FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Chomsky Grammatik zur Überprüfung der restlosen Teilbarkeit durch Sechs

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA

Fakultät für Mathematik und Informatik

Molekulare Algorithmen SS 2026

Linda Hinüber
Salim Alkhaddoor

Inhaltsverzeichnis

1 Einleitung

Die Technik des DNA-Computings ist eine zunehmend an Bedeutung gewinnende Technologie, welche die DNA als Datenträger und Speichermedium verwendet. Dabei dienen DNA-Stränge als Informationsträger, welche Wörter (Strings) codieren, auf die biochemische Reaktionen angewandt werden können. Die Vorteile dieser Methode liegen in der hohen Speicherdichte, hoher Parallelisierbarkeit, langer Lebensdauer und der Energieeffizienz der Berechnungen. Auf der anderen Seite ist es ein höchst komplexes Medium, welches sich nur kompliziert bearbeiten lässt, sich durch hohe Kosten auszeichnet und auf viele Einwirkungen, wie zum Beispiel UV-Licht, sehr empfindlich reagiert.[1, 2]

Durch seine technischen Eigenschaften steht das DNA-Computing in enger Verbindung mit Chomsky-Grammatiken [2]. Grammatiken im Allgemeinen sind endliche Erzeugungssysteme, durch welche sich formale Sprachen erzeugen lassen. Sie bestehen aus einem Tupel $G = (V, \Sigma, P, S)$. Dabei ist V das Alphabet der Nichtterminalsymbole, Σ das Alphabet der Terminalsymbole. P ist eine endliche, nicht leere Menge von Produktionsregeln der Form $p \to q$, bestehend aus Terminal- und Nichtterminalsymbolen. Sie fangen immer beim Startsymbol S an. Durch Ableitungsschritte wird die Eingabe umgewandelt beziehungsweise bearbeitet; dadurch können verschiedene Berechnungsabläufe formal dargestellt und angewendet werden [3, 4, 2]. Chomsky-Grammatiken beziehen sich dabei auf eine Hierarchie solcher Grammatiken. Durch die Form der Regeln werden die Grammatiken in unterschiedliche Subklassen, wie folgt, aufgeteilt:

- Typ 0 $\equiv$ Rekursiv aufzählbare Grammatiken.
- Typ 1 $\equiv$ Kontextsensitive Grammatiken.
- Typ 2 $\equiv$ Kontextfreie Grammatiken.
- Typ 3 $\equiv$ Reguläre Grammatiken.

Nicht jedes berechenbare Problem lässt sich durch Grammatiken höherer Komplexitätsklassen (wie Typ 2 oder Typ 3) lösen. Es gilt allerdings eine Teilmengenbeziehung, bei der Grammatiken vom Typ 3 $\subset$ Typ 2 $\subset$ Typ 1 $\subset$ Typ 0. Das heißt, jede Typ 3 Grammatik lässt sich auch als eine der anderen Typen darstellen (siehe 1.1). Allerdings gilt andersherum, dass, sollte eine Grammatik sich nicht als Typ 1 darstellen lassen, sie sich auch nicht als Typ 2 oder 3 darstellen lässt [3, 4].
Eine Grammatik bietet damit eine Möglichkeit, die Bearbeitungsschritte, welche auf die DNA angewendet werden, zu beschreiben. In diesem Fall werden die Produktionsregeln durch biochemische Reaktionen umgesetzt. Das Herausschneiden eines Teils kann beispielsweise durch Restriktionsenzyme umgesetzt werden oder das Verknüpfen durch Ligasen. Im Laufe der Jahre konnte gezeigt werden, dass durch DNA-Computing Typ 0 Grammatiken, welche das Erzeugen aller rekursiv aufzählbaren Sprachen ermöglichen, emuliert werden können [5]. Während Typ 0 Grammatiken keinerlei Einschränkungen für die Struktur von p und q besitzen, gilt für Typ 1 Grammatiken eine Einschränkung. Bei diesen darf eine Produktionsregel

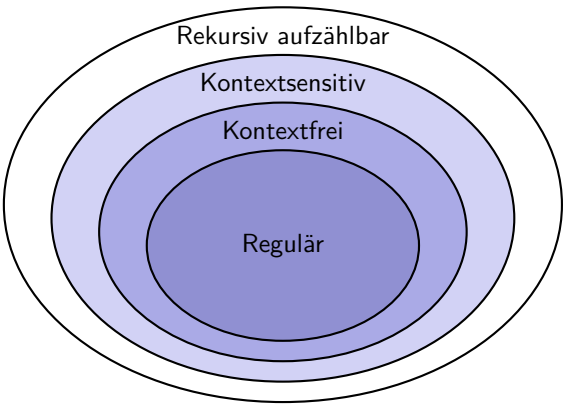

Abbildung 1.1: Schema der Chomsky-Hierarchie für formale Grammatiken.

