde{\boldsymbol{Q}} = \langle Q_0, \ldots, Q_{l-1}, Q_H\rangle$ und Folgezustandsvariablen $\tilde{\boldsymbol{Q}}' = \langle Q'_0, \ldots, Q'_{l-1}, Q'_H\rangle$ und wird definiert durch

- die Übergangsfunktion
$$\tilde{\delta} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+1} & \rightarrow \quad \mathbb{B}^{l+1} \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto \quad \tilde{\boldsymbol{Q}}' \end{cases}$$
  mit
$$\begin{aligned} {Q'}_i &= \mathrm{ite}(Q_H, 1, \delta_i) \quad \forall i (0 \le i < l) \\ {Q'}_H &= \mathrm{ite}(Q_H, 1, \lambda) \end{aligned}$$
- die Ausgabefunktion
$$\tilde{\lambda} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+1} & \rightarrow \quad \mathbb{B} \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto \quad H \end{cases}$$
  mit $H := Q'_H$, d.h. die Ausgabefunktion entspricht der Zustandsfunktion von der Konstruktionsvariablen Q_H
- und dem Startzustand $\tilde{\boldsymbol{s}} = \boldsymbol{s}.(0)$.

Lemma 6.21. *Sei $\mathcal{S}$ ein synchroner Schaltkreis und f die zugehörige endliche online Funktion. Dann wird mit Konstruktion 6.20 zu der online Funktion* mem(f) *ein synchroner Schaltkreis $\mathcal{S}[0]$ erzeugt.*

Beweis: Solange $\mathcal{S}[0]$ keine 1 ausgibt, besitzt das Register zu der Variablen Q_H den Inhalt 0. Die Übergangsfunktionen $\tilde{\delta}_i$ für $(0 \le i < l)$ und die Ausgabefunktion $\tilde{\lambda}$ verhalten sich dann genauso wie die Übergangsfunktionen δ_i und Ausgabefunktion λ des synchronen Schaltkreises $\mathcal{S}$. Deshalb gibt $\mathcal{S}[0]$ genau dann das erste Mal eine 1 aus, wenn $\mathcal{S}$ das erste Mal eine 1 ausgibt.

Wenn nun die Ausgabe von $\mathcal{S}$ das erste Mal 1 ist, dann berechnet die Zustandsfunktion δ_l für das Register mit der Variable Q_H eine 1. Für alle weiteren Berechnungsschritte wird sich die boolesche Funktion der Variable Q_H wie die konstante 1-Funktion verhalten, d.h das Register mit der Variablen Q_H wird immer den Wert 1 besitzen. Da diese boolesche Funktion der Ausgabefunktion entspricht, gibt der synchrone Schaltkreis $\mathcal{S}$ immer eine 1 aus, sobald einmal eine 1 ausgegeben worden ist.

Insgesamt berechnet der synchrone Schaltkreis $\mathcal{S}[0]$ die online Funktion mem(f). □

Bemerkung 6.22. Die Übergangsfunktion $\tilde{\delta}$ verhält sich nur solange wie die Übergangsfunktion δ, bis zum Zeitpunkt t das erste Mal eine 1 ausgegeben wird. Ab dem Zeitpunkt $t+1$ geben alle $\tilde{\delta}_i$ $(0 \leq i < l)$ nur noch 1 aus, d.h. ab dem Zeitpunkt $t+2$ befindet sich der synchrone Schaltkreis $\tilde{\mathcal{S}}$ immer in dem Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$. Damit gelingt es in vielen Fällen, die Anzahl der erreichbaren Zustände stark einzuschränken.

Bemerkung 6.23. Die zusätzliche Zustandsvariable Q_H verändert die Größe der OBDDs von $\tilde{\lambda}$ und $\tilde{\delta}$ nur um jeweils einen Knoten, wenn die Variable Q_H als erstes in die Variablenordnung eingefügt wird. Wenn die booleschen Funktionen λ und δ nun mit einer passenden Variablenordnung sehr gut kodiert sind, können ihre Größen mit dieser erweiterten Variablenordnung erhalten werden.

Ist nun bereits ein synchroner Schaltkreis $\mathcal{S}[j]$ für die online Funktion $f[j]$ berechnet worden, so kann mit folgender Konstruktion aus $\mathcal{S}[j]$ ein synchroner Schaltkreis $\mathcal{S}[j+1]$ für die online Funktion $f[j+1]$ berechnet werden.

Konstruktion 6.24. Sei $\mathcal{S}[j] = (\delta, \lambda, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ $(l, m, 1)$. Seien $\boldsymbol{X}$ die Eingabevariablen, $\boldsymbol{Q}$ die Zustandsvariablen, $\boldsymbol{Q}'$ die Folgezustandsvariablen und H die Ausabevariable.

Der Automat $\mathcal{S}[j+1] = (\delta, \tilde{\lambda}, \boldsymbol{s})$ vom Typ $(l, m, 1)$ wird konstruiert mit der neuen Ausgabefunktion

$$\tilde{\lambda} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow & \mathbb{B} \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & H \end{cases}$$

mit

$$H = \left(\forall_X \lambda\right)\Big|_{Q_i = \delta_i (0 \leq i < l)}$$

Lemma 6.25. *Sei f eine endliche online Funktion und $f[j]$ die endliche online Funktion aus Iteration 6.10, die von dem synchronen Schaltkreis $\mathcal{S}[j] = (\delta, \lambda, \boldsymbol{s})$ vom Typ $(l, m, 1)$ berechnet wird. Dann wird mit Konstruktion 6.24 zu der online Funktion $f[j+1]$ ein synchroner Schaltkreis $\mathcal{S}[j+1] = (\delta, \tilde{\lambda}, \boldsymbol{s})$ vom Typ $(l, m, 1)$ erzeugt.*

Beweis: Mit

$$\lambda'(Q) = \forall_X \lambda(\boldsymbol{X}, \boldsymbol{Q})$$

erhält man einen synchronen Schaltkreis $\mathcal{S}[j]' = (\delta, \lambda', \boldsymbol{s})$, der die online Funktion

$$f[j]' = \bigwedge_{b_0, \ldots, b_{m-1} \in \mathbb{B}} f[j]^{(0)}(b_0, \ldots, b_{m-1}) + \sum_{t \geq 1} \left(\bigwedge_{b_0, \ldots, b_{m-1} \in \mathbb{B}} f[j]^{(t)}(\xi_0^{[t-1]}, b_0, \ldots, \xi_{m-1}^{[t-1]}, b_{m-1}) \right) 2^t$$

berechnet. Mit

$$\tilde{\lambda}(\boldsymbol{X}, \boldsymbol{Q}) = \lambda'(\boldsymbol{Q})|_{Q_i = \delta_i (0 \leq i < l)}$$

erhält man schließlich $\mathcal{S}[j+1] = (\delta, \tilde{\lambda}, \boldsymbol{s})$, der die online Funktion

$$\sum_{t\geq 0}\left(\bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} f[j]^{(t)}(\xi_0^{[t]}, b_0, \ldots, \xi_{m-1}^{[t]}, b_{m-1})\right)2^t =$$

$$\bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}}\left(\sum_{t\geq 0} f[j]^{(t)}(\xi_0^{[t]}, b_0, \ldots, \xi_{m-1}^{[t]}, b_{m-1})\right)2^t = \bigwedge_{b_0,\ldots,b_{m-1}\in\mathbb{B}} f[j]_{\langle b_0,\ldots,b_{m-1}\rangle} = f[j+1]$$

berechnet. □

Bemerkung 6.26. Da die Quantifizierung im ungünstigsten Fall die Zeitkomplexität $O(|\lambda|^{2^m})$ besitzt, wird mit dieser ein OBDD mit der maximalen Größe $|\lambda|^{2^m}$ berechnet. Die OBDD-Operation $f|_{X=g}$ hat deshalb nach Bemerkung 2.29 im ungünstigsten Fall eine Laufzeit von $O(|f|^2 \cdot |g|)$. Da m Substitutionen durchgeführt werden müssen, hat die Konstruktion 6.24 im ungünstigsten Fall die Laufzeitkomplexität

$$O((|\lambda|^{2^m})^{2^l}\prod_{i=0}^{l-1}|\delta_i|^{2^i})$$

mit der Substitutionsreihenfolge[6] $\delta_0, \delta_1, \ldots$.

Üblicherweise hat die Berechnung eine wesentlich günstigere Laufzeit.

- Mit Hilfe eines Caches (siehe 2.6.2) können viele Teilberechnungen eingespart werden.
- Die m Substitutionen können in einer Rekursion durchgeführt werden, sodaß gemeinsame Berechnungsschritte der m Substitutionen zusammengefaßt werden können.
- Mit der Iterationsvorschrift 6.10 wird ein synchroner Schaltkreis erzeugt, der eine erweiterbare online Funktion berechnet. Zu Beginn erhöht sich dabei oft die Knotenanzahl der OBDDs von $\tilde{\lambda}$ im Vergleich zu λ, jedoch nach wenigen Iterationsschritten sind sich die booleschen Funktionen sehr ähnlich und können durch ähnlich große OBDDs dargestellt werden.

Ist nun $\mathcal{S}$ ein synchroner Schaltkreis und f die zugehörige online Funktion, so kann mit diesen Konstruktionen ein synchroner Schaltkreis $\mathcal{S}[*]$ zu der online Funktion $f[*]$ berechnet werden.

Algorithmus 6.27 (Berechnung von $S[*]$).
// Input: Automat $\mathcal{S} = (\delta, \lambda, S)$ vom Typ $(l, m, 1)$ mit $\delta = (\delta_0, \ldots, \delta_{l-1})$
// Output: Automat $\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$
Automat mache_erweiterbar(δ, λ, S)
{
(1) Konstruiere nach 6.20 $\mathcal{S}[0] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$ mit $\tilde{\delta} = (\delta_0, \ldots, \delta_{l-1}, \delta_l)$
(2) **do** {
(3) `Ausgabefunktion_alt` $= \tilde{\lambda}$;
(4) $\tilde{\lambda}(\tilde{\boldsymbol{Q}}) = \forall_X \tilde{\lambda}$;
(5) $\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) = \tilde{\lambda}^-|_{Q_i=\delta_i(0\leq i\leq l)}$;
(6) } **while**(`Ausgabefunktion_alt`$^-! = \tilde{\lambda}$)
(7) **return**$(\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S}))$;

}

[6] $(\forall_V \lambda)|_{X_0=\delta_0}$ besitzt im ungünstigsten Fall mit $k = |\lambda|^{2^m}$ ein OBDD der Größe $k^2|\delta_0|$, $((\forall_V \lambda)|_{X_0=\delta_0})|_{X_1=\delta_1}$ ein OBDD der Größe $(|k|^2|\delta_0|)^2|\delta_1|$, u.s.w.

Lemma 6.28. *Nach endlich vielen Schritten terminiert Algorithmus 6.27.*

Beweis: Die Konstruktionen 6.20 und 6.24 gelingen unabhängig von dem Startzustand: Sind die Übergangsfunktion δ und Ausgabefunktion λ zu einem $(l, m, 1)$ Automaten gegeben, so berechnet dieser mit einem Startzustand $s \in \mathbb{B}^l$ eine online Funktion f_s. Im j-ten Iterationsschritt von Algorithmus 6.27 berechnet der Automat $\mathcal{S}[j] = (\tilde{\delta}, \tilde{\lambda}, \tilde{s})$ vom Typ $(l+1, m, 1)$ für einen Startzustand $s \in \mathbb{B}^{l+1}$ mit $\tilde{s} = s.(0)$ die online Funktion $f_s[j]$.

Nach Lemma 6.15 gibt es also ein $m_{\tilde{s}}$ mit $f_s[m_{\tilde{s}}] = f_s[m_{\tilde{s}} + j]$ für alle $j \geq 0$. Ist $R_{\tilde{s}}$ die Menge der erreichbaren Zustände von dem Automaten $\mathcal{S}[0]$ ausgehend von dem Startzustand $\tilde{s}$, so verändert sich im $(m_{\tilde{s}} + 1)$-ten Iterationsschritt die Ausgabefunktion im Bereich der erreichbaren Zustände $R_{\tilde{s}}$ nicht mehr. Es gilt demnach

$$\texttt{Ausgabefunktion\_alt} \wedge R_{\tilde{s}} = \tilde{\lambda} \wedge R_{\tilde{s}} \tag{25}$$

Für jeden Startzustand $\tilde{s} = s.(1)$ mit $s \in \mathbb{B}^l$ berechnet der Automat $(\tilde{\delta}, \tilde{\lambda}, \tilde{s})$ nach Bemerkung 6.29 nur noch die Ausgabe 1, d.h. die Ausgabefunktion ist von Anfang an die konstante 1-Funktion.

Es gibt also insgesamt für jeden Startzustand $\tilde{s} \in \mathbb{B}^{l+1}$ ein $m_{\tilde{s}}$, sodaß die Gleichung (25) erfüllt ist. Aus

$$\mathbb{B}^{l+1} = \bigcup_{\tilde{s} \in \mathbb{B}^{l+1}} R_{\tilde{s}}$$

folgt dann im Iterationsschritt $m + 1$ mit

$$m = \max\{m_{\tilde{s}} \mid \tilde{s} \in \mathbb{B}^{l+1}\}$$

die Gleichheit: $\texttt{Ausgabefunktion\_alt} = \tilde{\lambda}$. □

Bemerkung 6.29. Sei $\mathcal{S}[i] = (\delta, \lambda[i], \boldsymbol{s})$ der synchrone Schaltkreis vom Typ $(l+1, m, 1)$ in der i-ten Iteration von Algorithmus 6.27.

Startet der Automat $\mathcal{S}[i]$ in einem Zustand $Z = b.(1)$ mit $b \in \mathbb{B}^l$, so kann die Ausgabefunktion $\lambda[i]$ nur noch den Wert 1 berechnen:

Die Ausgabe von $\mathcal{S}[i]$ in dem Zustand $Z = b.(1)$ mit $b \in \mathbb{B}^l$ entspricht für jede Eingabe X dem Wert 1, d.h. es gilt

$$\forall b \in \mathbb{B}^m : \lambda[i](b, Z) = 1 \tag{26}$$

Für den speziellen Zustand $Z = (1, \ldots, 1) \in \mathbb{B}^{l+1}$ ist dies erfüllt: Es gilt für $\mathcal{S}[0]$ nach Konstruktion 6.20. Gilt dies nun für $\mathcal{S}[i-1]$ mit $i > 0$, so berechnet auch $\forall_X \lambda[i-1]$ im Zustand Z die Ausgabe 1. Da nach Bemerkung 6.22 der Zustand Z nicht mehr verlassen werden kann, berechnet auch $\lambda[i] = \left(\forall_X \lambda\right)\Big|_{Q_r = \delta_r (0 \leq r \leq l)}$ in diesem Zustand für jede Eingabe den Wert 1.

Nun folgt dies auch für einen Zustand $Z = b.(1) \neq (1, \ldots 1)$ mit $b \in \mathbb{B}^l$. Für $\mathcal{S}[0]$ gilt dies wieder nach Konstruktion. Gilt dieses Ausgabeverhalten nun für $\mathcal{S}[i-1]$ mit $i > 0$, so berechnet auch $\forall_X \lambda[i-1]$ im Zustand Z die Ausgabe 1. Da der Folgezustand von Z der Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ ist, wird von $\lambda[i] = \left(\forall_X \lambda\right)\Big|_{Q_r = \delta_r (0 \leq r \leq l)}$ ebenfalls für jede Eingabe eine 1 berechnet.

Da der Folgezustand von $Z = b.(1)$ mit $b \in \mathbb{B}^l$ der Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ ist, der nicht mehr verlassen werden kann, wird ausgehend von Z immer eine 1 ausgegeben.

Mit Hilfe folgender Überlegung können die erreichbaren Zustände ähnlich wie in Konstruktion 6.20 eingeschränkt werden.

Lemma 6.30. *Sei $\mathcal{S}[j] = (\delta, \lambda[i], \boldsymbol{s})$ der synchrone Schaltkreis vom Typ $(l+1, m, 1)$ in der j-ten Iteration von Algorithmus 6.27 und $\delta = (\delta_0, \ldots, \delta_l)$ die Übergangsfunktion von $\mathcal{S}[j]$.*

Dann kann die Zustandsfunktion δ_l von der Konstruktionsvariable Q_H in dem synchronen Schaltkreis $\mathcal{S}[j]$ mit einer berechneten Ausgabefunktion $\lambda[i]$ vom Iterationsschritt i mit $0 \leq i \leq j$ ausgetauscht werden, ohne das Ausgabeverhalten von $\mathcal{S}[j]$ zu verändern.

Beweis: Im folgenden ist $\tilde{\mathcal{S}}[i]$ der synchrone Schaltkreis mit der Übergangsfunktion

$$\tilde{\delta} = (\delta_0, \ldots, \delta_{l-1}, \lambda[i])$$

d.h. die Zustandsfunktion δ_l von der Konstruktionsvariable Q_H wird mit der berechneten Ausgabefunktion $\lambda[i]$ ausgetauscht.

Solange $\lambda[i]$ eine 0 ausgibt, besitzt das Register Q_H von $\tilde{\mathcal{S}}[i]$ und wegen (26) auch das Register Q_H von $\tilde{\mathcal{S}}[i]$ den Inhalt 0. Deshalb entspricht die Zustandsfunktion $\tilde{\delta}[i]$ der Zustandsfunktion $\delta[i]$, solange $\lambda[i]$ eine 0 ausgibt.

Wenn $\lambda[i]$ das erste Mal eine 1 berechnet, dann gerät der Automat $\tilde{\mathcal{S}}$ in einen Zustand $Z = b.(1)$ mit $b \in \mathbb{B}^l$. In diesem berechnet $\lambda[i]$ nach Bemerkung 6.29 die Ausgabe 1.

Damit verhalten sich alle Zustandsfunktionen δ_r mit $(0 \leq r < l)$ und λ wie die boolesche 1-Funktion. Der synchrone Schaltkreis $\tilde{\mathcal{S}}[i]$ gelangt demnach in den Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$, der wegen (6.29) für jede Eingabe immer eine 1 ausgibt. Da das Register Q_H nur den Inhalt 1 besitzen kann, wird dieser Zustand nicht mehr verlassen und $\tilde{\lambda}$ berechnet immer die Ausgabe 1.

Da nach Bemerkung 6.11 $\mathcal{S}[i]$ nur Ausgabefolgen aus $0^\omega \cup 0^* 1^\omega$ berechnen kann, wird auch von $\mathcal{S}[i]$ nur noch 1 ausgegeben. Insgesamt besitzen $\tilde{\mathcal{S}}[i]$ und $\mathcal{S}[i]$ dasselbe Ausgabeverhalten und berechnen daher die gleiche online Funktion $f[j]$.

Mit Konstruktion 6.24 kann in einem Berechnungsschritt i die Zustandsfunktion δ_l durch $\lambda[i]$ ersetzt werden und mit diesem Automaten dann iterativ $\tilde{\mathcal{S}}[j]$ berechnet werden, d.h. δ_l ist mit $\lambda[i]$ in $\mathcal{S}[j]$ ausgetauscht worden. □

Bemerkung 6.31. Wenn das OBDD $\tilde{\lambda}$ von $\mathcal{S}[j]$ im Algorithmus 6.27 kleiner ist als δ_l, so kann δ_l mit $\tilde{\lambda}$ ausgetauscht werden. Die Substitution in Konstruktion 6.24 kann dann effizienter ausgeführt werden.

Ist der Fixpunkt $\mathcal{S}[*]$ berechnet worden, so werden außerdem die erreichbaren Zustände weiter eingeschränkt, wenn die Zustandsfunktion δ_l mit $\tilde{\lambda}$ ausgetauscht wird. Denn sobald $\delta_l := \tilde{\lambda}$ im Berechnungsschritt t eine 1 berechnet, wird sich $\mathcal{S}[*]$ im Berechnungsschritt $t+2$ in dem Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ befinden, der in den folgenden Berechnungsschritten nicht mehr verlassen werden kann.

Insgesamt ergibt sich dann folgende iterative Berechnung von $\mathcal{S}[*]$.

Algorithmus 6.32 (Berechnung von $\mathcal{S}[*]$).
// Input: Automat $\mathcal{S} = (\delta, \lambda, S)$ vom Typ $(l, m, 1)$ mit $\delta = (\delta_0, \ldots, \delta_l)$
// Output: Automat $\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$
Automat mache_erweiterbar(δ, λ, S)
{
Konstruiere nach 6.20 $\mathcal{S}[0] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$ mit $\tilde{\delta} = (\delta_0, \ldots, \delta_{l-1}, \delta_l)$
do
{
`Ausgabefunktion_alt` $= \tilde{\lambda}$;
$\tilde{\lambda}(\tilde{\boldsymbol{Q}}) = \forall_X \tilde{\lambda}$;
$\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) = \tilde{\lambda}^- |_{Q_i = \delta_i (0 \leq i \leq l)}$;
if($|\tilde{\lambda}| < |\delta_l|$)
$\delta_l^- = \tilde{\lambda}^+$;
}
while(`Ausgabefunktion_alt`$^-$ $! = \tilde{\lambda}$)
$\delta_l^- = \tilde{\lambda}^+$;
return($\mathcal{S}[*] = (\tilde{\lambda}, \tilde{\delta}, \tilde{S})$);
}

Der Beweis der Terminierung entspricht dem Beweis von Lemma 6.28. Der Beweis verwendet dabei die Idee, daß die Ausgabefunktion $\tilde{\lambda}$ nach m Iterationen in dem Bereich der erreichbaren Zustände R_m von $\mathcal{S}[m]$ sich nicht mehr verändert, d.h. es gilt

$$\texttt{Ausgabefunktion\_alt}(\tilde{\boldsymbol{Q}}) \wedge R_m(\tilde{\boldsymbol{Q}}) = \tilde{\lambda}(\tilde{\boldsymbol{Q}}) \wedge R_m(\tilde{\boldsymbol{Q}}) \tag{27}$$

Da im Algorithmus 6.27 die Übergangsfunktionen der $\mathcal{S}[j]$ $(j \geq 0)$ gleich bleiben, stimmen auch die erreichbaren Zustände für jeden Automaten $\mathcal{S}[j]$ überein. Es genügt für die Terminierungsabfrage (27) deshalb, nur die erreichbaren Zustände R von dem Automaten $\mathcal{S}[0]$ zu berechnen.

Wenn die erreichbaren Zustände R von $\mathcal{S}[0]$ errechnet sind, kann mit der Terminierungsabfrage (27) die Iteration im Vergleich zur Abfrage `Ausgabefunktion_alt` $= \tilde{\lambda}$ oftmals erheblich verkürzt werden (siehe Beweis von Lemma 6.28). Da die erreichbaren Zustände von $\mathcal{S}[0]$ nach Bemerkung 6.22 oft stark eingeschränkt werden, gelingt die Berechnung von R in vielen Fällen. In diesem Fall bietet sich dann die Terminierungsabfrage (27) an. Diese kann für OBDDs effizient durchgeführt werden mit

$$\text{INTERSECTS}(\texttt{Ausgabefunktion\_alt}(\tilde{\boldsymbol{Q}}) \oplus \tilde{\lambda}(\tilde{\boldsymbol{Q}}), R(\tilde{\boldsymbol{Q}}))$$

Die boolesche Funktion (`Ausgabefunktion_alt` $\oplus\, \tilde{\lambda}$) ist in dem Bereich R genau dann erfüllbar, wenn `Ausgabefunktion_alt` und $\tilde{\lambda}$ in dem Bereich R nicht übereinstimmen. Dieser Test kann mit der OBDD-Operation INTERSECTS effizient durchgeführt werden. Die Operation durchläuft dabei rekursiv die OBDDs `Ausgabefunktion_alt` und $\tilde{\lambda}$ bis eine Lösung gefunden wird. Bei dieser Suche werden insbesondere keine weiteren Speicherressourcen verwendet.

Außerdem kann die Ausgabefunktion $\tilde{\lambda}$, dargestellt durch ein OBDD, in dem Bereich der nicht erreichbaren Zustände reduziert werden, denn es gilt folgendes Lemma.

Lemma 6.33. *Sei $\mathcal{S}[0]$ der synchrone Schaltkreis vom Typ $(l+1, m, 1)$ aus Algorithmus 6.27 und $\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}})$ die Ausgabefunktion im Iterationsschritt j. Sei $R(\tilde{\boldsymbol{Q}})$ die Menge der erreichbaren Zustände von $\mathcal{S}[0]$.*

Gilt für eine boolesche Funktion μ die Beziehung $\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}}) = \mu(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}})$ dann folgt

$$\left(\forall_X \tilde{\lambda}\right)\Big|_{Q_i = \delta_i (0 \leq i \leq l)} (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}}) = \left(\forall_X \mu\right)\Big|_{Q_i = \delta_i (0 \leq i \leq l)} (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}})$$

Beweis: Sei im folgenden $f_1(\tilde{\boldsymbol{Q}}) := \forall_X \tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}})$ und $f_2(\tilde{\boldsymbol{Q}}) := \forall_X \mu(\boldsymbol{X}, \tilde{\boldsymbol{Q}})$. Aus $\tilde{\lambda} \wedge R = \mu \wedge R$ folgt

$$\begin{aligned} f_1 \wedge R = (\forall_X \tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}})) \wedge R(\tilde{\boldsymbol{Q}}) &= \forall_X (\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}})) = \\ &= \forall_X (\mu(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}})) = (\forall_X \mu(\boldsymbol{X}, \tilde{\boldsymbol{Q}})) \wedge R(\tilde{\boldsymbol{Q}})) = f_2 \wedge R \end{aligned}$$

Wenn ein Zustand $q \in \mathbb{B}^l$ in R liegt, so befindet sich auch ein Folgezustand $\delta(x, q)$ mit $x \in \mathbb{B}^m$ in R. Deswegen folgt für jedes $x \in \mathbb{B}^m$ und $q \in R$

$$f_1|_{Q_i=\delta_i}(x, q) = f_1(\delta(x, q)) = f_2(\delta(x, q)) = f_1|_{Q_i=\delta_i}(x, q)$$

und daraus folgt $f_1|_{Q_i=\delta_i}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}}) = f_2|_{Q_i=\delta_i}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \wedge R(\tilde{\boldsymbol{Q}})$. □

Mit Hilfe der REDUCE-Operation 5.2 darf dann mit Lemma 5.3 das OBDD $\tilde{\lambda}$ mit

$$\tilde{\lambda}_{red} = \text{REDUCE}(\tilde{\lambda}, R)$$

reduziert werden, wobei im Zustandsbereich R sich $\tilde{\lambda}_{red}$ wie $\tilde{\lambda}$ verhält.

Insgesamt ergibt sich nun folgender Algorithmus, wobei sich die $\tilde{\lambda}$ in dem Bereich der erreichbaren Zustände R im Iterationsschritt j wie die $\tilde{\lambda}$ im Algorithmus 6.27 verhalten.

Algorithmus 6.34 (Berechnung von $S[*]$).
// Input: Automat $\mathcal{S} = (\delta, \lambda, S)$ vom Typ $(l, m, 1)$ mit $\delta = (\delta_0, \ldots, \delta_{l-1})$
// Output: Automat $\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$
Automat mache_erweiterbar(δ, λ, S)
{
Konstruiere nach 6.20 $\mathcal{S}[0] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$ mit $\tilde{\delta} = (\delta_0, \ldots, \delta_{l-1}, \delta_l)$
Berechne die erreichbaren Zustände R von $\mathcal{S}[0]$.
do
{
$\tilde{\lambda}_{red} = \text{REDUCE}(\tilde{\lambda}, R)$;
if$(|\tilde{\lambda}_{red}| < |\tilde{\lambda})|$
$\tilde{\lambda}^- = \tilde{\lambda}_{red}$;
else
FREE$(\tilde{\lambda}_{red})$;
`Ausgabefunktion_alt` $= \tilde{\lambda}$;
$\tilde{\lambda}(\tilde{\boldsymbol{Q}}) = \forall_X \tilde{\lambda}$;
$\tilde{\lambda}(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) = \tilde{\lambda}^-|_{Q_i=\delta_i(0 \le i < l)}$;
}
while(INTERSECTS(`Ausgabefunktion_alt` $\oplus \tilde{\lambda}, R$))
return($\mathcal{S}[*] = (\tilde{\delta}$, `Ausgabefunktion_alt`, $\tilde{S})$);
}

Bemerkung 6.35. Wenn die von den Algorithmen 6.27 und 6.32 konstruierten synchronen Schaltkreise $\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{S})$ vom Typ $(l+1, m, 1)$ in den Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ gelangen, so wird nach Bemerkung 6.29 von λ für jede weitere Eingabefolge nur noch die 1 ausgegeben. Wenn der synchrone Schaltkreis $\mathcal{S}[*]$ eine 1 ausgeben kann, so auch $\mathcal{S}[0]$. Damit befindet sich der Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ in den erreichbaren Zuständen R von $\mathcal{S}[0]$. Deshalb besitzt $\mathcal{S}[*]$ von Algorithmus 6.34 in dem Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ dasselbe Ausgabeverhalten wie $\mathcal{S}[*]$ von Algorithmus 6.27 und berechnet ebenfalls nur noch 1.

Gibt ein $\mathcal{S}[*]$ von einem der Algorithmen 6.27, 6.32 oder 6.34 im Berechnungsschritt t nun eine 1 aus, so berechnet dieser für jede weitere Eingabefolge immer die Ausgabe 1. Deshalb darf der synchrone Schaltkreis nach der Ausgabe einer 1 sofort in den Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ übergehen.

Sind $\tilde{\delta}_i$ für $(0 \leq i \leq l)$ die Übergangsfunktionen von $\mathcal{S}[*]$, so gelingt der Zustandsübergang im Schritt $t + 1$ in den Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ mit der Umformung

$$\delta'_i = \tilde{\delta}_i \vee \tilde{\lambda}$$

Solange $\tilde{\lambda}$ eine 0 berechnet, verhalten sich die Übergangsfunktionen δ'_i wie $\tilde{\delta}$. Sobald aber im Berechnungsschritt t von $\tilde{\lambda}$ eine 1 ausgegeben wird, befindet sich der zugehörige Automat im Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$. Da in diesem nur die Ausgabe 1 berechnet werden kann, wird das Ausgabeverhalten von $\mathcal{S}[*]$ nicht verändert.

