Institut für Formale Methoden der Informatik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Erkennung von Gruppen mit CC^0 Schaltkreisen

Joel Bienias

Studiengang:	Informatik
Prüfer/in:	Dr. Armin Weiß
Betreuer/in:	Dr. Armin Weiß
Beginn am:	22. Mai 2023
Beendet am:	22. November 2023

Kurzfassung

Seit Jahren gibt es ein Interesse den Zusammenhang zwischen Komplexitätsklassen von Schaltkreisen und abstrakten Algebren besser zu verstehen. Die ersten welche dieses Thema wirklich vorantrieben, waren Barrington und Thérien mit Ihrer Arbeit über die Klasse NC^1 und endliche Monoide. Jedoch ist NC^1 nicht die einzige interessante Schaltkreisklasse. Wir wollen einen genaueren Blick auf die Unterklasse CC^0 werfen und zeigen, dass sie dieselbe Aussagekraft wie auflösbare Gruppen haben. Wir werden zeigen, dass für jede auflösbare Gruppe ein Schaltkreis existiert unter Verwendung ihrer Normalreihe. Außerdem zeigen wir, dass für jeden CC^0 Schaltkreis eine Gruppe existiert, welche wir aus zyklischen Gruppen mit Kranzprodukten erzeugen.

Abstract

For years there has been an interest in the relation of the complexity classes of circuits and abstract algebras. The first to really promote this were Barrington and Thérien with their paper about the class NC^1 and finite Monoids. However NC^1 is not the only interesting class of circuits. We want to take a look at the subclass CC^0 and show that they have the same expressiveness as solvable groups. We will show that a circuit exists for every solvable group, using its subnormal series. Furthermore we show that we can create a group with wreath products of cyclic groups, for every CC^0 circuit.

Inhaltsverzeichnis

1 Einführung

In den 80er Jahren begann die Untersuchung des Zusammenhangs zwischen endlichen Monoiden und Komplexitätsklassen von Schaltkreisen. In seinem Paper von 1986 zeigte Barrington, dass jede Sprache, welche von einem NC^1 Schaltkreis erkannt werden kann, auch von einem Branching Programm erkannt werden kann. [Bar86] Daraufhin wurden solche Schaltkreise weiterhin untersucht und Barrington veröffentlichte 1988 eine Arbeit, in welcher er bestimmte Monoid Klassen bestimmten Schaltkreis Komplexitätsklassen zuordnete, bezüglich deren Aussagekraft. [BT88] Basierend auf dieser Entdeckung ist also die Frage naheliegend, ob man für jede Schaltkreisklasse, eine genaue Klasse von Monoiden angeben kann. 1991 zeigten Barrington und Thérien wie man NC^1 feiner aufteilen kann und sie erwähnten kurz CC^0 Schaltkreise und deren Zusammenhang mit endlichen abelschen Gruppen. [MPT91] Aufgrund dieser Arbeit wollen wir einen genaueren Blick auf CC^0 Schaltkreise und endliche auflösbare Gruppen werfen.

Wir wollen in dieser Arbeit einen konkreten Beweis für diese Randnotiz aus der Arbeit von McKenzie, Thérien und Péladeau liefern. In Ihrer Arbeit [MPT91] geben sie in Theorem 2.8 Gleichheiten von Schaltkreisklassen und Programmen über bestimmten Algebren ohne konkreten Beweis an. Wir liefern also eine Konstruktion von Programm über Algebra zu Schaltkreis und umgekehrt.

Eine Gruppe G heißt auflösbar, falls eine Normalreihe $G_1 \trianglelefteq G_2 \trianglelefteq \cdots \trianglelefteq G_n = G$ existiert, sodass $G_1 = \{ e \}$. Weiter gilt, dass jede abelsche Gruppe auflösbar ist. Zudem ist jede Untergruppe, Faktorgruppe und direktes Produkt von auflösbaren Gruppen, selbst wieder auflösbar. [KM17] Diese Eigenschaften bilden die Grundlage, auf welche wir immer wieder zurückkommen.

Das andere Relevante sind die CC^0 Schaltkreise. Ein Schaltkreis aus $\mathsf{CC}^0[m]$ berechnet eine Funktion in konstanter Tiefe, polynomieller Größe und verwendet ausschließlich $\mathtt{MOD}_m$-Gatter. Zusätzlich sind solche $\mathtt{MOD}$-Gatter *unbounded-fan-in*. Dies bedeutet, dass ein $\mathtt{MOD}_m$-Gatter mit unbegrenzt vielen Inputs verwendet werden kann. [Kom22] Im Gegensatz dazu stehen *bounded-fan-in* Gatter, welche nur für eine begrenzte Anzahl an Inputs definiert sind. Obgleich die eigentliche Definition der Klasse der CC^0 Schaltkreise auf ausschließlich $\mathtt{MOD}$-Gatter einschränkt ist, dehnen wir diese Definition leicht. Wir erlauben bounded-fan-in $\mathtt{Und}$-/$\mathtt{Oder}$-Gatter da, wie wir später sehen, diese problemlos durch $\mathtt{MOD}$-Gatter ersetzt werden können. Somit wird die Klasse CC^0 als $\mathsf{CC}^0 = \bigcup_{m>1} \mathsf{CC}^0[m]$ angegeben.

Zuerst betrachten wir wichtige Grundlagen und geben einige Hilfsschaltkreise an, welche wir später wiederverwenden. Wir haben uns für graphische Darstellungen der Schaltkreise in dieser Arbeit entschieden, welche in den Darstellungen von oben Inputs erhalten und unten ihre Outputs haben. Anschließend geben wir eine vollständige Konstruktion für einen Schaltkreis basierend auf einem Programm und umgekehrt an.

2 Voraussetzungen

Im Folgenden bezeichne $[n] = \{1, \ldots, n\}$ und alternativ $[0:n] = \{0, \ldots, n\}$. Für Mengen G, H bezeichnet G^H die Menge aller Funktionen $f : H \to G$. Wir bezeichnen mit $\odot$ komponentenweise Operationen auf Tupeln.

2.1 Das Wortproblem $\mathcal{W}(M)$

Sei M ein Monoid, dann definieren wir $\mathcal{W}(M)$ wie folgt. Sei $\mathcal{W}(M) = \{\eta_M^{-1}(m) \mid m \in M\}$ $\eta_M : M^* \to M$, mit η_M surjektiv. Also beschreibt $\mathcal{W}(M)$ die Menge, aller Urbilder der Elemente aus M. [MPT91]

2.2 $\mathcal{M}$-Erkennung

Sei $L \subseteq \Sigma^*$ eine Sprache. L wird von einem Monoid M $\mathcal{M}$-erkannt, genau dann wenn ein Homomorphismus $\phi : \Sigma^* \to M$ und $A \subseteq M$ existiert, sodass $\phi^{-1}(A) = L$. Es sei $\mathcal{M}(\Sigma^*, M) = \{L \subseteq \Sigma^* \mid L \text{ wird } \mathcal{M}\text{-erkannt}\}$. Es sei weiter $\mathcal{M}(M) = \bigcup_\Sigma \mathcal{M}(\Sigma^*, M)$. [MPT91]

2.3 M-Programme

Wir definieren Programme über einem Monoid $M = (M, *)$ wie folgt. Sei Σ ein endliches Alphabet und M ein Monoid. Dann definieren wir eine Folge $\phi = (\phi_n)_{n \in \mathbb{N}_0}$ welche wir M-Programm nennen. Jedes einzelne Folgenglied ist eine endliche Folge $\phi_n = (r_i)_{i \in [l(n)]}$ von Instruktionen der Form (i, f) mit $i \in [n]$ und $f \in M^\Sigma$, hierbei bezeichne $l(n)$ die Länge der Folge bezüglich n. Zusätzlich erlauben wir Konstanten als Instruktionen der Form $(0, t)$ mit $t \in M$. Dies ist äquivalent zu einer Instruktion der Form (i, f) mit $f \in M^\Sigma$, da für $t \in M$ eine Funktion $\forall x \in \Sigma : f(x) = t$ angegeben werden kann. Um Verwirrung zu vermeiden markieren wir solche konstanten Instruktion durch $i = 0$. Somit wird deutlich, dass für $(0, f);\ f \in M$ und $(i, f);\ f \in M^\Sigma$ für $i \neq 0$ gilt. Wir setzen nun für ein Wort $x = a_1 \ldots a_n; a_i \in \Sigma$ für $i \in [n]$, die Funktion $(i_j, f_j)(x) = f_j(a_{i_j})$ mit $i_j \in [n]; j \in [l(n)]$. Wir werten also die entsprechende Funktion für den Buchstaben an Stelle i_j aus und erhalten somit für $\phi_n(x) = f_1(a_{i_1}) * \cdots * f_{l(n)}(a_{i_{l(n)}})$. Es wird also $\phi_n(x)$ bezüglich der Verknüpfung aus M berechnet. Auf diese Weise können wir jedes ϕ_n als Funktion $\phi_n : \Sigma^n \to M$ auffassen und somit ϕ als $\phi : \Sigma^* \to M$. Es wird auf diese Weise eine Programmfamilie definiert, es gilt also $\exists \phi_n \forall x \in \Sigma^n : \phi_n(x) \in M$. Wir definieren weiter $(F_n)_{n \in \mathbb{N}_0}$, sodass $\forall i \in \mathbb{N}_0 : F_i \subseteq M$. [MPT91]

2.4 $\mathcal{P}$-Erkennung

Wir definieren $\mathcal{P}(M)$ als $\mathcal{P}$-Erkennung über M-Programme folgendermaßen. Sei $L \subseteq \Sigma^*$ eine Sprache. Wir sagen L wird $\mathcal{P}$-erkannt, falls für ein Monoid M ein M-Programm $\phi = (\phi_n)_{n\in\mathbb{N}_0}$ polynomialer Länge und eine Folge $(F_n)_{n\in\mathbb{N}_0}$, mit $\forall F_n : \phi_n^{-1}(F_n) = L \cap \Sigma^n$ existiert.

2.5 Schaltkreise

Wir befassen uns mit Schaltkreisfamilien $C = (C_n)_{n\in\mathbb{N}}$. Eine Schaltkreisfamilie C berechnet eine Funktion $f : A^* \to A$. Hierbei gilt, dass ein einzelner Schaltkreis nicht beliebig viele Eingaben abdeckt, weshalb wir für jede Eingabe der Länge $n \in \mathbb{N}$ einen eigenen Schaltkreis C_n angeben. Diese Folge an Schaltkreisen bildet dann die Schaltkreisfamilie für unsere Funktion. Für bessere Lesbarkeit werden wir zwei-stellige `Und`- und `Oder`-Gatter zulassen, jedoch können diese leicht durch `MOD`-Gatter ersetzt werden, wie wir später sehen.