$p \to q$ nicht verkürzend sein, das bedeutet, dass die Länge des resultierenden Wortes q mindestens so lang sein muss wie die des Ausgangswortes p ($|p| \leq |q|$). Die Ausnahme dabei ist das Startsymbol, welches auf das leere Wort abbilden darf, sofern S in keiner Regel auf der rechten Seite vorkommt. Dies stellt sicher, dass auch das leere Wort Teil der Sprache sein kann [4].

Der Schwerpunkt dieser Arbeit liegt auf der Entwicklung einer Chomsky-Grammatik für das Überprüfen der restlosen Teilbarkeit durch die Zahl Sechs.

2 Grammatik

Die Eingabe der Grammatik besteht aus einer nicht-negativen Zahl bestehend aus den Ziffern von 0 bis 9. Sie kann beliebig lang sein und hat keine weiteren Eingrenzungen bezüglich der Reihenfolge oder dem Vorkommen von Ziffern.

2.1 Symbolmenge und Produktionsregeln

Die mathematische Prüfung der Teilbarkeit durch 6 wird durch die formale Grammatik $G = (V, \Sigma, P, S)$ abgebildet. Die einzelnen Komponenten des 4-Tupels sind wie folgt definiert:

$$
\begin{aligned}
V &= \{S, L, R, R_0, R_1, R_2\} \cup \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
\Sigma &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
S &\in V \quad \text{(Startsymbol)}
\end{aligned}
$$

Während das Startsymbol S definitionsgemäß den Beginn jeder Ableitung darstellt, lassen sich die Nichtterminalen in zwei funktionale Gruppen unterteilen: Die Symbole L und R dienen als statische Randbegrenzungen, welche die Eingabe vollständig umschließen. Die Symbole R_0, R_1 und R_2 fungieren hingegen als variable Restklassen-Marker, welche den aktuellen Zustand der Modulo-Berechnungen repräsentieren.
Die endliche Menge der Produktionsregeln P ist in der nachfolgenden Tabelle 2.1 detailliert ausgeführt.

Grammatik G

Gruppe 1: Startregel			
S	$\rightarrow$	$Lz_1z_2\cdots z_nR$	(I) Initialisierung
Gruppe 2: Modulo 2			
$1R, 3R, 5R, 7R, 9R$	$\rightarrow$	{no}	(II) Ungerade Zahl
$0R$	$\rightarrow$	R_0	
$2R$	$\rightarrow$	R_2	
$4R$	$\rightarrow$	R_1	(III) Gerade Zahl
$6R$	$\rightarrow$	R_0	
$8R$	$\rightarrow$	R_2	
Gruppe 3: Modulo 3			
$0R_0, 3R_0, 6R_0, 9R_0$	$\rightarrow$	R_0	
$0R_1, 3R_1, 6R_1, 9R_1$	$\rightarrow$	R_1	(IV) Restlos
$0R_2, 3R_2, 6R_2, 9R_2$	$\rightarrow$	R_2	
$1R_0, 4R_0, 7R_0$	$\rightarrow$	R_1	
$1R_1, 4R_1, 7R_1$	$\rightarrow$	R_2	(V) Rest 1
$1R_2, 4R_2, 7R_2$	$\rightarrow$	R_0	
$2R_0, 5R_0, 8R_0$	$\rightarrow$	R_2	
$2R_1, 5R_1, 8R_1$	$\rightarrow$	R_0	(VI) Rest 2
$2R_2, 5R_2, 8R_2$	$\rightarrow$	R_1	
Gruppe 4: Termination			
LR_0	$\rightarrow$	{yes}	(VII) Ende
LR_1, LR_2	$\rightarrow$	{no}	

Tabelle 2.1: Formale Definition der Produktionsregeln für die Grammatik G.