Während nach Bemerkung 6.31 mit Algorithmus 6.32 ein synchroner Schaltkreis $\mathcal{S}[*]$ berechnet wird, der bei einer Ausgabe 1 sich erst ab dem Berechnungsschritt $t+2$ in dem Zustand $(1, \ldots, 1) \in \mathbb{B}^{l+1}$ befindet, gelingt der Übergang mit dieser Umformung bereits im Berechnungsschritt $t + 1$.

Die erreichbaren Zustände können nun noch weiter eingeschränkt werden, da Zustände $Z = b.(1)$ mit $b \in \mathbb{B}^l$ vermieden werden. Diese Modifikation besitzt jedoch den Nachteil, daß die Übergangsfunktionen sehr viel größer werden können. Überwiegt nun der Aspekt, einen synchronen Schaltkreis $\mathcal{S}[*]$ mit möglichst wenigen Zuständen zu erhalten, so kann mit dieser Modifikation der Übergangsfunktionen die Anzahl der erreichbaren Zustände weiter reduziert werden.

6.3.2 Konstruktion einer reproduktiven allgemeinen parametrisierten Lösung aus einem erweiterbaren synchronen Schaltkreis

Sei $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ ein erweiterbarer synchroner Schaltkreis vom Typ $(l, m, 1)$ mit den Eingabevariablen $\boldsymbol{X} = \langle X_0, \ldots, X_{m-1}\rangle$, Zustandsvariablen $\boldsymbol{X} = \langle Q_0, \ldots, Q_{l-1}\rangle$ und der zugehörigen erweiterbaren online Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$.

Aus Satz 6.9 und seinem Beweis erhält man eine Beschreibung, wie eine reproduktive allgemeine parametrisierte Lösung $\mathcal{S}_L$ aus dem erweiterbaren synchronen Schaltkreis $\mathcal{S}$ konstruiert werden kann.

Im ersten Berechnungsschritt muß $\mathcal{S}_L$ das Ausgabeverhalten der Funktionen $(\tau_0^{(0)}, \ldots, \tau_{m-1}^{(0)})$ mit

$$\begin{aligned}\tau_i^{(0)}(X_0, \ldots, X_i) = \mathrm{ite}(X_i, \neg\ & f^{(0)}_{\{m-1-i\}}(\tau_0^{(0)}(X_0), \ldots, \tau_{i-1}^{(0)}(X_0, \ldots, X_{i-1}, 1),\\ & f_{\{m-1-i\}}(\tau_0^{(0)}(X_0), \ldots, \tau_{i-1}^{(0)}(X_0, \ldots, X_{i-1}, 0))\end{aligned}$$

und in den weiteren Berechnungsschritten $t > 0$ das Ausgabeverhalten

$$\begin{aligned}&\tau_i^{(t)}(\zeta_0^{[t-1]}, X_0, \ldots, \zeta_i^{[t-1]}, X_i, \zeta_i^{[t-1]}, \ldots, \zeta_{m-1}^{[t-1]})\\ &\qquad = \mathrm{ite}(X_i, \neg\ f^{(t)}_{\{m-1-i\}}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, \tau_i^{[t-1]}, 1, \tau_{i+1}^{[t-1]}, \ldots, \tau_{m-1}^{[t-1]}),\\ &\qquad\qquad f^{(t)}_{\{m-1-i\}}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, \tau_i^{[t-1]}, 0, \tau_{i+1}^{[t-1]}, \ldots, \tau_{m-1}^{[t-1]}))\end{aligned}$$

besitzen. Der Aufbau des synchronen Schaltkreises $\mathcal{S}_L = (\tilde{\delta}, \tilde{\lambda}, \tilde{\boldsymbol{s}})$ kann deshalb aus zwei Phasen zusammengesetzt werden.

1. Die gesuchte Ausgabefunktion $\tilde{\lambda}$ mit Eingabe X muß sich im Berechnungsschritt $t = 0$ wie

$$\tau^{(0)} = (\tau_0^{(0)}, \ldots, \tau_{m-1}^{(0)})$$

verhalten, wobei die $\tau_i^{(0)}$ nur von den Variablen X abhängen. Die $\tau_i^{(0)}$ können nun von booleschen Funktionen $\tilde{\lambda}_i^{(0)}$ berechnet werden, die von den Zustandsvariablen $\boldsymbol{Q}$ von $\mathcal{S}$ und den Eingabevariablen $\boldsymbol{X}$ abhängen

Befinden sich die Register von $\boldsymbol{Q}$ in dem Startzustand $\boldsymbol{s}$, so wird mit $\lambda(\boldsymbol{X}, \boldsymbol{Q})$ die boolesche Funktion $f^{(0)}$ berechnet. Die $f^{(0)}_{\{m-1-i\}}$ können dann durch die μ_i für $(0 \leq i < m)$ mit

$$\mu_i(\boldsymbol{X}, \boldsymbol{Q}) = \forall_{\{X_i, \ldots, X_{m-1}\}}\, \lambda$$

umgesetzt[7] werden. Mit diesen erhält man dann $\tilde{\lambda}^{(0)} = (\tilde{\lambda}_0^{(0)}, \ldots, \tilde{\lambda}_{m-1}^{(0)})$, das $\tau^{(0)}$ realisiert. $\tilde{\lambda}_0^{(0)}$ gewinnt man aus

$$\tilde{\lambda}_0^{(0)}(\boldsymbol{X}, \boldsymbol{Q}) = \mathrm{ite}(X_0, \neg\mu_0, \mu_0)$$

Sind $\tilde{\lambda}_0^{(0)}, \ldots, \tilde{\lambda}_{i-1}^{(0)}$ bereits berechnet, so kann

$$\tilde{\lambda}_i^{(0)}(\boldsymbol{X}, \boldsymbol{Q}) = \mathrm{ite}(X_i, \neg\pi_i, \pi_i)$$

mit

$$\pi_i = \mu_i|_{X_j = \tilde{\lambda}_j^{(0)}\,(0 \leq j < i)}$$

berechnet[8] werden.

[7]Zuerst wird μ_{m-1} berechnet, indem ν mit der Variable X_{m-1} quantifiziert wird, anschließend wird μ_{m-2} berechnet, indem μ_{m-1} mit X_{m-2} quantifiziert wird, anschließend ...

[8]Die Substitution der τ_j in X_j können in einer OBDD-Operation in vielen Fällen effizient berechnet werden.

In den weiteren Berechnungsschritten wird der synchrone Schaltkreis $\mathcal{S}$ simuliert mit der Eingabe, die von $\mathcal{S}_L$ im vorherigen Berechnungsschritt ausgegeben worden ist. Aus diesem Grund werden für den Berechnungsschritt $t = 1$ die Zustandsregister von den Variablen $\boldsymbol{Q}$ mit dem Startzustand $\boldsymbol{s} = (s_0, \ldots, s_{l-1})$ von $\mathcal{S}$ initialisiert, d.h. die Zustandsfunktionen $\tilde{\delta}_i$ für Q_i verhalten sich im Berechnungsschritt $t = 0$ wie die konstante s_i-Funktion. Außerdem werden die Ausgaben von $\mathcal{S}_L$ in Registern von den Zustandsvariablen $\boldsymbol{Q}_x = \langle Q_{x_0}, \ldots, Q_{x_{m-1}} \rangle$ zwischengespeichert, d.h die booleschen Funktionen $\tilde{\lambda}_i^{(0)}$ berechnen neben der Ausgabe auch den Folgezustand für die Register von den Variablen $\boldsymbol{Q}_x$.

2. Im folgenden wird $\mathcal{S}_L$ in den weiteren Berechnungsschritten $t > 0$ beschrieben.

   Die Zustandsfunktionen $\tilde{\delta}_i$ berechnen für die Register der Zustandsvariablen Q_i den Folgezustand, den $\mathcal{S}$ besitzt, wenn dieser die Ausgabe von $\mathcal{S}_L$ aus dem letzten Berechnungsschritt als Eingabe erhält. Da die Ausgabe in den Registern mit den Variablen $\boldsymbol{Q}_x$ zwischengelagert ist, kann $\tilde{\delta}_i$ umgesetzt werden durch

$$\tilde{\delta}_i^{(1)}(\boldsymbol{Q}, \boldsymbol{Q}_x) = \delta_i|_{X_k = Q_{x_k} (0 \leq k < m)}$$

   Die gesuchte Ausgabefunktion $\tilde{\lambda}$ muß sich nun wie die boolesche Funktion $(\tau_0^{(t)}, \ldots, \tau_{m-1}^{(t)})$ verhalten. Diese $\tau_i^{(t)}$ können nun folgendermaßen mit booleschen Funktionen $\tilde{\lambda}_i^{(1)}$ berechnet werden, die von den Zustandsvariablen $\boldsymbol{Q}$, $\boldsymbol{Q}_x$ und den Eingabevariablen $\boldsymbol{X}$ abhängen.

   Um $\tilde{\lambda}_i^{(1)}$ zu berechnen, werden die

$$f_{\{m-1-i\}}^{(t)}(\tau_0^{[t-1]}, X_0, \ldots, \tau_{i-1}^{[t-1]}, X_{i-1}, \tau_i^{[t-1]} \ldots, \tau_{m-1}^{[t-1]}) \tag{28}$$

   für $(0 \leq i < m)$ mit den booleschen Variablen $\boldsymbol{X}$ für den t-ten Berechnungsschritt benötigt.

   Neben den Zustandsvariablen von $\mathcal{S}$ für den $(t-1)$-ten Berechnungsschritt, die in den Registern mit den Variablen $\boldsymbol{Q}$ vorliegen, sind für die Berechnung von (28) die aktuellen Eingaben $\boldsymbol{X}$ und die Ergebnisse $(\tau_0^{(t-1)}, \ldots, \tau_{m-1}^{(t-1)})$ aus dem vorherigen Rechenschritt $t-1$ erforderlich. Diese sind in den Registern mit den Variablen $\boldsymbol{Q}_x$ gespeichert.

   Die boolesche Funktion (28) wird nun durch die $\mu_i(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}_x)$ mit $(0 \leq i < m)$ berechnet, die ähnlich wie in Phase 1 gewonnen werden können.

   Da die Zustandsfunktionen $\tilde{\delta}_i^{(1)}$ den Folgezustand berechnen, in dem sich $\mathcal{S}$ mit der Eingabe aus dem letzten Berechnungsschritt von $\mathcal{S}_L$ befindet, gewinnt man mit

$$\nu(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}_x) = \lambda|_{Q_k = \tilde{\delta}_k^{(1)} (0 \leq k < m)}$$

   deshalb eine boolesche Funktion, die

$$f^{(t)}(\tau_0^{[t-1]}, X_0, \ldots, \tau_{i-1}^{[t-1]}, X_{i-1}, \tau^{[t-1]}, X_i \ldots, \tau_{m-1}^{[t-1]}, X_{m-1})$$

   umsetzt. Mit dieser können nun mit

$$\mu_i(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}_x) = \forall_{\{X_i, \ldots, X_{m-1}\}} \nu$$

   die μ_i erzeugt werden.

   Aus den μ_i kann dann $\tilde{\lambda}^{(1)} = (\tilde{\lambda}_0^{(1)}, \ldots, \tilde{\lambda}_{m-1}^{(1)})$ erstellt werden. $\tilde{\lambda}_0^{(1)}$ gewinnt man aus

$$\tilde{\lambda}_0^{(1)}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}_x) = \mathrm{ite}(X_0, \neg\mu_0, \mu_0)$$

   Sind $\tilde{\lambda}_0^{(1)}, \ldots, \tilde{\lambda}_{i-1}^{(1)}$ bereits berechnet, so kann

$$\tilde{\lambda}_i^{(1)}(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}_x) = \mathrm{ite}(X_i, \neg\pi_i, \pi_i)$$

mit

$$\pi_i = \mu_i|_{X_j = \tilde{\lambda}_j^{(1)} (0 \le j < i)}$$

berechnet werden.

Diese $\tilde{\lambda}_i^{(1)}$ berechnen neben der Ausgabefunktion zugleich den Folgezustand für die Register mit den Variablen $\boldsymbol{Q}_x$, d.h. das Ergebnis von $\mathcal{S}_L$, das $(\tau_0^{(t)}, \ldots, \tau_{m-1}^{(t)})$ entspricht, wird in diesen Registern für den nächsten Berechnungsschritt zwischengespeichert.

Insgesamt ergibt sich folgende Konstruktion. Für diese wird ein weiteres Register mit der Variable Q_{Init} eingeführt, mit der die zwei Berechnungsphasen unterschieden werden.

Konstruktion 6.36. Sei $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ $(l, m, 1)$ mit dem Startzustand $s = (s_0, \ldots, s_{l-1})$. Seien $\boldsymbol{X}$ die Eingabevariablen, $\boldsymbol{Q}$ die Zustandsvariablen und $\boldsymbol{Q}'$ die Folgezustandsvariablen.

Sei Q_{Init} eine zusätzliche Zustandsvariable und Q'_{Init} die zugehörige Folgezustandsvariable, um die zwei Berechnungsphasen zu unterscheiden.

Seien $Q_{x_0}, \ldots, Q_{x_{m-1}}$ weitere Zustandsvariablen und $Q'_{x_0}, \ldots, Q'_{x_{m-1}}$ die Folgevariablen, um die Ergebnisse aus dem letzten Berechnungsschritt im aktuellen Schritt zur Verfügung zu haben.

Der Automat $\mathcal{S}[*] = (\tilde{\delta}, \tilde{\lambda}, \tilde{s})$ vom Typ $(l+m+1, m, m)$ besitzt die Eingabevariablen $\boldsymbol{X}$, Ausgabevariablen $\boldsymbol{H} = \langle H_0, \ldots, H_{m-1} \rangle$, Zustandsvariablen $\tilde{\boldsymbol{Q}} = \langle Q_0, \ldots, Q_{l-1}, Q_{x_0}, \ldots, Q_{x_{m-1}, Q_{\text{Init}}} \rangle$ und Folgezustandsvariablen $\tilde{\boldsymbol{Q}}' = \langle Q'_0, \ldots, Q'_{l-1}, Q'_{x_0}, \ldots, Q'_{x_{m-1}}, Q'_{\text{Init}} \rangle$ und wird definiert durch

- die Ausgabefunktion

$$\tilde{\lambda} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+m+1} & \to & \mathbb{B}^{l+m+1} \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto & \boldsymbol{H} \end{cases}$$

mit

$$H_i = \text{ite}(Q_{\text{Init}}, \tilde{\lambda}_i^{(1)}, \tilde{\lambda}_i^{(0)}) \quad \forall i (0 \le i < m)$$

- die Übergangsfunktion

$$\tilde{\delta} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+m+1} & \to & \mathbb{B}^{l+m+1} \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto & \tilde{\boldsymbol{Q}}' \end{cases}$$

mit

$$\begin{aligned} Q_{\text{Init}} &= 1 \\ {Q'}_i &= \text{ite}(Q_{\text{Init}}, \tilde{\delta}_i^{(1)}, s_i) \quad \forall i (0 \le i < l) \\ Q'_{x_i} &= H_i \quad \forall i (0 \le i < m) \end{aligned}$$

d.h. Zustandsfunktionen von den Konstruktionsvariablen Q_{x_i} entsprechen den Ausgabefunktionen

- und dem Startzustand $\boldsymbol{s} = (s_0, \ldots, s_{l-1}, \underbrace{0}_{\text{Phase 1}}, \underbrace{0, \ldots, 0}_{\text{beliebig}})$.

Aus der obigen Beschreibung von $\mathcal{S}_L$ folgt:

Lemma 6.37. *Sei $\mathcal{S}$ ein erweiterbarer synchroner Schaltkreis. Dann wird mit Konstruktion 6.36 eine reproduktive allgemeine parametrisierte Lösung $\mathcal{S}[*]$ von $\mathcal{S}$ erzeugt.*

Bemerkung 6.38. Die Zustandsvariablen besitzen untereinander folgende Ordnung

$$Q_{\text{Init}} < Q_{x_0} < \ldots < Q_{x_{m-1}} < Q$$

wobei die Ordnung der Variablen in Q beibehalten wird. Mit dieser Strategie werden die OBDDs der Zustandsfunktionen nur um einen konstanten Faktor bzgl. der Knotenanzahl erweitert.

6.4 Lösung von parameterbehafteten online Gleichungen

Im folgenden sind alle online Funktionen endlich, insbesondere

$$f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$$

Gesucht ist zu einem Parameter $\boldsymbol{\pi} \in {}_2\mathbb{Z}^p$ die Menge aller Lösungen $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ zu der parameterbehafteten Gleichung

$$f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega$$

d.h. die Lösungsmenge

$$L_{\boldsymbol{\pi}} = \{\boldsymbol{\xi} \in {}_2\mathbb{Z}^m \mid f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega\}$$

Mit Hilfe einer reproduktiven allgemeinen parametrisierten Lösung $\sigma : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m$ kann jedoch nur die Lösungsmenge für alle Parameter berechnet werden

$$L = \bigcup_{\boldsymbol{\pi} \in {}_2\mathbb{Z}^p} L_{\boldsymbol{\pi}}$$

Will man gezielt zu einem Parameter $\boldsymbol{\pi}$ die Lösungsmenge $L_{\boldsymbol{\pi}}$ berechnen, so muß eine allgemeine parametrisierte Lösung

$$\sigma : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$$

gesucht werden, sodaß gilt

$$L_{\boldsymbol{\pi}} = \{\sigma(\boldsymbol{\pi}, \boldsymbol{\zeta}) \mid \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m\}$$

Dieses σ ist in der Regel aber nicht mehr online, ja sogar nicht einmal offline. Besitzt die endliche online Funktion f jedoch gewisse Eigenschaften, die noch näher bestimmt werden, so kann eine parameterbehaftete reproduktive allgemeine parametrisierte online Funktion berechnet werden, die einen eingeschränkten Teil der Nullstellen beschreibt.

Definition 6.39. Sei $f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine endliche online Funktion. Eine online Funktion $\sigma : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ heißt *parameterbehaftete parametrisierte Lösung* von der Gleichung

$$f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega$$

(kurz: parameterbehaftete parametrisierte Lösung von f), wenn gilt

$$\forall \boldsymbol{\pi} \in {}_2\mathbb{Z}^p \forall \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m (f(\boldsymbol{\pi}, \sigma(\boldsymbol{\pi}, \boldsymbol{\zeta})) = 0^\omega)$$

Eine parameterbehaftete parametrisierte Lösung σ von f heißt *allgemein*, wenn zusätzlich gilt

$$\forall \boldsymbol{\pi} \in {}_2\mathbb{Z}^p \forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m (f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega \Rightarrow \exists \boldsymbol{\zeta} \in {}_2\mathbb{Z}^m (\boldsymbol{\xi} = \sigma(\boldsymbol{\pi}, \boldsymbol{\zeta})))$$

Eine parameterbehaftete parametrisierte allgemeine Lösung σ von f heißt *reproduktiv*, wenn gilt

$$\forall \boldsymbol{\pi} \in {}_2\mathbb{Z}^p \forall \boldsymbol{\xi} \in {}_2\mathbb{Z}^m (f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega \Rightarrow \sigma(\boldsymbol{\pi}, \boldsymbol{\xi}) = \boldsymbol{\xi})$$

Bemerkung 6.40. Eine sehr einschränkende Bedingung, die an eine parameterbehaftete Lösung gestellt wird, ist dabei, daß zu jedem Parameter $\boldsymbol{\pi} \in {}_2\mathbb{Z}^p$ eine Eingabe $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ existieren muß mit

$$f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega$$

Bemerkung 6.41. Im folgenden wird herausgearbeitet, welche Eigenschaften für f notwendig sind, damit eine parameterbehaftete parametrisierte Lösung σ für f existiert.

In jedem Berechnungsschritt $t > 0$ ist der Lösung σ nur der Anfang $\boldsymbol{\pi}^{[t-1]}$ des Parameters $\boldsymbol{\pi} \in {}_2\mathbb{Z}^p$ bekannt. Aus dieser Information muß σ in der Lage sein, eine bereits berechnete Anfangslösung $\boldsymbol{\lambda}^{[t-1]} \in \mathbb{B}^{tm}$ zu $\boldsymbol{\lambda}^{[t]} \in \mathbb{B}^{(t+1)m}$ fortzusetzen. Für alle möglichen Parameter mit dem Anfang $\boldsymbol{\pi}^{[t-1]}$ muß deshalb f eine Lösung besitzen, die aus dem Anfang $\boldsymbol{\lambda}^{[t-1]} \in \mathbb{B}^{(t+1)m}$ fortgesetzt werden kann.

Dieser Gedanke wird nun noch genauer gefaßt. Sei $\sigma = \sum_{t\geq 0} \sigma^{(t)} 2^t$ mit

$$\sigma^{(t)} : \begin{cases} \mathbb{B}^{p(t+1)} \times \mathbb{B}^{m(t+1)} & \rightarrow \quad \mathbb{B}^m \\ (\boldsymbol{\pi}^{[t]}, \boldsymbol{\xi}^{[t]}) & \mapsto \quad (\tau_0^{(t)}, \ldots, \tau_{m-1}^{(t)}) \end{cases}$$

eine endliche online Funktion. Folgende Bedingungen erfüllt

$$f(\boldsymbol{\pi}, \boldsymbol{\xi}) = \sum_{t\geq 0} f^{(t)}(\boldsymbol{\pi}^{[t]}, \boldsymbol{\xi}^{[t]}) 2^t$$

wenn σ eine parameterbehaftete parametrisierte Lösung zu f ist.

1. Im ersten Berechnungsschritt $t = 0$ muß für alle Parameteranfänge $(b_0, \ldots, b_{p-1}) \in \mathbb{B}^p$ ein Lösungsanfang $(b_p, \ldots, b_{p+m-1}) \in \mathbb{B}^m$ existieren mit

   $$f^{(0)}(b_0, \ldots, b_{p+m-1}) = 0$$

   Dies kann auch formuliert werden durch

   $$\exists_{\{y_0,\ldots,y_{p-1}\}} \forall_{\{y_p,\ldots,y_{p+m-1}\}} f^{(0)}(y_0, \ldots, y_{p+m-1}) = 0 \tag{29}$$

2. Sei im $t-1$-ten Berechnungsschritt der Lösungsanfang $\boldsymbol{\lambda}^{[t-1]}$ zu dem Parameteranfang $\pi^{[t-1]} = (\pi_0^{[t-1]}, \ldots, \pi_{p-1}^{[t-1]})$ und Eingabenanfang $(\xi_0^{[t-1]}, \ldots, \xi_{m-1}^{[t-1]})$ berechnet worden, d.h. es gilt

   $$\boldsymbol{\lambda}^{[t-1]} = (\tau_0^{[t-1]}(\boldsymbol{\pi}^{[t-1]}, \boldsymbol{\xi}^{[t-1]}), \ldots, \tau_{m-1}^{[t-1]}(\boldsymbol{\pi}^{[t-1]}, \boldsymbol{\xi}^{[t-1]}))$$

   Dann muß zu jedem Parameteranfang

   $$\boldsymbol{\pi}^{[t]} = (\pi_0^{[t]}, b_0, \ldots, \pi_{p-1}^{[t]}, b_p)$$

   mit $b_0, \ldots, b_{p-1} \in \mathbb{B}$ für den Berechnungsschritt t von σ ein Lösungsanfang

   $$\boldsymbol{\lambda}^{[t]} = (\lambda_0^{[t]}, b_p, \ldots, \lambda_{m-1}^{[t]}, b_{p+m-1})$$

   mit $b_p, \ldots, b_{p+m-1} \in \mathbb{B}$ existieren, sodaß gilt

   $$f^{(t)}(\pi_0^{[t-1]}, b_0, \ldots, \pi_{p-1}^{[t-1]}, b_{p-1}, \lambda_0^{[t-1]}, b_p, \ldots, \lambda_{m-1}^{[t-1]}, b_{p+m-1}) = 0$$

   Diese Eigenschaft kann auch ausgedrückt werden durch

   $$\begin{aligned} \exists_{\{y_0,\ldots,y_{p-1}\}} \forall_{\{y_p,\ldots,y_{p+m-1}\}} f^{(t+1)}(\pi_0^{[t]}, y_0, \ldots, \pi_{p-1}^{[t]}, y_{p-1}, \\ \lambda_0^{[t]}, y_p, \ldots, \lambda_{m-1}^{[t]}, y_{p+m-1}) = 0 \quad \forall t > 0 \end{aligned} \tag{30}$$

Diese zwei Eigenschaften sind für f notwendig, damit eine parameterbehaftete parametrisierte Lösung σ zu der Gleichung $f = 0^\omega$ existiert. Eine endliche online Funktion f mit diesen Eigenschaften heißt im folgenden *kontrollierbar*. Mit der folgenden äquivalenten Formulierung kann die Eigenschaft kontrollierbar definiert werden.

Definition 6.42. Sei $f \in \mathrm{eON}_{p+m,1}$. Die parameterbehaftete online Gleichung $f(\boldsymbol{\pi}, \boldsymbol{\xi}) = 0^\omega$ besitzt eine *kontrollierbare Lösungsmenge* (kurz: f ist kontrollierbar), wenn gilt

$$(\forall \boldsymbol{\pi}_1 \exists \boldsymbol{\xi}_1)(\forall \boldsymbol{\pi}_2 \exists \boldsymbol{\xi}_2): \; (f(\boldsymbol{\pi}_1, \boldsymbol{\xi}_1) = 0^\omega \wedge f(\boldsymbol{\pi}_2, \boldsymbol{\xi}_2) = 0^\omega \wedge {}_2|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2| \leq {}_2|\boldsymbol{\pi}_1 - \boldsymbol{\pi}_2|)$$

Existiert eine parameterbehaftete parametrisierte Lösung zu f, so kann diese nach obiger Bemerkung 6.41 folgende kontrollierbare Nullstellen beschreiben

Definition 6.43. $\boldsymbol{\xi}_1 \in {}_2\mathbb{Z}^m$ ist eine *kontrollierbare Nullstelle* für den Parameter $\boldsymbol{\pi}_1 \in {}_2\mathbb{Z}^p$, wenn $f(\boldsymbol{\pi}_1, \boldsymbol{\xi}_1) = 0^\omega$ und

$$\forall \boldsymbol{\pi}_2 \exists \boldsymbol{\xi}_2: \; (f(\boldsymbol{\pi}_1, \boldsymbol{\xi}_1) = 0^\omega \wedge f(\boldsymbol{\pi}_2, \boldsymbol{\xi}_2) = 0^\omega \wedge {}_2|\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2| \leq {}_2|\boldsymbol{\pi}_1 - \boldsymbol{\pi}_2|)$$

gilt.

Besitzt eine kontrollierbare endliche online Funktion f nur kontrollierbare Nullstellen, so wird sich später zeigen, daß eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung für f konstruiert werden kann.

Es ist deshalb erstrebenswert, aus einer kontrollierbaren online Funktion alle nicht kontrollierbaren Nullstellen zu eliminieren. Dies gelingt mit folgender Iteration.

Definition 6.44. Sei $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine endliche online Funktion und $f_{\langle b_0, \ldots, b_{m-1} \rangle}$ nach Definition 3.15 der Prädiktor zu $(b_0, \ldots, b_{p+m-1}) \in \mathbb{B}^m$. Dann wird folgende Iterationsvorschrift für f definiert

$$\begin{aligned} f[0] &:= \mathrm{mem(f)} \\ f[j+1] &:= \bigvee_{(b_0,\ldots,b_{p-1}) \in \mathbb{B}^p} \; \bigwedge_{(b_p,\ldots,b_{p+m-1}) \in \mathbb{B}^m} f[j]_{\langle b_0, \ldots, b_{p+m-1} \rangle} \end{aligned}$$

Die nachfolgenden Überlegungen zur Berechnung eines Fixpunktes sind identisch mit der Beweisführung aus Abschnitt 6.2.2. Die endlichen online Funktionen $f[j]$ können dargestellt werden durch

$$f[j] = \bigvee_{\alpha_0,\ldots,\alpha_{p-1} \in \mathbb{B}^j} \; \bigwedge_{\alpha_p,\ldots,\alpha_{p+m-1} \in \mathbb{B}^j} f[0]_{\langle \alpha_0, \ldots, \alpha_{p+m-1} \rangle}$$

Mit Hilfe der partiellen Ordnung $\leq$ aus Definition 6.13 kann nun auf ähnliche Weise wie bei den Lemmata 6.14 und 6.15 gezeigt werden, daß es ein $M \in \mathbb{N}_0$ gibt mit

$$f[M] = f[M+1]$$

und daraus folgt $f[M] = f[M+j]$ für alle $j \geq 0$. Die Terminierung der Iteration ist deshalb garantiert.

Definition 6.45. Sei $f \in \mathrm{eON}_{m,1}$. Die online Funktion mit $f[M] = f[M+j]$ für alle $j \geq 0$ mit $M \in \mathbb{N}_0$ heißt *Fixpunkt* $f[*]$ von der Iterationsvorschrift 6.44.

In [Stu96] wird für den Fall $f : {}_2\mathbb{Z} \times {}_2\mathbb{Z} \to {}_2\mathbb{Z}$ ausgeführt, daß bei dem Fixpunkt $f[*]$ genau die nicht kontrollierbaren Nullstellen eliminiert werden. Diese Beweise sind sehr leicht auf den Fall $f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ übertragbar. Im folgenden werden deshalb nur die Ergebnisse für spätere Überlegungen vorgestellt.

Notwendig zur Berechnung einer parameterbehafteten parametrisierten Lösung von f ist die Kontrollierbarkeit von f. Wenn aus dem Fixpunkt $f[*]$ eine Lösung gewonnen werden soll, so muß also garantiert sein, daß auch $f[*]$ kontrollierbar ist.

Lemma 6.46. *Wenn f kontrollierbar ist, dann ist $f[j]$ für alle $j \in \mathbb{N}_0$ kontrollierbar.*

Mit der Iteration wird der kontrollierbare Anteil der Nullstellen nicht verändert. Deshalb ist es gleichwertig, von einer online Funktion $f[j]$ die kontrollierbaren Nullstellen zu suchen.

Lemma 6.47. *Sei f kontrollierbar. Dann gilt für alle $j \in \mathbb{N}_0$*

$\boldsymbol{\xi}$ ist genau dann eine kontrollierbare Nullstelle von f, wenn $\boldsymbol{\xi}$ eine kontrollierbare Nullstelle von $f[j]$ ist.