Wir definieren die Belegung eines einzelnen $\texttt{MOD}_p$-Gatters mit den Werten a_i für die Variablen x_i. Es soll gelten, dass $a_i \in \{0, 1\}$ und somit der Variablen x_i einen Wahrheitswert zuordnet, also $\mathcal{A} : \{x_1, \ldots, x_n\} \to \{0, 1\}$ und demnach $a_i = \mathcal{A}(x_i)$. So können wir dann den Output eines $\texttt{MOD}_p$-Gatters wie folgt berechnen.

$$\mathfrak{C}_n : \{0,1\}^n \to \{0,1\}$$
$$(a_1, \ldots, a_n) \mapsto \begin{cases} 1 & \text{, falls } \sum_{i=1}^{n} \alpha_i a_i \mod p \equiv 0 \\ 0 & \text{, sonst} \end{cases}$$

In dieser Definition berücksichtigen wir, dass eine Variable x_i multiple Male in ein Gatter gespeist werden kann, hier als α_i vermerkt. Es sei $\alpha_i \in \mathbb{N}$ und wird für jede Variable fest von C_n festgesetzt. Wir wollen hier noch anmerken, dass unsere Definition rein technisch nicht für n Inputs, sondern für deutlich mehr Inputs definiert wird, da wir multiple Nutzung einzelner Variablen zulassen. Dies kann jedoch hier vernachlässigt werden, da unsere `MOD`-Gatter unbounded-fan-in sind.

Da wir uns im Folgenden mit Schaltkreisen zur Berechnung in endlichen Gruppen beschäftigen, ist es naheliegend den Output direkt auf das Wortproblem abzubilden. So können wir für eine Gruppe G und eine Sprache $L \subseteq \Sigma^*$ mit Homomorphismus $\phi : \Sigma^* \to G$ und eine Teilmenge $A \subseteq G$, L durch $\phi^{-1}(A) = L$ erkennen. Für ein $x \in G$ legen wir fest, mit einem Schaltkreis 1 auszugeben, falls $x \in A$ und 0 falls $x \notin A$.

Wir stellen aber fest, dass sobald wir Gruppen als Kette von semidirekten Produkte darstellen, möglicherweise essenzielle Information verloren gehen und sich so Fehler kumulativ sammeln. Um dies zu vermeiden, setzen wir eine einheitliche Kodierung fest, die sowohl für Eingabe als auch Ausgabe verwendet wird. Dazu nutzen wir die One-Hot-Kodierung für Gruppenelemente. So kodieren wir zum Beispiel das neutrale Element e jeder Gruppe als $0^{n-1}1 \in \mathsf{Cod}_n$ in einer Gruppe von n Elementen.

Im Folgenden werden wir verschiedene Schaltkreise betrachten und führen die Konvention ein, dass dicke Linien mehrere Bits führen und dünne Linien einzelne Bits führen.

2.5.1 One-Hot-Kodierung

Allgemein gilt, wenn wir ein Element x aus einer Gruppe der Größe p betrachten, kodieren wir dieses mit p-Bits. Somit stellen wir sicher, dass jedes Bit exakt einem Element der Gruppe zugeordnet werden kann. Wir definieren $\mathsf{Cod}_p \subseteq \{0,1\}^*$ als die Sprache der One-Hot-Kodierung für p-Bits.

$$\mathsf{Cod}_p = \{ x \in \{0,1\}^p \mid |x|_1 = 1 \}$$

Im Falle einer zyklischen Gruppe duplizieren wir jedes Bit entsprechend seines Exponenten, bezüglich des Generators. Das heißt in einer Gruppe $C_p = \langle g \rangle$, entspricht Bit i, dem Element g^i und wir duplizieren den Input in das Gatter i-mal.

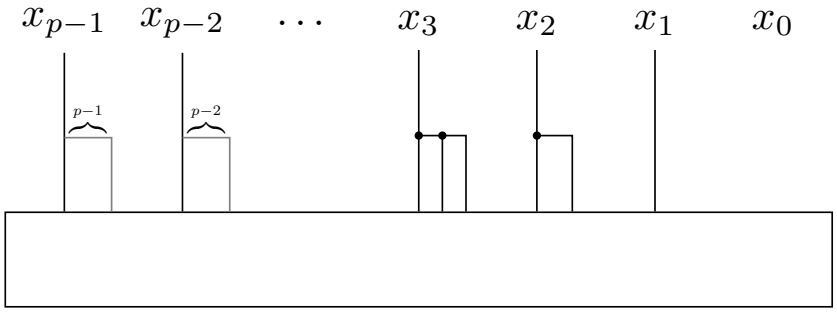


Abbildung 2.1: Schematische Darstellung für Inputs eines Elements x aus C_p

Im Folgenden werden wir solche Inputs $x_i;\ i \in \mathbb{N}$ in einen singulären Input x zusammenfassen. Dies ermöglicht eine bessere Lesbarkeit da wir mehrere Gruppen Elemente (z. B. g_0, g_1) als solche direkt in den Schaltkreis einfügen können und uns so die Aufteilung in einzelne Bits sparen, da diese meist nicht von sonderlicher Relevanz ist. Es gilt noch zu beachten, dass wir für Gruppen mit einer Normalreihe länger zwei einen Rekodierungs-Schaltkreis benötigen. Dieser bekommt normalerweise zwei unterschiedliche Kodierungen und erzeugt eine neue Kodierung. Dies ist für alle Schaltkreise nötig, die wir später aus zyklischen Schaltkreisen zusammensetzen. Dieser Schaltkreis kann als Blackbox verstanden werden, welcher aus Inputs g und h einen Output y in neuer Kodierung erzeugt. Es gilt $\#\text{bits}(g) \cdot \#\text{bits}(h) = \#\text{bits}(y)$, wobei hier # als Anzahl verstanden wird. Ein solches `recode`-Gatter kann mit Tiefe 1 realisiert werden und benötigt ausschließlich 2-stellige `Und`-Gatter. Wir können dies auch als Funktion der Form $recod_{a,b} : \mathsf{Cod}_a \times \mathsf{Cod}_b \to \mathsf{Cod}_{ab}$ angeben.

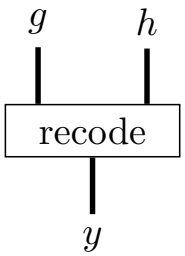


Abbildung 2.2: Blackbox Darstellung eines `recode`-Schaltkreises

Die andere Richtung funktioniert hier analog, und wir können ein Inverses `recode`-Gatter angeben. Dieses bekommt einen kodierten Input einer Gruppe G und spaltet diesen in Elemente aus den Gruppen N und Q auf. Hierbei ist zu beachten, dass $N \trianglelefteq G$ und $Q = G/N$ ist. Ein solches `splitCode`-Gatter kann mit Tiefe 1 realisiert werden und benötigt ausschließlich 2-stellige `Oder`-Gatter.

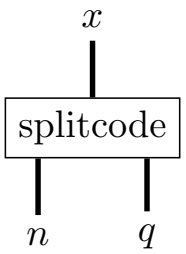


Abbildung 2.3: Blackbox Darstellung eines Inversen `recode`-Schaltkreises

2.5.2 `Und`-/`Oder`-Gatter

Solange $\texttt{MOD}_n$-Gatter mit $n \geq 3$ zur Verfügung stehen, kann jedes 2-stellige `Und`- bzw. `Oder`-Gatter durch ein $\texttt{MOD}_n$-Gatter ersetzt werden. Wir zeigen die Konversion anhand eines $\texttt{MOD}_3$-Gatters. Es gilt zu beachten, dass für $n \geq 3$ man $n - 2$ konstante Einsen in das $\texttt{MOD}_n$-Gatter hinzufügen muss, damit die Funktionalität zu einem `Und`-Gatters identisch bleibt. Im Falle eines `Oder`-Gatters muss nichts geändert werden, solange wir eine Negation am Output des `MOD`-Gatter zulassen. Ansonsten müssen wir den Output durch ein zweites `MOD`-Gatter negieren.

x_1	x_0	$x_0 \wedge x_1$	$(x_0 + x_1 + 1) \mod 3$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Tabelle 2.1: Wahrheitstafel `Und`-Gatter

			$\overbrace{\quad\quad\quad}^{A}$	
x_1	x_0	$x_0 \vee x_1$	$(x_0 + x_1) \mod 3$	$A \mod 3$
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

Tabelle 2.2: Wahrheitstafel `Oder`-Gatter

2.5.3 Zyklische Gruppen

Wir definieren einen Schaltkreis für eine zyklische Gruppe $C_p = \langle g \rangle$ unter Verwendung von $\texttt{MOD}_p$-Gattern wie folgt. Jedes Element $g^k \in C_p$ kodieren wir als x mit den einzelnen Bits i:

$$x_i \mapsto \begin{cases} 1 & \text{, falls } i \equiv k \mod p \\ 0 & \text{, sonst} \end{cases}$$

Wir können nun mit einem einzelnen $\texttt{MOD}_p$-Gatter überprüfen, ob die Summe multipler Inputs $x_i;\ i \in \mathbb{N}$, in Summe p ergibt oder nicht, indem wir die Bits der Inputs wie in Abb. 2.1 mit dem Gatter verbinden. Da wir auf diese Weise leicht überprüfen können, ob das Produkt der Elemente $(g_i)_{i\in[0:p]}$ dem neutralen Element e entspricht, können wir dies ausnutzen, um jedes andere Element mit einem eigenen $\texttt{MOD}_p$-Gatter zu berechnen. Dafür benötigt jedes $\texttt{MOD}_p$-Gatter einen eigenen Offset, um alle Zahlen zwischen 0 und $p-1$ abzudecken. Wir benötigen also insgesamt p viele $\texttt{MOD}_p$-Gatter, um die Gruppe zu berechnen. Wir zeigen dies schematisch mit drei Inputs $(x_i)_{i\in[0:2]}$, welche einen Output y liefern, welchen wir hier in seine einzelnen Bits $y_i;\ 0 \le i \le p$ aufteilen. In Abb. 2.4 wird dies schematisch dargestellt.