2.2 Algorithmische Idee

Zum Lösen der Modulo-6-Berechnungen wird das Problem in eine Prüfung der Kongruenz modulo 2 und modulo 3 zerlegt. Dies funktioniert aufgrund der Teilerfremdheit von 2 und 3 mathematisch durch den Chinesischen Restsatz [6]. Dies ermöglicht eine signifikante Reduktion der Regelkomplexität.
Die Grammatik startet per Definition mit der Eingabekonfiguration $Lz_1z_2 \dots R$, welche aus dem Startsymbol S abgeleitet wird. Die Symbole L und R dienen dabei als Randbegrenzer zwischen welchen die zu prüfende Ziffernabfolge steht.
Im ersten Verarbeitungsschritt erfolgt die Paritätsprüfung (modulo 2). Dies kann bewerkstelligt werden durch das Betrachten der am rechten Rand stehenden Ziffer. Da das Symbol R unmittelbar neben der letzten Ziffer z_n platziert ist, kann der numerische Wert direkt isoliert betrachtet werden. Endet das Eingabewort auf einer ungeraden Ziffer ($z_n \in \{1, 3, 5, 7, 9\}$), greift die Abbruchregel, welche das Wort direkt ablehnt. Ist die Endziffer hingegen gerade ($z_n \in \{2, 4, 6, 8\}$) kann das Kriterium der Teilbarkeit durch 2 direkt als erfüllt bewiesen werden.
Das System initialisiert nun die modulo 3 Prüfung, indem die gerade Ziffer z_n gelöscht wird und das Randsymbol R zu einem „Restmarker“ $R_i(i, \in \{0, 1, 2\})$ umgewandelt wird. Der Index i speichert dabei den mathematischen Rest für modulo 3 Operationen (R_0 für {0,6}, R_1 für {4} und R_2 für {2,8}), die dabei als letztes stehende Ziffer wird bei der Regelanwendung „rausgelöscht“. Damit beginnt die modulo 3 Phase. Der Marker R_i dient dabei als Restzähler und weiterhin Marker des letzten Symbols. Bei jedem Ableitungsschritt wird die links vom Marker stehende Ziffer in Kombination mit dem Restmarker betrachtet. Bei der Ableitung wird dann die Ziffer gelöscht und der Restmarker aktualisiert. Die Aktualisierung des Restmarkers geschieht zyklisch:

- Ziffern der Restklasse 0 ($\{0, 3, 6, 9\}$) induzieren keine Indexänderung ($R_i \to R_i$).
- Ziffern der Restklasse 1 ($\{1, 4, 7\}$) erhöhen den Index zyklisch um 1 ($R_i \to R_{(i+1) \bmod 3}$).
- Ziffern der Restklasse 2 ($\{2, 5, 8\}$) erhöhen den Index zyklisch um 2 ($R_i \to R_{(i+2) \bmod 3}$).

Dabei ist zu beachten, dass Indizes ab dem Wert 3 zyklisch auf die Werte 0, 1 oder 2 reduziert werden, sodass stets der exakte Modulo-3-Rest abgebildet wird.
Dies wiederholt sich so lange, bis es keine Ziffern mehr gibt und dadurch die Randsymbole R_I und L nebeneinander stehen. Durch diese Kombination kommt es dann zur Termination der Grammatik. Liegt die Konfiguration LR_0 vor ist die Grammatik sowohl restlos durch 3 als auch durch 2 teilbar und damit auch durch 6. Diese Zahlen werden als teilbar akzeptiert und yes ausgegeben. Sollte LR_1 oder LR_2 vorliegen, ist die Grammatik nicht restlos durch 3 teilbar und wird damit mit einem no abgelehnt.
Ein entscheidender Punkt der Grammatik ist die Möglichkeit, die Zahl von hinten nach vorne ziffernweise abzuarbeiten. Diese Idee basiert auf der Quersummenregel. Da für die Basis des Dezimalsystems $10 \equiv 1 \pmod 3$ gilt, ist jede Zehnerpotenz 10^k ebenfalls kongruent zu 1 (mod 3). Daraus folgt, dass eine Zahl N genau dann restlos durch 3 teilbar ist, wenn ihre Quersumme restlos durch 3 teilbar ist.
Da die Addition einer solchen Eingabe, wie sie hier vorliegt, nicht ohne Weiteres mit einer formalen Grammatik umsetzbar ist, muss hier auf ein anderes Verfahren zurückgegriffen werden. In diesem Fall wird, anstatt die Quersumme aufzusummieren, direkt modulo 3 reduziert. Dadurch wird indirekt die Quersumme gebildet, allerdings wird sie sofort in jedem Schritt modulo 3 reduziert.