Entscheidend ist nun, daß es mit Hilfe der Iteration gelingt, alle nicht kontrollierbaren Nullstellen zu eliminieren. Mit Hilfe dieser "bereinigten" online Funktion gelingt es dann, eine parameterbehaftete parametrisierte Lösung für die kontrollierbaren Nullstellen zu berechnen.

Lemma 6.48. *Ist $\boldsymbol{\xi}$ eine Nullstelle von $f[*]$ für einen Parameter $\boldsymbol{\pi}$, dann ist $\boldsymbol{\xi}$ eine kontrollierbare Nullstelle für den Parameter $\boldsymbol{\pi}$.*

Zusammenfassend kann festgestellt werden, daß zu einer kontrollierbaren endlichen online Funktion f mit Hilfe der Iterationsvorschrift 6.44 ein Fixpunkt $f[*]$ berechnet werden kann, der

- kontrollierbar ist,
- alle kontrollierbaren Nullstellen von f und
- keine nicht kontrollierbaren Nullstellen besitzt.

Analog zum Beweis von Lemma 6.18 kann gezeigt werden, daß der Fixpunkt $f[*]$ erweiterbar ist, wenn f mindestens eine kontrollierbare Nullstelle besitzt. Mit der Iteration ist deshalb gewährleistet, daß die Überlegungen aus Abschnitt 6.2.1 direkt angewendet werden können.

Lemma 6.49. *Sei f eine endliche online Funktion mit mindestens einer kontrollierbaren Nullstelle. Dann ist $f[*]$ erweiterbar.*

Aus einem solchen Fixpunkt $f[*]$ kann deshalb eine reproduktive allgemeine parametrisierte Lösung σ für die online Funktion $f : {}_2\mathbb{Z}^m \times {}_2\mathbb{Z}^p \to {}_2\mathbb{Z}^m \times {}_2\mathbb{Z}^p$ berechnet werden.

Ist f außerdem kontrollierbar, so ist σ eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung von $f[*]$. Dieses σ beschreibt dann alle kontrollierbaren Nullstellen von f.

Satz 6.50. *Besitzt eine kontrollierbare endliche online Funktion f nur kontrollierbare Nullstellen, dann gibt es eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung σ zu der Gleichung $f = 0^\omega$.*

Beweis:

Sei $f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine endliche kontrollierbare online Funktion, die nur kontrollierbare Nullstellen besitzt. Im folgenden sei $n = m + p$.

Da f kontrollierbar ist, gibt es eine kontrollierbare Nullstelle und folglich ist f erweiterbar. Deshalb kann eine reproduktive allgemeine parametrisierte Lösung

$$\sigma : \begin{cases} {}_2\mathbb{Z}^n & \to \quad {}_2\mathbb{Z}^n \\ (\boldsymbol{\zeta}_0, \ldots, \boldsymbol{\zeta}_{n-1}) & \mapsto \quad (\tau_0(\boldsymbol{\zeta}), \ldots, \tau_{n-1}(\boldsymbol{\zeta})) \end{cases}$$

mit $\tau_i = \sum \tau_i^{(t)} 2^t$ wobei

$$\begin{aligned}
\tau_i^{(0)}(\zeta_0^{(0)}, \ldots, \zeta_i^{(0)}) &= \operatorname{ite}(z_i^{(0)}, \neg\ f_{\{n-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 1), \\
&\qquad f_{\{n-1-i\}}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 0)) \\
\tau_i^{(t)}(\boldsymbol{\zeta}^{[t]}) &= \operatorname{ite}(z_i^{(t)}, \neg\ f_{\{n-1-i\}}^{(t)}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, \tau_i^{[t-1]}, 1, \tau_i^{[t-1]}, \ldots, \tau_{n-1}^{[t-1]}), \\
&\qquad f_{\{n-1-i\}}^{(t)}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, \tau_i^{[t-1]}, 0, \tau_{i+1}^{[t-1]}, \ldots, \tau_{n-1}^{[t-1]})) \quad t > 0
\end{aligned}$$

zu $f : {}_2\mathbb{Z}^n \to {}_2\mathbb{Z}^n$ berechnet werden.

Da f kontrollierbar ist und nur kontrollierbare Nullstellen besitzt, gelten die Eigenschaften (29) und (30) aus Bemerkung 6.41 für alle Anfangslösungen von σ.

Im folgenden muß gezeigt werden, daß

$$\tau_i(\zeta_0, \ldots, \zeta_{n-1}) = \sum_{t \geq 0} \tau_i^{(t)} 2^t = \zeta_i \quad \forall i (0 \leq i < p)$$

gilt.

Für alle $0 \leq i < p$ ist dies für $t = 0$ wegen

$$\begin{aligned}
&\tau_i^{(0)}(\zeta_0^{(0)}, \ldots, \zeta_{n-1}^{(0)}) = \zeta_i^{(0)} \\
&\quad \Leftrightarrow f_{\{n-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 1) \vee f_{\{n-1-i\}}^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, 0) = 0 \\
&\quad \Leftrightarrow \exists_{\{y_{i-1}\}} \forall_{\{y_i, \ldots, y_{n-1}\}} f^{(0)}(\tau_0^{(0)}, \ldots, \tau_{i-1}^{(0)}, y_{i-1}, y_i, \ldots, y_{n-1}) = 0
\end{aligned}$$

und Gleichung (29) erfüllt und wegen

$$\begin{aligned}
&\tau_i^{(t)}(\zeta_0^{[t]}, \ldots, \zeta_{n-1}^{[t]}) = \zeta_i^{(t)} \\
&\quad \Leftrightarrow\ f_{\{n-1-i\}}^{(t)}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, 1, \tau_i^{[t-1]}, \ldots, \tau_{n-1}^{[t-1]}) \vee \\
&\qquad\ f_{\{n-1-i\}}^{(t)}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, 0, \tau_i^{[t-1]}, \ldots, \tau_{n-1}^{[t-1]}) = 0 \\
&\quad \Leftrightarrow \exists_{\{y_{i-1}\}} \forall_{\{y_i, \ldots, y_{n-1}\}} f^{(0)}(\tau_0^{[t]}, \ldots, \tau_{i-1}^{[t]}, y_{i-1}, \ldots, \tau_{n-1}^{[t]}, y_{n-1}) = 0
\end{aligned}$$

und Gleichung (30) für $t > 0$ erfüllt. $\square$

6.5 Berechnung der Nullstellen eines parameterbehafteten synchronen Schaltkreises

Im folgenden ist $\mathcal{S}$ ein synchroner Schaltkreis $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ vom Typ $(l, m+p, 1)$ mit den Gesamt-Eingabevariablen $\tilde{\boldsymbol{X}} = \langle \boldsymbol{P}, \boldsymbol{X} \rangle$, wobei $\boldsymbol{X}$ die m Eingabevariablen , und $\boldsymbol{P}$ die p Parametereingaben sind. Sei $f : {}_2\mathbb{Z}^p \times {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ die parameterbehaftete online Funktion, die von $\mathcal{S}$ berechnet wird.

Ist f eine kontrollierbare online Funktion, so kann nach Kapitel 6.4 eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung von f berechnet werden, die alle kontrollierbaren Nullstellen beschreibt:

1. Berechne einen synchronen Schaltkreis $\mathcal{S}[*]$, sodaß die zugehörige online Funktion $f[*]$ genau die kontrollierbaren Nullstellen von f besitzt. Dies gelingt für f mit folgender Iteration aus Definition 6.44

$$\begin{aligned} f[0] &:= \text{mem(f)} \\ f[j+1] &:= \bigvee_{(b_0,\ldots,b_{p-1}) \in \mathbb{B}^p} \; \bigwedge_{(b_p,\ldots,b_{p+m-1}) \in \mathbb{B}^m} f[j]_{\langle b_0,\ldots,b_{p+m-1} \rangle} \end{aligned}$$

   Alle Konstruktionen und Algorithmen aus Abschnitt 6.3.1 können übernommen werden, indem die Quantifizierung

$$\forall_X \, \lambda$$

   zu

$$\exists_P \forall_X \, \lambda(\boldsymbol{P}, \boldsymbol{X}, \boldsymbol{Q})$$

   erweitert wird.

2. Wird zu $f[*]$ ein synchroner Schaltkreis $\mathcal{S}_L$ mit Konstruktion 6.36 erzeugt, so ist die zugehörige online Funktion nach Satz 6.50 eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung von $f[*]$.

7 Lösung von offline Gleichungen

In diesem Kapitel wird eine reproduktive allgemeine parametrisierte Lösung von einem synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ konstruiert.

Nach Satz 3.13 ist eine 2-adische Funktion $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ genau dann offline, wenn es einen möglicherweise unendlichen synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ gibt, der f berechnet. Die offline Funktionen, die von einem endlichen synchronen Schaltkreis berechnet werden können, werden als endliche offline Funktionen bezeichnet.

Sei nun $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n) und $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ die zugehörige (endliche) offline Funktion. Gesucht wird eine reproduktive allgemeine parametrisierte Lösung $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ zu der Gleichung $f = 0^\omega$.

Im folgenden wird eine Funktion $\tilde{f} : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^n$ definiert, die endlich online ist und die gleiche Nullstellenmenge wie die offline Funktion f besitzt.

Aus dieser Definition läßt sich dann direkt aus dem synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ ein synchroner Schaltkreis $\tilde{\mathcal{S}} = (\tilde{\delta}, \tilde{\lambda}, \tilde{\boldsymbol{s}})$ konstruieren.

Mit diesem synchronen Schaltkreis kann dann mit den Konstruktionen aus Abschnitt 6.3 eine reproduktive allgemeine parametrisierte Lösung $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ berechnet werden.

Ist die offline Funktion $f = (f_0, \ldots, f_{n-1})$ gegeben durch

$$f_i : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t \geq 0} f^{(t)}(\boldsymbol{\xi}^{[s^{(t)}]}) 2^t \end{cases}$$

so wird die online Funktion $f = (\tilde{f}_0, \ldots, \tilde{f}_{n-1})$ definiert durch

$$\tilde{f}_i : \begin{cases} {}_2\mathbb{Z}^m & \to \quad {}_2\mathbb{Z} \\ \boldsymbol{\xi} & \mapsto \quad \sum_{t \geq 0} \tilde{f}_i^{(t)}(\boldsymbol{\xi}^{[t]}) 2^t \end{cases}$$

mit

$$\tilde{f}_i^{(t)} = \begin{cases} 0, & \text{wenn} \quad 0 \leq t < s^{(0)} \\ f_i^{(r)}, & \text{wenn} \quad s^{(r)} \leq t < s^{(r+1)} \end{cases}$$

Da $\tilde{f}_i^{(t)}$ für $0 \leq t < s^{(0)}$ eine 0 ausgibt, entsprechen die Nullstellen von f genau den Nullstellen von $\tilde{f}$. Mit folgender Konstruktion kann nun aus dem synchronen Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ ein synchroner Schaltkreis $\tilde{\mathcal{S}}$ erzeugt werden, der die online Funktion $\tilde{f}$ berechnet.

Mit diesem synchronen Schaltkreis $\tilde{\mathcal{S}}$ kann dann mit den Konstruktionen aus Abschnitt 6.3 eine Lösung für $\tilde{f} = 0^\omega$ und damit für $f = 0^\omega$ berechnet werden.

Konstruktion 7.1. Sei $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n). Seien $\boldsymbol{X}$ die Eingabevariablen, $\boldsymbol{H}$ die Hilfsausgabevariablen, $\boldsymbol{Q}$ die Zustandsvariablen und $\boldsymbol{Q}'$ die Folgezustandsvariablen.

Seien $Q_{H_0}, \ldots, Q_{H_{n-1}}$ zusätzliche Zustandsvariable und $Q'_{H_0}, \ldots, Q'_{H_{n-1}}$ die zugehörigen Folgezustandsvariablen.

Der synchrone Schaltkreis $\tilde{\mathcal{S}} = (\tilde{\delta}, \tilde{\lambda}, \tilde{\boldsymbol{s}})$ vom Typ $(l + n, m, n)$ besitzt die Eingabevariablen $\boldsymbol{X}$, Ausgabevariablen $\boldsymbol{H}$, Zustandsvariablen $\tilde{\boldsymbol{Q}} = \langle Q_0, \ldots, Q_{l-1}, Q_{H_0}, \ldots, Q_{H_{n-1}} \rangle$ mit den zugehörigen Folgezustandsvariablen $\tilde{\boldsymbol{Q}}' = \langle Q'_0, \ldots, Q'_{l-1}, Q'_{H_0}, \ldots, Q'_{H_{n-1}} \rangle$ und wird definiert durch

- die Übergangsfunktion

$$\tilde{\delta} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+n} & \rightarrow & \mathbb{B}^{l+n} \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto & \tilde{\boldsymbol{Q}}' \end{cases}$$

  mit

$$\begin{aligned} {Q'}_i &= \delta_i \quad \forall i (0 \leq i < l) \\ Q'_{H_i} &= \mathrm{ite}(\alpha_i, \lambda_i, Q_{H_i}) \quad \forall i (0 \leq i < n) \end{aligned}$$

- die Ausgabefunktion

$$\tilde{\lambda} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^{l+n} & \rightarrow & \mathbb{B}^n \\ (\boldsymbol{X}, \tilde{\boldsymbol{Q}}) & \mapsto & \boldsymbol{H} \end{cases}$$

  mit $H_i := Q'_{H_i}$, d.h. die Ausgabefunktion entspricht der Zustandsfunktion von den Konstruktionsvariablen Q_H

- und die Startmenge $\tilde{\boldsymbol{s}} = \boldsymbol{s}.(0, \ldots, 0)$.

Soll nun zu dem synchronen Schaltkreis $\tilde{\mathcal{S}}$ eine reproduktive allgemeine parametrisierte Lösung konstruiert werden, so wird verlangt, daß die Ausgabe von $\tilde{f}$ einstellig ist. Mit $\mathcal{S}' = (\tilde{\delta}, \lambda', \tilde{\boldsymbol{s}})$, wobei

$$\lambda' = \bigvee_{0 \leq i < n} \tilde{\lambda}_i$$

kann dann ein synchroner Schaltkreis mit einer Ausgabe berechnet werden, sodaß die zugehörigen online Funktionen dieselbe Nullstellenmenge besitzen.

Dieser synchrone Schaltkreis besitzt dann jedoch m zusätzliche Zustandsvariablen.

Mit folgender Konstruktion kann der synchrone Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ zuerst zu einen Automaten $\tilde{\mathcal{S}}$ mit einer einstelligen Ausgabe umgeformt werden, sodaß die zugehörigen offline Funktionen dieselben Nullstellen besitzen. Anschließend kann aus $\tilde{\mathcal{S}}$ mit Konstruktion 7.1 ein synchroner Schaltkreis $\mathcal{S}'$ berechnet werden, der nur eine zusätzliche Zustandsvariable benötigt.

Konstruktion 7.2. Sei $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n) mit den Eingabevariablen $\boldsymbol{X}$, Zustandsvariablen $\boldsymbol{Q}$ und Folgezustandsvariablen $\boldsymbol{Q}'$.

Der synchrone Schaltkreis $\tilde{\mathcal{S}} = (\tilde{\delta}, \lambda, \tilde{\alpha}, \boldsymbol{s})$ vom Typ $(l, m, 1)$ besitzt die Eingabevariablen $\boldsymbol{X}$, Hilfsausgabevariable H, Ausgabeberechtigungsvariable A, Zustandsvariablen $\boldsymbol{Q}$, Folgezustandsvariablen $\boldsymbol{Q}'$ und wird definiert durch

- die Hilfsausgabe

$$\tilde{\lambda} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow & \mathbb{B} \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & H \end{cases}$$

  mit

$$H = \bigvee_{0 \leq i < n} (\lambda_i \wedge \alpha_i)$$

- die Ausgabeberechtigung

$$\tilde{\alpha} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l & \rightarrow & \mathbb{B} \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & A \end{cases}$$

  mit

$$A = \bigvee_{0 \leq i < n} \alpha_i$$

8 Lösung von Ungleichungen

Eine interessante Frage ist, welche Eingaben möglich sind, sodaß ein synchroner Schaltkreis $\mathcal{S}$ nicht die Nullfolge ausgibt. Ist $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ die zugehörige online Funktion, dann wird die Lösungsmenge zu der Ungleichung

$$f(\boldsymbol{\xi}) \neq 0^\omega$$

gesucht.

Soll die Lösungsmenge von einem endlichen synchronen Schaltkreis $\mathcal{S}_L$ berechnet werden, so muß dieser für alle Eingaben alle Lösungen auflisten, mit denen $\mathcal{S}$ nicht die Nullfolge ausgibt.

Dabei gibt es zwei Arten von Nullstellen für $\mathcal{S}$.

Definition 8.1. Sei $f : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}$ eine endliche online Funktion. $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ mit $f(\boldsymbol{\xi}) = 0^\omega$ heißt

- *universelle Nullstelle* von f, wenn gilt

$$\exists t \in \mathbb{N}_0 \forall \boldsymbol{v} \in {}_2\mathbb{Z}^m ({}_2|\boldsymbol{v} - \boldsymbol{\xi}| \leq 2^{-t} \Rightarrow f(\boldsymbol{v}) = 0^\omega$$

- *existentielle Nullstelle* von f, wenn gilt

$$\forall t \in \mathbb{N}_0 \exists \boldsymbol{v} \in {}_2\mathbb{Z}^m ({}_2|\boldsymbol{v} - \boldsymbol{\xi}| \leq 2^{-t} \wedge f(\boldsymbol{v}) \neq 0^\omega)$$

Mit folgender Iteration gelingt es, die universellen Nullstellen von f zu eliminieren.

Definition 8.2. Für $f \in \mathrm{eON}_{m,1}$ wird folgende Iteration definiert.

$$\begin{aligned} f[0] &:= \neg\mathrm{mem(f)} \\ f[j+1] &:= \bigwedge_{b_0,\ldots,b_{m-1} \in \mathbb{B}} f[j]_{\langle b_0,\ldots,b_{m-1}\rangle} \end{aligned}$$

Zu dieser Iteration kann analog zur Iteration 6.10 folgendes gezeigt werden.

- Es gibt einen Fixpunkt $f[*]$, der bestimmt ist mit $f[M] = f[M+1]$.
- $f[*](\boldsymbol{\xi}) \neq 0^\omega$ gilt genau dann, wenn $\boldsymbol{\xi}$ keine universelle Nullstelle ist.
- $f[*]$ ist eine erweiterbare online Funktion, wenn $f[*] \neq 0^\omega$ gilt.

Mit einer reproduktiven allgemeinen parametrisierten Lösung $\sigma : {}_2\mathbb{Z}^m \to {}_2\mathbb{Z}^m$ werden dann alle Folgen berechnet, die für f

- entweder keine Nullstellen
- oder existentielle Nullstellen sind.

Wenn ein synchroner Schaltkreis alle Nicht-Nullstellen berechnen soll, so berechnet er - wie in der Einleitung bereits erwähnt - zusätzlich alle existentiellen Nullstellen mit. Denn existiert in f eine existentielle Nullstelle, dann gibt es zu jeder Anfangsfolge dieser Nullstelle nach Definition eine Nicht-Nullstelle mit der gleichen Anfangsfolge. Da alle Nicht-Nullstellen berechnet werden können, kann jedes Anfangsstück der Länge $k \in \mathbb{N}_0$ von der existentiellen Nullstelle und damit auch die existentielle Nullstelle selbst berechnet werden. Die Lösungsmenge einer Ungleichung kann also im allgemeinen nicht mit Hilfe eines synchronen Schaltkreises berechnet werden.

Umgekehrt kann jedoch jeder Anfang einer existentiellen Nullstelle $\boldsymbol{\xi} = (\xi_0, \ldots, \xi_{m-1}) \in {}_2\mathbb{Z}^m$ fortgesetzt werden zu einer Nicht-Nullstelle. Ist nun $\boldsymbol{\xi}^{[t]}$ ein Anfang von $\boldsymbol{\xi}$ der Länge $t+1$, so gibt es eine Fortsetzung $b_0, \ldots, b_m \in \mathbb{B}^r$ mit

$$f^{(t+r)}(\xi_0^{[t]}, b_0, \ldots, \xi_{m-1}^{[t]}, b_{m-1}) = 1$$

M.a.W. kann zu jeder existentiellen Nullstelle $\boldsymbol{\xi}$ ein Prädiktor $f_{\langle b_0,\ldots,b_{m-1}\rangle}$ von f gefunden werden, sodaß $f_{\langle b_0,\ldots,b_{m-1}\rangle}(\boldsymbol{\xi}) \neq 0^\omega$ gilt. Da es nach Bemerkung 3.16 für eine endliche online Funktion nur endlich viele Prädiktoren gibt, werden nur endlich viele Folgen endlicher Länge benötigt, um alle existentiellen Nullstellen zu einer Nicht-Nullstelle fortsetzen zu können.

Mit dieser Überlegung kann nun eine online Funktion

$$\tau : \begin{cases} {}_2\mathbb{Z} \times {}_2\mathbb{Z}^m & \rightarrow \quad {}_2\mathbb{Z}^m \\ (\pi, \boldsymbol{\xi}) & \mapsto \quad \tau(\pi, \boldsymbol{\xi}) \end{cases}$$

konstruiert werden, mit der unter der Einschränkung $\pi \neq 0^\omega$ alle Nicht-Nullstellen beschrieben werden können. τ hat dabei folgendes Verhalten:

- Solange $\pi_0, \ldots, \pi_i = 0$: Ausgabeverhalten von σ, d.h.
  es wird der Anfang aller Nicht-Nullstellen und aller existentiellen Nullstellen ausgegeben.
- Wenn das erste Mal $\pi_i = 1$ gilt:
  Sei $\boldsymbol{\lambda}^{[t]} = (\lambda_0^{[t]}, \ldots, \lambda_{m-1}^{[t]})$ für $t \leq i$ die berechnete Ausgabe von τ bzw. σ im Berechnungsschritt t.
  - Wenn es ein $t \leq i$ gibt mit $f^{(t)}(\boldsymbol{\lambda}^{[t]}) = 1$: Ausgabeverhalten wie σ, da
    jedes $\xi \in {}_2\mathbb{Z}^m$ mit dem Anfangsstück $\boldsymbol{\lambda}^{[t]}$ bereits eine Nicht-Nullstellen ist.
  - sonst:
    * Gib in den nächsten r Schritten eine Ausgabe[9] $b_0, \ldots, b_{m-1} \in \mathbb{B}^r$ aus, sodaß gilt

$$f^{(r+i)}(\lambda_0^{[i]}, b_0, \ldots, \lambda_{m-1}^{[i]}, b_{m-1}) = 1 \tag{31}$$

    * Anschließend: beliebige Ausgabe

Wenn nun garantiert ist, daß $\pi \neq 0^\omega$ gilt, dann wird auf jeden Fall eine Nicht-Nullstelle von f ausgegeben. Denn jede Anfangsfolge $\boldsymbol{\lambda}^{[t]}$ von σ ist die Anfangsfolge einer Nicht-Nullstelle oder einer existentiellen Nullstelle. Mit obigen Überlegungen kann daher $\boldsymbol{\lambda}^{[t]}$ zu einer endlichen Folge $\boldsymbol{\lambda}^{[t+r]}$ = fortgesetzt werden, sodaß $f^{(t+r)}(\boldsymbol{\lambda}^{[t+r]}) = 1$ gilt.

Außerdem werden mit allen Eingaben $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$,$\pi \in {}_2\mathbb{Z}$ mit $\pi \neq 0^\omega$ alle Nicht-Nullstellen aufgelistet: Sei $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$ eine Nicht-Nullstelle von f. Ist nun $f^{(t)}(\boldsymbol{\xi}^{[t]}) \neq 0$, wobei t minimal ist für ein $\boldsymbol{\xi} \in {}_2\mathbb{Z}^m$, dann wird mit der Anfangsfolge $\pi^{[t]} = \langle 0, \ldots, 0, 1\rangle$ von einem π das Ausgabeverhalten von τ immer wie σ sein. Da σ eine reproduktive Lösung ist, berechnet τ für die Eingaben $\boldsymbol{\xi}$ und π die Ausgabe $\boldsymbol{\xi}$.

Mit dieser Strategie kann die Beschreibung der Lösungsmenge von der Ungleichung $f(\boldsymbol{\xi}) \neq 0^\omega$ mit τ folgendermaßen umgesetzt werden.

$$f(\boldsymbol{\xi}) \neq 0^\omega \Leftrightarrow \exists \pi \in {}_2\mathbb{Z} \setminus \{0^\omega\} : \boldsymbol{\xi} = \tau(\pi, \boldsymbol{\xi})$$

[9]Dies ist möglich, da alle universellen Nullstellen eliminiert sind.

Ein synchroner Schaltkreis $\mathcal{S}_L$, der die online Funktion τ berechnet, kann direkt konstruiert werden. Dieser verhält sich mit den Eingaben $\boldsymbol{X} = \langle X_0, \ldots, X_{m-1} \rangle$ und der Zusatzeingabe P im Normalfall wie σ.

Nur wenn die Eingabe P das erste Mal 1 wird und der synchrone Schaltkreis $\mathcal{S}$ mit der berechneten Eingabe von $\mathcal{S}_L$ noch keine 1 ausgegeben hat, dann wird eine Eingabefolge ausgegeben, mit der der synchrone Schaltkreis $\mathcal{S}$ eine 1 ausgeben muß.

Im folgenden wird dargelegt, wie Berechnungsschritt (31) von τ umgesetzt werden kann. In einem ersten Schritt werden die Eingabefolgen, mit denen $\mathcal{S}$ von einem Zustand ausgehend eine 1 ausgeben kann, mit einer booleschen Funktion `1_Folge` kodiert. Anschließend wird vorgestellt, wie der synchrone Schaltkreis $\mathcal{S}_L$ mit `1_Folge` konstruiert werden kann, damit dieser die online Funktion τ berechnet.

8.1 Kodieren der Eingabefolgen

Sei $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ (l, m, n) mit den m Eingabevariablen $\boldsymbol{X}$, l Zustandsvariablen $\boldsymbol{Q}$, Folgezustandsvariablen $\boldsymbol{Q}'$ und der Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$.

Zusätzlich werden für die folgende Konstruktion die l Hilfsvariablen $\tilde{\boldsymbol{Q}}$ eingeführt, um Zustände von $\mathcal{S}$ zu kodieren und die Variablen $\boldsymbol{X}^{(i)}$ für $i \geq 0$ mit $\boldsymbol{X}^{(i)} = \langle X_0^{(i)}, \ldots, X_{m-1}^{(i)} \rangle$ zur Kodierung der Eingabefolgen.

Zu allen Zuständen, von denen $\mathcal{S}$ ausgehend eine 1 berechnen kann, wird nun eine entsprechende Eingabe gesucht. Dabei wird für diese Zustände genau eine Eingabefolge in der Relation `1_Folge` kodiert.

In einem Berechnungsschritt $i \geq 2$ sind in einer charakteristischen Funktion $\texttt{erledigt}(\tilde{\boldsymbol{Q}})$ alle Zustände gesammelt, für die eine Folge bis zur Länge $i - 1$ gefunden worden ist, sodaß $\mathcal{S}$ eine 1 ausgeben muß.

Diese Eingabefolgen zu den Zuständen $\texttt{erledigt}(\tilde{\boldsymbol{Q}})$ sind in

$$\texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)})$$

kodiert. Es gilt dabei $\texttt{1\_Folge}(q, x^{(0)}, \ldots, x^{(i-2)}) = 1$ genau dann, wenn von dem Zustand $q \in \mathbb{B}^l$ ausgehend mit den Eingaben $x^{(0)}, \ldots, x^{(i-2)} \in \mathbb{B}^m$ eine 1 ausgegeben wird.

Weiter liegt vor dem Berechnungschritt i eine Übergangsrelation $\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{Q})$ vor, mit

$$\delta^{(*)}(q, x^{(0)}, \ldots, x^{(i-2)}, q') = 1 \Leftrightarrow \delta(x^{(i-2)}, \delta(x^{(i-3)}, \ldots, \delta(\delta(x^{(0)}, q)) \ldots)) = q'$$

d.h. von dem Zustand $q \in \mathbb{B}^l$ aus wird mit den Eingaben $x^{(0)}, \ldots, x^{(i-2)} \in \mathbb{B}^m$ in den Zustand $q' \in \mathbb{B}^l$ übergegangen.

Im Schritt i werden diese Relationen nun für die Eingabefolgen $x^{(0)}, \ldots, x^{(i-1)} \in \mathbb{B}^m$ erweitert.

In $\texttt{1\_Folge}_{neu}$ werden dabei alle Eingaben der Länge i zu einem Zustand $\boldsymbol{Q}$ berechnet, mit denen von $\boldsymbol{Q}$ ausgehend eine 1 ausgegeben wird.

$$\texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}) = \exists_Q \left[\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{Q}) \wedge \lambda(\boldsymbol{X}^{(i-1)}, \boldsymbol{Q})\right]$$

Mit $\texttt{erfüllt}(\tilde{\boldsymbol{Q}})$ werden nun alle Zustände kodiert, sodaß $\mathcal{S}$ von diesen ausgehend mit einer Eingabefolge der Länge i eine 1 ausgeben kann.

$$\texttt{erfüllt}(\tilde{\boldsymbol{Q}}) = \exists_{\{\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}\}} \texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)})$$

In $\texttt{erledigt}(\tilde{\boldsymbol{Q}})$ werden dann mit

$$\texttt{erledigt}(\tilde{\boldsymbol{Q}}) = \texttt{erledigt}(\tilde{\boldsymbol{Q}}) \vee \texttt{erfüllt}(\tilde{\boldsymbol{Q}})$$

alle Zustände gesammelt, von denen ausgehend $\mathcal{S}$ mit einer Eingabefolge bis zur Länge i eine 1 ausgeben kann.

Genauso werden mit

$$\texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}) = \texttt{1\_Folge}(\tilde{\boldsymbol{Q}}) \vee \texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)})$$

alle Eingabefolgen der Länge i zusammengefaßt, mit denen $\mathcal{S}$ von den Zuständen $\texttt{erledigt}(\tilde{\boldsymbol{Q}})$ ausgehend eine 1 berechnet.

Anschließend kann mit

$$\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}, \boldsymbol{Q}') = \exists_Q \left[\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{Q}) \wedge T(\boldsymbol{X}^{(i-1)}, \boldsymbol{Q}, \boldsymbol{Q}')\right]$$

die Übergangsrelation zur Eingabe $\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}$ erzeugt werden.