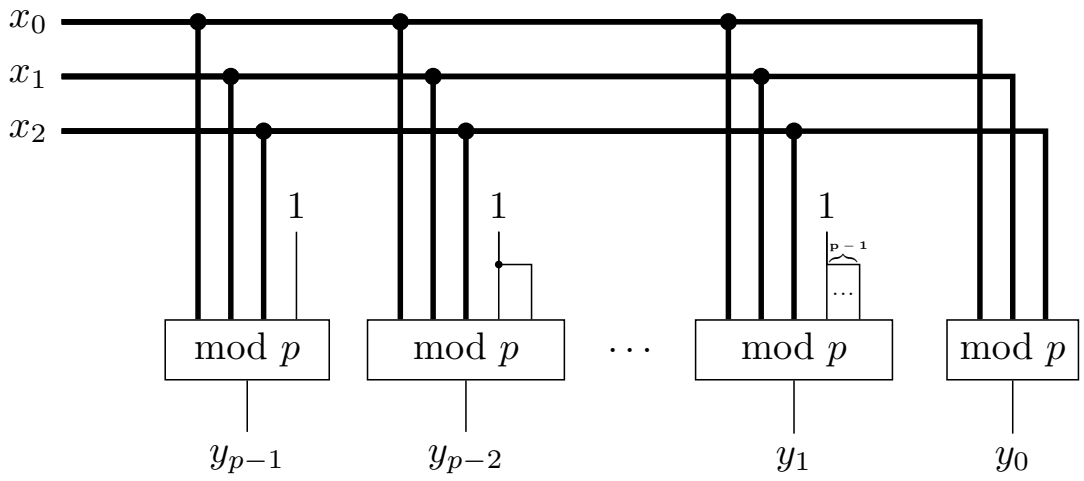


Abbildung 2.4: Schematische Darstellung zur Berechnung von C_p. Inputs kodiert / Output Einzelbits

Auf diese Weise können zyklische Gruppen mit Tiefe 1 und konstanter Breite berechnet werden. Es gilt noch zu beachten, dass für eine Gruppe C_p nicht $\texttt{MOD}_p$-Gatter verwendet werden müssen, sondern auch $\texttt{MOD}_q$-Gatter $\forall p \in \mathbb{N}\ \exists q \in \mathbb{N} : q \ge p$. Wir nutzen im späteren Verlauf die abkürzende Schreibweise aus Abb. 2.5.

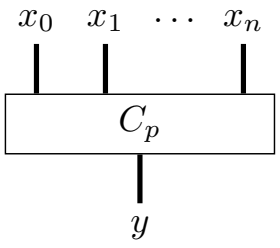


Abbildung 2.5: C_p Schaltkreis

Dieser Schaltkreis kann auf diese Weise mit beliebig vielen Inputs verwendet werden und ist bereits die Darstellung für die gesamte Schaltkreisfamilie von C_p, da MOD-Gatter mit beliebig vielen Inputs angesteuert werden können. Die Beschriftung C_p sollte im Speziellen angepasst werden (z. B. C_3) und impliziert die Verwendung von $\texttt{MOD}_p$-Gattern.

2.5.4 Abelsche Gruppen und direkte Produkte

Jede abelsche Gruppe kann durch ein direktes Produkt von endlich erzeugten abelschen Gruppen dargestellt werden. Auf diese Weise können wir für jede Gruppe, welche Teil des direkten Produkts ist, direkt ein C_p-Gatter (vgl. 2.5) nutzen oder induktiv wieder einen Schaltkreis für eine abelsche Gruppe angeben. Zu beachten ist, dass in jeder Stufe die Kodierung angepasst werden muss. Dies stellt kein Problem dar, weil die Kodierung nur abhängig von der Gruppe ist und somit in den Schaltkreis direkt eingebettet werden kann. Im Beispiel von $V_4 = C_2 \times C_2$, der Kleinschen Vierergruppe, müssen wir nach Berechnen der einzelnen Komponenten einen recode-Schaltkreis einfügen, welcher aus zwei 2 Bit Outputs, einen 4 Bit Output erzeugt. In Abb. 2.6 ist der Schaltkreis nochmals schematisch für drei Inputs gegeben. Es gilt zu beachten, dass beim direkten Produkt für Tupel $(x, y), (x', y')$ die Rechenvorschrift $(x, y)(x', y') = (xx', yy')$ gilt. Wir können auch $(x, y)(x', y') = (x, y) \odot (x', y')$ schreiben. Dies erklärt auch bereits weshalb wir außer den C_p Gattern nur zusätzlich ein recode-Gatter brauchen.

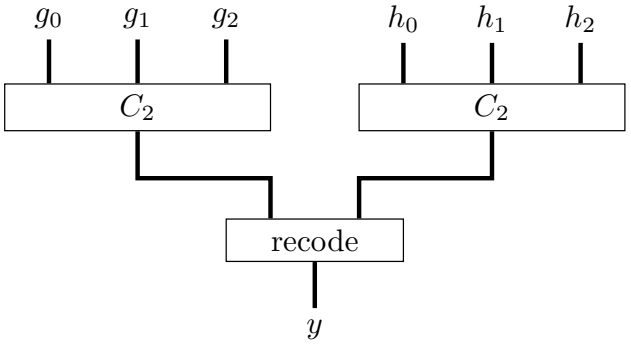


Abbildung 2.6: Schaltkreis zur Erkennung von V_4

In Abb. 2.7 sehen wir, wie solch eine Rekodierung Aussehen kann. Hier ist zu beachten, dass wir sowohl bei Input als auch bei Output die einzelnen Bits betrachten. Das recode-Gatter verrechnet die Inputs g_Π und h_Π, welche wir jeweils als Output aus den C_2-Gattern erhalten haben. Da die gegebene Gruppe nicht zyklisch ist, gilt hier zu beachten, dass bis auf die Kodierung des neutralen Elements - hier 0001 - die Kodierung nach Belieben angepasst werden kann. Für das Beispiel wurde sich für folgende Kodierung entschieden.

$$recod_{2,2} : \mathsf{Cod}_2 \times \mathsf{Cod}_2 \to \mathsf{Cod}_4$$

$$(g, h) \mapsto \begin{cases} 0001 & \text{, falls } g = h = 01 \\ 0010 & \text{, falls } (g = 01) \wedge (h = 10) \\ 0100 & \text{, falls } (g = 10) \wedge (h = 01) \\ 1000 & \text{, falls } g = h = 10 \end{cases}$$

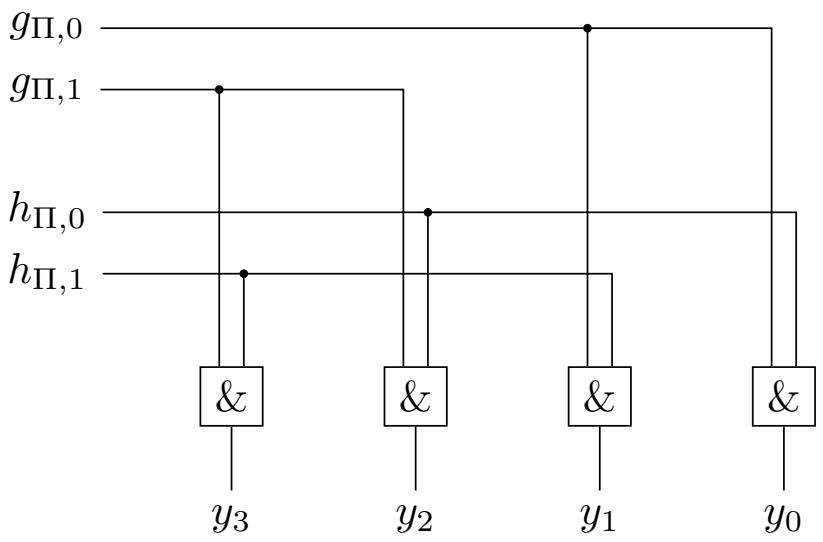


Abbildung 2.7: Recode Schaltkreis von V_4

2.5.5 Semidirekte Produkte

Eine Gruppe G kann als semidirektes Produkt von einer Gruppe N und Q dargestellt werden, als $N \rtimes Q = G$, falls $N \trianglelefteq G$ und $Q \leq G$. Für $G = N \cdot Q$ und $N \cap Q = \{ e \}$ gilt:

1. $Q \cong G/N$
2. $\forall g \in G \; \exists! \, n \in N \; \exists! \, q \in Q : g = nq$
3. $\varphi : Q \to \mathrm{Aut}(N), q \mapsto \varphi_q$ mit $\varphi_q : N \to N, n \mapsto qnq^{-1}$

Aufgrund der Berechnungsvorschrift $(n, q)(n', q') = (n\varphi_q(n'), qq') = (nqn'q^{-1}, qq')$, ist es hier nicht möglich die einzelnen Komponenten unabhängig zu berechnen. Die zweite Komponente des Tupels kann wie beim direkten Produkt unabhängig der ersten Komponente berechnet werden. Wir stellen für Tupel (n_i, q_i) mit $i \in [k]$ fest, wenn wir die Darstellung $n_i q_i$ verwenden:

$$\begin{aligned}
\prod_{i=1}^{k} n_i q_i &= n_1 q_1 n_2 q_2 \dots n_k q_k \\
&= n_1 \cdot \underbrace{q_1 n_2 q_1^{-1}}_{q_1 n_2} \cdot \underbrace{q_1 q_2 n_3 q_2^{-1} q_1^{-1}}_{q_1 q_2 n_3} \cdot \quad \dots \quad \cdot \underbrace{q_1 \dots q_{k-1} n_k q_{k-1}^{-1} \dots q_1^{-1}}_{q_1 q_2 \cdots q_{k-1} n_k} \cdot q_1 \dots q_k \\
&= \prod_{i=1}^{k} \prod_{j=1}^{i-1} q_j \; n_i \; \prod_{i=1}^{k} q_i
\end{aligned}$$

Auf diese Weise sehen wir, wie wir k Inputs berechnen können. Wir sehen leicht, dass jedes n_i konjugiert wird von einem q_i' was bedeutet, dass ${}^{q_i'} n_i = n_i' \in N$, da N ein Normalteiler in G ist. Somit erhalten wir nun wieder, wie beim direkten Produkt, zwei Komponenten welche wir unabhängig voneinander berechnen können. Somit ist $\prod_{i=1}^{k} (n_i, q_i) = \bigodot_{i=1}^{k} (n_i', q_i)$ und wir berechnen dies analog zum direkten Produkt.

Beispiel

Wir betrachten die symetrische Gruppe $S_3 = C_3 \rtimes C_2$ als semidirektes Produkt. Es gilt $C_3 \trianglelefteq S_3$, es sei $C_3 = \langle r \rangle$ und $C_2 = \langle s \rangle$, dann erhalten wir folgende Verknüpfungstafel.

$\cdot$	(e,e)	(r,e)	(r^2,e)	(e,s)	(r,s)	(r^2,s)
(e,e)	(e,e)	(r,e)	(r^2,e)	(e,s)	(r,s)	(r^2,s)
(r,e)	(r,e)	(r^2,e)	(e,e)	(r,s)	(r^2,s)	(e,s)
(r^2,e)	(r^2,e)	(e,e)	(r,e)	(r^2,s)	(e,s)	(r,s)
(e,s)	(e,s)	(r^2,s)	(r,s)	(e,e)	(r^2,e)	(r,e)
(r,s)	(r,s)	(e,s)	(r^2,s)	(r,e)	(e,e)	(r^2,e)
(r^2,s)	(r^2,s)	(r,s)	(e,s)	(r^2,e)	(r,e)	(e,e)

Tabelle 2.3: Verknüpungstafel $C_3 \rtimes C_2$

Nun lässt sich für k Elemente der Schaltkreis aus Abb. 2.9 konstruieren. In Tabelle 2.4 und Tabelle 2.5 sehen wir nochmals alle möglichen Konjugationen der Elemente aus C_3 mit C_2. Basierend hierauf geben wir in Abb. 2.8 die interne Struktur des `conj`-Gatters an.