2.3 Beispielanwendungen

Beispielableitung 1 (Ablehnungsbeispiel)

Gegeben sei die ungerade Eingabe $N = 625483$. Die Initialisierung erfolgt über die Startregel (I).

$$\begin{aligned} S &\Rightarrow L\,625483\,R && \text{(Initialisierung)} \\ &\Rightarrow \{\text{no}\} && \text{(Regel } 3R \to \{\text{no}\}\text{)} \end{aligned}$$

Da die Kette durch die 3 am Ende ungerade ist und damit die modulo 2 Prüfung nicht besteht wird das Wort abgelehnt.

Beispielableitung 2 (Akzeptanzbeispiel)

Gegeben sei die gerade Eingabe $N = 625482$. Die Ableitungskette zeigt den schrittweisen Abbau von rechts nach links:

$$\begin{aligned} S &\Rightarrow L\,625482\,R && \text{(Initialisierung)} \\ &\Rightarrow L\,62548\,R_2 && \text{(Regel } 2R \to R_2\text{)} \\ &\Rightarrow L\,6254\,R_1 && \text{(Regel } 8R_2 \to R_1\text{, da } 2+8=10\equiv 1\text{)} \\ &\Rightarrow L\,625\,R_2 && \text{(Regel } 4R_1 \to R_2\text{, da } 1+4=5\equiv 2\text{)} \\ &\Rightarrow L\,62\,R_1 && \text{(Regel } 5R_2 \to R_1\text{, da } 2+5=7\equiv 1\text{)} \\ &\Rightarrow L\,6\,R_0 && \text{(Regel } 2R_1 \to R_0\text{, da } 1+2=3\equiv 0\text{)} \\ &\Rightarrow L\,R_0 && \text{(Regel } 6R_0 \to R_0\text{, da } 0+6=6\equiv 0\text{)} \\ &\Rightarrow \{\text{yes}\} && \text{(Termination über } LR_0 \to \{\text{yes}\}\text{)} \end{aligned}$$

Das Wort wird erfolgreich akzeptiert, da es auf eine gerade Ziffer endete und das Wort die Modulo 3 Prüfung besteht.

2.4 Implementierung als interaktive Visualisierung

Um die Arbeitsweise der Grammatik anschaulich nachvollziehen zu können, wurde ein Java-Programm entwickelt, das den schrittweisen Ableitungsprozess simuliert und jede Regelanwendung inklusive des aktuellen Wortes ausgibt. Der Quellcode befindet sich in der Datei `mod6/src/Main.java` und ist wie folgt aufgebaut:
Die `Main`-Klasse kapselt den gesamten Ableitungsvorgang. Im Konstruktor wird die Eingabe N gemäß der Startregel (I) in die Konfiguration $L\,N\,R$ überführt und die erste Ableitungszeile ausgegeben. Die Methode `run()` steuert den Ablauf: Zunächst wird die letzte Ziffer mitsamt des rechten Randmarkers R durch `Mod2()` verarbeitet (Gruppe 2). Ist die Ziffer ungerade, bricht das Programm mit `No` ab. Andernfalls folgt die Modulo-3-Prüfung, in der innerhalb einer Schleife wiederholt `TrimString()` das jeweils nächste Ziffer-Marker-Paar abtrennt und `Mod3()` die entsprechende Regel aus Gruppe 3 anwendet. Sobald die Randsymbole L und R_i nebeneinanderstehen, entscheidet `end()` über die Akzeptanz (Gruppe 4). Jede Regelanwendung wird durch `WriteRule()` mit der Phasenbezeichnung I–VI auf der Konsole ausgegeben, sodass die gesamte Ableitungskette lückenlos protokolliert wird.
Die Methoden des Programms bilden die Produktionsregeln der formalen Grammatik direkt ab: Die Fallunterscheidung in `Mod2()` prüft die Parität der Endziffer und setzt den