Da mit der Übergangsrelation $\delta^{(*)}$ nur noch die Übergänge von den Zuständen $\neg\texttt{erledigt}(\boldsymbol{Q})$ betrachtet werden müssen, kann $\delta^{(*)}$ eingeschränkt werden durch

$$\delta^{(*)} = \delta^{(*)} \wedge \neg\texttt{erledigt}$$

Wenn $R(\tilde{\boldsymbol{Q}})$ die erreichbaren Zustände von $\mathcal{S}$ sind, die nach w Übergängen von dem Startzustand erreicht werden können, so kann nach maximal $i = w$ Iterationen abgebrochen werden. Denn spätestens dann werden mit allen Eingabefolgen der Länge $w + 1$ nur noch bereits durchlaufene Zustände erreicht.

Es kann bereits vorher abgebrochen werden, wenn $R(\tilde{\boldsymbol{Q}}) \subseteq \texttt{erledigt}(\tilde{\boldsymbol{Q}})$ gilt. Für alle erreichbaren Zustände ist dann eine Eingabefolge berechnet worden. Dies tritt aber nur dann ein, wenn es keinen erreichbaren Zustand gibt, von dem aus keine 1 mehr berechnet werden kann, d.h. wenn es keine universellen Nullstellen für f gibt.

Im Berechnungsschritt $i = 1$ werden die charakteristischen Funktionen folgendermaßen initialisiert.

$$\delta^{(*)} = (\tilde{\boldsymbol{Q}} \equiv \boldsymbol{Q}), \quad \texttt{erledigt} = \texttt{1\_Folge} = 0$$

Nach der Berechnung von $\texttt{1\_Folge}$ mit r Iterationen gilt $\texttt{1\_Folge}(q, x^{(0)}, \ldots, x^{(r-1)}) = 1$ genau dann, wenn von dem Zustand q ausgehend innerhalb der Eingabe $x^{(0)}, \ldots, x^{(t-1)}$ von $\mathcal{S}$ eine 1 ausgegeben worden ist.

Für die spätere Konstruktion von $\mathcal{S}_L$ wird außerdem die Eigenschaft benötigt, daß zu jedem Zustand höchstens eine Eingabefolge in $\texttt{1\_Folge}$ kodiert ist.

Dies gelingt mit folgender strikten lexikographischen Ordnung $<$ aus Beispiel 2.25 auf den Wörtern $\langle x^{(0)}, \ldots, x^{(r-1)}\rangle \in \mathbb{B}^{mr}$ mit $x^{(i)} = \langle x_0^{(i)}, \ldots, x_{m-1}^{(i)}\rangle \in \mathbb{B}^m$:

$$\begin{aligned}&\langle x^{(0)}, \ldots, x^{(r-1)}\rangle > \langle \tilde{x}^{(0)}, \ldots, \tilde{x}^{(r-1)}\rangle \Leftrightarrow \exists i(0 \leq i < r)\ \exists j(0 \leq j < m) : \\ &\left(x^{(0)} = \tilde{x}^{(0)} \wedge \ldots \wedge x^{(i-1)} = \tilde{x}^{(i-1)} \wedge \left(x_0^{(i)} = \tilde{x}_0^{(i)} \wedge \ldots \wedge x_{j-1}^{(i)} = \tilde{x}_{j-1}^{(i)} \wedge (x_{(j)}^{(i)} = 1 \wedge \tilde{x}_j^{(i)} = 0)\right)\right)\end{aligned}$$

Sind $\boldsymbol{X}^{(i)} = (X_0^{(i)}, \ldots, X_{m-1}^{(i)})$ die Eingabevariablen mit $0 \leq i < r$, so werden zusätzlich die Hilfsvariablen $\tilde{\boldsymbol{X}}^{(i)} = (\tilde{X}_0^{(i)}, \ldots, \tilde{X}_{m-1}^{(i)})$ benötigt, um diese Ordnung mit Hilfe einer booleschen Funktion zu kodieren:

$$O : \left\{ \begin{array}{lcl} \mathbb{B}^{2mr} & \rightarrow & \mathbb{B} \\ (Y, \tilde{Y}) & \mapsto & \left\{ \begin{array}{ll} 1, & \text{wenn} \quad Y > \tilde{Y} \\ 0, & \text{sonst} \end{array} \right. \end{array} \right.$$

wobei $Y = \langle \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)} \rangle$ und $\tilde{Y} = \langle \tilde{\boldsymbol{X}}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(r-1)} \rangle$

Von den möglichen Eingabefolgen zu einem Zustand wird nun jeweils eine Eingabefolge ausgewählt durch folgende Strategie:

$$\{(\tilde{q}, x) \in \mathbb{B}^{l+mr} \mid \neg(\exists(\tilde{q}, \tilde{x}) \in \texttt{1\_Folge} : x > \tilde{x})\} \cap \texttt{1\_Folge}$$

d.h. zu jedem Zustand, wenn möglich, wird die kleinste Eingabefolge bzgl. der Ordnung $<$ ausgewählt.

Mit der Operation

$$\begin{aligned}&\texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)}) = \texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)})\\ &\wedge \neg \exists_{\tilde{X}^{(0)} \cup \ldots \cup \tilde{X}^{(r-1)}} \left[O(\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)}, \tilde{\boldsymbol{X}}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(r-1)}) \wedge \texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{X}}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(r-1)}) \right]\end{aligned}$$

wird dann für jeden Zustand $q \in \mathbb{B}^l$, der in der Relation berücksichtigt wird, genau die Eingabefolge ausgewählt, die bzgl. der strikten lexikographischen Ordnung am kleinsten ist.

Insgesamt kann `1_Folge` nun folgendermaßen berechnet werden:

Algorithmus 8.3 (Berechne `1_Folge`).
// Input: Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$, Ausgabefunktion λ, erreichbare Zustände $R(\tilde{\boldsymbol{Q}})$, Erreichbarkeitstiefe `tiefe`
// Output: $\texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}_0, \ldots, \boldsymbol{X}_r)$
bdd berechne_1_Folge
{
$\delta^{(*)} = (\tilde{\boldsymbol{Q}} \equiv \boldsymbol{Q})$; $\texttt{erledigt} = \mathbf{0}$; $\texttt{1\_Folge} = \mathbf{0}$;
$i = 0$;
do
{
$i{+}{+}$;
$\texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X}) = \exists_Q \left[\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{Q}) \wedge \lambda(\boldsymbol{X}, \boldsymbol{Q}) \right]$;
$\texttt{erfüllt}(\tilde{\boldsymbol{Q}}) = \exists_{\{\boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X}\}} \texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X})$;
$\texttt{erledigt}(\tilde{\boldsymbol{Q}}) = \texttt{erledigt}(\tilde{\boldsymbol{Q}}) \vee \texttt{erfüllt}(\tilde{\boldsymbol{Q}})$;
$\delta^{(*)} = \delta^{(*)} \wedge \neg \texttt{erledigt}^-$;
$\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X}, \boldsymbol{Q}') = \exists_Q \left[\delta^{(*)^-}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{Q}) \wedge T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \right]$;
$\texttt{1\_Folge}_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}) = \texttt{1\_Folge}^-_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X} \leftarrow \boldsymbol{X}^{(i-1)})$;
$\delta^{(*)}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}, \boldsymbol{Q}) = \delta^{(*)^-}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-2)}, \boldsymbol{X} \leftarrow \boldsymbol{X}^{(i-1)}, \boldsymbol{Q}' \leftarrow \boldsymbol{Q})$;
$\texttt{1\_Folge}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}) = \texttt{1\_Folge}^-(\tilde{\boldsymbol{Q}}) \vee \texttt{1\_Folge}^-_{neu}(\tilde{\boldsymbol{Q}}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)})$;
}
while$(R(\tilde{\boldsymbol{Q}}) \nsubseteq \texttt{erledigt}(\tilde{\boldsymbol{Q}})$ && $i < \texttt{tiefe})$;
FREE(`erledigt`);
$\texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}) = \texttt{1\_Folge}(\tilde{\boldsymbol{Q}} \leftarrow \boldsymbol{Q}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)})$;
$\texttt{1\_Folge}'(\boldsymbol{Q}, \tilde{\boldsymbol{X}}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(i-1)}) = \texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}^{(0)} \leftarrow \tilde{\boldsymbol{X}}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)} \leftarrow \tilde{\boldsymbol{X}}^{(i-1)})$;
$\texttt{1\_Folge} = \texttt{1\_Folge}^- \wedge$
$\quad \neg \exists_{\tilde{X}^{(0)} \cup \ldots \cup \tilde{X}^{(i-1)}} \left[O(\boldsymbol{X}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(i-1)}) \wedge \texttt{1\_Folge}'^-(\boldsymbol{Q}, \tilde{\boldsymbol{X}}^{(0)}, \ldots, \tilde{\boldsymbol{X}}^{(i-1)}) \right]$;
return$(\texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(i-1)}))$;

}

8.2 Automatenkonstruktion

Der Automat $\mathcal{S}_L = (\lambda^{(L)}, \delta^{(L)}, \boldsymbol{s}^{(L)})$ kann ähnlich wie die Komposition $\mathcal{S}' = \mathcal{S}_\sigma \to \mathcal{S}$ konstruiert werden, wobei neben der Ausgabefunktion λ' von $\mathcal{S}'$ außerdem die Ausgabefunktion λ_σ von $\mathcal{S}_\sigma$ gegeben ist. Im folgenden sind auch die Zustandsvariablen $\boldsymbol{Q} = (Q_0, \ldots, Q_{l-1})$ von $\mathcal{S}$ gegeben, in denen sich $\mathcal{S}$ befindet, wenn dieser die Ausgaben von $\mathcal{S}_\sigma$ als Eingaben erhält. Die zugehörigen Zustandsfunktionen von diesen Q_i werden im folgenden mit δ_i' bezeichnet.

Zu Beginn der Berechnung entspricht die Ausgabe von $\mathcal{S}_L$ der Ausgabe von $\mathcal{S}_\sigma$.

In jedem Berechnungsschritt von $\mathcal{S}_\sigma \to \mathcal{S}$ wird nun getestet, ob $\mathcal{S}$ zur Eingabe $\mathcal{S}_\sigma$ eine 1 berechnet. Mit einem zusätzlichen Register mit der Zustandsvariablen N wird dieser Test protokolliert.

$$\delta_N = \mathrm{ite}(N, 1, \lambda')$$

Wenn nun das zugehörige Register den Wert 1 besitzt und noch nicht begonnen wurde, eine 1-Folge zu berechnen, dann entspricht das zukünftige Ausgabeverhalten immer dem von σ.

Wird dagegen zum Zeitpunkt t in dem Automaten $\mathcal{S}_L$ in Eingabevariable P eine 1 eingegeben, dann wird ab dem Zeitpunkt $t+1$ eine 1-Folge ausgegeben, wenn im aktuellen Berechnungsschritt t von δ_N keine 1 berechnet worden ist. Dies trifft genau dann zu, wenn in diesem Berechnungsschritt t und den vorherigen von $\mathcal{S}_\sigma \to \mathcal{S}$ keine 1 berechnet worden ist.

Ist diese Bedingung erfüllt, dann wird im nächsten Schritt begonnen, eine 1-Folge zu berechnen. In einem Register mit der Zustandsvariable π wird dabei vermerkt, daß der Automat eine 1-Folge ausgeben wird. Die Übergangsfunktion δ_π von π sieht dabei folgendermaßen aus:

$$\delta_\pi = \mathrm{ite}(\pi, 1, \mathrm{ite}(\delta_N, 0, P))$$

Um eine korrekte Berechnung der 1-Folge zu garantieren, werden die Zustandsvariablen $\boldsymbol{Q}$ von $\mathcal{S}$ "eingefroren", indem die Übergangsfunktionen δ_i' von Q_i für $\mathcal{S}_L$ folgendermaßen verändert werden.

$$\mathrm{ite}(\pi, Q_i, \delta_i')$$

In der folgenden Ausführung wird nun angenommen, daß zum Zeitpunkt t die Berechnung für die 1-Folge aktiviert wird, d.h. in dem Register zur Zustandsvariablen π wird die Berechnung der 1-Folge protokolliert.

Da die Eingabe von $\mathcal{S}$ bisher von $\mathcal{S}_\sigma$ bestimmt worden ist, befindet sich $\mathcal{S}$ in einem Zustand $\boldsymbol{Q}$, von dem aus eine 1 ausgegeben werden kann. In $\texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)})$ wird deshalb zu dem Zustand $\boldsymbol{Q}$ genau eine Eingabefolge kodiert, mit der $\mathcal{S}$ eine 1 ausgeben wird.

Ist nun im Berechnungsschritt t im Register zu der Variable π eine 1 vermerkt, dann wird ab dem Berechnungsschritt $t+1$ die kodierte Eingabefolge aus `1_Folge` ausgegeben.

Mit Hilfe von Registern zu den Zustandsvariablen $Z_0, \ldots, Z_{r-1}$ wird ab dem Berechnungsschritt $t+1$ unär gezählt. Im Berechnungsschritt $(t+1 \leq i \leq t+r)$ befindet sich im Register Z_{i-t-1} dann eine 1 und in den übrigen Z_j $(j \neq i)$ eine 0.

Da nun die Zustandsregister $\boldsymbol{Q}$ von $\mathcal{S}$ in den nächsten Berechnungsschritten $i > t$ nicht mehr verändert werden, kann mit Hilfe folgender booleschen Funktion L_i die zugehörige 1-Folge zu dem Zustand Q_i in den Berechnungsschritten $t+1, \ldots, t+1+r$ berechnet werden.

$$\begin{aligned} & L_i(\boldsymbol{Q}, Z_0, \ldots, Z_{r-1}) \\ & \quad = \exists_{\{X^{(0)}, \ldots, X^{(r-1)}\}} \left[\texttt{1\_Folge}(\boldsymbol{Q}, \boldsymbol{X}^{(0)}, \ldots, \boldsymbol{X}^{(r-1)}) \wedge ((Z_0 \wedge X_i^{(0)}) \vee, \ldots, \vee (Z_{r-1} \wedge X_i^{(r-1)})) \right] \end{aligned}$$

Das Ausgabeverhalten von $\mathcal{S}_L$ besitzt insgesamt folgendes Verhalten.

$$\lambda_i^{(L)} = \mathrm{ite}(\pi, L_i, \sigma_i^{(L)}) \quad (0 \leq i < m)$$

Der beschriebene synchrone Schaltkreis kann nun folgendermaßen konstruiert werden.

Konstruktion 8.4. Sei $\mathcal{S} = (\delta, \lambda, \boldsymbol{s})$ ein synchroner Schaltkreis vom Typ (l, m, n) mit den Eingabevariablen $\boldsymbol{X}$, Zustandsvariablen $\boldsymbol{Q}$ und Folgezustandsvariablen $\boldsymbol{Q}'$.

Sei $\mathcal{S}_\sigma = (\lambda^{(\sigma)}, \delta^{(\sigma)}, s^{(\sigma)})$ vom Typ $(l^{(\sigma)}, m, m)$ die reproduktive allgemeine parametrisierte Lösung zu dem synchronen Schaltkreis, der aus der Iteration 8.2 gewonnen wird. Dieser besitzt die gleichen Eingabevariablen $\boldsymbol{X}$ wie $\mathcal{S}$, die Ausgabevariablen $H^{(\sigma)}$ und die Zustandsvariablen $Q^{(\sigma)}$.

Weiter sei P eine zusätzliche Eingabevariable und N, π, $Z_0, \ldots, Z_{r-1}$ zusätzliche Zustandsvariablen mit den Folgezustandsvariablen N', π', $Z'_0, \ldots, Z'_{r-1}$.

Der Automat $\mathcal{S}_L = (\lambda^{(L)}, \delta^{(L)}, S^{(L)})$ vom Typ $(l + l^{(\sigma)} + 2 + r, m, m)$ zu der online Funktion τ wird konstruiert durch

- die Übergangsfunktion

$$\delta^{(L)} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l \times \mathbb{B}^{l^{(\sigma)}} \times \mathbb{B}^2 \times \mathbb{B}^r & \rightarrow \quad \mathbb{B}^m \times \mathbb{B}^l \times \mathbb{B}^{l^{(\sigma)}} \times \mathbb{B}^2 \times \mathbb{B}^r \\ (\boldsymbol{X}, \boldsymbol{Q}, Q^{(\sigma)}, \langle N, P \rangle, \langle Z_0, \ldots, Z_{r-1} \rangle) & \mapsto \quad (\boldsymbol{Q}', Q'^{(\sigma)}, \langle N', P' \rangle, \langle Z'_0, \ldots, Z'_{r-1} \rangle) \end{cases}$$

  mit

$$\begin{aligned} Q'_i &= \mathrm{ite}(\pi, Q_i, \delta_i|_{X_j = \lambda_j^{(\sigma)} (0 \leq j < m)}) \quad \forall i (0 \leq i < l) \\ Q'^{(\sigma)}_i &= \delta_i^{(\sigma)} \quad \forall i (0 \leq i < l^{(\sigma)}) \\ N' &= \mathrm{ite}(N, 1, \lambda|_{X_j = \lambda_j^{(\sigma)} (0 \leq j < m)}) \\ \pi' &= \mathrm{ite}(\pi, 1, \mathrm{ite}(N', 0, P)) \\ Z'_0 &= \mathrm{ite}(\pi, 0, 1) \\ Z'_i &= \mathrm{ite}(\pi, Z_{i-1}, 0) \quad (1 \leq i < r) \end{aligned}$$

- die Ausgabefunktion

$$\lambda^{(L)} : \begin{cases} \mathbb{B}^m \times \mathbb{B}^l \times \mathbb{B}^{l^{(\sigma)}} \times \mathbb{B}^2 \times \mathbb{B}^r & \rightarrow \quad \mathbb{B}^m \\ (\boldsymbol{X}, \boldsymbol{Q}, Q^{(\sigma)}, \langle N, P \rangle, \langle Z_0, \ldots, Z_{r-1} \rangle) & \mapsto \quad (H_0^{(\sigma)}, \ldots, H_{m-1}^{(\sigma)}) \end{cases}$$

  mit

$$H_i^{(\sigma)} = \mathrm{ite}(\pi, L_i, \lambda_i^{(\sigma)}) \quad (0 \leq i < m)$$

- und den Startzustand $s^{(L)} = S.S^{(\sigma)}.(0, 0).(1, 0 \ldots, 0)$

9 Konstruktion des Minimalautomaten

Sei im folgenden $\mathcal{S} = (\delta, \lambda, \alpha, \boldsymbol{s})$ ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ (l, m, n), der die Eingabevariablen $\boldsymbol{X}$ und Zustandsvariablen $\boldsymbol{Q}$ besitzt.

Mit den Konstruktionen aus Kapitel 6 werden synchrone Schaltkreise $\mathcal{S}$ so modifiziert, daß das Ausgabeverhalten von Zuständen vereinfacht wird. In vielen Fällen berechnen solche Zustände zu allen Eingabefolgen dieselben Ausgabefolgen.

Definition 9.1. Zwei Zustände $q_0, q_1 \in \mathbb{B}^l$ von $\mathcal{S}$ heißen äquivalent, kurz

$$q_0 \sim q_1$$

wenn für jede Eingabefolge von q_0 und q_1 ausgehend dieselbe Ausgabefolge berechnet wird.

Wegen ihrer Reflexivität, Symmetrie und Transitivität ist $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$ eine Äquivalenzrelation.

Synchrone Schaltkreise können nun derart minimalisiert werden, daß alle äquivalenten Zustände zu einem Repräsentanten zusammengefaßt werden können. Damit kann dann die Anzahl der erreichbaren Zustände reduziert werden. Eventuell vereinfachen sich dann auch die booleschen Funktionen, sodaß der synchrone Schaltkreis vereinfacht realisiert werden kann.

Diese Konstruktion, alle erreichbaren Zustände, die äquivalent sind, zu einem Repräsentanten zusammenzufassen, wird im folgenden als *Minimalisierung* bezeichnet und der erzeugte synchrone Schaltkreis als der *Minimalautomat* von $\mathcal{S}$.

Definition 9.2. Sei R die Menge der erreichbaren Zustände von $\mathcal{S}$

Ein synchroner Schaltkreis mit Ausgabeberechtigung $\mathcal{S}$ heißt *Minimalautomat* bzgl. seiner erreichbaren Zustände, kurz Minimalautomat von $\mathcal{S}$, wenn gilt

$$\forall q_0, q_1 \in R : q_0 \sim q_1 \Rightarrow q_0 = q_1$$

Es gibt nun drei Berechnungsschritte für die Minimalisierung von $\mathcal{S}$

1. Die Äquivalenzrelation $\sim$ auf der Menge der erreichbaren Zustände wird berechnet.
2. Für jede Äquivalenzklasse wird ein Repräsentant ausgewählt.
3. Die Zustandsfunktion wird derart modifiziert, daß sie nicht den Folgezustand $q \in \mathbb{B}^l$, sondern den Repräsentanten aus seiner Äquivalenzklasse berechnet.

9.1 Berechnung der äquivalenten Zustände

Das Ziel ist im folgenden, die Äquivalenzrelation $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$ mit Hilfe einer booleschen Funktion $E(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})$ mit

$$E(q_0, q_1) = 1 \Leftrightarrow q_0 \sim q_1$$

zu berechnen, wobei weitere l Hilfsvariablen $\tilde{\boldsymbol{Q}}$ benötigt werden, um die Zustandspaare zu kodieren.

Für die Berechnung der äquivalenten Zustände gibt es viele Berechnungsstrategien. Das effizienteste Verfahren von Hopcroft wird in [Bra84] ausführlich vorgestellt. Voraussetzung für dieses Verfahren sind aufwendige Mengenoperationen, wie das Aufspalten von Partitionen. Dieses Verfahren mit OBDDs umzusetzen, erscheint daher hoffnungslos. Im folgenden wird ein sehr einfaches Verfahren gewählt, das keine raffinierten Zwischenschritte für die Berechnung der Äquivalenzrelation einsetzt. Genau diese Einfachheit ermöglicht es, OBDDs effizient einzusetzen. So kann es gelingen, eine große Äquivalenzrelation $\sim\subseteq \mathbb{B}^l \times \mathbb{B}^l$ zu berechnen.

Mit Hilfe folgender Relation $\nsim_k\subseteq \mathbb{B}^l \times \mathbb{B}^l$ gelingt ein sehr einfaches Verfahren zur Berechnung der Äquivalenzrelation E.

Definition 9.3. Zwei Zustände $q_0, q_1 \in \mathbb{B}^l$ heißen *k-unterscheidbar*, kurz

$$q_0 \nsim_k q_1$$

wenn es eine Eingabefolge gibt, sodaß sich die Ausgabefolgen, die von q_0 und q_1 ausgehend berechnet werden, nach spätestens k Berechnungsschritten unterscheiden.

Die Relation $\nsim_k$ wird umgesetzt durch folgende charakteristische Funktion.

Definition 9.4. Sei $U_k : \mathbb{B}^l \times \mathbb{B}^l \rightarrow \mathbb{B}$ die charakteristische Funktion aller k-unterscheidbaren Zustände:

$$U_k(q_0, q_1) = 1 \Leftrightarrow q_0 \nsim_k q_1$$

Wenn zwei Zustände k-unterscheidbar sind, dann sind sie natürlich auch $k+1$-unterscheidbar, d.h. es gilt

$$U_k \subseteq U_{k+1}$$

Da der Zustandsraum $\mathbb{B}^l$ endlich ist, gibt es demnach ein s mit

$$U_s = U_{s+1}$$

Tritt dieser Fall ein, so sind alle U_k bereits berechnet, denn es gilt

Lemma 9.5. *Sei $U_s = U_{s+1}$. Dann folgt $U_s = U_{s+i}$ für alle $i \geq 1$.*

Beweis: Sei $U_s = U_{s+1}$. Für $i = 1$ ist nichts zu zeigen. Sei nun $i \geq 2$. Gibt es ein i mit

$$U_s = U_{s+1} = \cdots = U_{s+i-1} \subset U_{s+i}$$

dann gibt es zwei Zustände q_0, q_1, die nicht $(s+i-1)$-unterscheidbar sind, jedoch mit einer Eingabefolge $\langle I_0, \ldots, I_{s+i-1}\rangle \in \mathbb{B}^{(s+i)m}$ $(s+i)$-unterscheidbar sind. Die zwei Zustände $q_0' = \lambda(I_0, q_0)$ und $q_1' = \lambda(I_0, q_1)$ sind deshalb mit der Eingabefolge $\langle I_1, \ldots, I_{s+i-1}\rangle$ $(s+i-1)$-unterscheidbar und wegen $U_{s+i-2} = U_{s+i-1}$ auch $(s+i-2)$-unterscheidbar. Es gibt also eine Eingabefolge $\langle I_1', \ldots I_{s+i-1}'\rangle \in \mathbb{B}^{(s+i-1)m}$, mit der sich q_0', q_1' in der Ausgabe unterscheiden. Damit sind dann q_0, q_1 mit der Eingabefolge $\langle I_0, I_1', \ldots I_{s+i-1}\rangle$ $(s+i-1)$-unterscheidbar- im Widerspruch zur Annahme. □

Dieses U_k steht nun in direkter Beziehung zu der gesuchten Äquivalenzrelation $E(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})$.

Lemma 9.6. *Gilt $U_k = U_{k+1}$ so folgt $E = \neg U_k$*

Beweis: Gilt $U_k = U_{k+1}$, so folgt mit Lemma 9.5 $U_k = U_{k+i}$ für alle $i \geq 0$. Es gilt also $U_k(q_0, q_1) = 0$ genau dann, wenn es keine Eingabefolge gibt, mit der die Zustände q_0 und q_1 bzgl. ihres Ausgabeverhaltens unterscheidbar sind. Mit anderen Worten gilt

$$U_k(q_0, q_1) = 0 \Leftrightarrow q_0 \sim q_1$$

□

Berechnung der Äquivalenzrelation $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$

Im folgenden wird gezeigt, wie die Mengen U_k berechnet werden können. Dabei werden neben den Variablen $\boldsymbol{Q}$ und $\tilde{\boldsymbol{Q}}$ jeweils die Variablen $\boldsymbol{Q}'$ und $\tilde{\boldsymbol{Q}}'$ eingeführt.

Zwei Zustände q_0 und q_1 sind 1-unterscheidbar, wenn es eine Eingabe $x \in \mathbb{B}^m$ gibt, bei der sich das Ausgabeverhalten der beiden Zustände unterscheidet, d.h wenn es eine Eingabe $x \in \mathbb{B}^m$ gibt mit

$$\alpha(x, q_0) \neq \alpha(x, q_1)$$

oder

$$\alpha_i(x, q_0) = 1 \wedge \lambda_i(x, q_1) \neq \lambda_i(x, q_1)$$

für ein i mit $(0 \leq i < n)$.

U_1 kann also folgendermaßen berechnet werden.

$$U_1 = \exists_X \left[\bigvee_{0 \leq i < n} \Big((\lambda_i(\boldsymbol{X}, \boldsymbol{Q}) \oplus \lambda_i(\boldsymbol{X}, \tilde{\boldsymbol{Q}})) \wedge \alpha_i(\boldsymbol{X}, \boldsymbol{Q}) \Big) \right] \vee \exists_X \left[\bigvee_{0 \leq i < n} \Big(\alpha_i(\boldsymbol{X}, \boldsymbol{Q}) \oplus \alpha_i(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \Big) \right]$$

Die Berechnung von U_{k+1} aus U_k gelingt mit folgendem Zusammenhang.

Lemma 9.7. *Sei $k > 0$. Zwei Zustände q_0, q_1 sind genau dann $k+1$-unterscheidbar, wenn sie k-unterscheidbar sind oder es eine Eingabe $x \in \mathbb{B}^m$ gibt, sodaß die zugehörigen Folgezustände q_0', q_1' k-unterscheidbar, aber nicht $(k-1)$-unterscheidbar sind.*

Beweis: "$\Rightarrow$" Sei $q_0 \not\sim_{k+1} q_1$. Es gibt also eine Eingabe $\langle I_0, \ldots, I_k \rangle \in \mathbb{B}^{(k+1)m}$, sodaß sich nach maximal $k+1$ Berechnungsschritten die Ausgabe unterscheidet. Sind q_0, q_1 k-unterscheidbar, so ist nichts weiter zu zeigen. Seien sie also nicht k-unterscheidbar. Seien q_0', q_1' die Folgezustände von q_0, q_1 zur Eingabe $I_0 \in \mathbb{B}^m$. Von den Folgezuständen beginnend unterscheidet sich dann mit der Eingabe $\langle I_1, \ldots, I_k \rangle$ nach spätestens k-Berechnungsschritten die Ausgabe, d.h. q_0', q_1' sind k-unterscheidbar. Mit $k = 1$ sind die Zustände wegen $U_0 = \emptyset$ natürlich nicht 0-unterscheidbar. Sei also $k > 1$. Wären q_0', q_1' nun bereits mit einer Eingabefolge $\langle I_1', \ldots, I_k' \rangle \in \mathbb{B}^{(k-1)m}$ $(k-1)$-unterscheidbar, dann wären q_0, q_1 mit der Eingabefolge $\langle I_0, I_1', \ldots, I_k' \rangle$ k-unterscheidbar - im Widerspruch zu Annahme. Folglich sind q_0', q_1' nicht $(k-1)$-unterscheidbar.

"$\Leftarrow$" Umgekehrt folgt aus $q_0 \not\sim_k q_1$ auch $q_0 \not\sim_{k+1} q_1$. Sind außerdem zwei Folgezustände q_0', q_1' von q_0, q_1 k-unterscheidbar, dann gibt es eine Eingabefolge, sodaß die Vorgängerzustände q_0, q_1 nach maximal $k+1$ Berechnungen ein unterschiedliches Ausgabeverhalten besitzen. Die Zustände q_0, q_1 sind also $(k+1)$-unterscheidbar. □

Es gilt also die Beziehung

$$U_{k+1} = \texttt{neu}_{k+1} \cup U_k$$

mit

$$\texttt{neu}_{k+1} = \{(q_0, q_1) \mid \exists q_0', q_1' \in U_k \setminus U_{k-1} \exists x \in \mathbb{B}^m : q_0' = \delta(x, q_0), q_1' = \delta(x, q_1)\}$$

Wird anstelle von $U_k \setminus U_{k-1}$ die Menge A_k mit

$$U_k \setminus U_{k-1} \subseteq A_k \subseteq U_k$$

gewählt, so bleibt die Beziehung bestehen, da die Menge $\texttt{neu}_{k+1}$ nach Lemma 9.7 nur mit Tupeln aus U_k erweitert wird.