${}^{e}e$	$=$	e	${}^{s}e$	$=$	e
${}^{e}r$	$=$	r	${}^{s}r$	$=$	r^2
${}^{e}r^2$	$=$	r^2	${}^{s}r^2$	$=$	r

Tabelle 2.4: Konjugation für C_3

${}^{01}001$	$=$	001	${}^{10}001$	$=$	001
${}^{01}010$	$=$	010	${}^{10}010$	$=$	100
${}^{01}100$	$=$	100	${}^{10}100$	$=$	010

Tabelle 2.5: Konjugation für C_3 kodiert

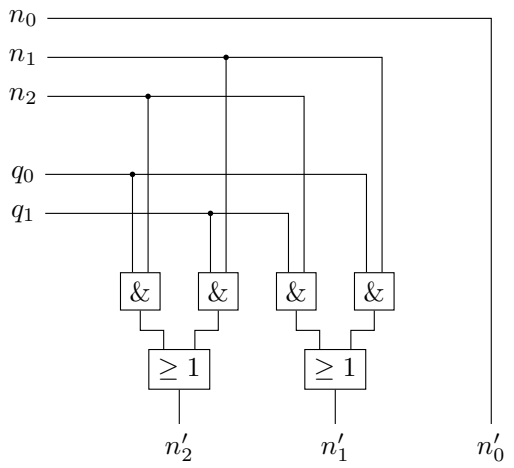


Abbildung 2.8: Konjugations-Schaltkreis für S_3

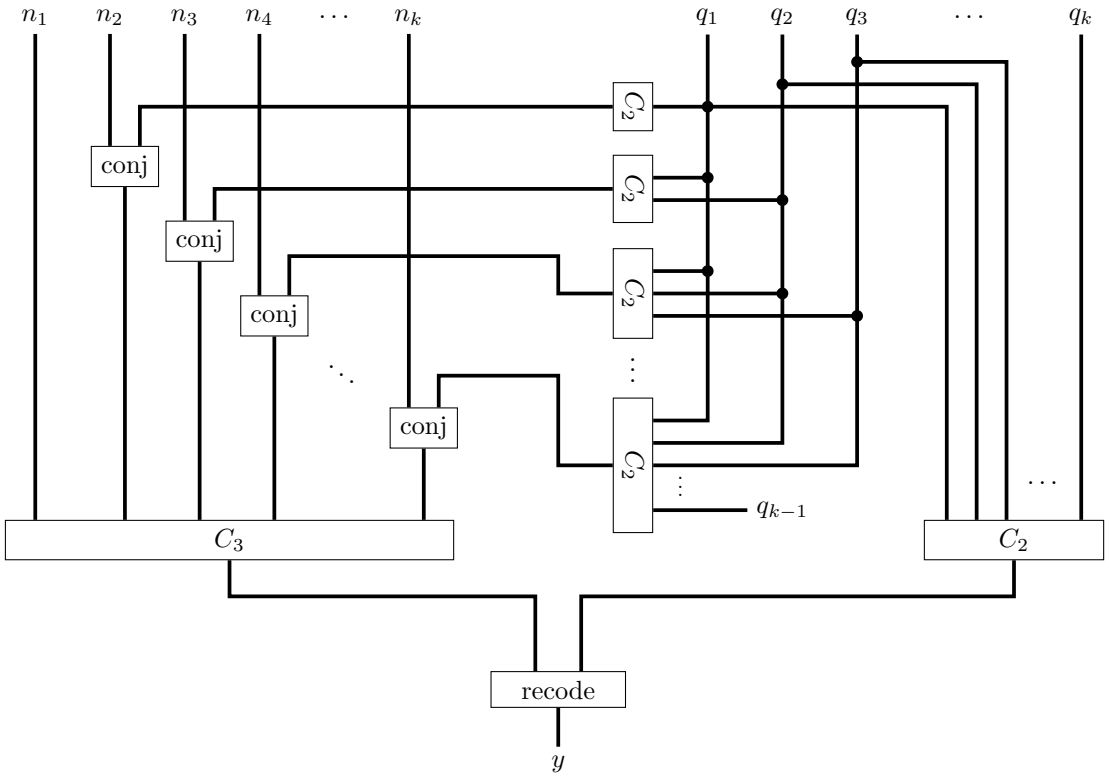


Abbildung 2.9: Schematische Darstellung zur Berechnung von S_3

2.6 Kranzprodukt

Das Kranzprodukt von einer Gruppe $G \wr H$ ist definiert als das semidirekte Produkt von $\underbrace{(G \times G \times \cdots \times G)}_{|H|-\text{ mal}} \rtimes H$. Wir können auch sagen $G^H \rtimes_\varphi H$.

Das φ ist $\varphi : H \to \text{Aut}(G^H)$. Es wird hier also ein Element von H auf eine Funktion, beziehungsweise einen Automorphismus, abgebildet. Wichtig ist hierbei, dass φ ein Homomorphismus ist. φ wird meist in der Schreibweise weggelassen, da es einfach definiert ist als:

$$\begin{aligned} \varphi : H &\to \text{Aut}(G^H) \\ h &\mapsto \varphi_h \end{aligned}$$

Wir definieren nun φ_h für $\forall h \in H$:

$$\begin{aligned} \varphi_h : G^H &\to G^H \\ f &\mapsto g = \varphi_h(f) = {}^h f \end{aligned}$$

Wir bilden also ein Element von G^H auf ein anderes solches Element ab. Wir stellen aber fest, dass wir G^H auch als Menge von Funktionen darstellen können, und zwar als $G^H = \{f : H \to G\}$. Das liegt daran, dass ein $|H|$-Tupel mit $|G|$ vielen Möglichkeiten pro Stelle, genau $|G|^{|H|}$ viele verschiedene Möglichkeiten bietet und es auch nur genau so viele Funktionen von $f : H \to G$ gibt, weil jedem Input $|G|$ viele verschiedene Möglichkeiten zugeordnet werden können.

Da wir wissen, dass Elemente von G^H als Funktionen aufgefasst werden können, definieren wir nun noch $\forall h \in H, f \in G^H$:

$$\varphi_h(f)(x) = {}^h f(x) = f(h^{-1}x)$$

Wir zeigen, dass φ ein Homomorphismus ist:

$$\varphi(hk) = \varphi_{hk} = \varphi_h \circ \varphi_k = \varphi(h)\varphi(k)$$

Es folgt $\forall x \in H$:

$$\begin{aligned}\varphi_{hk}(f)(x) &= f((hk)^{-1}x) = f(k^{-1}h^{-1}x)\\ &= \varphi_k(f(h^{-1}x)) = \varphi_h(\varphi_k(f))(x)\\ &= (\varphi_h \circ \varphi_k)(f)(x)\end{aligned}$$

Desweiteren bildet das neutrale Element aus H auf das neutrale Element aus $\mathrm{Aut}(G^H)$ ab:

$$\varphi(e) = \varphi_e$$

Es folgt $\forall x \in H$:

$$\varphi_e(f)(x) = f(e^{-1}x) = f(ex) = f(x)$$

Somit ist gezeigt das φ ein Homomorphismus ist.

Wir stellen nun noch einmal die erste Homomorphismus Eigenschaft in anderer Form dar, weil wir sie in dieser Form später nutzen.

$$\begin{aligned}{}^{(hk)} f(x) &= f((hk)^{-1}x) = f(k^{-1}h^{-1}x)\\ &= {}^k f(h^{-1}x) = {}^h({}^k f(x))\end{aligned}$$

Für zwei Elemente $(f, a), (g, b) \in G \wr H$ ist die zweistellige Operation $\cdot : (G \wr H)^2 \to G \wr H$ wie folgt definiert.

$$(f, a)(g, b) = (f\ {}^a g, ab)$$

Lemma 2.6.1
Seien $f, g, h \in G^H$ und $x, y, z \in H$. Sei weiter e_A das neutrale Element in G^H und e_H das neutrale Element in H. [BMMN98]

1. $(f, x)(e_A, e_H) = (f, x) = (e_A, e_H)(f, x)$
2. $((f, x)(g, y))(h, z) = (f, x)((g, y)(h, z))$
3. $(e_A, x)(f, e_H)(e_A, x)^{-1} = ({}^x f, e_H)$

Beweis. (1) folgt aus Definition. (2) ergibt sich durch direktes Nachrechnen wie folgt

$$\begin{aligned}((f, x)(g, y))(h, z) &= (f\ {}^x g, xy)(h, z) = (f\ {}^x g\ {}^{xy} h, xyz)\\ &= (f\ {}^x g\ {}^x({}^y h), xyz) = (f\ {}^x(g\ {}^y h), xyz)\\ &= (f, x)(g\ {}^y h, yz) = (f, x)((g, y)(h, z))\end{aligned}$$

(3) folgt aus direktem Nachrechen analog zu (2).

3 Beweis $\mathsf{CC}^0 = P(solvableGroup)$

3.1 $\mathcal{P}(solvableGroup) \subseteq \mathsf{CC}^0$

Wir zeigen per Induktion, dass eine beliebige auflösbare Gruppe G mit CC^0 Schaltkreisen $\mathcal{P}$-erkannt werden kann. Zunächst beweisen wir, dass man für jede beliebige auflösbare Gruppe G einen Schaltkreis angeben kann, welcher k kodierte Inputs g_i $i \in [k]$ verrechnen kann und als Ausgabe $g' = g_1 g_2 \dots g_k$ liefert. Es gilt $G_1 \trianglelefteq G_2 \trianglelefteq \dots \trianglelefteq G_n = G$ mit G_i auflösbar für $i \in [n]$. Es gilt für einen Normalteiler $N \trianglelefteq G$ und einen Quotienten $Q = G/N$ wieder, dass diese auflösbare Gruppen sind. Im Allgemeinen können wir nicht von $G = N \rtimes Q$ ausgehen, jedoch erfolgt die Konstruktion für G auf diese Weise nach demselben Prinzip. Diese Restriktion verschafft uns den Vorteil, dass unser Quotient Q immer eine abelsche Gruppe ist, was wiederum nur eine zyklische Gruppe oder ein direktes Produkt aus zyklischen Gruppen ist.