Restmarker R_i entsprechend des modulo-3-Wertes. In `Mod3()` werden die Ziffern anhand ihrer modulo-3-Klasse (Rest 0, 1 oder 2) und des aktuellen Markers R_i über Switch-Cases ausgewertet. Die Bedingung LR_0 bzw. LR_1, LR_2 in `end()` realisiert die Terminationsregeln. Auch wenn das Programm nicht jede einzelne Produktionsregel als String-Ersetzung codiert, sondern die mod-3-Klassifizierung arithmetisch berechnet, bildet es dennoch jeden Ableitungsschritt korrekt ab.

2.4.1 Ausführungsbeispiele

Die folgenden Terminalausgaben zeigen das Programmverhalten an drei repräsentativen Eingaben.

Beispiel 1: Ungerade Eingabe $N = 625483$

```
I       S       -->     L625483R
II      3R      -->     No
```

Beispiel 2: Durch 6 teilbare Eingabe $N = 625482$

```
I       S       -->     L625482R
III     2R      -->     L62548R2
VI      8R2     -->     L6254R1
V       4R1     -->     L625R2
VI      5R2     -->     L62R1
VI      2R1     -->     L6R0
IV      6R0     -->     LR0
VII     LR0     -->     Yes
```

Beispiel 3: Gerade, aber nicht durch 3 teilbare Eingabe $N = 123454$

```
I       S       -->     L123454R
III     4R      -->     L12345R1
VI      5R1     -->     L1234R0
V       4R0     -->     L123R1
IV      3R1     -->     L12R1
VI      2R1     -->     L1R0
V       1R0     -->     LR1
VII     LR1     -->     No
```

In Beispiel 1 wird die Eingabe bereits nach der Modulo-2-Prüfung abgelehnt, da die Endziffer 3 ungerade ist. Beispiel 2 durchläuft die vollständige Ableitungskette und terminiert mit `Yes`, da die Zahl sowohl durch 2 als auch durch 3 teilbar ist. Beispiel 3 zeigt eine gerade Zahl, die die Modulo-3-Prüfung nicht besteht und daher mit `No` abgelehnt wird.

2.5 Vollständigkeit

Um die formale Zuverlässigkeit der Chomsky-Grammatik nachzuweisen, muss gezeigt werden, dass das System für jede syntaktisch korrekte Eingabe nach endlich vielen Schritten terminiert und stets das korrekte Ergebnis liefert.

Die Vollständigkeit und die garantierte Termination basieren darauf, dass für jede beliebige Startkonfiguration $Lz_0z_1 \dots z_nR$ eine lückenlose Fallabdeckung und eine strikte Monotonie vorliegt. Die Produktionsregeln der Grammatik decken alle Kombinationen aus allen möglichen Ziffern und Zustandsmarker lückenlos ab. Für jede Ziffer besteht mit dem aktuellen Restmarker genau eine passende Regel. Dadurch bleibt das System nie einfach stecken. Da jede Produktionsregel die aktuelle Ziffer ersatzlos aus dem verbleibenden Wort entfernt, verkürzt sich die Länge der Ziffernkette bei jedem Schritt um genau eins. Da jede Eingabe aus einer endlichen Menge an Ziffern besteht, ist das Wort nach einer endlichen Anzahl an Schritten vollständig abgebaut. In dieser finalen Konfiguration stehen die Randsymbole unmittelbar nebeneinander, wodurch die Terminationsregeln greifen, welche den Marker eliminieren und das System endgültig stoppen. Endlose Ableitungsketten oder zyklische Schleifen sind somit durch den schrittweisen Abbau strukturell ausgeschlossen.