Wird A_k als boolesche Funktion in Variablen $\boldsymbol{Q}$ und $\tilde{\boldsymbol{Q}}$ beschrieben und ist $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$ die Transitionsrelation von $\mathcal{S}$, so kann U_{k+1} berechnet werden mit

$$U_{k+1}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \exists_{X \cup Q' \cup \tilde{Q}'} \left[A_k(\boldsymbol{Q}', \tilde{\boldsymbol{Q}}') \wedge T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge T(\boldsymbol{X}, \tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{Q}}') \right] \vee U_k \tag{32}$$

A_k muß derart gewählt werden, daß sich A_k als charakteristische Funktion für alle Elemente aus $\neg U_{k-1}$ wie U_k verhält. Ansonsten kann sich A_k beliebig verhalten. Nach Lemma 5.3 kann mit Hilfe der REDUCE-Operation

$$A_k = \mathrm{REDUCE}(U_k, \neg U_{k-1})$$

ein A_k mit diesen Eigenschaften bestimmt werden. Dabei berechnet die REDUCE-Operation eine boolesche Funktion, dessen OBDD möglichst klein ist. Da die REDUCE-Operation nicht immer eine bessere Lösung als das OBDD $U_k \wedge \neg U_{k-1}$ berechnet, sollte in einem Größenvergleich der OBDDs das kleinere OBDD ausgewählt werden.

Insgesamt ergibt sich nun folgender Algorithmus

Algorithmus 9.8 (Berechnen der äquivalenten Zustände).
// Input: Startmenge $S(\boldsymbol{Q})$, Transitionsrelation $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$
// Output: Äquivalenzrelation $\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})$
bdd `berechne_äquivalente_Zustände`
{
(1) $U(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \exists_X \left[\bigvee_{0 \le i < n} \left((\lambda_i(\boldsymbol{X}, \boldsymbol{Q}) \oplus \lambda_i(\boldsymbol{X}, \tilde{\boldsymbol{Q}})) \wedge \alpha_i(\boldsymbol{X}, \boldsymbol{Q}) \right) \right] \vee$
$\exists_X \left[\bigvee_{0 \le i < n} \left(\alpha_i(\boldsymbol{X}, \boldsymbol{Q}) \oplus \alpha_i(\boldsymbol{X}, \tilde{\boldsymbol{Q}}) \right) \right];$
(2) $A = U^+;$
(3) **do**
{
(4) $A(\boldsymbol{Q}', \tilde{\boldsymbol{Q}}') = A^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}', \tilde{\boldsymbol{Q}} \leftarrow \tilde{\boldsymbol{Q}}');$
(5) $A(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \exists_{X \cup Q' \cup \tilde{Q}'} \left[A^-(\boldsymbol{Q}', \tilde{\boldsymbol{Q}}') \wedge T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge T(\boldsymbol{X}, \tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{Q}}') \right];$
(6) $A = A \wedge \neg U;$
(7) **if**$(A! = \mathbf{0})$ {
(8) $U_{neu} = A \vee U;$
(9) $\texttt{reduziert}_A = \mathrm{REDUCE}(U_{neu}, (\neg U)^-);$
(10) $U^- = U_{neu};$
(11) **if**$(|\texttt{reduziert}_A| < |A|)$
(12) $A^- = \texttt{reduziert}_A;$
(13) **else** FREE($\texttt{reduziert}_A$);
}
}
(14) **while**$(A! = \mathbf{0})$
(15) **return**$(\neg U);$

}

Berechnung der Äquivalenzrelation $\sim \subseteq R \times R$

Die Äquivalenzrelation $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$ kann in vielen Fällen nur durch ein sehr großes OBDD dargestellt werden. Da in dieser Arbeit nur die erreichbaren Zustände R eines synchronen Schaltkreises betrachtet werden, reicht es jedoch aus, wenn $\sim \subseteq \mathbb{B}^l \times \mathbb{B}^l$ auf die erreichbaren Zustände eingeschränkt wird. Die Relation $\sim \subseteq R \times R$ bleibt dabei natürlich eine Äquivalenzrelation. Im folgenden sei nun

$$R^{\times}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = R(\boldsymbol{Q}) \wedge R(\tilde{\boldsymbol{Q}})$$

die charakteristische Funktion zu $R \times R$.

Es gelingt nun, mit einem OBDD die Äquivalenzrelation $\sim \subseteq R \times R$ darzustellen, wobei außerhalb des Bereiches $R^{\times}$ das OBDD so modifiziert wird, daß die Knotenanzahl möglichst klein wird.

Lemma 9.9. *Seien für $k > 0$ die U'_{k-1}, U'_k und B_k so gewählt, daß sich B_k in dem Bereich von $R \times R$ wie A_k , U'_{k-1} wie U_{k-1} und U'_k wie U_k verhalten. Dann gilt:*

Wird U'_{k+1} bestimmt, indem in Berechnung (32) A_k mit B_k, U_{k-1} mit U'_{k-1} und U_k mit U'_k ausgetauscht werden, so verhält sich U'_{k+1} in dem Bereich von $R \times R$ wie U_{k+1}.

Beweis: Im folgenden seien $Y = X \cup Q' \cup \tilde{Q}'$, $\tilde{T} = T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge T(\boldsymbol{X}, \tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{Q}}')$. Da sich nach Voraussetzung A_k und B_k in dem Bereich R gleich verhalten, gibt es ein $C : \mathbb{B}^l \times \mathbb{B}^l \to \mathbb{B}$ mit

$$B_k = (A_k \wedge R^{\times}) \vee (C \wedge \neg R^{\times})$$

und es gilt

$$\begin{aligned}
\exists_Y \left[\tilde{T} \wedge B_k\right] &= \exists_Y \left[\tilde{T} \wedge ((A_k \wedge R^{\times}) \vee (C \wedge \neg R^{\times}))\right] \\
&= \exists_Y \left[A_k \wedge R^{\times} \wedge \tilde{T}\right] \vee \underbrace{\exists_Y \left[C \wedge \neg R^{\times} \wedge \tilde{T}\right]}_{=:D(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})} \qquad (33)
\end{aligned}$$

Es gilt $D \wedge R^{\times} = \emptyset$: Wegen der Konstruktion von D gibt es für jedes Zustandspaar (q_0, q_1) aus D eine Eingabe, sodaß die Folgezustände (q'_0, q'_1) in $C \wedge \neg R^{\times}$ liegen. Wenn nun $D(q_0, q_1) = 1$ ist und $q_0, q_1 \in R$ gilt, dann gibt es also eine Eingabe, sodaß die Folgezustände q'_0, q'_1 in $C \wedge \neg R^{\times}$ liegen. Diese sind aber wegen $q_0, q_1 \in R$ von $\mathcal{S}$ erreichbar und befinden sich daher in R - im Widerspruch zur Annahme $q'_0, q'_1 \in C \wedge \neg R^{\times}$.

Es folgt also

$$\begin{aligned}
R^{\times} \wedge U'_{k+1} &= R^{\times} \wedge (\exists_Y \left[\tilde{T} \wedge B_k\right] \vee U'_k) = R^{\times} \wedge \exists_Y \left[\tilde{T} \wedge B_k\right] \vee R^{\times} \wedge U'_k \\
&= R^{\times} \wedge \exists_Y \left[\tilde{T} \wedge A_k\right] \vee R^{\times} \wedge U_k = R^{\times} \wedge (\exists_Y \left[\tilde{T} \wedge A_k\right] \vee U_k) = R^{\times} \wedge U_{k+1}
\end{aligned}$$

□

Wenn U'_{k-1}, U'_k in dem Bereich $R \times R$ mit U_k, U_{k-1} übereinstimmen und B_k die Beziehung

$$U'_{k-1} \wedge R^{\times} \subseteq B_k \wedge R^{\times} \subseteq U'_k \wedge R^{\times}$$

erfüllt, wird nach Lemma 9.9 mit

$$U'_{k+1}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \exists_{X \cup Q' \cup \tilde{Q}'} \left[B_k(\boldsymbol{Q}', \tilde{\boldsymbol{Q}}') \wedge T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \wedge T(\boldsymbol{X}, \tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{Q}}')\right] \vee U'_k$$

ein U'_{k+1} berechnet, das mit U_{k+1} im Bereich $R \times R$ übereinstimmt.

Sind nun die Mengen U'_i auf diese Art berechnet, so ist durch ein U'_k mit

$$U'_k \wedge R^{\times} = U'_{k+1} \wedge R^{\times} \tag{34}$$

auf dem Bereich $R \times R$ die Äquivalenzrelation $\sim \subseteq R \times R$ bestimmt.

Bemerkung 9.10. Mit Hilfe der REDUCE-Operation gelingt es nun, die OBDDs U'_k und B_k mit der Kenntnis von U'_{k-1} und $R^{\times}$ möglichst klein zu wählen. Dabei sollte das OBDD $R^{\times}$ von $R \times R$ nicht explizit berechnet werden, da dieses sehr viele Knoten besitzen kann.

1. B_k darf mit $C = \text{REDUCE}(U'_k, \neg U'_{k-1})$ bestimmt werden, denn in dem Bereich $\neg U'_{k-1}$ verhält sich C wie U'_k. Es gilt für $B_k = \text{REDUCE}(U'_k, \neg U'_{k-1})$ also insbesondere

$$U'_{k-1} \wedge R^{\times} \subseteq B_k \wedge R^{\times} \subseteq U'_k \wedge R^{\times}$$

2. Ist nun U'_{k+1} berechnet, so gelingt mit Hilfe der REDUCE-Operation, in vielen Fällen ein möglichst kleines OBDD

$$U''_{k+1} = \text{REDUCE}(U'_{k+1}, R^{\times})$$

zu bestimmen mit den verlangten Eigenschaften aus Lemma 9.9. Die Berechnung von $R^{\times}$ kann dabei vermieden werden, indem die Reduzierung aufgeteilt wird zu

$$C = \text{REDUCE}(U'_{k+1}, R(\boldsymbol{Q})), \quad U'''_k = \text{REDUCE}(C, R(\tilde{\boldsymbol{Q}}))$$

Dabei hat sich sogar herausgestellt, daß in vielen Fällen mit U'''_k ein kleineres OBDD als mit U''_k entsteht.

In manchen Fällen ist das OBDD R so ungünstig strukturiert, daß die Reduzierung von C nicht gelingt. Wenn dieser Fall eintritt, sollte ebenfalls die Reduzierung von C in dem Bereich $\neg R(\tilde{\boldsymbol{Q}})$ vermieden werden, da die Relation U_k symmetrisch ist und die Variablen $Q_i, \tilde{Q}_i$ benachbart angeordnet sind (siehe Bemerkung 9.12). Die Reduzierung verhält sich deswegen ähnlich ungünstig.

3. Das relationale Produkt (32) sollte getrennt berechnet werden mit

$$\begin{aligned} B'(\boldsymbol{X}, \boldsymbol{Q}, \tilde{\boldsymbol{Q}}') &= \exists_{Q'} \left[B_k(\boldsymbol{Q}', \tilde{\boldsymbol{Q}}') \wedge T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}') \right] \\ B(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) &= \exists_{X \cup \tilde{Q}'} \left[B'(\boldsymbol{X}, \boldsymbol{Q}, \tilde{\boldsymbol{Q}}') \wedge T(\boldsymbol{X}, \tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{Q}}') \right] \\ U'_{k+1}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) &= B \vee U_k \end{aligned}$$

Dadurch wird oftmals vermieden, daß die Knoten während der Berechnung die Speichergrenze überschreiten. Außerdem darf der Zwischenschritt B' in dem Bereich $\neg R^{\times}$ bzgl. der Knotenanzahl weiter reduziert werden.

Denn für die Berechnung von U'_{k+1} muß nur gewährleistet sein, daß B in dem Bereich von $R \times R$ nicht verändert wird. Es kann dabei ähnlich wie für Lemma 9.9 begründet werden, daß B und B' außerhalb des Bereichs $R^{\times}$ verändert werden dürfen, ohne das resultierende U'_{k+1} in dem Bereich von $R^{\times}$ zu verändern.

Nach der Berechnung B' darf folglich mit der Operation

$$\texttt{reduziert}_{B'} = \text{REDUCE}(B'(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}'), R(\boldsymbol{Q}))$$

der Zwischenschritt B' und mit $\texttt{reduziert}_B = \text{REDUCE}(B(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}), R(\tilde{\boldsymbol{Q}}))$ das Ergebnis B reduziert werden.

Wenn die Struktur des OBDDs $R(\boldsymbol{Q})$ die Reduzierung von B' nicht ermöglicht, hat sich meistens gezeigt, daß ebenfalls die Reduzierung von B in dem Bereich $R(\tilde{\boldsymbol{Q}})$ fehlschlägt. Eine Kontrollabfrage vermeidet dann diese meistens unrentable Berechnung.

Für die Terminierungsabfrage (34) sollte nicht die Operation $U'_k \wedge R^\times$ durchgeführt werden, da somit die Reduzierung wieder zunichte gemacht wird. Tatsächlich muß nur abgefragt werden, ob die boolesche Funktion $\mathtt{neu}_k = (U'_k \wedge \neg U'_{k-1})$ mit einem Zustandspaar aus $R \times R$ erfüllt werden kann. Mit Hilfe der OBDD-Operation

$$\text{INTERSECTS}(\mathtt{neu}_k, R(\boldsymbol{Q}), R(\tilde{\boldsymbol{Q}}))$$

gelingt nun diese Testabfrage ohne die explizite Berechnung von $R^\times$. Die Abfrage durchläuft dabei rekursiv die drei OBDDs, bis eine Lösung gefunden wird.

Insgesamt ergibt sich nun Algorithmus 9.11 zur Berechnung der Äquivalenzrelation.

Dieser Algorithmus unterscheidet sich von dem Basis-Algorithmus 9.8, indem folgende Reduktionen mit Hilfe der erreichbaren Zustände $R(\boldsymbol{Q})$ durchgeführt werden.

- In den Zeilen (3)-(6) wird Bemerkung 9.10.(2) zur Reduktion von U_0 umgesetzt.
- In den Zeilen (9)- (17) wird anschließend Bemerkung 9.10.(3) zur Reduktion von U_k während seiner Berechnung durchgeführt.
- In den Zeilen (21)-(25) wird danach Bemerkung 9.10.(1) zur Reduktion von B_k realisiert.
- Und in den Zeilen (26)- (29) schließlich wird Bemerkung 9.10.(2) zur Reduktion von U_k durchgeführt.

Algorithmus 9.11 (Berechnen der äquivalenten Zustände aus R).

// Input: Startmenge $S(\boldsymbol{Q})$, Transitionsrelation $T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q}')$ und Erreichbarkeitsmenge $R(\boldsymbol{Q})$
// Output: Äquivalenzrelation $\tilde{E}(\boldsymbol{Q},\tilde{\boldsymbol{Q}})$
bdd `berechne_äquivalente_Zustände`
{
$\tilde{R} = R(\boldsymbol{Q} \leftarrow \tilde{\boldsymbol{Q}})$; $R^{\times}(\boldsymbol{Q},\tilde{\boldsymbol{Q}}) = R \wedge \tilde{R}$;
$U(\boldsymbol{Q},\tilde{\boldsymbol{Q}}) = \exists_X \left[\bigvee_{0\le i<n}\Big((\lambda_i(\boldsymbol{X},\boldsymbol{Q}) \oplus \lambda_i(\boldsymbol{X},\tilde{\boldsymbol{Q}})) \wedge \alpha_i(\boldsymbol{X},\boldsymbol{Q})\Big)\right] \vee$
$\qquad\exists_X \left[\bigvee_{0\le i<n}\Big(\alpha_i(\boldsymbol{X},\boldsymbol{Q}) \oplus \alpha_i(\boldsymbol{X},\tilde{\boldsymbol{Q}})\Big)\right]$;
$\mathtt{reduziert}_U = \mathrm{REDUCE}(U,R)$;
if($|\mathtt{reduziert}_U| < |U|$) {
  $U^- = \mathrm{REDUCE}(\mathtt{reduziert}_U^-, \tilde{R})$;
else FREE($\mathtt{reduziert}_U$);
$B = U^+$;
do
{
  $B(\boldsymbol{Q}',\tilde{\boldsymbol{Q}}') = B^-(\boldsymbol{Q} \leftarrow \boldsymbol{Q}', \tilde{\boldsymbol{Q}} \leftarrow \tilde{\boldsymbol{Q}}')$; $B'(\boldsymbol{X},\boldsymbol{Q},\tilde{\boldsymbol{Q}}') = \exists_{Q'}\left[T(\boldsymbol{X},\boldsymbol{Q},\boldsymbol{Q}') \wedge B^-(\boldsymbol{Q}',\tilde{\boldsymbol{Q}}')\right]$;
  $\mathtt{reduziert}_{B'} = \mathrm{REDUCE}(B',R)$;
  if($|\mathtt{reduziert}_{B'}| < |B'|$) {
    FREE(B');
    $B(\boldsymbol{Q},\tilde{\boldsymbol{Q}}) = \exists_{\boldsymbol{X}\cup\tilde{Q}'}\left[T(\boldsymbol{X},\tilde{\boldsymbol{Q}},\tilde{\boldsymbol{Q}}') \wedge \mathtt{reduziert}_{B'}^-(\boldsymbol{X},\boldsymbol{Q},\tilde{\boldsymbol{Q}}')\right]$;
    $B = \mathrm{REDUCE}(B^-,\tilde{R})$;
  }**else**{
    FREE($\mathtt{reduziert}_{B'}$);
    $B(\boldsymbol{Q},\tilde{\boldsymbol{Q}}) = \exists_{\boldsymbol{X}\cup\tilde{Q}'}\left[T(\boldsymbol{X},\tilde{\boldsymbol{Q}},\tilde{\boldsymbol{Q}}') \wedge B'^-(\boldsymbol{X},\boldsymbol{Q},\tilde{\boldsymbol{Q}}')\right]$;
  }
  $B = B^- \wedge \neg U$;
  if(INTERSECTS($B, R(\boldsymbol{Q}), R(\tilde{\boldsymbol{Q}})$)$^-! = \mathbf{0}$) {
    $U_{neu} = B \vee U$;
    $\mathtt{reduziert}_B = \mathrm{REDUCE}(U_{neu}, (\neg U)^-)$;
    $U^- = U_{neu}$;
    if($|\mathtt{reduziert}_B| < |B|$)
      $B^- = \mathtt{reduziert}_B$;
    else FREE($\mathtt{reduziert}_B$);
    $\mathtt{reduziert}_U = \mathrm{REDUCE}(U, R(\boldsymbol{Q}))$;
    if($|\mathtt{reduziert}_U| < |U|$)
      $U^- = \mathrm{REDUCE}(\mathtt{reduziert}_U^-, R(\tilde{\boldsymbol{Q}}))$;
    else FREE($\mathtt{reduziert}_U$);
  } **else** $B^- = \mathbf{0}$;
}
while($B! = \mathbf{0}$)
return($\neg U$);
}

Bemerkung 9.12. Die Variablen $\boldsymbol{Q},\tilde{\boldsymbol{Q}},\boldsymbol{Q}',\tilde{\boldsymbol{Q}}'$ besitzen eine sehr starke Abhängigkeit. Deswegen sollten diese Variablen immer als Nachbarn in der Variablenordnung zusammengefaßt werden, um die OBDDs zur Berechnung der Äquivalenzrelation $\sim$ möglichst klein zu halten.

9.2 Berechnung der Repräsentanten

Ist $\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})$ die berechnete boolesche Funktion von Algorithmus 9.11, so ist

$$E(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge R(\boldsymbol{Q}) \wedge R(\tilde{\boldsymbol{Q}}) \tag{35}$$

die Äquivalenzrelation $\sim \subseteq R \times R$ der äquivalenten Zustände von $\mathcal{S}$. Ist nun $q = (q_0, \ldots, q_{l-1}) \in R$ ein erreichbarer Zustand, so ist

$$K_q(\boldsymbol{Q}) = E|_{\tilde{Q}_i = q_i (0 \leq i < l)}$$

die Äquivalenzklasse von q auf der Menge R. Das Ziel ist nun, zu jeder Klasse $K_q(\boldsymbol{Q})$ einen Repräsentanten zu berechnen.

Grundlegendes Verfahren

Dies gelingt mit folgender strikten lexikographischen Ordnung aus Beispiel 2.25 auf den Wörtern $q = (q_0, \ldots, q_{l-1}) \in \mathbb{B}^l$:

$$q > \tilde{q} \Leftrightarrow (\exists i (0 \leq i < l) \; (q_0 = \tilde{q}_0 \wedge \ldots \wedge q_{i-1} = \tilde{q}_{i-1} \wedge q_i = 1 \wedge \tilde{q}_i = 0))$$

Die boolesche Funktion O kodiert diese Ordnung $>$ mit

$$O : \begin{cases} \mathbb{B}^{2l} & \rightarrow \quad \mathbb{B} \\ (\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) & \mapsto \quad \begin{cases} 1, & \text{wenn} \quad \boldsymbol{Q} > \tilde{\boldsymbol{Q}} \\ 0, & \text{sonst} \end{cases} \end{cases}$$

Eine Menge P_M der Repräsentanten auf dem Bereich $M \subseteq R$ kann bestimmt werden mit

$$P_M = \{q \mid \neg(\exists \tilde{q} \in K_q \cap M : q > \tilde{q})\} \cap M$$

d.h. aus jeder Klasse K_q wird der kleinste Zustand aus M bzgl. der Ordnung $<$ ausgewählt. Mit Hilfe von booleschen Funktionen läßt sich dies folgendermaßen formulieren:

$$P_M(\boldsymbol{Q}) = \neg \exists_{\tilde{\boldsymbol{Q}}} \left[E(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \right] \wedge M(\boldsymbol{Q})$$

Wegen $M \subseteq R$ folgt

$$\neg \exists_{\tilde{Q}} \left[E(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \right] = \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge R(\boldsymbol{Q}) \wedge R(\tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right] =$$
$$\neg \left(\exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right] \wedge R(\boldsymbol{Q}) \right) = \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right] \vee \neg R(\boldsymbol{Q})$$

und damit kann $P_M(\boldsymbol{Q})$ auch mit

$$P_M(\boldsymbol{Q}) = \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge M(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right] \wedge M(\boldsymbol{Q}) \tag{36}$$

berechnet werden. Damit kann die Menge P_M anstelle von E mit $\tilde{E}$ berechnet werden. Die Operation (35) sollte nämlich vermieden werden, da mit der Berechnung $\tilde{E} \wedge R^{\times}$ die Reduktion des OBDDs $\tilde{E}$ aus Algorithmus 9.11 wieder rückgängig gemacht wird.

Für $M := R$ kann folglich mit Berechnung (36) die gesuchte Menge $P_R(\boldsymbol{Q})$ berechnet werden, die für jede Klasse der Äquivalenzrelation $\sim \subseteq R \times R$ genau einen Repräsentanten beinhaltet.

Verfahren mit Erreichbarkeitsberechnung

Sind Zustände äquivalent, so sind auch die Folgezustände äquivalent. Da mit Berechnung (36) diese Nachbarschaft nicht berücksichtigt wird, gelingt es oftmals nicht, benachbarte Zustände als Repräsentanten auszuwählen. Mit folgender Strategie wird nun versucht, auch die Folgezustände eines Repräsentanten als Repräsentanten für die entsprechende Klasse auszuwählen.

Die Idee besteht darin, in jeder Iteration i die Zustände des Automaten $\mathcal{S}$ in der Erreichbarkeitstiefe i zu durchlaufen. Aus diesen Zuständen werden nun Repräsentanten aus den Klassen ausgewählt, von denen in den bisherigen Berechnungsschritten noch kein Repräsentant bestimmt worden ist.

Im folgenden ist $\mathrm{Id}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})$ eine charakteristische Funktion[10] mit

$$\mathrm{Id}(q, \tilde{q}) = 1 \Leftrightarrow q_i = \tilde{q}_i \forall (0 \leq i < l)$$

Mit der Berechnung der charakteristischen Funktion

$$\text{finde\_Repräsentanten}(\boldsymbol{Q}) = \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge (\neg \mathrm{Id}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})) \wedge R(\tilde{\boldsymbol{Q}}) \right] \wedge R(\boldsymbol{Q})$$

erhält man nun alle Zustände aus $R \times R$, deren Klassen mindestens die Kardinalität 2 besitzen. Aus den Klassen mit diesen Zuständen muß nun zwischen mehreren Kandidaten ein Repräsentant bestimmt werden. Diese Repräsentanten werden im folgenden in $P(\boldsymbol{Q})$ gesammelt. Mit

$$\text{ein\_elementig}(\boldsymbol{Q}) = R(\boldsymbol{Q}) \wedge \neg \text{finde\_Repräsentanten}(\boldsymbol{Q})$$

werden außerdem alle Repräsentanten der einelementigen Klassen ermittelt. Die Vereinigung von $P(\boldsymbol{Q})$ und ein_elementig($\boldsymbol{Q}$) ergibt dann insgesamt eine Menge der Repräsentanten.

Im Verlauf der Berechnung von $P(\boldsymbol{Q})$ wird in finde_Repräsentanten($\boldsymbol{Q}$) protokolliert, zu welchen Zuständen noch ein Repräsentant benötigt wird.

In einem Iterationsdurchlauf werden nun die neu berechneten Zustände $\texttt{neu}(\boldsymbol{Q})$ getrennt je nachdem,

- ob sie einer Klasse mit mehreren Elementen angehören, zu der noch kein Repräsentant berechnet worden ist,

$$\texttt{neu}_n(\boldsymbol{Q}) = \text{finde\_Repräsentanten}(\boldsymbol{Q}) \wedge \texttt{neu}(\boldsymbol{Q})$$

- oder ob zu dem Zustand bereits ein Repräsentant berechnet worden ist, bzw. die Klasse nur einelementig ist.

$$\texttt{neu}_1(\boldsymbol{Q}) = neu(\boldsymbol{Q}) \wedge (\neg neu_n(\boldsymbol{Q}))$$

Für die Zustände $\texttt{neu}_n$ können dann wegen (36) mit

$$\texttt{neu}_P(\boldsymbol{Q}) = \neg \exists_{\tilde{\boldsymbol{Q}}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge \texttt{neu}_n(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right] \wedge \texttt{neu}_n(\boldsymbol{Q})$$

die zugehörigen Repräsentanten berechnet werden.

Mit $\exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge \texttt{neu}_P(\tilde{\boldsymbol{Q}}) \right] \wedge R(\boldsymbol{Q})$ werden dabei alle äquivalenten Zustände aus R der aktuell berechneten Repräsentanten ermittelt. Wegen finde_Repräsentanten $\subseteq R$ werden dann mit

$$\text{finde\_Repräsentanten}(\boldsymbol{Q}) = \text{finde\_Repräsentanten} \wedge \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge \texttt{neu}_P(\tilde{\boldsymbol{Q}}) \right]$$

[10]Die Größe des zugehörigen OBDDs besitzt ähnlich wie das OBDD O aus Beispiel 2.25 eine lineare Komplexität in Bezug auf die Zustandsanzahl, wenn jeweils die Zustandsvariablen Q_i und $\tilde{Q}_i$ benachbart angeordnet sind.

die verbleibenden Zustände bestimmt, zu denen noch ein Repräsentant berechnet werden muß. Mit der Berechnung

$$P = P \vee P_{neu}$$

wird schließlich die Menge der Repräsentanten vervollständigt.

Insgesamt ergibt sich nun folgender Algorithmus

Algorithmus 9.13 (Repräsentanten mit Erreichbarkeitsberechnung).
// Input: Transition $T(\boldsymbol{X}, \boldsymbol{Q}, \boldsymbol{Q}')$, erreichbare Zustände R, Äquivalenzrel. $\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \subseteq R \times R$
// Output: Repräsentantenmenge von $\tilde{E}$
bdd `berechne_transition_min`
{

$\text{finde\_Repräsentanten} = \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge (\neg \mathrm{Id}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})) \wedge R(\tilde{\boldsymbol{Q}})\right] \wedge R(\boldsymbol{Q});$

$\text{ein\_elementig} = R \wedge \neg \text{finde\_Repräsentanten};$

$\mathtt{neu}_n(\boldsymbol{Q}) = \text{finde\_Repräsentanten}(\boldsymbol{Q}) \wedge S(\boldsymbol{Q});$

$\text{finde\_Repräsentanten} = \text{finde\_Repräsentanten} \wedge \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge S(\tilde{\boldsymbol{Q}})\right];$

$\mathtt{erreicht} = S^+;\ \mathtt{neu} = S^+;\ P = S^+;$

}

$\textbf{while}(\text{finde\_Repräsentanten}! = \mathbf{0});$

$\quad \mathtt{neu}(\boldsymbol{Q}') = \exists_{Q \cup X} [\mathtt{neu}(\boldsymbol{Q}) \wedge T(\boldsymbol{Q}, \boldsymbol{Q}')];$

$\quad \mathtt{neu}(\boldsymbol{Q}) = \mathtt{neu}(\boldsymbol{Q}' \leftarrow \boldsymbol{Q});$

$\quad \mathtt{neu}(\boldsymbol{Q}) = \mathtt{neu}^- \wedge (\neg \mathtt{erreicht})^-;$

$\quad \mathtt{erreicht} = \mathtt{erreicht}^- \vee \mathtt{neu};$

$\quad \mathtt{neu}_n(\boldsymbol{Q}) = \text{finde\_Repräsentanten}(\boldsymbol{Q}) \wedge \mathtt{neu}(\boldsymbol{Q});$

$\quad \mathtt{neu}_1(\boldsymbol{Q}) = \mathtt{neu}(\boldsymbol{Q}) \wedge (\neg \mathtt{neu}_n(\boldsymbol{Q}))$

$\quad \mathtt{neu}_P(\boldsymbol{Q}) = \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge \mathtt{neu}_n(\tilde{\boldsymbol{Q}}) \wedge O(\boldsymbol{Q}, \tilde{\boldsymbol{Q}})\right] \wedge \mathtt{neu}_n(\boldsymbol{Q});$

$\quad \text{finde\_Repräsentanten} = \text{finde\_Repräsentanten} \wedge \neg \exists_{\tilde{Q}} \left[\tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge \mathtt{neu}_P(\tilde{\boldsymbol{Q}})\right];$

$\quad P = P^- \vee P_{neu};$

$\quad \mathtt{neu} = P_{neu}^- \vee \mathtt{neu}_1^-;$

}

$\text{FREE}(\text{finde\_Repräsentanten}); \text{FREE}(\mathtt{neu}); \text{FREE}(\mathtt{erreicht});$

$P = P^- \vee \text{ein\_elementig}^-;$

$\textbf{return}(P);$

}

9.3 Berechnung der Übergangsfunktionen

Im folgenden wird eine Transitionsrelation T_{min} für den Minimalautomaten von $\mathcal{S}$ berechnet. Diese kodiert auf dem Bereich $R \times R$ die Übergangsfunktionen $\tilde{\delta}_i : R \to R$ von $\mathcal{S}_{min}$.