Es möchte noch angemerkt werden, dass wir ohne Beschränkung der Allgemeinheit davon ausgehen können, ausschließlich mit einem $\mathtt{MOD}_r$-Gatter zu arbeiten. Da jede auflösbare Gruppe eventuell in ein Produkt zyklischer Gruppen aufgebrochen wird. Durch Ihre Normalreihe wissen wir bereits welche MOD-Gatter für jede zyklische Gruppe benötigt werden. Es ist daher naheliegend das kgV dieser Zahlen zu bilden, um r zu erhalten. Wir können nun davon ausgehen, dass jedes $\mathtt{MOD}_p$-Gatter, jeden einzelnen Input $\frac{r}{p}$-mal vervielfacht. Auf diese Weise agiert ein $\mathtt{MOD}_r$-Gatter wie ein $\mathtt{MOD}_p$-Gatter. Duplizierte Inputs stellen zudem kein Problem dar, da sie weder die Tiefe des Schaltkreises beeinflussen, noch die Anzahl der erlaubten Inputs an den MOD-Gattern übersteigen, da alle MOD-Gatter unbounded-fan-in sind. Wir sehen also, es ist problemlos möglich eine Vielzahl unterschiedlicher MOD-Gatter zu nutzen und wir müssen uns in der Bezeichnung auch nicht auf ein p beschränken, da es nur eine Formalität ist alle Gatter zu normieren.

Induktionsanfang

Als Induktionsanfang können wir jede beliebige einfache endliche abelsche Gruppe wählen, also jede zyklische Gruppe mit Prim Ordnung. Wir haben bereits eine allgemeine Konstruktion für zyklische Gruppen in 2.5.3 gegeben.

Induktionsvoraussetzung

Als Induktionsvoraussetzung setzen wir nun, $\exists\, G_{n-1} \trianglelefteq G_n, H \leq G_n : G_n = G_{n-1} \rtimes H$ es existiert ein Schaltkreis für H und G_{n-1} mit konstanter Tiefe.

Induktionsschritt

Wir geben nun eine Konstruktion für $G_{n+1} = G_n \rtimes H$ an. Sei $(x_i)_{i\in[0:k]}$ eine Folge von Inputs mit $x_i \in G_{n+1}$. Wir können ohne Beschränkung der Allgemeinheit annehmen, dass $(x_i) = (g_i, h_i)$ mit $g_i \in G_n$ und $h_i \in H$. Da per Annahme H immer nur abelsch ist, brauchen wir hier keinen größeren Aufwand, da es mit einem Schaltkreis analog zu Abb. 2.5 oder Abb. 2.6 berechnet werden kann. Um G_{n+1} zu berechnen, müssen wir jedes g_i mit $\prod_{j=0}^{i-1} h_j$ konjugieren und anschließend in den Schaltkreis für G_n einspeisen. Wie bereits in Abb. 2.3 können wir die Inputs in die Kodierung der kleineren Erzeugergruppen aufteilen. Dies nutzen wir, um die konjugierten g Terme wiederum aufzuspalten, falls G_n nicht bereits zyklisch ist wie in Abb. 2.9. So können wir die Induktionsvoraussetzung hier nutzen, da wir bereits einen Schaltkreis für G_n besitzen. In Abb. 3.1 ist nochmal die Konstruktion abgebildet, wobei wir die `splitCode`-Gatter hier zur Übersichtlichkeit weggelassen haben. Wir können davon ausgehen, dass diese direkt nach den `conj`-Gatter platziert werden und die Inputs für den G_n Schaltkreis wieder Tupel sind. Somit ist dies ein valider Schaltkreis für G_{n+1}.

Auf diese Weise ist es möglich einen Schaltkreis für komplexere Gruppen zu bauen. So kann man z.B. über die Kette, $C_2 \trianglelefteq V_4 \trianglelefteq A_4 \trianglelefteq S_4$, leicht einen Schaltkreis für S_4 angeben. Jedoch ist die Tiefe eines solchen Schaltkreises meist nicht optimal. Für das gegebene Beispiel würde man $S_4 = A_4 \rtimes C_2$, dann $A_4 = V_4 \rtimes C_3$ und $V_4 = C_2 \times C_2$ darstellen. Wir stellen fest, dass jeder H Schaltkreis, nach vorangegangener Bezeichnung, immer direkt in einer Ebene dargestellt werden kann, sodass die Gesamttiefe des Schaltkreises hauptsächlich von G beeinflusst wird.

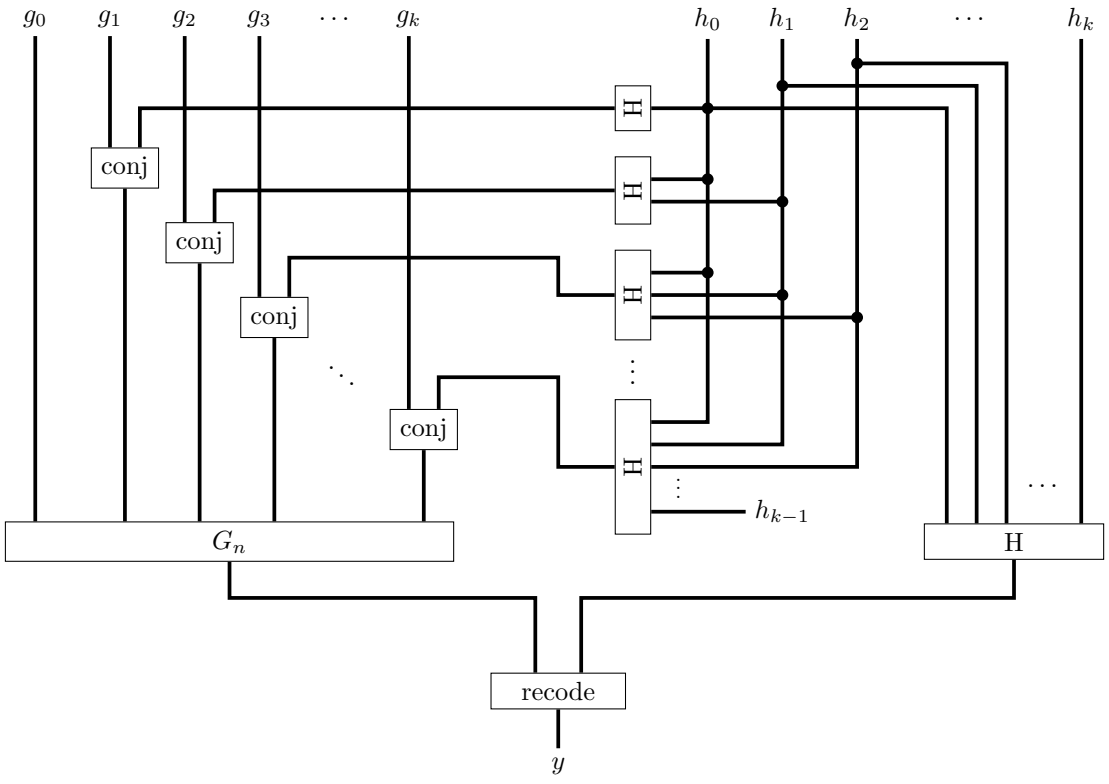


Abbildung 3.1: Schaltkreis für G_{n+1}

Wir betrachten nun noch den allgemeinen Fall. Wie bereits erwähnt müssen wir in der Regel davon ausgehen, dass wir nicht immer solch eine Kette vorliegen haben. In solch einem Fall kann nicht jede Stufe einfach als semidirektes Produkt dargestellt werden, insbesondere wenn wir versuchen den Schaltkreis möglichst flach zu halten. Ein Beispiel für eine Gruppe welche nicht durch ein semidirektes oder direktes Produkt dargestellt werden kann, ist die Quaternionen Gruppe Q_8. In solchen Fällen ist $Q = G/N$ nicht unbedingt eine Untergruppe von G, kann jedoch mit einem Repräsentantensystem in G eingebettet werden.

Im Allgemeinen gilt, falls $N \trianglelefteq G$, dann existiert eine Abbildung $\pi : G \twoheadrightarrow Q;\ g \mapsto gN$, und π ist ein Epimorphismus. Für ein Repräsentantensystem $R \subseteq G$ für Q gilt, $\forall q \in Q : \left|\pi^{-1}(q) \cap R\right| = 1$ und $\forall q \in Q \exists!\, r_q \in R : \pi(r_q) = q$. Es sei $f_Q : Q \to R$, die Repräsentantenfunktion von Q bijektiv. Sei $\psi : N \times R \to G; (n, r) \mapsto nr$, ist bijektiv. Es gilt $\forall g \in G\ \exists!\, r \in R\ \exists!\, n \in N : g = nr$. Somit können wir unter Verwendung von N und R beliebige Elemente aus G berechnen. Für $g_i \in G, n_i \in N, r_i \in R$ gilt:

$$\begin{aligned}
\prod_{i=0}^{k} g_i &= \prod_{i=0}^{k} n_i r_i = n_0 r_0 n_1 r_1 \dots n_k r_k \\
&= n_0\ r_0 n_1 r_0^{-1}\ r_0 r_1 n_2 \quad \dots \quad r_0 \dots r_{k-1} n_k r_{k-1}^{-1} \dots r_0^{-1}\ r_0 \dots r_k \\
&= n_0\ {}^{r_0}n_1\ {}^{r_0 r_1}n_2\ {}^{w}iederr_0 \dots r_{k-1} n_k\ r_0 \dots r_k \\
&= \prod_{i=0}^{k} {}^{\prod_{j=0}^{i-1} r_j} n_i \prod_{i=0}^{k} r_i = \prod_{i=0}^{k} \tilde{n}_i \prod_{i=0}^{k} r_i \neq g
\end{aligned}$$

Wir sehen, dass dies wieder ähnlich dem semidirekten Produkt berechnet werden könnte. Demnach könnten wir auch hier wieder Komponentenweise alle $\tilde{n}_i$ und alle r_i berechnen und würden wieder g erhalten. Da aber $\exists\, r, r' \in R : rr' \notin R$, aber $\forall\, r, r' \in R : rr' \in G$ müssen wir diese Repräsentanten zunächst nach Q abbilden, um in Q die Repräsentanten zu verrechnen, um dann den korrekten Repräsentanten zu bestimmen. Daher berechnen wir $o_i = \pi(\prod_{j=0}^{i-1} r_j)$, somit ist $o_i \in Q$. Wir brauchen aber den korrespondieren Repräsentanten der Nebenklasse in Q und erhalten diesen mit $f_Q(o_i) = a_i \in R$. Jedoch gibt es noch ein letztes Problem, nämlich dass ${}^{a_i}n_i$ verschoben sein kann. Wir brauchen daher noch einen Korrekturterm $w_i = a_i r_i a_{i+1}^{-1} \in N$, um keine Rechenfehler zu akkumulieren. Da ${}^{a_i}n_i \in N$ und $w_i \in N$ ist damit auch ${}^{a_i}n_i w_i \in N$. Es ergibt sich die neue Formel:

$$\prod_{i=0}^{k} g_i = \prod_{i=0}^{k} {}^{a_i}n_i w_i \prod_{i=0}^{k} r_i = \bigodot_{i=0}^{k} ({}^{a_i}n_i w_i, r_i) = \bigodot_{i=0}^{k} (n_i', r_i)$$

Es kann leicht ein Schaltkreis mit konstanter Tiefe, für die Funktion f_Q, angegeben werden. Da $|R| = [G : N]$ verhält sich die Kodierung hier wie bei Abb. 3.1 für H. Intern in den H Schaltkreisen fügen wir ein Gatter für $\pi(r_i) = q_i$ ein, welches auch in konstanter Tiefe möglich ist und bestimmen in dem Schaltkreis für Q das Ergebnis, welches wir mit f_Q auf ein Element aus R abbilden. Diese Elemente sind wieder unsere a_i welche wir mit den n_i in den `conj`-Gattern verrechnen. Als Letztes brauchen wir ein zusätzliches Gatter um w_i zu berechnen. Wir haben bereits die Outputs a_i und können diese auch negieren. Zudem können wir ein Gatter konstanter Tiefe mit Output w_i angeben, da es nur endlich viele Möglichkeiten für w_i gibt. Nun können die berechneten ${}^{a_i}n_i$ Outputs der `conj`-Gatter und die berechneten w_i in das G_n-Schaltkreis gespeist werden. Somit erhalten wir eine Konstruktion für den allgemeinen Fall und können auf diese Weise jede auflösbare Gruppe mithilfe

einer Normalreihe berechnen. Als Letztes stellen wir fest, dass sich diese Konstruktion für jede beliebige Anzahl an Inputs auf diese Weise bilden lässt und können so für jede Eingabelänge k einen eigenen Schaltkreis angeben.