2.6 Typ der Chomsky-Grammatik

Die in diesem Kapitel beschriebene Grammatik klassifiziert sich formal als eine Typ-0 Grammatik, da sie verkürzende Regeln beinhaltet, bei denen die Länge der rechten Regelseite kleiner ist als die der linken Seite. Allerdings lässt sich dieser Ansatz auch als nicht-verkürzende Typ-1 Grammatik lösen. In diesem Fall würde man statt die verarbeiteten Ziffern zu löschen den Restmarker R nach links durch tauschen. Eine entsprechende Regel würde beispielsweise wie folgt lauten:

$$\begin{aligned} 0R &\to R_0 \\ \Rightarrow 0R &\to R_00 \end{aligned}$$

Dadurch würde auch die ursprüngliche Zahl erhalten bleiben. Dies macht die Grammatik allerdings komplexer ohne einen direkten Vorteil daraus zu gewinnen, weswegen sich hier dagegen entschieden wurde. Eine Typ-2 Grammatik, bei welcher auf der linken Seite nur genau ein nichtterminales Symbol stehen darf ist für die Herangehensweise über Modulo 2 und 3 nicht möglich. Dies liegt daran, dass eine Grammatik die korrekte Abarbeitung der Reihenfolge nach garantieren muss, in diesem Fall muss also modulo 2 vor modulo 3 überprüft werden. Damit muss hinten angefangen werden mit der Abarbeitung, da sonst durch Regeln basierend auf R und L nicht klar ist welche Regel zuerst angewendet werden muss und die Ziffer immer eingelesen werden muss und nicht generiert werden kann.

2.7 Alternativer Ansatz

Das Problem lässt sich alternativ auch über eine direkte Abbildung von modulo 6 mit insgesamt sechs Restklassen lösen. Die algorithmische Grundidee folgt dem gleichen Prinzip wie die Modulo-3-Prüfung, erfolgt jedoch mit mehr Restklassen. Dieser Ansatz hat allerdings zwei entscheidende Nachteile, der erste ist, dass es zu einer Verdopplung der Produktionsregelanzahl kommt. Dies liegt daran, dass es doppelt so viele Restklassen gibt, welche durch unterschiedlich eingelesene Ziffern in alle der anderen Restklassen übergehen könnte. Der zweite Nachteil ist, dass dieser Ansatz als Standard nicht den frühen Abbruch durch modulo 2 ermöglicht, natürlich ließen sich diese Regeln noch mit integrieren, dies würde aber in weiteren Regeln enden. Um den dafür notwendigen Kontrollfluss und die Marker-Verschiebungen zu realisieren ist diese Grammatik ebenfalls von Typ-1- (kontextsensitiv) oder Typ-0-Grammatik (unbeschränkt).

3 Komplexitätsanalyse

Die Komplexität der entwickelten Chomsky-Grammatik wird im Folgenden anhand der Anzahl der notwendigen Regelanwendungen sowie der dafür benötigten Zeitschritte analysiert. Sei n die Anzahl der Ziffern der als Eingabe übergebenen Dezimalzahl N.

3.1 Regelanwendungen (Schrittkomplexität)

Bei der Auswertung der Grammatik müssen zwei Szenarien unterschieden werden:

Best-Case (Ungerade Zahlen):
Ist die gegebene Dezimalzahl ungerade, greift nach der Initialisierung sofort die Paritätsprüfung am rechten Rand. Da die Endziffer $z_n \in \{1, 3, 5, 7, 9\}$ ist, bricht das System die Ableitung unverzüglich ab. Hierfür sind exakt zwei Regelanwendungen notwendig:

1. Die Startregel (I): $S \to Lz_1z_2 \cdots z_nR$
2. Die Abbruchregel (II): $z_nR \to \{\text{no}\}$

Dies resultiert in einer konstanten Schrittanzahl von exakt 2 Regelanwendungen.

Worst-Case / Average-Case (Gerade Zahlen):
Ist die Zahl gerade ($z_n \in \{0, 2, 4, 6, 8\}$), muss die gesamte Ziffernkette von rechts nach links vollständig abgearbeitet werden, um die Teilbarkeit durch 3 zu prüfen. Eine Verkürzung des Verfahrens ist hierbei nicht möglich. Die Gesamtzahl der Regelanwendungen setzt sich wie folgt zusammen:

- 1 Regelanwendung für die Initialisierung der Grammatik durch das Startsymbol ($S \to LwR$).
- 1 Regelanwendung für die Paritätsprüfung und gleichzeitige Transformation des rechten Begrenzers in den ersten Restmarker ($z_nR \to R_i$).
- $(n-1)$ Regelanwendungen für das schrittweise Einlesen und Eliminieren aller verbleibenden Ziffern links des Markers.
- 1 Regelanwendung für die finale Terminationsentscheidung am linken Rand ($LR_i \to \{\text{yes}\}$ oder $\{\text{no}\}$).