Aus der Transitionsrelation T_{min} lassen sich dann die Übergangsfunktionen zu den Zustandsvariablen Q_i berechnen:

$$\tilde{\delta}_i = \exists_{Q_i'} \left[T_{min}(\boldsymbol{Q}, \boldsymbol{Q}') \wedge Q_i'\right] \tag{37}$$

Grundlegendes Verfahren

Mit der Menge der Repräsentanten kann nun eine neue Transitionsrelation T_{min} berechnet werden, die einen Zustand auf den Repräsentanten seines Folgezustandes abbildet. T_{min} kann bei dieser Berechnung leider nicht im partitionierten Zustand berechnet werden. Wenn der Minimalautomat also eine Transitionsrelation besitzt, die nicht in den Hauptspeicher paßt, so wird die Berechnung hier scheitern.

Mit der Berechnung

$$E'(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) = \tilde{E}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \wedge P(\boldsymbol{Q} \leftarrow \tilde{\boldsymbol{Q}})$$

erhält man eine Relation auf $R \times R$ mit $E'(q, q') = 1$ genau dann, wenn der Zustand q' der Repräsentant von q ist, d.h die Relation E beschreibt eine Funktion $e : R \to R$, die einen Zustand q auf seinen Repräsentanten q' abbildet.

Mit folgender Berechnung

$$T_{min}(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}') = \exists_{\tilde{Q}} \left[T(\boldsymbol{Q}, \boldsymbol{Q}') \wedge E'(\boldsymbol{Q} \leftarrow \boldsymbol{Q}', \tilde{\boldsymbol{Q}} \leftarrow \tilde{\boldsymbol{Q}}') \right]$$

entsteht eine Relation, die den Repräsentanten zu dem Folgezustand von q in Relation zu q setzt. Da T und E' Relationen zu Funktionen auf R sind, beschreibt natürlich auch T_{min} eine Funktion - eine Übergangsfunktion des Minimalautomaten von $\mathcal{S}$ auf den Zuständen R.

Mit Hilfe der Operation

$$S(\tilde{\boldsymbol{Q}}) := \exists_{Q} \left[S(\boldsymbol{Q}) \wedge E'(\boldsymbol{Q}, \tilde{\boldsymbol{Q}}) \right]$$

kann die Startmenge des Minimalautomaten berechnet werden, indem Zustände einer Klasse durch ihren Repräsentanten ersetzt werden.

Berechnen einer Transitionsrelation, die nur für Repräsentanten modifiziert wird

Mit der obigen Strategie berechnen die $\tilde{\delta}_i$ für einen Zustand aus R den Repräsentanten des Folgezustandes. Unnötigerweise werden die nicht benötigten Zustände aus $\neg R$ mit Operation (37) willkürlich auf andere Folgezustände abgebildet. Oftmals ist dieses Übergangsverhalten im Bereich $\neg R$ sehr verschieden gegenüber der ursprünglichen Übergangsfunktionen δ_i. Da die Variablenordnung auf die ursprünglichen OBDDs abgestimmt ist, eignet sich diese meistens nicht für die OBDDs der neuen Übergangsfunktionen $\tilde{\delta}_i$, sodaß diese in vielen Fällen größer sind als die ursprünglichen OBDDs.

Tatsächlich wird aber nur der Übergang von Repräsentanten auf Repräsentanten benötigt. Für die übrigen Zustände kann das alte Übergangsverhalten beibehalten werden. Sind P die Repräsentantenmenge auf R, T die Transitionsrelation von $\mathcal{S}$ und T_{min} die Transitionsrelation von $\mathcal{S}_{min}$, dann bleibt für

$$T'_{min} = \mathrm{ite}(P, T_{min}, T)$$

die Struktur des OBDDs sehr ähnlich und die Variablenordnung paßt besser zu den leicht veränderten OBDDs. Für deren Berechnung dürfen allerdings die OBDDs P, T und T_{min} nicht zu groß sein.

10 OBDD-Reduzierung

Ein synchroner Schaltkreis vom Typ (l, m, n) darf bzgl. seiner booleschen Funktionen verändert werden, solange die Übergangsfunktion die erreichbaren Zustände korrekt auf die Folgezustände abbildet und die Hilfsausgabefunktion und Ausgabeberechtigung für alle Eingaben in den erreichbaren Zuständen die korrekte Ausgabe berechnet. Für die Eingaben $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ kann das Verhalten der booleschen Funktionen also beliebig verändert werden.

Die booleschen Funktionen von $\mathcal{S}$ können daher in dem Bereich $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ als unspezifizierte Funktionen angesehen werden.

Definition 10.1. Eine *unvollständig spezifizierte boolesche Funktion* f ist definiert durch

$$f : \mathbb{B}^m \to \mathbb{B} \cup \{*\}$$

Bemerkung 10.2. Eine unvollständig spezifizierte boolesche Funktion $f : \mathbb{B}^m \to \mathbb{B} \cup \{*\}$ kann durch ein Tupel $(f^{\text{on}}, f^{\text{off}})$ von vollständig spezifizierten Funktionen $f^{\text{on}}, f^{\text{off}} : \mathbb{B}^m \to \mathbb{B}$ ausgedrückt werden mit

$$\begin{aligned} f^{\text{on}}(b) = 1 \quad &\Leftrightarrow \quad f(b) = 1 \\ f^{\text{off}}(b) = 1 \quad &\Leftrightarrow \quad f(b) = 0 \end{aligned}$$

f^{on} entspricht dabei genau der Eingabemenge, deren Elemente von f zu 1 abgebildet werden, d.h. f ist die on-Menge von f. Analog ist f^{off} die off-Menge von f, d.h. f^{off} sind genau die Elemente, die von f zu 0 abgebildet werden. f^{dc} beschreibt die "don't care" Eingaben mit $f^{\text{dc}} = \neg(f^{\text{on}} \vee f^{\text{off}})$, d.h. Eingaben von f, die unvollständig spezifiziert sind.

Wenn f in dem Bereich $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ beliebig verändert werden darf, so entspricht diese Menge der dc-Menge f^{dc}. Die on-Menge und off-Menge kann berechnet werden mit

$$f^{\text{on}} = f \wedge R, \quad f^{\text{off}} = \neg f \wedge R$$

Definition 10.3. Eine boolesche Funktion $\tilde{f} : \mathbb{B}^m \to \mathbb{B}$ heißt *Überdeckung* von $(f^{\text{on}}, f^{\text{off}})$, wenn gilt

$$f^{\text{on}} \subseteq \tilde{f}, \quad f^{\text{off}} \subseteq \neg\tilde{f}$$

Das Ziel besteht nun darin, zu einer unvollständig spezifizierten Funktion f eine Überdeckung $\tilde{f}$ zu finden, sodaß das zugehörige OBDD mit möglichst wenigen Knoten beschrieben wird.

Wenn die booleschen Funktionen eines synchronen Schaltkreises mit Ausgabeberechtigung mit OBDDs dargestellt werden, können die OBDDs für Eingaben $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ derart verändert werden, daß deren Graph mit einer kleineren Anzahl von Entscheidungsknoten repräsentiert werden kann.

Dies leistet der bereits vorgestellte REDUCE-Algorithmus 5.2. Dabei kann das OBDD F mit

$$F' = \text{REDUCE}(F, R)$$

so reduziert werden, daß sich F' in dem Bereich R wie F verhält. In diesem Algorithmus werden die OBDDs F und R rekursiv durchlaufen und in Abhängigkeit von R mit Hilfe der Reduktionsregeln 1 und 2 aus Abschnitt 2.3 Knoten von F zusammengefaßt.

Verschiedene Knoten mit der gleichen Variablenbeschriftung können jedoch nur zusammengefaßt werden, wenn diese den gleichen Vater-Knoten besitzen. Oft können dagegen viele Knoten mit derselben Variablenbeschriftung zu einem gemeinsamen Knoten zusammengefaßt werden. Es wird nun ein Verfahren nach [CCMS94] vorgestellt, mit dem es möglich ist, möglichst viele Knoten mit derselben Variablenbeschriftung zusammenzufassen.

Dabei können dann wesentlich bessere Ergebnisse als mit der REDUCE-Operation erzielt werden. Dieses Verfahren ist aber auch sehr viel komplexer aufgebaut und benötigt deshalb wesentlich mehr Zeit und Speicherressourcen. Für die beschriebenen Konstruktionen aus den vorherigen Kapiteln ist es deshalb vorteilhaft, mit der REDUCE-Operation zu arbeiten. Wenn jedoch eine Darstellung der Automaten mit möglichst kleinen OBDDs benötigt wird, dann eignet sich besonders gut das folgende Verfahren.

10.1 Der Reduktions-Algorithmus

Seien im folgenden F^{on} und F^{off} die OBDDs mit der Variablenordung $Y_0 < \ldots, < Y_{m-1}$ zu den booleschen Funktionen f^{on} und f^{off}.

Der wesentliche Trick besteht nun darin, $f = (f^{\mathrm{on}}, f^{\mathrm{off}})$ mit Hilfe eines OBDDs $\mathrm{ext}[F]$ darzustellen, das eine zusätzliche Variable z besitzt und eine vollständige boolesche Funktion kodiert.

$$\mathrm{ext}[F] = \mathrm{ITE}(z, F^{\mathrm{on}}, F^{\mathrm{off}})$$

Gilt $z < Y_0$ so existiert genau ein Knoten mit Variable z, bei dem der THEN-Knoten die on-Menge und der ELSE-Knoten die off-Menge von f kodiert.

Ist z nun mit $Y_i < z < Y_{i+1}$ in der Variablenordnung einsortiert, so kodieren die Knoten, die mit z beschriftet sind, unvollständig spezifizierte Funktionen f', wobei die THEN- und ELSE-Knoten die on- und off-Mengen von f' beschreiben. Soll nun eine Überdeckung von f gefunden werden, sodaß das zugehörige OBDD möglichst klein wird, so müssen zu allen Funktionen, die den Knoten mit der Variablenbeschriftung z entsprechen, Überdeckungen gefunden werden.

Damit kann die Berechnung einer Überdeckung von einem sehr großen OBDD auf die Berechnung einer Überdeckung mit mehreren kleinen OBDDs reduziert werden.

Befindet sich z nun in der Tiefe i des OBDDs $\mathrm{ext}[F]$, so werden z-Knoten zu einer Menge P zusammengefaßt, deren zugehörige unvollständig spezifizierten Funktionen $\mathcal{F}$ eine gemeinsame Überdeckung besitzen. Das OBDD zu der Überdeckung kann nun alle Knoten aus P ersetzen. Anschließend kann Variable z in der Variablenordnung von dem modifizierten OBDD in die Tiefe $i+1$ geschoben werden, indem z mit Y_{i+1} ausgetauscht wird. Hier kann dann das gleiche Verfahren wiederholt werden.

Um eine möglichst hohe Flexibilität zu erhalten, sollte dabei in Tiefe i nicht eine gemeinsame Überdeckung von den Funktionen aus $\mathcal{F}$ berechnet werden, sondern eine unvollständig spezifizierte boolesche Funktion $\tilde{f}_{\mathcal{F}}$, wobei jede Überdeckung von $\tilde{f}_{\mathcal{F}}$ dann auch eine Überdeckung von allen Funktionen aus $\mathcal{F}$ ist.

Definition 10.4. $\tilde{f} = (\tilde{f}^{\mathrm{on}}, \tilde{f}^{\mathrm{off}})$ heißt *unvollständige Überdeckung* von einer unvollständigen booleschen Funktion $f = (f^{\mathrm{on}}, f^{\mathrm{off}})$, wenn gilt

$$\tilde{f}^{\mathrm{on}} \subseteq f^{\mathrm{on}} \text{ und } \tilde{f}^{\mathrm{off}} \subseteq f^{\mathrm{off}}$$

$\tilde{f}$ heißt unvollständige Überdeckung von einer unvollständigen booleschen Funktionsmenge $\mathcal{F}$, wenn $\tilde{f}$ für alle $f \in \mathcal{F}$ eine unvollständige Überdeckung ist.

Für den Schritt in Tiefe $i+1$ können so wesentlich mehr Knoten zusammengefaßt werden, da Bereiche der on-Mengen und off-Mengen solange wie möglich erhalten bleiben. Diese Idee wird im folgenden Algorithmus genauer gefaßt

Algorithmus 10.5 (Reduziere OBDD**).**
// Input: $F = (F^{\text{on}}, F^{\text{off}})$
// Output: Überdeckung $\tilde{F}$ von F
bdd *Reduktion*$(F^{\text{on}}, F^{\text{off}})$
{

(1) $\tilde{F} = \text{ITE}(z, F^{\text{on}}, F^{\text{off}})$

(2) Erzeuge eine neue Variable z mit $z < Y_0$

(3) Berechne $Z = \{k \mid \text{Knoten } k \text{ aus } \tilde{F} \text{ ist mit Variable } z \text{ beschriftet}\}$

(4) Partitioniere $Z = P_0 + \ldots + P_p$, wobei gilt:

- Alle unvollständig spezifizierten booleschen Funktionen zu den Knoten aus einem P_i mit $(0 \leq i \leq p)$ besitzen eine gemeinsame Überdeckung.
- Die Anzahl p der Klassen ist minimal.

(5) Berechne zu jeder Klasse P_i eine unvollständige Überdeckung f_i.

(6) Kodiere jeweils das $f_i = ({f_i}^{\text{on}}, {f_i}^{\text{off}})$ für ein P_i mit dem OBDD $F_i = \text{ITE}(z, {F_i}^{\text{on}}, {F_i}^{\text{off}})$.

(7) Leite alle Kanten in $\tilde{F}$, die zu einem Knoten aus P_i führen, um zu dem OBDD F_i.

(8) Sei $Y_i < z < Y_{i+1}$. Vertausche die Variablen z und Y_{i+1} in der Ordnung.

(9) Gehe zu Schritt 3, bis die Variablenordnung $Y_{m-1} < z$ vorliegt.

(10) **return**$(F|_{z=1})$;

}

Dieses Verfahren ist korrekt. In jedem Durchlauf wird die unvollständige boolesche Funktion $\tilde{f}$ des OBDDs $\tilde{F}$ näher an eine Überdeckung von f herangeführt. Dabei wird im Schritt (4) und (5) garantiert, daß alle Überdeckungsmöglichkeiten von $\tilde{f}$ nach einem Iterationsschritt auch eine Überdeckungsmöglichkeit von dem $\tilde{f}$ vor dem Iterationsschritt ist. Indem die Variablen Y_{i+1}, z in Schritt 8 vertauscht werden, wird dieselbe unvollständig spezifizierte boolesche Funktion beschrieben und daher die Kodierung der on-Mengen und off-Mengen in dem OBDD $\tilde{F}$ nicht verändert.

Sei $\tilde{F}$ für $Y_{m-1} < z$ das berechnete OBDD. Jede Überdeckung der kodierten unspezifizierten booleschen Funktion $\tilde{f}$ von $\tilde{F}$ ist nun eine Überdeckung von der usprünglichen unvollständig spezifizerten Funktion f. Die Knoten mit der Variablenbeschriftung z kodieren nun die verbleibenden unvollständig spezifizierten Teilfunktionen von einer Überdeckung zu f. Da der THEN-Knoten nur der **1**- oder **0**-Knoten sein kann, werden von den z-Knoten bereits vollständig spezifizierte Funktionen kodiert: die 1 oder 0 Funktionen. Mit $\tilde{F}|_{z=1}$ wird dann jede Kante, die zu einem z-Knoten hinführt, entsprechend auf den **1**- oder **0**-Knoten umgeleitet. Damit liegt dann eine Überdeckung von f vor.

Im folgenden werden noch die Berechnungsschritte (4) und (5) genauer ausgeführt.

10.1.1 Heuristisches Verfahren für das minimale Clique-Überdeckungsproblem

Z beinhaltet alle OBDD-Knoten mit der Variablenbeschriftung z, deren THEN-Knoten die on-Mengen und ELSE-Knoten die off-Mengen der unvollständig spezifizierten booleschen Funktionen kodieren.

Auf der Menge Z wird nun im Berechnungsschritt (4) eine Partition $Z = P_0 + P_1 + \cdots + P_p$ gesucht, sodaß die zugehörigen unvollständig spezifizierten booleschen Funktionen aus P_i eine gemeinsame Überdeckung besitzen und p minimal ist.

Dieses Problem kann umformuliert werden, indem folgender ungerichtete Graph G eingeführt wird mit

- G-Knoten[11] V für die OBDD-Menge Z und
- ungerichtete Kanten $E \subseteq V \times V$. Es gilt $(k_i, k_j) \in E$ genau dann, wenn die zugehörigen unvollständig spezifizierten Funktionen f_i, f_j eine gemeinsame Überdeckung besitzen.

Dieser Graph wird im folgenden mit Hilfe von booleschen Funktionen kodiert. Jeder OBDD-Knoten aus Z muß dabei mit einer injektiven Funktion nach $\mathbb{B}^r$ abgebildet werden. Damit kann eine charakteristische Funktion $V : \mathbb{B}^r \to \mathbb{B}$ für die Knotenmenge $V \subseteq \mathbb{B}^r$ erzeugt werden.

Diese Abbildung wird direkt aus dem Knoten-Zeiger des zugehörigen OBDDs aus Z gewonnen. Die boolesche Funktion $V : \mathbb{B}^{32} \to \mathbb{B}$ besitzt dabei die Abbildung

$$V(p) = 1 \Leftrightarrow p \text{ ist ein OBDD-Zeiger aus } Z$$

Diese Kodierung wurde gewählt, da sie sehr leicht zu implementieren ist und die Dekodierung[12] von einem Knoten aus V zu dem zugehörigen OBDD aus Z sehr einfach ist. Außerdem liegen die OBDD-Zeiger nicht in dem ganzen Bereich von $\mathbb{B}^{32}$, sondern befinden sich in der Regel in mehreren zusammengehörigen Teilbereichen. Dies ist jedoch genau die Stärke von OBDDs: Werden Bits aus dem Zeiger nicht verwendet, so werden auch die zugehörigen Variablen des OBDDs $V(\boldsymbol{X})$ in der reduzierten Form eliminiert. Im folgenden wird angenommen, daß n Variablen $X = \{X_0, \ldots, X_{n-1}\}$ mit $\boldsymbol{X} = \langle X_0, \ldots, X_n \rangle$ benötigt werden.

Zu den Knoten V kann nun die Menge der Kanten $E : \mathbb{B}^n \times \mathbb{B}^n \to \mathbb{B}$ berechnet werden, wobei

$$E(b_0, b_1) = 1$$

genau dann gilt, wenn die OBDDs mit den Zeigern b_0, b_1 aus Z unvollständig spezifizierte boolesche Funktionen kodieren, die eine gemeinsame Überdeckung besitzen. Diese Kantenmenge kann nun mit Hilfe folgenden Lemmas berechnet werden.

Lemma 10.6. *Seien $\mathcal{F} = \{f_0, \ldots, f_r\} (r > 0)$ unvollständig spezifizierte boolesche Funktionen mit $f_i = ({f_i}^{\text{on}}, {f_i}^{\text{off}})$.*

Es gibt eine gemeinsame Überdeckung $\tilde{f}$ von den unvollständig spezifizierten Funktionen aus $\mathcal{F}$ genau dann, wenn gilt

$$\forall f_i, f_j \in \mathcal{F} : ({f_i}^{\text{on}} \wedge {f_j}^{\text{off}} = 0) \wedge ({f_j}^{\text{on}} \wedge {f_i}^{\text{off}} = 0)$$

Für die boolesche Funktion $E(\boldsymbol{X}, \tilde{\boldsymbol{X}})$ werden zusätzlich die n Variablen $\tilde{\boldsymbol{X}}$ eingeführt.

Seien $F \subseteq V$ und $\mathcal{F}$ die zugehörigen unvollständigen booleschen Funktionen. Eine Überdeckung kann zu $\mathcal{F}$ berechnet werden genau dann, wenn gilt:

$$\forall k_0, k_1 \in F : E(k_0, k_1) = 1$$

Eine Knotenmenge $F \subseteq V$ mit dieser Eigenschaft wird *Clique* genannt.

Wird eine Partition $Z = P_0 + P_1 + \cdots + P_p$ mit minimalem p gesucht, so entspricht dies der Suche einer minimalen Anzahl von Cliquen, wobei diese eine Partition auf der Knotenmenge V bilden.

[11] Im folgenden muß zwischen den Graph Knoten (G-Knoten) und den OBDD-Knoten unterschieden werden.

[12] Es muß nur berücksichtigt werden, daß sich die Variablenordnung mit Hilfe von heuristischen Verfahren verändern kann.

Dieses Problem wird als *minimales Clique-Überdeckungsproblem* für einem Graphen G bezeichnet. Die Berechnung solcher überdeckenden Cliquen mit minimaler Anzahl ist leider NP-vollständig.

Mit folgender heuristischen Berechnung kann dieses Problem auf die Berechnung von maximalen Cliquen verlagert werden.

1. Finde Clique $C : \mathbb{B}^n \to \mathbb{B}$ mit maximaler Knotenanzahl auf Graphen $G = (V, E)$
2. Streiche alle Knoten, die in Clique C vorkommen
$$V(\boldsymbol{X}) := V(\boldsymbol{X}) \wedge \neg C(\boldsymbol{X})$$
3. Streiche alle Kanten, von denen ein Knoten in C liegt
$$E(\boldsymbol{X}, \tilde{\boldsymbol{X}}) := E(\boldsymbol{X}, \tilde{\boldsymbol{X}}) \wedge \neg C(\boldsymbol{X}) \wedge \neg C(\tilde{\boldsymbol{X}})$$
4. Wiederhole diese Schritte bis $E = 0$ gilt.

Indem jeweils eine maximale Clique C_i im Iterationsschritt i für ein P_i ausgewählt wird, gelingt es in vielen Fällen, die Knotenmenge V mit so wenigen Cliquen wie möglich zu überdecken.

Es wird nun ein sehr einfaches rekursives Verfahren 10.7 in [CPR95] vorgestellt, mit dem eine maximale Clique $C(\boldsymbol{X})$ aus dem Graphen $V(\boldsymbol{X}), E(\boldsymbol{X}, \tilde{\boldsymbol{X}})$ berechnet werden kann. Wegen seiner Einfachheit können die verwendeten Mengen mit Hilfe von OBDD-Operationen effizient berechnet werden.

Für die Berechnung der maximalen Clique werden Kanten mit $E(b, b) = 1$ entfernt durch die Operation
$$E := E(\boldsymbol{X}, \tilde{\boldsymbol{X}}) \wedge \neg \mathrm{Id}(\boldsymbol{X}, \tilde{\boldsymbol{X}})$$

In `maxClique`$(\boldsymbol{X})$ wird die größte bisher berechnete Clique protokolliert. In `maxClique_Anz` wird dabei die Anzahl der in `maxClique` kodierten G-Knoten gespeichert, d.h. es gilt $\|\texttt{maxClique}\| = \texttt{maxClique\_Anz}$.

In der Rekursionstiefe i von Algorithmus 10.7 ist in `clique`$(\boldsymbol{X})$ eine Clique mit i G-Knoten aus $V(\boldsymbol{X})$ berechnet worden.

In $\tilde{V}(\boldsymbol{X})$ sind die G-Knoten außerhalb der Clique enthalten, die alle `clique`$(\boldsymbol{X})$ zu einer Clique der Größe $i + 1$ erweitern können.

Im Rekursionsschritt i werden nun alle G-Knoten aus $\tilde{V}(\boldsymbol{X})$ durchlaufen. Sei also im folgenden `neu`$(\boldsymbol{X})$ ein ausgewählter G-Knoten aus $\tilde{V}(\boldsymbol{X})$.

Aus `neu`$(\boldsymbol{X})$ und `clique`$(\boldsymbol{X})$ kann nun eine neue Clique
$$\texttt{clique}_{neu}(\boldsymbol{X}) = \texttt{clique}(\boldsymbol{X}) \vee \texttt{neu}(\boldsymbol{X})$$
mit der G-Knotenanzahl $i + 1$ berechnet werden.

Zu dem G-Knoten `neu`$(\boldsymbol{X})$ wird die Menge aller verbleibenden G-Knoten berechnet, die alle mit `neu`$(\boldsymbol{X})$ verbunden sind und damit jeweils mit $\texttt{clique}_{neu}(\boldsymbol{X})$ eine Clique der Größe $i + 2$ bilden können.
$$\texttt{Knoten\_verbunden}(\tilde{\boldsymbol{X}}) = \exists_X \left[\texttt{neu}(\boldsymbol{X}) \wedge E(\boldsymbol{X}, \tilde{\boldsymbol{X}})\right] \wedge \tilde{V}(\tilde{\boldsymbol{X}})$$

Wenn nun $\|\texttt{Knoten\_verbunden}\| + \|\texttt{clique}\| + 1 = \|\texttt{Knoten\_verbunden}\| + i + 1$ nicht größer ist als `maxClique_Anz`, so kann dieser Rekursionsschritt beendet werden, da die Größe `maxClique_Anz` nicht mehr verbessert werden kann.

Wenn nun `Knoten_verbunden` $= \mathbf{0}$ gilt, so kann die berechnete Clique der Größe $i+1$ nicht weiter vergrößert werden. Da $i+1 >$ `maxClique_Anz` gilt, ist damit die bisher größte Clique berechnet worden. Nachdem `maxClique` und `maxClique_Anz` aktualisiert worden sind, wird der Rekursionsschritt beendet.

Ansonsten wird mit $\texttt{berechne\_MaxClique}(\texttt{Knoten\_verbunden}(\boldsymbol{X}), \texttt{clique}_{neu}, \texttt{tiefe}+1)$ die Rekursion fortgesetzt.

Insgesamt ergibt sich nun folgender Algorithmus, der mit $\texttt{berechne\_MaxClique}(V(\boldsymbol{X}), \mathbf{0}, 0)$ aufgerufen wird.

Algorithmus 10.7 (Bestimme maximale Clique).

// Input: G-Knoten $\tilde{V}(\boldsymbol{X})$, Kanten $E(\boldsymbol{X}, \tilde{\boldsymbol{X}})$, $\texttt{clique}(\boldsymbol{X})$, Rekursionstiefe $\texttt{tiefe}{=}0$
extern: $\texttt{maxClique} = \mathbf{0}$, $\texttt{maxClique\_Anz} = 0$
// Output: - extern: maximale Clique `maxClique`, Größe `maxClique_Anz`
$\texttt{berechne\_MaxClique}(\text{bdd } \tilde{V}, \text{bdd } \texttt{clique}, \text{int } \texttt{tiefe})$
{
(1) **if**$(\tilde{V} = \mathbf{0})$ {
(2) $\texttt{maxClique\_Anz} = \texttt{tiefe}$; $\texttt{maxClique}^- = \texttt{clique}^+$;
(3) **return**;
}
(4) $\tilde{V}_{rest}(\boldsymbol{X}) = \tilde{V}^+(\boldsymbol{X})$;
(5) **while**$(\tilde{V}_{rest} != \mathbf{0})$ {
(6) $\texttt{neu}(\boldsymbol{X}) = \texttt{hole\_Element}(\tilde{V}_{rest}, \boldsymbol{X})$;
(7) $\texttt{Knoten\_verbunden}(\tilde{\boldsymbol{X}}) = \exists_X \left[\texttt{neu}(\boldsymbol{X}) \wedge E(\boldsymbol{X}, \tilde{\boldsymbol{X}})\right]$;
(8) $\texttt{Knoten\_verbunden}(\boldsymbol{X}) = \texttt{Knoten\_verbunden}^-(\tilde{\boldsymbol{X}} \leftarrow \boldsymbol{X})$;
(9) $\texttt{Knoten\_verbunden}(\boldsymbol{X}) = \texttt{Knoten\_verbunden}^- \wedge \tilde{V}$;
(10) **if**$(\texttt{tiefe} + 1 + \|\texttt{Knoten\_verbunden}\| > \texttt{maxClique\_Anz})$ {
(11) $\texttt{clique}_{neu} = \texttt{clique} \vee \texttt{neu}$;
(12) $\texttt{berechne\_MaxClique}(\texttt{Knoten\_verbunden}, \texttt{clique}_{neu}, \texttt{tiefe}+1)$;
(13) $\text{FREE}(\texttt{clique}_{neu})$;
}
(14) $\tilde{V}_{rest}(\boldsymbol{X}) = \tilde{V}_{rest}^- \wedge (\neg(\texttt{neu}^-))^-$;
(15) $\text{FREE}(\texttt{Knoten\_verbunden})$;
}

}

Das Berechnen einer maximalen Clique allerdings ist wie das minimale Clique-Überdeckungsproblem NP-vollständig. Der eben vorgestellte Algorithmus muß im ungünstigsten Fall $\|V\|!$ Rekursionen durchlaufen.

Allerdings gelingt es mit einer einfachen Umformung des Algorithmus, ein heuristisches Verfahren zu erzeugen, das ausreichend gute Ergebnisse für das heuristische Verfahren des minimalen Clique-Überdeckungsproblems liefert und im ungünstigsten Fall eine Komplexität

$$O\left(\sum_{i=0}^{|V|} i\right) = O\left(\frac{|V| \cdot (|V|+1)}{2}\right)$$

besitzt.