Da wir nun wissen, dass man beliebige Elemente verrechnen kann, müssen wir nun abschließend unser Programm P mit einem solchen Schaltkreis verbinden. Sei also P ein Programm mit m Inputs, dann hat P die Länge k. Wir müssen also für Inputs $w \in \Sigma$ die Ergebnisse $x \in G$ berechnen. Wir verwenden auch für w's eine One-Hot-Kodierung und stellen fest, dass wenn f nicht injektiv ist `Oder`-Gatter benötigt werden und sonst ohne Gatter umkodiert werden kann (vgl. Abb. 3.2). Sei (i, f) eine Instruktion aus P, dann kann f in den Schaltkreis vor x_i eingebettet werden. Wir erhalten somit unsere Inputs $x_i; i \in [0 : k]$ welche mit `splitCode`-Gattern in die erforderten g_i, h_i aufgeteilt werden. Mit dem Ergebnis des Schaltkreises aus Abb. 3.1 müssen wir nur noch aus dem kodierten Ergebnis einen singulären Wahrheitswert errechnen. Es gilt $F \subseteq G$ enthält alle Elemente, welche erkannt werden müssen. Jetzt müssen nur diejenigen Bits in einem `Oder`-Gatter akkumuliert werden, welche für alle Elemente aus F mit einer 1 belegt sind. Dies liefert uns schließlich das Ergebnis als Wahrheitswert, unabhängig davon ob die Eingabe erkannt wird oder nicht.

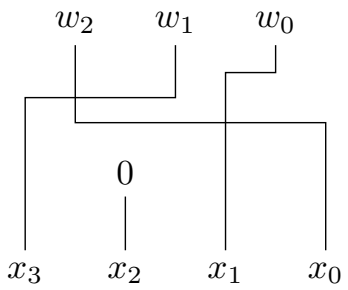
Als Letztes betrachten wir noch die Tiefe des Schaltkreises und überprüfen, ob wir immer noch konstante Tiefe haben. Zuallererst muss die Eingabe in Inputs für den Schaltkreis umgewandelt werden. In 3.2 sehen wir, dass die Tiefe dieser Komponente nur von der gewählten Kodierung abhängt, es kann hier bis zu einer $\log_2$ tiefen Verschachtelung von 2-stelligen `Oder`-Gattern kommen, bezüglich der Größe der Kodierung. Jedoch ist dies wiederum nur konstant, da sich die Kodierung nicht mit der Eingabelänge verändert. Als Nächstes wird ein Element in einem `splitCode`-Gatter aufgespalten, welches nur konstante Tiefe für die Konversion zwischen Kodierungen braucht. Dasselbe trifft auch für `conj`-Gatter, `recode`-Gatter, Schaltkreise zur Berechnung von Repräsentanten und Schaltkreise zur Berechnung von Ausgleichstermen zu. Zudem gilt, dass die Gruppe sich nicht ändert und die gesamte Berechnung von Gruppenelementen somit feste Tiefe besitzt. Die letzte Komponente ist der Schaltkreis zur Erkennung der Eingabe, welcher unabhängig von der Eingabe seine Tiefe hat. Da dieser letzte Schritt bis auf wenige Grenzfälle - G ist trivial - immer eine surjektive Abbildung ist, müssen auch hier nur im Worst-Case $\log_2$ tief verschachtelte 2-stellige `Oder`-Gatter verwendet werden bezüglich der Kodierung von G. Somit hat jede Komponente des Schaltkreises konstante Tiefe und somit auch der ganze Schaltkreis.

Als Beispiel betrachten wir zwei Funktionen $f_1, f_2 \in {C_4}^{\Sigma}$ mit $\Sigma = \{a, b, c\}$ und $C_4 = \{0, 1, 2, 3\}$. Wir kodieren $a = 001$, $b = 010$, $c = 100$ und C_4 wie gewohnt. w_i bezeichne die Bits für $w \in \Sigma$ und x_i die Bits für $x \in C_4$.

$$
\begin{aligned}
f_1 : \Sigma &\rightarrowtail C_4 \\
a &\mapsto 1 \\
b &\mapsto 3 \\
c &\mapsto 0
\end{aligned}
$$

$$
\begin{aligned}
f_2 : \Sigma &\to C_4 \\
a, c &\mapsto 1 \\
b &\mapsto 2
\end{aligned}
$$

f_1:



f_2:

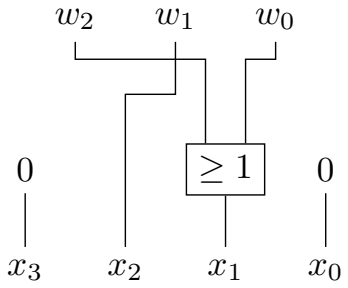


Abbildung 3.2: Schematische Schaltkreise für f_1 und f_2

Somit haben wir gezeigt, dass ein beliebiges Programm P über einer auflösbaren Gruppe G in einen Schaltkreis mit konstanter Tiefe konvertiert werden kann und dabei ausschließlich `MOD`-Gatter verwendet. Es gilt $P(solvableGroup) \subseteq \mathsf{CC}^0$.

■

3.2 $\mathcal{P}(solvableGroup) \supseteq \mathsf{CC}^0$

Wir zeigen für jeden Schaltkreis C, existiert ein M-Programm, sodass das Ergebnis des Programms in der Akzeptanzmenge $F \subseteq G$ liegt, falls C `true` liefert und sonst nicht. Ohne Einschränkung können wir von einem Schaltkreis mit 1 Bit Output ausgehen, da jeder Schaltkreis mit multiplen Outputs bzw. multi-Bit Output als mehrere einzelne Schaltkreise betrachtet werden und jedes einzelne Output Bit isoliert betrachtet werden kann. Wir können stets davon ausgehen, dass ein Gatter auf Ebene $i + 1$ Outputs aus Ebene i als Inputs bekommt. Es macht hier keinen Unterschied ob diese Inputs tatsächlich Outputs aus Ebene i sind, da Inputs sowieso nur 1 oder 0 sein können.

Induktionsanfang

Tiefe 1:

Für einen Schaltkreis der Tiefe 1, gibt es nur die Möglichkeit aus einem einzelnen `MOD`-Gatter zu bestehen. Wir können also direkt eine zyklische Gruppe nutzen deren Ordnung dem `MOD`-Gatter entspricht. Wir können somit leicht ein Programm angeben, indem wir jede Inputvariable des Gatters als Programmzeile abbilden und wir können jedes Mal die gleiche Funktion verwenden. Zusätzlich ist es nicht notwendig die Instruktionen zu restrukturieren oder zu vervielfachen. Jedoch müssen wir multi-Inputs einzelner Variablen berücksichtigen. Sei der Schaltkreis ein $\texttt{MOD}_p$-Gatter mit n Inputs. Sei $G = \langle g \rangle$ zyklisch und $|G| = p$. Wir definieren als Alphabet $\Sigma = \{ 0, 1 \}$ und $f : \Sigma \to G$. Unser Programm P muss nun für jede Eingabelänge definiert werden. Wir erhalten das n-Input Programm $\phi_n = r_1 \dots r_n$. Zeilen geben wir in der Form $r = (i, f(0), f(1))$ an. Wir erhalten somit:

$$\begin{aligned} r_1 &= (1, e, g^{\alpha_1}) \\ &\vdots \\ r_n &= (n, e, g^{\alpha_n}) \end{aligned}$$

Wir sehen für jedes $w = a_1 \dots a_n \in \Sigma^n$, gilt $\phi_n(w) = g^s$ mit $s = \sum_{i=1}^{n} \alpha_i \, |a_i|_1$. Sei nun $\forall n, m \in \mathbb{N} : F_n = F_m = \{ e \}$. Wir haben somit ein Programm angegeben, welches die Sprache $L = \{ w \in \Sigma^* \mid \sum_{i=1}^{n} \alpha_i \, |a_i|_1 \equiv 0 \mod p \}$ erkennt, was exakt dem Output des Schaltkreises entspricht.

Tiefe 2:

Wir zeigen noch für einen Schaltkreis der Tiefe 2, wie sich das Programm zur Berechnung des Schaltkreises ergibt. Wie bereits im Programm für einen Schaltkreis der Tiefe 1, gehen wir nun wieder von einem einzelnen `MOD`-Gatter aus, welches beliebig viele Inputs aus der Ebene zuvor bekommt vgl. 3.3. Hierbei gibt es keinen Unterschied, ob diese Inputs Outputs aus der vorherigen Ebene oder neue Inputs sind. Analog zur anderen Richtung können wir wieder davon ausgehen, dass die Outputs der vorherigen Ebene, bezüglich desselben `MOD`-Gatters berechnet wurde, da man auch hier wieder das kgV der einzelnen Gatter bilden kann. Somit sind alle Inputs unseres Gatters normiert.