Daraus ergibt sich für jede gerade Zahl eine Summe von:

$$1 + 1 + (n-1) + 1 = n + 2 \text{ Regelanwendungen}$$

3.2 Zeitschritte und asymptotische Komplexität

Da die Produktionsregeln der Grammatik strikt deterministisch aufgebaut sind und die Ersetzung ausschließlich am aktuellen Standort des Markers R_i stattfinden kann, ist eine parallele Verarbeitung verschiedener Textabschnitte ausgeschlossen. Zu jedem diskreten Zeitpunkt kann stets genau eine einzige Regel der Grammatik angewendet werden. Folglich entspricht die Anzahl der benötigten Zeitschritte exakt der Anzahl der ausgeführten Regelanwendungen.
Für die asymptotische Zeitkomplexität betrachtet man das Verhalten bei einer gegen unendlich laufenden Ziffernlänge ($n \to \infty$). Da die additiven Konstanten der linearen Funktion $f(n) = n + 2$ in der asymptotischen Betrachtung vernachlässigt werden können, liegt die Zeitkomplexität der Grammatik in der Komplexitätsklasse

$$\mathcal{O}(n)$$

Das System arbeitet somit streng linear zur Länge der Eingabe.

4 Zusammenfassung

Diese Arbeit stellt eine Chomsky-Grammatik vom Typ 0 vor, welche überprüft, ob sich eine beliebige positive natürliche Zahl restlos durch sechs teilen lässt. Dafür teilt sie das Problem in eine Modulo-2- und eine Modulo-3-Prüfung auf, wodurch die Grammatik mit einem stark reduzierten Regelsatz auskommt. Des Weiteren kann die Grammatik durch die Modulo-2-Prüfung im Fall von ungeraden Zahlen bereits frühzeitig abbrechen, wobei die Worst-Case-Laufzeit dennoch in $\mathcal{O}(n)$ liegt.
Diese theoretische Ausarbeitung inklusive der Java-Implementierung dient als Proof of Concept. Statt aufwendig jeden Berechnungsschritt direkt auf die DNA anzuwenden, kann ein solches Verfahren zunächst simuliert und validiert werden. Da beim DNA-Computing jede Produktionsregel einer physischen, biochemischen Reaktion entspricht, würde die hier gezeigte Reduktion der Regeln eine entscheidende Kosten- und Materialersparnis für die Laborarbeit bedeuten.

Literaturverzeichnis

[1] Mingzhi Zhang und Da Han. „Concept, development and applications of DNA computation“. In: *Fundamental Research* 6.1 (2026), S. 178–183. ISSN: 2667-3258. DOI: https://doi.org/10.1016/j.fmre.2023.06.015. URL: https://www.sciencedirect.com/science/article/pii/S2667325823002352.

[2] Thomas Hinze und Monika Sturm. „Eine DNA-Arithmetik auf der Basis von Chomsky-Grammatiken“. In: *Tagungsband 14. Theorietag Automaten und Formale Sprachen*. Caputh bei Potsdam, 2004, S. 71–75. URL: https://www.researchgate.net/publication/239566333_Eine_DNA-Arithmetik_auf_der_Basis_von_Chomsky-Grammatiken.

[3] John E. Hopcroft, Rajeev Motwani und Jeffrey D. Ullman. *Einführung in Automatentheorie, Formale Sprachen und Berechenbarkeit*. 3., aktualisierte Auflage. München: Pearson Studium, 2011. ISBN: 978-3-86894-082-4.

[4] Noam Chomsky. „On certain formal properties of grammars“. In: *Information and Control* 2.2 (1959), S. 137–167. DOI: 10.1016/S0019-9958(59)90362-6.

[5] Chang-Muk Lee u. a. „DNA Computing the Hamiltonian Path Problem“. In: *Molecules and Cells* 9.5 (1999), S. 464–469. ISSN: 1016-8478. DOI: 10.1016/S1016-8478(23)13571-0. URL: https://www.sciencedirect.com/science/article/pii/S1016847823135710.

[6] Encyclopedia of Mathematics. *Chinese remainder theorem*. European Mathematical Society. URL: https://encyclopediaofmath.org/wiki/Chinese_remainder_theorem (besucht am 21. 06. 2026).