Die Rekursion wird dort nicht mehr in der while-Schleife durchgeführt, sondern erst nach ihrer Beendigung. In der while-Schleife wird nun der G-Knoten $\texttt{neu}(\boldsymbol{X})$ in $\tilde{V}(\boldsymbol{X})$ gesucht, sodaß die zugehörige Menge $\texttt{Knoten\_verbunden}(\boldsymbol{X})$ maximal wird. Die Wahrscheinlichkeit ist hier besonders hoch, daß aus den G-Knoten $\texttt{Knoten\_verbunden}(\boldsymbol{X})$, $\texttt{neu}(\boldsymbol{X})$ und $\texttt{clique}(\boldsymbol{X})$ eine sehr große Clique erzeugt werden kann. Nach Beendigung der while-Schleife wird dann die Rekursion mit

$$\texttt{berechne\_MaxClique}(\texttt{Knoten\_verbunden}_{max}(\boldsymbol{X}), \texttt{clique}_{neu}, \texttt{tiefe} + 1)$$

aufgerufen.

Insgesamt ergibt sich nun folgender heuristischer Algorithmus

Algorithmus 10.8 (Bestimme möglichst große Clique).

// Input: G-Knoten $\tilde{V}(\boldsymbol{X})$, Kanten $E(\boldsymbol{X}, \tilde{\boldsymbol{X}})$, $\texttt{clique}(\boldsymbol{X})$, Rekursionstiefe $\texttt{tiefe}{=}0$
// Output: - extern: maximale Clique $\texttt{maxClique}$, Größe $\texttt{maxClique\_Anz}$
$\texttt{heuristisch\_MaxClique}(\text{bdd } \tilde{V}, \text{bdd } \texttt{clique}, \text{int } \texttt{tiefe})$
{
(1) **if**($\tilde{V} = \mathbf{0}$) {
(2) $\texttt{maxClique\_Anz} = \texttt{tiefe}$; $\texttt{maxClique} = \texttt{clique}$;
(3) **return**;
}
(4) $\tilde{V}_{rest}(\boldsymbol{X}) = \tilde{V}(\boldsymbol{X})^{+}$;
(5) $\texttt{Knoten\_verbunden}_{max} = \mathbf{0}$; $\texttt{neu}_{max} = \mathbf{0}$; $\texttt{Anz\_max} = -1$;
(6) **while**($\tilde{V}_{rest} != \mathbf{0}$) {
(7) $\texttt{neu}(\boldsymbol{X}) = \texttt{hole\_Element}(\tilde{V}_{rest}, \boldsymbol{X})$;
(8) $\texttt{Knoten\_verbunden}(\tilde{\boldsymbol{X}}) = \exists_X \left[\texttt{neu}(\boldsymbol{X}) \wedge E(\boldsymbol{X}, \tilde{\boldsymbol{X}})\right]$;
(9) $\texttt{Knoten\_verbunden}(\boldsymbol{X}) = \texttt{Knoten\_verbunden}(\tilde{\boldsymbol{X}} \leftarrow \boldsymbol{X})^{-}$;
(10) $\texttt{Knoten\_verbunden}(\boldsymbol{X}) = \texttt{Knoten\_verbunden}^{-} \wedge \tilde{V}$;
(11) **if**($\|\texttt{Knoten\_verbunden}\| > \texttt{Anz\_max}$) {
(12) $\texttt{neu}^{-}_{max} = \texttt{neu}^{+}$;
(13) $\texttt{Anz\_max} = \|\texttt{Knoten\_verbunden}\|$;
(14) $\texttt{Knoten\_verbunden}^{-}_{max} = \texttt{Knoten\_verbunden}^{+}$;
}
(15) $\tilde{V}_{rest} = \tilde{V}^{-}_{rest} \wedge (\neg\texttt{neu}^{-})^{-}$;
(16) FREE($\texttt{Knoten\_verbunden}$);
}
(17) $\texttt{clique} = \texttt{clique}^{-} \vee \texttt{neu}_{max}$;
(18) FREE($\tilde{V}$);
(19) $\texttt{berechne\_MaxClique}(\texttt{Knoten\_verbunden}_{max}, \texttt{clique}, \texttt{tiefe} + 1)$;

}

10.1.2 Berechnung einer optimalen unvollständigen Überdeckung

Sei im Berechnungsschritt (5) von Algorithmus 10.5 nun $\mathcal{F}$ die Menge der unvollständig spezifizierten Funktionen zu den OBDDs aus P_i, die gemeinsam von einer booleschen Funktion überdeckt werden können.

Gesucht wird eine unvollständige Überdeckung $\tilde{f}$ von $\mathcal{F}$. Wünschenswert dabei ist, daß $\tilde{f}$ nur soweit genauer spezifiziert wird, daß $\tilde{f}$ gerade eine unvollständige Überdeckung von $\mathcal{F}$ wird.

Gilt $\mathcal{F} = \{(f_0{}^{\text{on}}, f_0{}^{\text{off}}), \ldots, (f_t{}^{\text{on}}, f_t{}^{\text{off}})\}$ so wird mit

$$\tilde{f}^{\text{on}} = \bigvee_{j=0}^{t} f_j{}^{\text{on}}, \quad \tilde{f}^{\text{off}} = \bigvee_{j=0}^{t} f_j{}^{\text{off}}$$

eine unspezifizierte Überdeckung $\tilde{f}$ von $\mathcal{F}$ berechnet.

Außerdem muß für jede Überdeckung $g = (g^{\text{on}}, g^{\text{off}})$ von $\mathcal{F}$ gelten

$$\tilde{f}^{\text{on}} \subseteq g \quad \tilde{f}^{\text{off}} \subseteq \neg g$$

d.h. g ist eine Überdeckung von $\tilde{f}$. Folglich ist jede Überdeckung von $\tilde{f}$ auch eine Überdeckung von $\mathcal{F}$.

Mit $\tilde{f}$ besteht also der höchst mögliche Freiheitsgrad, um im Algorithmus 10.5 in einer späteren Iteration möglichst viele Knoten zusammenzufassen.

10.2 Reduktion eines booleschen Funktionsvektors

Zu einer unvollständig spezifizierten booleschen Funktion $f : \mathbb{B}^m \to \mathbb{B} \cup \{*\}$ kann mit Hilfe des obigen Reduktionsalgorithmus das zugehörige OBDD in vielen Fällen stark verkleinert werden.

Wenn das OBDD eines unvollständig spezifizierten booleschen Funktionsvektors

$$f : \mathbb{B}^m \to (\mathbb{B} \cup \{*\})^n$$

reduziert werden soll, ist es jedoch ungeschickt, jedes OBDD der unvollständig spezifizierten Funktionen $f_i : \mathbb{B}^m \to \mathbb{B} \cup \{*\}$ einzeln zu reduzieren. Oftmals besitzen die OBDDs gemeinsame Knoten, die in den einzelnen Berechnungen unterschiedlich verändert werden. So sind dann zwar die einzelnen OBDDs relativ klein, die Summe der Knoten aller OBDDs kann dann jedoch größer sein als bei den ursprünglichen OBDDs, da viele gemeinsame Knoten verloren gegangen sind.

Die OBDDs können nun so modifiziert werden, daß nicht einzelne OBDDs möglichst wenige Knoten besitzen, sondern daß die gesamte Knotenanzahl aller OBDDs möglichst gering wird. Dies wird erreicht, indem die einzelnen OBDDs so modifiziert werden, daß möglichst viele Knoten von den verschiedenen OBDDs zu einem Knoten zusammengefaßt werden können.

Liegen nun die OBDDs ${F_i}^{\text{on}}, {F_i}^{\text{off}}$ zu den unvollständig spezifizierten booleschen Funktionen $f_i = ({f_i}^{\text{on}}, {f_i}^{\text{off}})$ vor, so können diese mit den OBDDs

$$\tilde{F}_i = \text{ITE}(z, {F_i}^{\text{on}}, {F_i}^{\text{off}})$$

kodiert werden. Anschließend kann die Menge

$$Z = \{\tilde{F}_0, \ldots, \tilde{F}_{n-1}\}$$

erzeugt werden.

Im folgenden wird angenommen, daß Z die Knoten eines gesamten OBDDs sind, die mit der Variable z beschriftet sind. Der Reduktions-Algorithmus 10.5 wird nun ab Zeile 4 mit der Menge Z gestartet. In Zeile 7 werden nun anstelle des OBDDs $\tilde{F}$ die OBDDs $\tilde{F}_0, \ldots, \tilde{F}_{n-1}$ entsprechend modifiziert.

Insgesamt erhält man so OBDDs, die zusammen möglichst wenige Knoten benötigen.

Mit der dc-Menge $\mathbb{B}^m \times (\mathbb{B}^l \setminus R)$ können die OBDDs $\delta_0, \ldots, \delta_l$, $\lambda_0, \ldots, \lambda_{n-1}$, $\alpha_0, \ldots, \alpha_{n-1}$ eines synchronen Schaltkreises mit Ausgabeberechtigung nun mit einer möglichst kleinen Gesamtanzahl von Knoten reduziert werden.

Dieses Gesamt-OBDD mit mehreren Wurzelknoten kann nun in booleschen Formeln übersetzt werden, in denen möglichst viele Teilformeln gemeinsam verwendet werden. Aus den konstruierten Automaten können so eventuell kompakte boolesche Formeln berechnet werden, die dann direkt mit Hardwarebausteinen realisiert werden können.

11 Implementierung

Der Kern der Implementierung ist eine Bibliothek “libaut.a”, mit der die oben beschriebenen Automatenkonstruktionen ausgeführt werden können. Die Implementierung ist mit dem GNU-C++ Compiler umgesetzt worden. Mit Hilfe der OBDD-Bibliothek “libbdd.a” von [Lon93] werden dabei die booleschen Funktionen der synchronen Schaltkreise mit OBDDs umgesetzt. Im folgenden wird der grundsätzliche Aufbau der Implementierung der Automaten-Bibliothek libaut.a vorgestellt.

11.1 Aufbau der Automaten-Bibliothek

Die Automatenbibliothek setzt sich aus drei Teilen zusammen.

11.1.1 OBDD-Verwaltung

Die OBDD-Verwaltung stellt die Operationen zur Verfügung, mit denen die OBDDs verwaltet und bearbeitet werden können.

Indem ein Objekt BDD der Klasse bdd_Paket durch

bdd_Paket BDD(int max_Knoten_Anzahl);

erzeugt wird, wird ein OBDD-Manager von der OBDD-Bibliothek [Lon93] initialisiert, mit dem maximal max_Knoten_Anzahl Knoten verwaltet werden können. Nähert sich die Knotenanzahl an einen gewissen Grenzwert, so wird mit dem Verfahren “garbage collecting” versucht, nicht mehr benötigte Knoten zu entfernen (siehe Abschnitt 2.6.4). Reicht diese Knotenelimination nicht aus, so wird mit einer Neuordnung der Variablen versucht, die OBDDs zu verkleinern. Mit dem Befehl

BDD.NeuOrdnung(*Ordnungsstrategie*)

wird dabei festgelegt, nach welcher Ordnungsstrategie die Variablen neu geordnet werden. Dabei kann zwischen den Ordnungsstrategien “bdd_reorder_stable_window3” und “bdd_reorder_sift” gewählt werden (siehe 2.6.5). Die Neuordnung sollte aber nur dann aktiviert werden, wenn die Knotenanzahl den maximalen Grenzwert überschreitet, da die Neuordnung sehr viel Zeit in Anspruch nimmt. Mit dem Befehl “BDD.Keine_NeuOrdnung()” wird der Neuordnungsmechanismus außer Kraft gesetzt. Dies ist auch die Grundeinstellung, wenn das OBDD-Paket mit “bdd_Paket BDD” initialisiert wird.

Für das OBDD-Paket von [Lon93] wird ein Zeiger bddm auf seinen BDD-Manager benötigt, um OBDD-Operationen auszuführen. Mit diesem Zeiger bddm, können mit den OBDDs f, g, h z.B. die Operationen

$$\begin{aligned}&\text{bdd\_ite}(\text{bdd\_manager bddm}, \text{bdd } f, \text{bdd } g, \text{bdd } h)\\&\text{bdd\_free}(\text{bdd\_manager bddm}, \text{bdd } f)\\&\text{bdd\_not}(\text{bdd\_manager bddm}, \text{bdd } f)\end{aligned}$$

ausgeführt werden. Dieser Zeiger bddm kann mit dem Befehl

bddm=BDD.bekomme_bdd_manager()

initialisiert werden. Damit kann der Anwender dann OBDDs bearbeiten und modifizieren.

Mit

BDD.Statistik();

werden statistische Informationen über die OBDD-Verwaltung ausgegeben.

11.1.2 Variablen-Verwaltung

Die Variablenverwaltung stellt den Automaten verschiedene Typen von Variablen zur Verfügung. Dabei wird grundsätzlich zwischen Eingabevariablen und Zustands-/Folgezustandsvariablen unterschieden. Nach Abschnitt 4.2 werden dabei die Eingabevariablen vor den Zustands- und Folgezustandsvariablen in der Variablenordnung eingefügt. Die Zustands- und jeweiligen Folgezustandsvariablen werden außerdem benachbart angeordnet.

Mit dem Befehl

$$\text{VarManager VarM(bdd\_Paket BDD}, \text{int max\_Variablen\_Anz});$$

wird nun ein Objekt VarM der Klasse VarManager erzeugt, das maximal max_Variablen_Anz viele Variablen verwalten kann. In dem OBDD-Paket von [Lon93] wird dabei eine Variable X als ein OBDD ITE$(X, \mathbf{1}, \mathbf{0})$ interpretiert.

Der Anwender kann sich nun mit

VarM.hole_EingabeVar(int Anzahl, bdd *Var),   VarM.hole_ZustandsVar(int Anzahl, bdd *Var)

in ein Array “bdd Var[Anzahl]” die entsprechende Anzahl von Eingabe- und Zustandsvariablen ausgeben lassen. Mit diesen Variablen können nun die OBDDs für die booleschen Funktionen eines synchronen Schaltkreises erzeugt werden. Auf die Folgezustandsvariablen kann dabei nur intern für die Transitionsrelation zugegriffen werden.

Die Verwaltung der Variablen ist dabei so organisiert, daß möglichst viele Variablen von verschiedenen Automaten gemeinsam genutzt werden. Damit wird die Auslastung der OBDDs erhöht, wenn ein OBDD-Knoten (eine boolesche Unterfunktion) von verschiedenen Automaten gemeinsam genutzt wird (siehe Bemerkung 2.15).

Für die Zustands- und Folgezustandsvariablen Q und Q' existieren zusätzlich die Hilfsvariablen $\tilde{Q}$ und $\tilde{Q}'$. Mit diesen gelingt es dann, z.B. die Minimalisierung eines Automaten zu berechnen. Diese Hilfsvariablen $\tilde{Q}, \tilde{Q}'$ werden dabei nach Bemerkung 9.12 benachbart zu den Variablen Q, Q' angeordnet. Insgesamt werden für jede Zustandsvariable eines Automaten also drei weitere Variablen benötigt.

Für die Konstruktionen werden oftmals zusätzliche Zustandsvariablen benötigt. Dabei ist es von Vorteil, wenn diese als erstes in die Variablenordnung eingefügt werden (siehe 6.23,6.38). Um dies zu ermöglichen, werden zusätzliche Konstruktionsvariablen eingeführt, die immer vor den Zustands- und Eingabevariablen in der Variablenordnung angeordnet werden. Ansonsten besitzen diese die gleiche Umsetzung wie die herkömmlichen Zustandsvariablen, d.h. für jede Zustandsvariable werden drei zusätzliche Variablen benötigt.

Wird mit dem Befehl “BDD.NeuOrdnung(*Ordnungsstrategie*)” die Neuordnung der Variablen veranlaßt, so werden die Eingabe-, Zustands- und Konstruktionsvariablen seperat neu geordnet, eine Durchmischung ist also nicht möglich. Bei der Neuordnung der Zustands- und Konstruktionsvariablen werden dabei die jeweils vier zugehörigen Variablen $Q_i, Q'_i, \tilde{Q}_i$ und $\tilde{Q}'_i$ nicht getrennt.

11.1.3 Implementierung der Automaten und ihre Konstruktionen

Die Automaten-Bibliothek stellt die Operation

$$\begin{aligned}\text{Automat Automat}(&\text{bdd\_Paket \&BDD}, \text{VarManager \&VarM},\\ &\text{int Zustand\_Anz}, \text{int Eingabe\_Anz}, \text{int Ausgabe\_Anz},\\ &\text{bdd} *\text{ZustandFkt}, \text{bdd} *\text{AusgabeFkt}, \text{bdd} *\text{Ausgabeberechtigung},\\ &\text{int} **\text{Zustand\_Partition\_vorwärts}, \text{int} **\text{Zustand\_Partition\_rueckwärts}\\ &\text{int Startzustand})\end{aligned}$$

zur Verfügung, mit der ein synchroner Schaltkreis mit Ausgabeberechtigung $(\delta, \lambda, \alpha, s)$ vom Typ (l, m, n) eingegeben werden kann. Mit dieser Operation wird genauer ein Automat erzeugt, der

- mit dem OBDD-Manager BDD arbeitet,
- den Variablenmanager VarM verwendet,
- ein Automat vom Typ (l = Zustand_Anz, m = Eingabe_Anz, n = Ausgabe_Anz) ist,
- die booleschen Funktionen

  $$\begin{aligned}\delta_i &= \text{ZustandFkt}[i], 0 \leq i < l\\ \lambda_i &= \text{AusgabeFkt}[i], 0 \leq i < n\\ \alpha_i &= \text{Ausgabeberechtigung}[i], 0 \leq i < n\end{aligned}$$

  besitzt,
- den Startzustand $s = (s_0, \ldots, s_{l-1})$ besitzt, wobei $s_i = 1$ genau dann 1 ist, wenn das i-te Bit im Integerwert Startzustand gesetzt ist
- und dessen Transitionsrelationen bzgl. des Vorschlags der Zustandspartitionierungen

  $$\text{int} **\text{Zustand\_Partition\_vorwärts}, \text{int} **\text{Zustand\_Partition\_rueckwärts}$$

  versucht werden zu zerlegen (siehe Abschnitt 4.3.2).

Die Eingabevariablen können dabei weiter unterschieden werden zwischen herkömmlichen Eingabevariablen und Parametern. Mit Hilfe der Operation

$$A.\text{Parametrisiere}(\text{int Anzahl})$$

werden dabei die ersten p = Anzahl Eingabevariablen von einem Automaten A als Parameter interpretiert.

Informationen über Automaten

Mit Hilfe der Operation A.print_Groesse() wird die Größe der OBDDs eines Automaten angezeigt. Wenn die partitionierte Transitionsrelation berechnet ist, so werden die Größen ihrer Teil-OBDDs und die jeweilige Anzahl ihrer Variablen angegeben.

Mit A.print_FktPartition() kann die vom Anwender vorgeschlagene Partitionierung der Zustandsfunktionen von dem Automaten A betrachtet werden.

Außerdem werden mit A.print_Automat() die OBDDs der Zustandsfunktionen, Ausgabefunktionen und Ausgabeberechtigungen angezeigt. Dieser Befehl sollte jedoch nur für sehr kleine Automaten angewendet werden.

Produkt-Automat und Komposition

Liegen zwei Automaten $A1$ vom Typ $(l1, m, n)$ und $A2$ vom Typ $(l2, m, n)$ mit der selben Eingabeanzahl m und Ausgabeanzahl n vor, so wird mit $A1 * A2$ der Produktautomat vom Typ $(l1 + l2, m, 1)$ nach Konstruktion 5.7 erzeugt.

(1) Automat $A1(\mathrm{BDD}, \mathrm{VarM}, l1, m, n, \ldots)$;

(2) Automat $A2(\mathrm{BDD}, \mathrm{VarM}, l2, m, n, \ldots)$;

(3) Automat $*\,\mathrm{Produkt} = A1 * A2$;

Zu einem Automaten $A1$ vom Typ $(l1, m, n1)$ und Automaten $A2$ vom Typ $(l2, n1, n)$ wird die Komposition $A1 \to A2$ vom Typ $(l1 + l2, m, n)$ entsprechend Konstruktion 5.9 erzeugt mit $A1 > A2$.

(1) Automat $A1(\mathrm{BDD}, \mathrm{VarM}, l1, m, n1, \ldots)$;

(2) Automat $A2(\mathrm{BDD}, \mathrm{VarM}, l2, n1, n, \ldots)$;

(3) Automat $*\,\mathrm{Komposition} = A1 > A2$;

Berechnung der erreichbaren Zustände und Testen des Ausgabeverhaltens eines Automaten

Mit Hilfe der Operation A.VorwaertsTraversal() werden die erreichbaren Zustände ausgehend vom Startzustand von dem Automaten A berechnet. Die Umsetzung entspricht dem Algorithmus 5.4. Mit A.RueckwaertsTraversal() kann entsprechend die Menge der Zustände berechnet werden, von denen aus der Startzustand erreicht wird.

(1) Automat $A(\ldots)$;

(2) bdd erreichbare_Zustände = A.VorwaertsTraversal();

(3) bdd rückwärts_erreichbare_Zustände = A.RueckwaertsTraversal();

Mit der Operation

$$A.\mathrm{print\_Zustand}()$$

werden im Iterationsschritt i die Anzahl der Zustände ausgegeben, die mit i Übergängen das erste Mal erreicht werden können.

Mit Hilfe der Operation A.Ist_Ausgabe_gleich(int `Wert`) kann getestet werden, ob der Automat A vom Typ (l, m, n) immer die Ausgabe 0^n für `Wert` $= 0$ bzw. 1^n für `Wert` $= 1$ berechnet. Diese Umsetzung entspricht Algorithmus 5.5.

Berechnung einer reproduktiven allgemeinen parametrisierten Lösung

Berechnet ein Automat A die online Funktion f, so wird mit Operation

$$\mathrm{int}\ \ A.\mathrm{Kern}(\mathrm{Modus}, \mathrm{Zustand\_red})$$

der Automat A umgewandelt, sodaß dieser eine reproduktive allgemeine parametrisierte Lösung von f realisiert, wenn dies möglich ist. Gelingt diese Berechnung, so liefert die Operation den Wert 1 zurück, ansonsten 0.

Wenn A außerdem Eingabeparameter besitzt, so wird - wenn möglich - eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung berechnet.

Die Operation Kern setzt sich dabei aus mehreren Teilberechnungsschritten zusammen.

1. Ist A ein Automat vom Typ (l, m, n), so wird mit der Operation

   $$A.\text{mache\_AusgabeAnz\_1}()$$

   der Automat A nach Konstruktion 7.2 zu einem Automaten vom Typ $(l, m, 1)$ umgeformt, wobei die Ausgabe genau dann 0 ist, wenn die Ausgabe von dem ursprünglichen Automaten $(0, \ldots, 0) \in \mathbb{B}^n$ ist.

2. Mit der Operation

   $$A.\text{mache\_online}()$$

   wird A zu einem synchronen Schaltkreis umgeformt, dessen online Funktion dieselbe Nullstellenmenge besitzt wie die offline Funktion von dem ursprünglichen Automaten A.

3. Mit der Operation

   $$A.\text{mache\_erweiterbar}(\text{Modus}, \text{Zustand\_red})$$

   wird Automat A - falls möglich - umgewandelt, dessen online Funktion erweiterbar ist und dieselbe Nullstellenmenge besitzt wie die online Funktion des ursprünglichen Automaten A. Besitzt der Automat außerdem Eingabeparameter, so werden alle nicht-kontrollierbaren Nullstellen eliminiert. Die Operation entspricht dabei dem Algorithmus 6.32 im Fall Modus = 1 und 6.34 im Fall Modus = 2. Für Zustand_red = 1 wird entsprechend Bemerkung 6.35 versucht, die Zustandsanzahl zu verringern.

   Für den Parameterfall werden entsprechend Abschnitt 6.5 die Konstruktionen 6.32 und 6.34 erweitert.

4. Mit der Abfrage

   $$A.\text{Ist\_erweiterbar}() == 1$$

   wird anschließend getestet, ob der Automat A erweiterbar ist. Diese Abfrage entspricht dem Test (24) auf Seite 72.

5. Ist A erweiterbar, so kann mit Operation

   $$A.\text{berechne\_rapl}()$$

   eine reproduktive allgemeine parametrisierte Lösung berechnet werden. Ist die online Funktion von A mit ihren Parametern sogar kontrollierbar, so liegt eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung vor.

Diese Teiloperationen werden neben der Operation Kern() für den Anwender zur Verfügung gestellt. Damit hat der Anwender die Möglichkeit, teilweise die Reihenfolge der Operationen zu verändern, statistische Informationen über die Zwischenschritte abzufragen oder weitere Operationen einzufügen, wie z.B. die Reduzierung von OBDDs oder die Minimalisierung der Automaten.

Lösen von Ungleichungen

Mit der Operation

$$\text{int}\ \ A.\text{Ungleichung}(\text{Modus}, \text{Zustand\_red})$$

wird der synchrone Schaltkreis nach Konstruktion 8.4 erzeugt. Die zugehörigen Parameter Modus und Zustand_red bewirken ein analoges Vorgehen für die Fixpunktberechnung wie bei Operation mache_erweiterbar().

Minimalisierung

Die Minimalisierung gelingt mit der Operation

$$A.\text{Minimalisierung}(\text{Modus\_rep}, \text{Modus\_tran})$$

Dabei wird zuerst die Äquivalenzrelation der äquivalenten Zustände berechnet. Dies erfolgt mit Algorithmus 9.11. Anschließend wird zu jeder Äquivalenzklasse ein Repräsentant ausgewählt. Dies gelingt durch Berechnung (36) auf Seite 107 mit Modus_rep = 0 oder durch Algorithmus 9.13 mit Modus_rep = 1. Anschließend wird eine Transitionsrelation für den Minimalautomaten berechnet. Die ursprüngliche Transitionsrelation kann mit Modus_tran = 1 zusätzlich berücksichtigt werden (siehe Seite 110).

Reduzierung

Die Zustandsfunktionen, Ausgabefunktionen und Ausgabeberechtigungen können sich im Bereich der nicht erreichbaren Zustände beliebig verhalten. Für eine Reduktion müssen nun die erreichbaren Zustände `erreicht` berechnet werden. Anschließend kann mit der Operation

$$A.\text{ReduziereFkt\_primitiv}(\texttt{erreicht})$$

versucht werden, die OBDDs des Automaten A mit der OBDD-Operation REDUCE (siehe 5.2) zu reduzieren. Gelingt dies bei einem OBDD nicht, so wird das ursprüngliche OBDD beibehalten. Mit der Operation

$$A.\text{ReduziereFkt\_komplex}(\texttt{erreicht}, \text{maximal})$$

wird dies anstelle der REDUCE-Operation mit Algorithmus 10.5 versucht, der in Abschnitt 10 vorgestellt wird. Mit maximal = 1 wird für dieses Verfahren dabei der Algorithmus 10.7 zur Berechnung einer maximalen Clique verwendet. Ansonsten wird die Reduzierung mit einem heuristischen Verfahren 10.8 zum Auffinden einer möglichst großen Clique verwendet.

Mit Operation

$$A.\text{ReduziereGesamtFkt\_komplex}(\texttt{erreicht}, \text{maximal})$$

wird derselbe Algorithmus angewendet, wobei jedoch nicht versucht wird, jedes einzelne OBDD zu reduzieren, sondern die Gesamtzahl der Knoten aller OBDDs möglichst klein zu wählen.

11.2 Beispiele

Die OBDDs eines synchronen Schaltkreises werden im folgenden mit Hilfe von Wertetabellen erzeugt. Betrachtet man ein OBDD als einen vollständigen binären Baum (siehe Abschnitt 2.1.2), so entspricht die Beschriftung eines terminalen Knotens genau dem Eintrag der Variablenbewertung in der Wertetabelle. Soll ein OBDD mit der Variablenanzahl n generiert werden, so entspricht ein Bitstring der Länge 2^n der Wertetabelle für das OBDD. Das i-te Bit entspricht dabei der Beschriftung des i-ten terminalen Knotens von dem binären Baum. Während der binäre Baum nun rekursiv zu den terminalen Knoten mit der Beschriftung bzgl. des Bitstrings durchlaufen wird, kann das OBDD reduziert in die unique_tabel eingelagert werden.

Mit Hilfe der Bitstrings, die aus Blöcken der Länge 32 zusammengesetzt werden, können so OBDDs mit beliebiger Variablenanzahl erzeugt werden.

Mit Hilfe der C-Funktion random() können so beliebige Wertetabellen für die OBDDs generiert werden. Um die folgenden Beispiele für verschiedene Computersysteme identisch zu halten, ist die

Funktion Zufall() implementiert, die aus einem Array 32-Bitfolgen ausliest, die nicht verändert werden können. Diese selbst sind mit Hilfe der random() Funktion erzeugt worden.

Die Zustandsfunktionen eines Automaten können nun durch die Operation

$$\begin{aligned}&\text{bdd ZufallAutomat\_BDD(bdd\_manager bddm, int Var\_Anz, tt } *\text{table,}\\ &\qquad\qquad\text{int Eingabe\_Anz, bdd } *\text{ZustandVar, bdd } *\text{EingabeVar)}\end{aligned}$$

erzeugt werden. Das OBDD besitzt dann maximal Var_Anz viele Variablen. Von diesen Variablen werden zuerst alle Eingabevariablen in der Reihenfolge des BDD-Strings EingabeVar[Eingabe_Anz] verwendet. Schließlich werden (Var_Anz-Eingabe_Anz) viele Zustandsvariablen in der Reihenfolge des BDD-Strings ZustandVar[Zustand_Anz] übernommen. Mit diesen Variablen wird dann das OBDD erzeugt, dessen Ausgabeverhalten von der Wertetabelle table[$2^{\text{Var\_Anz}-5}$], bestehend aus 32-Bitblöcken, bestimmt wird.