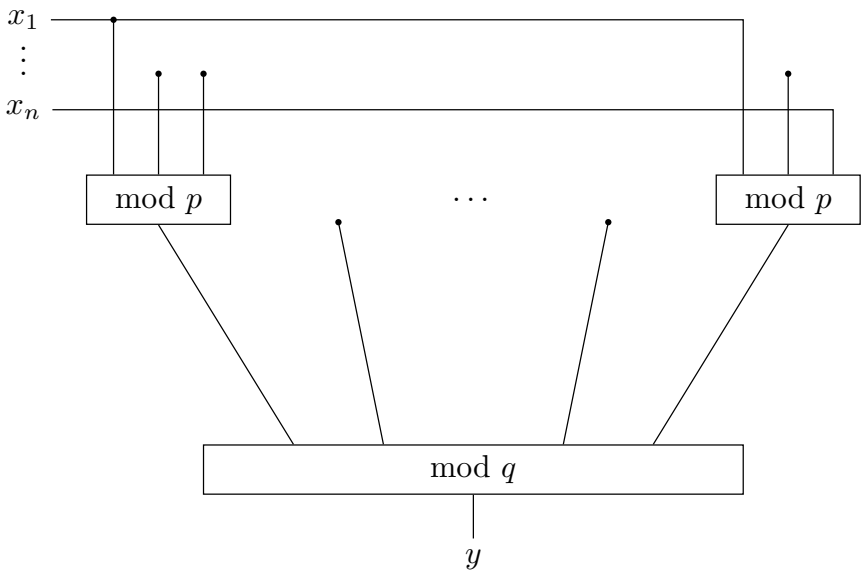


Abbildung 3.3: Grobe Illustration für einen Schaltkreis der Tiefe 2

Sei unser Gatter ein $\texttt{MOD}_q$-Gatter welches Inputs aus $\texttt{MOD}_p$-Gattern erhält, welche wir wieder als Teilschaltkreise $\mathfrak{C}_i$ bezeichnen mit den zugehörigen Programmen P_i. Die Programme P_i folgen dem selben Aufbau des Programms für Tiefe 1. Die Outputs der Schaltkreise $\mathfrak{C}_i$ sind $\mathfrak{C}_i(a_q, \ldots, a_n) = 1 \iff P_i(a_1, \ldots a_n) \in F$. Wir bilden also das Kranzprodukt $C_q \wr C_p$ für die Gruppen $C_p = \langle g \rangle, C_q = \langle h \rangle$ und konstruieren ein Programm über dieser Gruppe. Wir müssen also in unseren Programmen P_i alle $x \in C_p$ durch $((e, \ldots, e), g) \in C_q \wr C_p$ ersetzen. Für bessere Lesbarkeit definieren wir zwei Einbettungen und setzen $P_i := P_i(x_1, \ldots, x_n)$.

$$\begin{aligned} \eta_p : C_p &\hookrightarrow C_q \wr C_p \\ e &\mapsto ((e, \ldots, e), e) \\ x &\mapsto ((e, \ldots, e), x) \end{aligned} \qquad \begin{aligned} \eta_q : C_q &\hookrightarrow C_q \wr C_p \\ e &\mapsto ((e, \ldots, e), e) \\ x &\mapsto ((x, e, \ldots, e), e) \end{aligned}$$

Somit erhalten wir Programme P_i mit Programmzeilen der Form $(i, \eta_p(e), \eta_p(g^{\alpha_i}))$. Diese Programme geben nun 1 aus, falls $P_i \in F'$ und 0 sonst, mit $F' = \{ \eta_p(e) \}$. Nun müssen wir ein neues Programm P konstruieren, welches $(\eta_q(h))$ für jedes $P_i \in F'$ setzt und die erste Komponente

des $C_q^{C_p}$-Tupels auf e sonst. Dies können wir leicht durch das Kommutieren von $\eta_q(h)^{-1}$ und P_i erreichen und zusätzlich mit $\eta_q(h)$ multiplizieren.

$$\begin{aligned}\eta_q(h) \cdot [\eta_q(h)^{-1}, \eta_p(e)] &= \eta_q(h) \cdot \eta_q(h)^{-1} \cdot \eta_p(e) \cdot \eta_q(h) \cdot \eta_p(e)^{-1} \\ &= \eta_q(h) \cdot \eta_q(h)^{-1} \cdot \left({}^{\eta_p(e)}\eta_q(h)\right) \\ &= ((e, \ldots, e), e) \cdot ((h, \ldots, e), e) \cdot ((e, \ldots, e), e) \\ &= ((h, \ldots e), e) = \eta_q(h)\end{aligned}$$

$$\begin{aligned}\eta_q(h) \cdot [\eta_q(h)^{-1}, \eta_p(x)] &= \eta_q(h) \cdot \eta_q(h)^{-1} \cdot \eta_p(x) \cdot \eta_q(h) \cdot \eta_p(x)^{-1} \\ &= \eta_q(h) \cdot \eta_q(h)^{-1} \cdot \left({}^{\eta_p(e)}\eta_q(h)\right) \\ &= ((e, \ldots, e), x) \cdot ((h, \ldots, e), e) \cdot ((e, \ldots, e), x^{-1}) \\ &= ((e, \ldots, h, \ldots, e), e)\end{aligned}$$

Somit ist unsere Annahme erfüllt und wir erhalten h in der ersten Komponente des Tupels, falls $P_i \in F$ und e sonst. Wir setzen nun $\tilde{h} = \eta_q(h)$ und können nun das Programm P angeben. $P = \tilde{h}[\tilde{h}^{-1}, P_1] \cdot \ldots \cdot \tilde{h}[\tilde{h}^{-1}, P_k]$. Da $\tilde{h}$ eine Konstante ist, geben wir für $\tilde{h}$ Programmzeilen der Form $(0, \tilde{h})$ an. Das gesamte Programm ist somit nur eine Folge von all solchen Instruktionen, welche wir als P in Konkatenation dargestellt haben.

Es ist $F \leq C_q^{C_p}$ mit $F = F_t \times F_f = \{ e \} \times \{ x \in C_q \}^{|C_p|-1}$. Wir können hier die Komponente aus C_p ignorieren, da durch das Kommutieren diese Komponente stets e ist, demnach ist nur noch die erste Komponente relevant. Zudem teilen wir F' in zwei Abschnitte. Wir akzeptieren, falls die erste Komponente hier e ist und die restlichen Elemente des Tupels sind nicht von Bedeutung. Somit haben wir für einen Schaltkreis der Tiefe 2 ein Programm angegeben, welches 1 ausgibt, falls $P \in F'$ und 0 sonst. Es möchte angemerkt werden, dass die korrekte Zuweisung von Variablen nicht eingeschränkt wird, da wir für jeden Teilschaltkreis unbenutzte Variablen mit $\alpha = 0$ verzeichnen können. Dies führt im Programm zu Zeilen der Form $(i, \eta_p(e), \eta_p(e))$, was zwar das Programm verlängert, aber nichts an den Berechnungen ändert. Dies erlaubt uns eine leichtere Konstruktion des Programms. Zuletzt bemerken wir, dass das Programm extrem in Länge zunimmt, dies aber nur auf Basis des Schaltkreises basiert und die Länge des Programms bezüglich der Eingabe konstant ist. Die Länge ist somit polynomiell und P ist damit ein valides $(C_q \wr C_p)$-Programm.

Induktionsvoraussetzung

Sei $\mathfrak{C}$ ein Schaltkreis der Tiefe m, dann existiert ein Programm P polynomialer Länge über einer auflösbaren Gruppe G mit einer Untergruppe F, sodass F die akzeptierende Menge für P ist. Es gilt $\mathfrak{C}$ und P erkennen dieselbe Sprache. Es ist also $\mathfrak{C}(a_0, \ldots a_n) = 1 \iff P(a_0, \ldots, a_n) \in F$ und $F \leq G$.

Induktionsschritt

Wir zeigen ähnlich der Tiefe 2 Konstruktion, wie ein Programm für einen Schaltkreis der Tiefe $m + 1$ konstruiert werden kann. Zunächst definieren wir $\mathfrak{C}_{i,j}$ wobei i die Tiefe und j das Gatter selbst bezeichnet. Demnach ordnen wir $\mathfrak{C}_{i,j}$ das Programm $P_{i,j}$ zu. Wir konstruieren nun für den

Schaltkreis $\mathfrak{C}$ mit den Inputs $\mathfrak{C}_{m,j}$ $j \in [k]$ das Programm P unter Verwendung der Programme $P_{m,j}$ $j \in [k]$. Es gilt auch hier wieder, dass die einzelnen Ebenen per kgV normalisiert werden können und wir daher von einheitlichen Inputs ausgehen können. Die Programme $P_{m,j}$ sind über einer Gruppe R definiert, welche ein iteratives Kranzprodukt zyklischer Gruppen ist. Sei das Gatter der Tiefe $m+1$ ein $\mathtt{MOD}_p$-Gatter, dann müssen wir ein Programm über der Gruppe $C_p \wr R$ konstruieren.

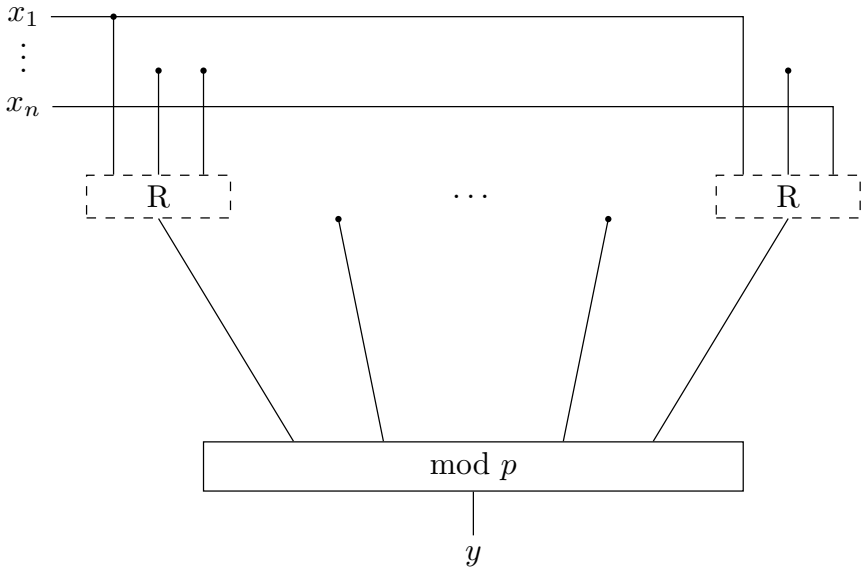


Abbildung 3.4: Grobe Illustration für einen Schaltkreis der Tiefe m+1

Aus der Induktionsvoraussetzung folgen bereits die Programme $P_{m,j}$, welche bereits dieselbe Funktion der Teilschaltkreise $\mathfrak{C}_{m,j}$ berechnen. Zur Übersichtlichkeit lassen wir den Tiefen-Index, ab jetzt fallen, da klar ist, dass wir von hier an mit P_j ein Programm für einen Schaltkreis der Tiefe m referenzieren. Wir gehen davon aus, dass $C_p = \langle g \rangle$ gilt. Es sei noch angemerkt, dass wir einzelne Variablen, welche nicht bereits aus Schaltkreisen der Tiefe m fallen, einfach kodieren können als (g, e), falls $a_i = 1$ oder (e, e), falls $a_i = 0$. Dies liefert uns simple Programmzeilen der Form $(i, (e, e), (g, e))$. Hierbei beschreibt $g \in {C_p}^R$ die Funktion

$$\begin{aligned} g : R &\to C_p \\ e &\mapsto g \\ x &\mapsto e \quad x \neq e_R \end{aligned}$$

Wir bemerken, dass Tupel aus ${C_p}^R$ auch als Funktionen aufgefasst werden können. Es ist $f = (g^{i_0}, \ldots, g^{i_n}) \in {C_p}^R$, identisch zu der Funktion $f : R \to C_p$ mit $f(e) = g^{i_0}$ und $f(x) = g^{i_x}$. Des Weiteren müssen nicht alle Inputs in unser $m+1$ Gatter direkt aus der Ebene m kommen, sondern können aus $m - n$ mit $n \leq m$ kommen. Wir haben aus der Induktion bereits akzeptierende Mengen F_i für jeden Teilschaltkreis, welchen wir als Input nutzen. Wir geben nun eine Funktion an, um Elemente in ${C_p}^R$ zu erhalten, welche entweder $f(e) = e$ oder $f(e) = g$ erfüllen, um somit in C_p das $\mathtt{MOD}_p$-Gatter zu simulieren.