Erzeugen von Automaten

Die Beispiel-Automaten werden nun folgendermaßen aufgebaut. Mit den Eingabevariablen EingabeVar[Eingabe_Anz]

VarM.hole_EingabeVar(Eingabe_Anz, EingabeVar)

und Zustandsvariablen ZustandVar[Zustand_Anz]

VarM.hole_ZustandsVar(Zustand_Anz, ZustandVar)

werden die Zustandsfunktionen durch

$$\begin{aligned}&\text{ZustandFkt}[2*i] = \text{ZufallAutomat\_BDD(bddm, Var\_Anz, table,}\\ &\qquad\qquad\text{Eingabe\_Anz, ZustandVar} + 2*i\text{, EingabeVar)}\end{aligned}$$

mit $0 \leq i < k$ wobei $k := \text{Zustand\_Anz} - (\text{Var\_Anz} - \text{Eingabe\_Anz})$ erzeugt. Die übrigen Zustandsfunktionen werden erzeugt durch

$$\begin{aligned}&\text{ZustandFkt}[i] = \text{ZufallAutomat\_BDD(bddm, Var\_Anz, table,}\\ &\qquad\qquad\text{Eingabe\_Anz, ZustandVar} + (\text{Zufall} \mod k)\text{, EingabeVar)}\end{aligned}$$

generiert.

Die Zustandsfunktionen selbst bestehen also aus den Eingabevariablen und Zustandsvariablen, deren zugehörige Zustandsfunktionen benachbart sind. Damit soll eine gewisse Lokalität eines synchronen Schaltkreises berücksichtigt werden. Mit dieser Konstruktion wird aber ebenso berücksichtigt, daß viele Zustandsfunktionen direkt oder indirekt von möglichst vielen Variablen abhängen.

Beispiel 1

Mit

$$\text{Automat } *A = \text{Beispiel\_Kern1(BDD, VarM);}$$

wird ein synchroner Schaltkreis mit Ausgabeberechtigung vom Typ $(20, 1, 1)$ nach obiger Strategie generiert, der eine online Funktion $f : {}_2\mathbb{Z} \to {}_2\mathbb{Z}$ berechnet. Mit

$$A\text{.print\_Groesse();}$$

erhält man die OBDD-Größen des Automaten A:

```
*************** Automatengroesse - gesamt:202 ******************************
************ Zustandsfunktionen - gesamt:193 *******************************
Fkt0: 12 , Fkt1: 11 , Fkt2: 12 , Fkt3: 13 , Fkt4: 12, Fkt5: 11 , Fkt6: 12 ,
Fkt7: 13 , Fkt8: 12 , Fkt9: 11 , Fkt10: 14 , Fkt11: 12 , Fkt12: 13 , Fkt13: 14 ,
Fkt14: 12 , Fkt15: 12 , Fkt16: 12 , Fkt17: 12 , Fkt18: 13 , Fkt19: 11 ,
************ Ausgabefunktionen - gesamt:11 *********************************
Fkt0: 11,
************ Ausgabeberechtigung -gesamt:6 *********************************
Fkt0: 6,
*************** Ende Automatengroesse***************************************
```

Mit dem Befehl

$$A.\text{print\_Zustand}()$$

erhält man nun die Anzahl der Zustände, die in der Erreichbarkeitstiefe das erste Mal erreicht werden.

Tiefe	neue Zustaende	erreichbare Zustaende	Knotenanzahl
0	1	1	21
1	2	3	55
2	4	7	113
3	8	15	204
4	16	31	342
5	32	63	540
6	64	127	831
7	126	253	1150
8	237	490	1619
9	431	921	2309
10	735	1656	3171
11	1162	2818	4153
12	1658	4476	5278
13	2080	6556	6341
14	2332	8888	7306
15	2319	11207	8026
16	2107	13314	8570
17	1749	15063	8971
18	1382	16445	9216
19	1027	17472	9369
20	738	18210	9455
21	489	18699	9530
22	319	19018	9585
23	190	19208	9614
24	126	19334	9627
25	78	19412	9630
26	53	19465	9641
27	38	19503	9643
28	27	19530	9642
29	19	19549	9653
30	10	19559	9655
31	7	19566	9657
32	5	19571	9659
33	4	19575	9660
34	2	19577	9660
35	0	19577	9660

Der Automat kann kopiert werden mit

$$\text{Automat } K = A;$$

Mit

$$K.\text{Kern}();$$

wird K nun zu einem synchronen Schaltkreis umgeformt, dessen online Funktion eine reproduktive allgemeine parametrisierte Lösung von f ist.

Entscheidend für diese Berechnung ist der Zwischenschritt

$$K.\text{mache\_erweiterbar}(2, 0);$$

Für diese Berechnung wird entsprechend Algorithmus 6.34 im Iterationsschritt i die Ausgabefunktion $\lambda[i]$ von dem synchronen Schaltkreis $\mathcal{S}[i]$ berechnet. Dabei wird die Größe des OBDDs der Ausgabefunktion angegeben:

```
lambda[0]:14
lambda[1]:22
lambda[2]:44
lambda[3]:46
lambda[4]:47
```

Zu diesem erweiterbaren synchronen Schaltkreis, der den Fixpunkt $f[*]$ aus der Iterationsvorschrift 6.10 berechnet, wird nun mit

$$K.\text{berechne\_rapl}()$$

ein synchroner Schaltkreis konstruiert, dessen online Funktion eine reproduktive allgemeine parametrisierte Lösung von f ist.

```
*************** Automatengroesse - gesamt:377 *******************
************ Zustandsfunktionen - gesamt:253 *******************
Fkt0: 15 , Fkt1: 14 , Fkt2: 15 , Fkt3: 16 , Fkt4: 15 , Fkt5: 14 ,
Fkt6: 15 , Fkt7: 16 , Fkt8: 15 , Fkt9: 14 , Fkt10: 17 , Fkt11: 15 ,
Fkt12: 16 , Fkt13: 17 , Fkt14: 15 , Fkt15: 15 , Fkt16: 15 , Fkt17: 15 ,
Fkt18: 16 , Fkt19: 14 ,
************ Konstruktionsfunktionen - gesamt:154 ****************
Fkt0: 18, Fkt1: 48, Fkt2: 1, Fkt3: 134,
************ Ausgabefunktionen - gesamt:134 *********************
Fkt0: 134,
************ Ausgabeberechtigung -gesamt:1 **********************
Fkt0: 1,
*************** Ende Automatengroesse*****************************
```

Mit

$$\text{Automat } \texttt{Teste} = K > A;$$

wird nun die Komposition von K mit A konstruiert. Mit der Abfrage

$$\texttt{Teste}.\text{Ist\_Ausgabe\_gleich}(0) == 1$$

kann dann getestet werden, ob mit den berechneten Ausgabefolgen von K tatsächlich der Automat A immer eine 0 berechnet.

Beispiel 2

Im folgenden wird eine reproduktive allgemeine parametrisierte Lösung zu der offline Funktion f konstruiert, die von dem Automaten A vom Typ $(22, 1, 1)$ mit folgenden OBDD-Größen berechnet wird.

```
*************** Automatengroesse - gesamt:295 ********************
************ Zustandsfunktionen - gesamt:280 *********************
Fkt0: 20 , Fkt1: 21 , Fkt2: 22 , Fkt3: 22 , Fkt4: 19 , Fkt5: 23 ,
Fkt6: 22 , Fkt7: 21 , Fkt8: 22 , Fkt9: 22 , Fkt10: 23 , Fkt11: 20 ,
Fkt12: 22 , Fkt13: 22 , Fkt14: 23 , Fkt15: 21 ,
************ Konstruktionsfunktionen - gesamt:0 ******************
************ Ausgabefunktionen - gesamt:11 ***********************
Fkt0: 11,
************ Ausgabeberechtigung -gesamt:13 **********************
Fkt0: 13,
*************** Ende Automatengroesse*****************************
```

Die Erreichbarkeitstiefe des Automaten A ist 30, wobei 10218 Zustände erreicht werden, die mit einem OBDD der Größe 4967 beschrieben werden können.

Mit Hilfe der Operation

$$K.\text{mache\_erweiterbar}(2,0);$$

erhält man nun einen synchronen Schaltkreis $S[*]$ mit der Ausgabefunktion $\lambda[i]$, mit dem die erweiterbare online Funktion $f[*]$ berechnet wird. Dabei werden in den Zwischenschritten von Algorithmus 6.34 folgende OBDDs berechnet.

```
lambda[0]:172, lambda[1]:498, lambda[2]:528, lambda[3]:570, lambda[4]:587, lambda[5]:581,
lambda[6]:589, lambda[7]:591, lambda[8]:587, lambda[9]:591, lambda[10]:596, lambda[11]:595,
lambda[12]:594, lambda[13]:593, lambda[14]:594, lambda[15]:595, lambda[16]:595
```

Mit diesem Zwischenschritt erhält man nun einen Automaten, der eine reproduktive allgemeine parametrisierte Lösung von f ist

```
*************** Automatengroesse - gesamt:4251 ********************
************ Zustandsfunktionen - gesamt:328 **********************
Fkt0: 23 , Fkt1: 24 , Fkt2: 25 , Fkt3: 25 , Fkt4: 22 , Fkt5: 26 ,
Fkt6: 25 , Fkt7: 24 , Fkt8: 25 , Fkt9: 25 , Fkt10: 26 , Fkt11: 23 ,
Fkt12: 25 , Fkt13: 25 , Fkt14: 26 , Fkt15: 24 ,
************ Konstruktionsfunktionen - gesamt:4013 *****************
Fkt0: 35, Fkt1: 597, Fkt2: 1, Fkt3: 3975,
************ Ausgabefunktionen - gesamt:3975 **********************
Fkt0: 3975,
************ Ausgabeberechtigung -gesamt:1 ************************
Fkt0: 1,
*************** Ende Automatengroesse******************************
```

Mit Hilfe der Operation

$$K.\text{mache\_erweiterbar}(1,0);$$

erhält man nun die Berechnung mit Algorithmus 6.32

```
lambda[0]:172, lambda[1]:1558, lambda[2]:2654, lambda[3]:3067, lambda[4]:3323, lambda[5]:3417,
lambda[6]:3466, lambda[7]:3497, lambda[8]:3525, lambda[9]:3555, lambda[10]:3574, lambda[11]:3597,
lambda[12]:3587, lambda[13]:3586, lambda[14]:3587, lambda[15]:3586, lambda[16]:3587,
lambda[17]:3587, lambda[18]:3585, lambda[19]:3585, lambda[20]:3585, lambda[21]:3586,
lambda[22]:3586
```

Diese Ausgabefunktion ist nun bereits zu groß, um eine reproduktive allgemeine parametrisierte Lösung mit einem herkömmlichen Rechnersystem zu erzeugen.

Beispiel 3

Gegeben ist der Automat A vom Typ $(13, 2, 1)$, dessen online Funktion kontrollierbar ist. Die OBDDs des Automaten A besitzen folgende Größe:

```
*************** Automatengroesse - gesamt:504 *********
************ Zustandsfunktionen - gesamt:473 **********
Fkt0: 42 , Fkt1: 41 , Fkt2: 42 , Fkt3: 41 , Fkt4: 43 ,
Fkt5: 44 , Fkt6: 44 , Fkt7: 44 , Fkt8: 42 , Fkt9: 42 ,
Fkt10: 43 , Fkt11: 42 , Fkt12: 43 ,
************ Konstruktionsfunktionen - gesamt:0 ********
************ Ausgabefunktionen - gesamt:29 *************
Fkt0: 29,
************ Ausgabeberechtigung -gesamt:12 ************
Fkt0: 12,
*************** Ende Automatengroesse*******************
```

Nach 16 Zustandsübergängen werden alle 872 Zustände erreicht. Diese können mit einem OBDD der Größe 618 kodiert werden.

Mit Hilfe der Operation

$$A.\text{Parametrisiere}(1)$$

wird nun die erste Eingabevariable als Parameter interpretiert.

Anschließend wird mit dem Befehl

$$K.\text{mache\_erweiterbar}(2, 0);$$

Algorithmus 6.32 ausgeführt, um Iteration 6.44 durchzuführen, d.h. um alle nicht-kontrollierbaren Nullstellen zu eliminieren. Im Iterationsschritt i wird dabei die Ausgabefunktion $\lambda[i]$ entsprechend modifiziert mit folgenden OBDD-Größen:

```
lambda[0]:248, lambda[1]:371, lambda[2]:469, lambda[3]:534, lambda[4]:581, lambda[5]:642,
lambda[6]:654, lambda[7]:710, lambda[8]:725, lambda[9]:714, lambda[10]:710, lambda[11]:694,
lambda[12]:671, lambda[13]:646, lambda[14]:598, lambda[15]:553, lambda[16]:471, lambda[17]:428,
lambda[18]:418, lambda[19]:418
```

Anschließend liegt ein Automat vor, dessen online Funktion nur noch kontrollierbare Nullstellen besitzt. Der Automat besitzt nun folgende OBDD-Größe:

```
*************** Automatengroesse - gesamt:914 **************************
************ Zustandsfunktionen - gesamt:486 ***************************
Fkt0: 43 , Fkt1: 42 , Fkt2: 43 , Fkt3: 42 , Fkt4: 44 , Fkt5: 45 , Fkt6: 45 ,
Fkt7: 45 , Fkt8: 43 , Fkt9: 43 , Fkt10: 44 , Fkt11: 43 , Fkt12: 44 ,
************ Konstruktionsfunktionen - gesamt:487 **********************
Fkt0: 70, Fkt1: 418,
************ Ausgabefunktionen - gesamt:418 ****************************
Fkt0: 418,
************ Ausgabeberechtigung -gesamt:1 *****************************
Fkt0: 1,
*************** Ende Automatengroesse***********************************
```

mit folgender Anzahl der erreichbaren Zustände

Tiefe	neue Zustaende	erreichbare Zustaende	Knotenanzahl
0	1	1	16
1	1	2	27
2	3	5	42
3	2	7	53
4	1	8	62
5	0	8	62

Wird mit

$$A.\text{Minimalisierung}();$$

dieser Automat minimalisiert, so entsteht ein Automat

```
*************** Automatengroesse - gesamt:1018 ************************
************ Zustandsfunktionen - gesamt:590 **************************
Fkt0: 69 , Fkt1: 56 , Fkt2: 70 , Fkt3: 71 , Fkt4: 71 , Fkt5: 59 , Fkt6: 61 ,
Fkt7: 57 , Fkt8: 57 , Fkt9: 59 , Fkt10: 60 , Fkt11: 57 , Fkt12: 60 ,
************ Konstruktionsfunktionen - gesamt:500 *********************
Fkt0: 84, Fkt1: 418,
************ Ausgabefunktionen - gesamt:418 ***************************
Fkt0: 418,
************ Ausgabeberechtigung -gesamt:1 ****************************
Fkt0: 1,
*************** Ende Automatengroesse***********************************
```

mit den erreichbaren Zuständen

Tiefe	neue Zustaende	erreichbare Zustaende	Knotenanzahl
0	1	1	16
1	1	2	27
2	2	4	38
3	1	5	37
4	0	5	37

Mit Hilfe der OBDD-Reduktion

$$A.\text{ReduziereFkt\_komplex}(\texttt{erreicht}, 0)$$

wobei $\texttt{erreicht} = A.\text{VorwaertsTraversal}()$, erhält man dann einen Automaten mit Größe

```
*************** Automatengroesse - gesamt:153 ********************
************ Zustandsfunktionen - gesamt:114 *********************
Fkt0: 7 , Fkt1: 1 , Fkt2: 7 , Fkt3: 1 , Fkt4: 7 , Fkt5: 1 , Fkt6: 32 ,
Fkt7: 22 , Fkt8: 7 , Fkt9: 32 , Fkt10: 32 , Fkt11: 1 , Fkt12: 32 ,
************ Konstruktionsfunktionen - gesamt:20 *****************
Fkt0: 1, Fkt1: 20, ************ Ausgabefunktionen - gesamt:32 ***********************
Fkt0: 32,
************ Ausgabeberechtigung -gesamt:1 ***********************
Fkt0: 1,
*************** Ende Automatengroesse******************************
```

Zu diesem Automaten kann nun mit dem Befehl A.berechne_rapl() eine parameterbehaftete reproduktive allgemeine parametrisierte Lösung erstellt werden, die von einem Automaten mit folgender Größe berechnet werden kann.

```
*************** Automatengroesse - gesamt:441 ********************
************ Zustandsfunktionen - gesamt:129 *********************
Fkt0: 8 , Fkt1: 1 , Fkt2: 8 , Fkt3: 1 , Fkt4: 8 , Fkt5: 1 , Fkt6: 35 ,
```

```
Fkt7: 24 , Fkt8: 8 , Fkt9: 35 , Fkt10: 35 , Fkt11: 1 , Fkt12: 35 ,
************ Konstruktionsfunktionen - gesamt:325 *****************
Fkt0: 1, Fkt1: 23, Fkt2: 1, Fkt3: 165, Fkt4: 272,
************ Ausgabefunktionen - gesamt:306 **********************
Fkt0: 165, Fkt1: 272,
************ Ausgabeberechtigung -gesamt:1 ***********************
Fkt0: 1, Fkt1: 1,
*************** Ende Automatengroesse*******************************
```

Da der Automat eine parameterbehaftete Lösung berechnet, gilt

$$\lambda_0 : \begin{cases} \mathbb{B}^2 \times \mathbb{B}^{18} & \rightarrow & \mathbb{B} \\ (\boldsymbol{X}, \boldsymbol{Q}) & \mapsto & X_0 \end{cases}$$

d.h. die 0-te Ausgabefunktion gibt die Eingabe der 0-ten Eingabevariable X_0 aus. Mit dem Befehl

$$A.\text{vereinfache\_Parameter}();$$

wird dann entsprechend die Ausgabefunktion vereinfacht: $\lambda_0 = \text{ITE}(X_0, \mathbf{1}, \mathbf{0})$.

Mit dem Befehl

$$A.\text{ReduziereGesamtFkt\_komplex}(\texttt{erreicht}, 0)$$

können dann die OBDDs des Automaten reduziert werden zu

```
*************** Automatengroesse - gesamt:67 *****************************
************ Zustandsfunktionen - gesamt:44 ******************************
Fkt0: 31 , Fkt1: 28 , Fkt2: 31 , Fkt3: 28 , Fkt4: 31 , Fkt5: 28 , Fkt6: 31 ,
Fkt7: 35 , Fkt8: 31 , Fkt9: 28 , Fkt10: 31 , Fkt11: 28 , Fkt12: 36 ,
************ Konstruktionsfunktionen - gesamt:60 *************************
Fkt0: 28, Fkt1: 28, Fkt2: 38, Fkt3: 32, Fkt4: 47,
************ Ausgabefunktionen - gesamt:50 *******************************
Fkt0: 32, Fkt1: 47,
************ Ausgabeberechtigung -gesamt:1 *******************************
Fkt0: 1, Fkt1: 1,
************Relationale Transition vorwaerts - gesamt: 1440 **************
RelTran0: 1127, RelTran1: 251, RelTran2: 75,
************Relationale Transition rueckwaerts - gesamt: 1359 ************
RelTran0: 1127, RelTran1: 14, RelTran2: 14, RelTran3: 107, RelTran4: 48,
RelTran5: 39, RelTran6: 36,
*************** Ende Automatengroesse*************************************
```

Wird dieselbe Berechnung ohne Minimalisierung durchgeführt, so erhält man einen Automaten der Größe

```
*************** Automatengroesse - gesamt:767 ********************
************ Zustandsfunktionen - gesamt:169 *********************
Fkt0: 18 , Fkt1: 20 , Fkt2: 14 , Fkt3: 19 , Fkt4: 17 , Fkt5: 20 , Fkt6: 13 ,
Fkt7: 31 , Fkt8: 4 , Fkt9: 20 , Fkt10: 14 , Fkt11: 20 , Fkt12: 17 ,
************ Konstruktionsfunktionen - gesamt:617 *****************
Fkt0: 18, Fkt1: 32, Fkt2: 1, Fkt3: 375, Fkt4: 536,
************ Ausgabefunktionen - gesamt:575 **********************
Fkt0: 375, Fkt1: 536,
*************** Ende Automatengroesse*******************************
```

Dieser besitzt bereits so große OBDDs, daß nur mit sehr großen Speicherressourcen getestet werden kann, ob dieser Automat tatsächlich eine parameterbehaftete Lösung berechnet.

Erst wenn die OBDDs von diesem Automaten z.B. mit

$$A.\text{ReduziereGesamtFkt\_komplex}(\texttt{erreicht}, 0)$$

reduziert werden, erhält man einen ausreichend klein dargestellten Automaten, der verifiziert werden kann:

```
*************** Automatengroesse - gesamt:63 *****************************
************ Zustandsfunktionen - gesamt:41 ******************************
Fkt0: 38 , Fkt1: 28 , Fkt2: 38 , Fkt3: 28 , Fkt4: 38 , Fkt5: 28 , Fkt6: 38 ,
Fkt7: 27 , Fkt8: 38 , Fkt9: 28 , Fkt10: 38 , Fkt11: 28 , Fkt12: 39 ,
************ Konstruktionsfunktionen - gesamt:54 *************************
Fkt0: 28, Fkt1: 28, Fkt2: 19, Fkt3: 23, Fkt4: 45,
************ Ausgabefunktionen - gesamt:48 *******************************
Fkt0: 23, Fkt1: 45,
************ Ausgabeberechtigung -gesamt:1 *******************************
Fkt0: 1, Fkt1: 1,
************Relationale Transition vorwaerts - gesamt: 8230 **************
RelTran0: 5752, RelTran1: 2463, RelTran2: 18,
************Relationale Transition rueckwaerts - gesamt: 6102 ************
RelTran0: 5752, RelTran1: 108, RelTran2: 90, RelTran3: 90, RelTran4: 41,
RelTran5: 60, RelTran6: 18,
*************** Ende Automatengroesse*************************************
```

Auffallend dabei ist nun, daß die Transitionsrelation von diesem Automaten wesentlich größer ist als bei der Konstruktion mit Minimalisierung.

Beispiel 4

Folgender Automat A wird minimalisiert:

```
*************** Automatengroesse - gesamt:232 ****************************
************ Zustandsfunktionen - gesamt:170 *****************************
Fkt0: 13 , Fkt1: 13 , Fkt2: 14 , Fkt3: 13 , Fkt4: 13 , Fkt5: 13 , Fkt6: 15 ,
Fkt7: 15 , Fkt8: 15 , Fkt9: 13 , Fkt10: 11 , Fkt11: 12 , Fkt12: 15 ,
Fkt13: 14 , Fkt14: 15 , Fkt15: 13 ,
************ Konstruktionsfunktionen - gesamt:0 **************************
************ Ausgabefunktionen - gesamt:64 *******************************
Fkt0: 64,
************ Ausgabeberechtigung -gesamt:12 ******************************
Fkt0: 12,
*************** Ende Automatengroesse*************************************
```

Dieser erreicht nach 18 Zustandsübergängen 615 Zustände, die von einem OBDD der Größe 722 kodiert werden können.

Mit dem Befehl

$$A.Minimalisierung(0, 0)$$

wird mit Algorithmus 9.11 im Iterationsschritt k die Relation der k-unterscheidbaren Zustände U_k im Bereich der erreichbaren Zustände errechnet:

```
U_0:1454, U_1:3090, U_2:1952, U_3:1063, U_4:957, U_5:957
```

Mit Hilfe der Strategie (36) auf Seite 107 wird anschließend die Transitionsrelation berechnet und daraus die Zustandsfunktionen erstellt. Es ergibt sich nun ein Automat mit den Zustandsfunktionen:

```
*************** Automatengroesse - gesamt:2738 ***********************
************ Zustandsfunktionen - gesamt:2673 ************************
Fkt0: 87 , Fkt1: 80 , Fkt2: 61 , Fkt3: 39 , Fkt4: 50 , Fkt5: 70 , Fkt6: 61 ,
Fkt7: 107 , Fkt8: 149 , Fkt9: 216 , Fkt10: 322 , Fkt11: 533 , Fkt12: 519 ,
Fkt13: 413 , Fkt14: 484 , Fkt15: 466 ,
*************** Ende Automatengroesse*********************************
```

Dieser erreicht nach 18 Zustandsübergängen 514 Zustände, die mit einem OBDD der Größe 654 kodiert werden können.

Mit Hilfe des Befehls

$$A.\text{ReduziereFkt\_primitiv}(\texttt{erreicht})$$

kann nun mit $\texttt{erreicht} = A.\text{VorwaertsTraversal}()$ ein Automat zu der Größe

```
*************** Automatengroesse - gesamt:1917 *************************
************ Zustandsfunktionen - gesamt:1855 **************************
Fkt0: 51 , Fkt1: 62 , Fkt2: 59 , Fkt3: 38 , Fkt4: 50 , Fkt5: 69 , Fkt6: 61 ,
Fkt7: 107 , Fkt8: 149 , Fkt9: 216 , Fkt10: 321 , Fkt11: 360 , Fkt12: 237 ,
Fkt13: 186 , Fkt14: 164 , Fkt15: 193 ,
*************** Ende Automatengroesse************************************
```

reduziert werden. Mit $A.\text{ReduziereFkt\_komplex}(\texttt{erreicht}, 0)$ kann dazu im Vergleich der Automat auf eine wesentlich kleinere Größe reduziert werden:

```
*************** Automatengroesse - gesamt:818 **************************
************ Zustandsfunktionen - gesamt:752 ***************************
Fkt0: 52 , Fkt1: 79 , Fkt2: 61 , Fkt3: 39 , Fkt4: 50 , Fkt5: 47 , Fkt6: 40 ,
Fkt7: 48 , Fkt8: 48 , Fkt9: 45 , Fkt10: 37 , Fkt11: 60 , Fkt12: 61 ,
Fkt13: 45 , Fkt14: 109 , Fkt15: 76 ,
*************** Ende Automatengroesse************************************
```

Wird nun mit

$$A.Minimalisierung(1, 1)$$

der Minimalautomat konstruiert, d.h. mit Algorithmus 9.13 die Menge der Repräsentanten berechnet und mit Hilfe der ursprünglichen Transitionsrelation die Transitionsrelation des Minimalautomaten erstellt, so gewinnt man einen Minimalautomaten mit folgenden OBDD-Größen der Zustandsfunktionen:

```
*************** Automatengroesse - gesamt:1275 **************************
************ Zustandsfunktionen - gesamt:1210 ***************************
Fkt0: 150 , Fkt1: 93 , Fkt2: 87 , Fkt3: 38 , Fkt4: 51 , Fkt5: 72 , Fkt6: 40 ,
Fkt7: 78 , Fkt8: 72 , Fkt9: 70 , Fkt10: 66 , Fkt11: 171 , Fkt12: 126 ,
Fkt13: 73 , Fkt14: 240 , Fkt15: 195 ,
*************** Ende Automatengroesse*************************************
```

Diese sind wesentlich kleiner im Vergleich zu dem ersten Minimalautomaten. Werden nun jedoch mit

$$\text{ReduziereFkt\_komplex}(\texttt{erreicht}, 0)$$

die OBDDs reduziert, so gewinnt man ein unbrauchbareres Ergebnis als mit dem ersten Minimalautomaten:

```
*************** Automatengroesse - gesamt:1001 *****************************
************ Zustandsfunktionen - gesamt:935 *******************************
Fkt0: 109 , Fkt1: 93 , Fkt2: 63 , Fkt3: 38 , Fkt4: 51 , Fkt5: 44 , Fkt6: 39 ,
Fkt7: 48 , Fkt8: 59 , Fkt9: 54 , Fkt10: 34 , Fkt11: 81 , Fkt12: 59 ,
Fkt13: 48 , Fkt14: 136 , Fkt15: 123 ,
*************** Automatengroesse ******************************************
```

Index

Literatur

[BCL+94] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1994.

[Bra84] Wilfried Brauer. *Automatentheorie*. B.G. Teubner, 1984.

[BRB90] Karl S. Brace, Richard L. Rudell, and Randal E. Bryant. Efficient implementation of a bdd package. *27th ACM/IEEE Design Automation Conference*, 1990.

[Bry85] Randel E. Bryant. Symbolic manipulation of boolean functions using a graphical representation. *22nd Design Automation Conference*, 1985.

[Bry86] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *Transaction on Computers*, 1986.

[BW96] Beate Bollig and Ingo Wegener. Improving the variable ordering of obdds is np-complete. *Transaction on Computers*, 1996.

[CBM89] Olivier Coudert, Christian Berthet, and Jean Cristophe Madre. Verification of sequential machines using boolean functional vectors, leuven. *Proc. of the IFIP International Workshop, Applied Formal Methods for Correct VLSI Design*, 1989.

[CCMS94] Shih-Chieh Chang, David Ihsin Cheng, and Malgorzata Marek-Sadowska. Minimizing robdd size of incompletely specified multiple output functions. *Proc. IEEE Europ. Design Automation Conference*, 1994.

[CM90] Olivier Coudert and Jean Christophe Madre. A unified framework for the formal verification of sequential circuits. *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990.

[CPR95] Fulvio Corno, Paolo Prinetto, and Matteo Sonza Reorda. Using symbolic techniques to find the maximum clique in very large sparse graphs. *Proc. IEEE Europ. Design Automation Conference*, 1995.

[FS90] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *Transaction on Computers*, 1990.

[HR68] Peter L. Hammer and Sergiu Rudeanu. *Boolean Methods in Operations Research and Relates Areas*. Springer, 1968.

[JED91] J.R.Burch, E.M.Clarke, and D.E.Long. Representing circuits more efficiently in symbolic model checking. *Proc. 28th ACM/IEEE Design Automation Conference*, 1991.

[JSL+90] Hervé J.Touati, Hamid Savoj, Bill Lin, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using bdd's. *Proc. IEEE Int. Conf. Computer-Aided Design*, 1990.

[LL92] H. T. Liaw and C. S. Lin. On the obdd-representation of general boolean functions. *Transaction on Computers*, 1992.

[Lon93] D. Long. Robdd packages. *Carnegie Mellon University*, 1993.

[MIY90] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. *27th ACM/IEEE Design Automation Conference*, 1990.

[Rud93] R. L. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *IWLS Workshop Notes*, 1993.

[Stu96] Burkhard Stubert. Lösen von 2-adischen gleichungen und ungleichungen. Master's thesis, Friedrich-Alexander Universität Erlangen-Nürnberg, Institut für Mathematische Maschinen und Datenverarbeitung, Lehrstuhl für Informatik I, 1996.

[Vui94] Jean E. Vuillemin. On circuits and numbers. *Transaction on Computers*, 1994.

[Weg94] Ingo Wegener. The size of reduced obdd's and optimal read-once branching programs for almost all boolean functions. *Transaction on Computers*, 1994.

[YBSV93] Eric Felt York, Rober Brayton, and Alberto Sangiovanni-Vincentelli. Dynamic variable reordering for bdd minimization. *Proc. Euro DAC'93*, 1993.