Es sei (f_i, e) das neue Fixelement für das Programm P_i, ähnlich dem Tiefe 2 Fixelement, welches wir wie folgt definieren:

$$f_i : R \to C_p$$
$$x \mapsto \begin{cases} g & , x^{-1} \in F_i \\ e & , x^{-1} \notin F_i \end{cases}$$

Es ist $f_i \in {C_p}^R$ und $e \in R$ und somit das Tupel $(f_i, e) \in C_p \wr R$. Hier müssen wir zwei Dinge beachten. Erstens, da F_i eine Untergruppe von R ist, gilt falls $x \in F_i$ auch $x^{-1} \in F_i$. Zweitens gilt, da jedes Programm sein eigens Fixelement hat, wird f_i stets richtig berechnet. Zur besseren Übersicht schreiben wir ab jetzt $P_i := (e, P_i)$, da P_i ohnehin ein Element aus R ist und die erste Komponente keine Relevanz hat. Außerdem setzen wir $f_i := (f_i, e)$, da bei unserem Fixelement die zweite Komponente keine Rolle spielt. Somit können wir einen Input analog zu Tiefe 2 als $f_i[f_i^{-1}, P_i] = {}^{P_i}f_i$ berechnen, wobei f_i wieder eine Konstante in unserem Programm ist. Wie bereits erwähnt, können wir für den Fall das $P_i = x_i$, also ein Teilschaltkreis nur eine Variable ist, $f_i[f_i^{-1}, P_i]$ komplett durch die zuvor genannte Instruktion $(i, (e, e), (g, e))$ ersetzen.

Wir rechnen noch nach, dass ein Fixelement f_i in Kombination mit einem Programm P_i uns tatsächlich das gewünschte Element liefert. Wir wollen also, falls $P_i \in F_i$, dass wir ein Tupel (f, r) erhalten wobei $f(e) = g$ und $r = e$. Im anderen Fall, falls $P_i \notin F_i$, erwarten wir für (f, r), dass $f(e) = e$ und $r = e$. In unserer Rechnung $f_i[f_i^{-1}, P_i] = {}^{P_i}f_i$ eliminiert sich bereits die zweite Komponente auf natürliche Weise, sodass

$$\begin{aligned} {}^{P_i}f_i &= (e, P_i)(f_i, e)(e, P_i^{-1}) \\ &= (P_i f_i P_i^{-1}, P_i P_i^{-1}) \\ &= ({}^{P_i}f_i, e) \end{aligned}$$

gilt. Wir sehen die zweite Komponente ist unabhängig von P_i und erfüllen somit bereits die Bedingung $r = e$. Wir betrachten noch $f_i(e)[f_i^{-1}, P_i](e) = {}^{P_i}f_i(e)$ für $P_i \in F_i$ und $P_i \notin F_i$.

1. Sei $P_i \in F_i$
$$ {}^{P_i}f_i(e) = f_i(P_i^{-1}e) = f_i(P_i^{-1}) = g $$

2. Sei $P_i \notin F_i$
$$ {}^{P_i}f_i(e) = f_i(P_i^{-1}e) = f_i(P_i^{-1}) = e $$

Dies ist korrekt und berechnet exakt die Elemente, welche wir haben wollen.

Somit liegen jetzt alle Inputs des `MOD`-Gatters in der Form $({}^{P_i}f_i, e)$ vor. Wir schreiben für $({}^{P_i}f_i, e) = (c_i, e)$. Somit können wir ein Programm P angeben, welches

$$\prod_{i=1}^{k} ({}^{P_i}f_i, e) = \prod_{i=1}^{k} (c_i, e) = (c_1 \dots c_k, e) = (c, e)$$

berechnet. Damit ist $P = f_1[f_1^{-1}, P_1] \dots f_k[f_k^{-1}, P_k]$ das Programm zur Berechnung des Schaltkreises $\mathfrak{C}$.

Als Letztes geben wir die Akzeptanzmenge $F = \{ e \} \times {C_p}^{|R-1|}$ für unser Programm P an. Wir überprüfen, ob $F \leq {C_p}^R$ eine Untergruppe ist. Es gilt, dass Elemente aus F komponentenweise verrechnet werden, es ist $F \cong {C_p}^{|R-1|}$ wieder nur ein direktes Produkt von zyklischen Gruppen und somit wieder eine Gruppe, also insbesondere eine Untergruppe von ${C_p}^R$.

Zum Schluss überprüfen wir noch die Länge unseres resultierenden Programmes und stellen sicher, dass wir auch tatsächlich nicht polynomielle Länge in der Eingabe überschreiten. Wir wissen, dass jedes Programm $P_{1,i}$ lineare Länge bezüglich der Eingabe besitzt, da solche Programme genau gleich viele Instruktionen wie Variablen haben. Wir fügen für die nächste Ebene jeweils eine einzelne Instruktion pro Input der Vorebene hinzu. Es werden auch hier konstant viele Instruktionen hinzugefügt und alle Programme $P_{1,i}$ mit konstanter Länge aggregiert. Auf dieselbe Weise wird jede weitere Ebene konstruiert. Somit ist das Programm zwar unvorstellbar lang, die Länge ist aber vorwiegend nur von dem Zugrunde liegenden Schaltkreis abhängig, welcher fest ist. Das Programm überschreitet daher nicht polynomielle Länge.

Wir stellen fest, dass in jedem Induktionsschritt, also für jede weitere Ebene des Schaltkreises, nur eine weitere Gruppe benötigt wird. Die Länge unserer auflösbaren Reihe $1 \leq G_1 \leq \cdots \leq G_k = G$ entspricht somit genau der Tiefe des Schaltkreises.

Wir haben somit gezeigt, dass ein beliebiger CC^0 Schaltkreis $\mathfrak{C}$ in ein Programm P, polynomialer Länge über einer auflösbaren Gruppe G, konvertiert werden kann. Es gilt $P(solvableGroup) \supseteq \mathsf{CC}^0$.

■

4 Fazit

Durch Zulassen von bounded-fan-in Und-/Oder-Gattern ist die Tiefe der Schaltkreise und die Länge der auflösbaren Reihen eine eins-zu-eins Konvertierung. Da beim Erzeugen eines Schaltkreises für jeden Normalteiler nur eine Ebene hinzugefügt wird und wir non-MOD-Gatter nicht in die Tiefe miteinbeziehen, fällt hier keine zusätzliche Tiefe an. In der Rückrichtung wird für jedes MOD-Gatter, welches die Tiefe erhöht, eine Gruppe in das Kranzprodukt aufgenommen. Somit könnte man theoretisch zwischen Schaltkreis und Gruppe problemlos wechseln, ohne die Tiefe des Schaltkreises bzw. die Länge der Reihe zu verändern. Nutzen wir aber die genaue Definition von CC^0, dann wird bei der Richtung Gruppe zu Schaltkreis die Tiefe verändert, da wir bei direkten Produkten zwangsweise Und-Gatter benötigen.

Es wäre noch interessant andere Kodierungen für die Schaltkreise zu betrachten, da durch die gewählte Kodierung viel Overhead entsteht. Eine unäre Kodierung würde bereits das Problem von Input Vervielfachung bei zyklischen Gruppen umgehen, wäre aber nicht unbedingt bei komplexeren Gruppen geeignet, welche nicht zyklisch sind.

In der Umformung von Schaltkreis zu Gruppe haben wir ein rechtsassoziatives Kranzprodukt verwendet. Man hätte auch ein Permutations-Kranzprodukt verwenden können, welches die Konstruktion komplett verändern würde. Es wäre interessant zu sehen, inwiefern sich die Programmlänge für ein solches Kranzprodukt von dem hier gewählten Kranzprodukt unterscheidet.

Literaturverzeichnis

[Bar86] D. A. Barrington. „Bounded-Width Polynomial-Size Branching Programs Recognize Exactly Those Languages in NC1“. In: *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*. STOC '86. Berkeley, California, USA: Association for Computing Machinery, 1986, S. 1–5. ISBN: 0897911938. DOI: 10.1145/12130.12131. URL: https://doi.org/10.1145/12130.12131 (zitiert auf S. 7).

[BMMN98] M. Bhattacharjee, D. Macpherson, R. G. Möller, P. M. Neumann. „Wreath products“. In: *Notes on Infinite Permutation Groups*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, S. 67–76. ISBN: 978-3-540-49813-1. DOI: 10.1007/BFb0092558. URL: https://doi.org/10.1007/BFb0092558 (zitiert auf S. 18).

[BT88] D. A. M. Barrington, D. Thérien. „Finite Monoids and the Fine Structure of NC1“. In: *J. ACM* 35.4 (Okt. 1988), S. 941–952. ISSN: 0004-5411. DOI: 10.1145/48014.63138. URL: https://doi.org/10.1145/48014.63138 (zitiert auf S. 7).

[KM17] C. Karpfinger, K. Meyberg. „Auflösbare Gruppen“. In: *Algebra: Gruppen - Ringe - Körper*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, S. 139–152. ISBN: 978-3-662-54722-9. DOI: 10.1007/978-3-662-54722-9_11. URL: https://doi.org/10.1007/978-3-662-54722-9_11 (zitiert auf S. 7).

[Kom22] M. Kompatscher. „CC-circuits and the expressive power of nilpotent algebras“. In: *Logical Methods in Computer Science* Volume 18, Issue 2 (Mai 2022). ISSN: 1860-5974. DOI: 10.46298/lmcs-18(2:12)2022. URL: http://dx.doi.org/10.46298/lmcs-18(2:12)2022 (zitiert auf S. 7).

[MPT91] P. McKenzie, P. Péladeau, D. Thérien. „NC1: The automata-theoretic viewpoint“. In: *Computational Complexity* 1.4 (Dez. 1991), S. 330–359. DOI: 10.1007/bf01212963. URL: https://doi.org/10.1007/bf01212963 (zitiert auf S. 7, 9).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift