Institut für Mathematik und Informatik
Ernst-Moritz-Arndt-Universität Greifswald

Möglichkeiten und Grenzen des DNA-Computings im Hinblick auf PSPACE

Wissenschaftliche Arbeit zur Erlangung
des akademischen Grades
Diplom-Biomathematiker

eingereicht von
Thomas Zerjatke

unter der Betreuung von
Dr. Christine Gaßner
Institut für Mathematik und Informatik
Dr. Monika Sturm
Institut für Theoretische Informatik, TU Dresden

Greifswald im April 2009

Inhaltsverzeichnis

Einleitung

In der Theoretischen Informatik gibt es eine ganze Reihe von Problemen, für die keine effizienten Algorithmen bekannt sind. Die Zeit, die benötigt wird, um diese oftmals praktisch relevanten Probleme aus den Komplexitätsklassen $\mathcal{NP}$ und $PSPACE$ mit heutigen Rechnern zu lösen, wächst exponentiell in der Eingabegröße. Dadurch kann es schnell zu Laufzeiten von mehreren Jahren oder sogar Jahrhunderten kommen; eine Lösung ist also selbst für relativ kleine Problemgrößen nicht möglich. Aus diesem Grund gibt es zahlreiche Versuche, neue Berechnungsmodelle zu entwickeln, mit denen diese Probleme auch in polynomiell wachsender Zeit und somit effizient berechnet werden können. Diese Modelle zeichnen sich dadurch aus, dass es in ihnen möglich ist, viele Rechenschritte gleichzeitig durchzuführen, die Berechnung erfolgt also *parallel*. Im Gegensatz dazu arbeiten alle heutigen Computer *sequenziell*, d.h. es wird immer nur ein Rechenschritt nach dem anderen ausgeführt. Neben dem Modell des Quantencomputers, welcher Prozesse der Quantenmechanik nutzt, gibt es einige Modelle, die durch biologische und biochemische Prozesse inspiriert sind; dazu gehört auch das *DNA-Computing*. Dabei wird das als Träger der Erbinformation dienende Molekül *DNA* als Speichermedium verwendet, Berechnungen erfolgen durch Manipulation dieser Moleküle mithilfe verschiedener molekularbiologischer Methoden.

Ähnlich wie bei Quantencomputern gibt es auch auf dem Gebiet des DNA-Computings zahlreiche theoretische Arbeiten, jedoch nur wenige praktische Versuche. Es ist daher noch nicht abzusehen, inwieweit sich das DNA-Computing praktisch nutzen lässt und wann dies der Fall sein könnte. Die Situation wird häufig mit der Entwicklung herkömmlicher Computer verglichen: Die ersten theoretischen Arbeiten erfolgten in den 1930er Jahren unter anderem durch Alan Turing und die Entwicklung des Berechnungsmodells der Turingmaschine, die ersten Computer wurden jedoch erst mehr als zwanzig Jahre später gebaut.

Auf dem Gebiet des DNA-Computings gibt es zwei prinzipielle Herangehensweisen: Zum einen die Entwicklung theoretischer Modelle, die mehr oder weniger stark vom Speichermedium DNA und den vorhandenen Methoden zu dessen Manipulation abstrahieren, und die Untersuchung dieser Modelle hinsichtlich ihrer Berechnungsstärke; zum anderen weitgehend unabhängig davon der Versuch, Algorithmen für spezielle bisher nicht effizient lösbare Probleme zu finden, die im Labor umgesetzt werden könnten.

Während es zu Problemen der Klasse $\mathcal{NP}$ bereits eine Vielzahl von Veröffentlichungen gibt und einige dieser Probleme auch schon im Labor für kleine Problemgrößen

praktisch gelöst werden konnten, sind aus der Literatur nur sehr wenige Veröffentlichungen zur größeren Klasse $\mathcal{PSPACE}$ bekannt. In der vorliegenden Arbeit wird eines der Probleme aus dieser Klasse, das Problem der Quantifizierten Booleschen Formeln, betrachtet; dieses ist vollständig in $\mathcal{PSPACE}$, d.h. alle Probleme der Klasse können darauf in effizienter Weise reduziert werden. Es sollen verschiedene Möglichkeiten zur Lösung dieses Problems durch DNA-Computing-Algorithmen gegeben und die Grenzen einer praktischen Umsetzung diskutiert werden.

Das erste Kapitel dieser Arbeit fasst einige Grundlagen aus der Komplexitätstheorie zusammen und stellt die Klasse $\mathcal{PSPACE}$ sowie das Problem der Quantifizierten Booleschen Formeln vor. Im zweiten Kapitel werden die molekularbiologischen Grundlagen erklärt und es wird ein Überblick über die Entwicklung des DNA-Computings und verschiedene Modellvorstellungen gegeben. Anschließend werden im dritten Kapitel DNA-Computing-Algorithmen zur Lösung des betrachteten Problems erarbeitet. Im vierten Kapitel wird diskutiert, inwieweit sich die vorgeschlagenen Algorithmen praktisch umsetzen ließen und welche Probleme dabei auftreten könnten. Abschließend wird eine kurze Zusammenfassung und ein Ausblick gegeben. Am Schluss der Arbeit ist ein Glossar zu finden, das die wichtigsten biologischen Begriffe noch einmal zum schnellen Nachschlagen zusammenfasst.

1 Komplexitätstheorie und die Klasse PSPACE

1.1 Mathematische Grundbegriffe

In diesem Abschnitt sollen einige Grundbegriffe definiert werden, die im Folgenden verwendet werden.

Ein *Alphabet* ist eine endliche, nichtleere Menge, die Elemente dieser Menge werden auch als *Zeichen*, *Buchstaben* oder *Symbole* bezeichnet.

Ein *Wort über einem Alphabet* Σ ist eine endliche Folge von Buchstaben aus Σ, es lässt sich also durch Hintereinanderschreiben (*Konkatenation*) von Elementen aus Σ bilden. Für zwei Wörter w und v sei wv ihre Konkatenation.

Für ein Alphabet Σ bezeichne $|\Sigma|$ seine Mächtigkeit, für ein Wort w bezeichne $|w|$ seine Länge, d.h. die Anzahl seiner Buchstaben; λ sei das Wort der Länge 0, das *leere Wort*.

$\Sigma^{(n)}$ sei die Menge aller Wörter über Σ der Länge n; die Angabe von n in Klammern erfolgt hierbei in Abgrenzung zum Begriff des *n-fachen Kartesischen Produktes*.

Σ^* sei die Menge aller Wörter über Σ, also

$$\Sigma^* := \bigcup_{n \in \mathbb{N}} \Sigma^{(n)} \ ,$$

Σ^+ sei die Menge aller Wörter über Σ außer dem leeren Wort λ, also

$$\Sigma^+ := \bigcup_{n \in \mathbb{N} \setminus \{0\}} \Sigma^{(n)} = \Sigma^* \setminus \{\lambda\} \ .$$

Eine *Sprache* ist eine Teilmenge von Σ^*. Für zwei Sprachen A und B, $A, B \subseteq \Sigma^*$, wird definiert:

$$A \cup B := \{w \mid w \in A \text{ oder } w \in B\}$$
$$A \cap B := \{w \mid w \in A \text{ und } w \in B\}$$
$$A \otimes B := \{wv \mid w \in A \text{ und } v \in B\}$$

Da Sprachen im Allgemeinen unendlich viele Wörter enthalten, ist eine endliche Beschreibungsmöglichkeit, eine *Grammatik*, notwendig.

Definition 1.1

Eine *Grammatik* ist ein 4-Tupel $G = (V, \Sigma, P, S)$, wobei

- V eine endliche Menge von *Variablen*,
- Σ eine endliche Menge von *Terminalsymbolen* mit $V \cap \Sigma = \emptyset$,
- $P \subset ((V \cup \Sigma)^* \otimes V \otimes (V \cup \Sigma)^*) \times (V \cup \Sigma)^*$ eine endliche Menge von *Regeln* (*Produktionen*) und
- S die *Startvariable* ist.

Für eine Regel $(y, y') \in P$ schreibt man $y \to y'$, y enthält dabei mindestens eine Variable.

Für Wörter $u, v \in (V \cup \Sigma)^*$ wird die Relation $u \Rightarrow_G v$ (u geht unter G unmittelbar über in v) definiert, falls u und v die Form $u = xyz$ und $v = xy'z$ haben und $y \to y'$ eine Regel in P ist.

Die reflexive und transitive Hülle von $\Rightarrow_G$ wird mit $\Rightarrow_G^*$ bezeichnet; $u \Rightarrow_G^* v$ bedeutet also, dass u in beliebig vielen Schritten in v über geht.

Die *von G erzeugte Sprache* ist

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$$

Die von G erzeugte Sprache $L(G)$ enthält also alle Wörter über dem Terminalalphabet Σ, die aus der Startvariablen S in beliebig vielen Schritten abgeleitet werden können.

Der Sprachtheoretiker Noam Chomsky teilte Grammatiken in vier Typen ein, *Typ 0* bis *Typ 3*:

Definition 1.2

Jede Grammatik ist vom Typ 0.

Eine Grammatik ist vom Typ 1 (*kontextsensitiv*), wenn für alle Regeln $y \to y'$ in P gilt, dass $|y| \leq |y'|$ ist, Wörter während der Ableitung also nicht kürzer werden können. Einzige Ausnahme bildet die Regel $S \to \lambda$, falls die Startvariable S nicht auf der rechten Seite einer Regel aus P auftritt.

Eine Grammatik ist vom Typ 2 (*kontextfrei*), wenn zusätzlich für alle Regeln $y \to y'$ in P gilt, dass $y \in V$, d.h. eine einzelne Variable ist.

Eine Grammatik ist vom Typ 3 (*regulär*), wenn zusätzlich für alle Regeln gilt, dass $y' \in \{\lambda\} \cup \Sigma \cup (\Sigma \otimes V)$ ist, d.h. die rechten Seiten der Regeln sind entweder das leere Wort, ein einzelnes Terminalzeichen oder ein Terminalzeichen, das von einer Variablen gefolgt wird.

Eine Sprache $L \subseteq \Sigma^*$ ist vom Typ 0 bis Typ 3, wenn es eine sie erzeugende Grammatik des entsprechenen Typs gibt.

Die *Turingmaschine* ist ein Berechnungsmodell, welches 1936 vom englischen Logiker Alan Turing entwickelt wurde. Ziel war die Beantwortung der Frage, welche mathematischen Funktionen durch algorithmische Verfahren berechnet werden können. Turing leitete sein Modell aus der Beobachtung, wie Menschen mathematisch-naturwissenschaftliche Probleme algorithmisch lösen, ab und schuf so die Grundidee für die von-Neumann-Rechnerarchitektur und somit den Bau realer Rechner. Eine Turingmaschine besitzt zunächst ein aus linear angeordneten Zellen bestehendes unendliches Band. Jede dieser Zellen kann ein Symbol eines gegebenen Bandalphabets enthalten. Die weiteren Bestandteile sind ein Lese-Schreib-Kopf und eine ihn steuernde Kontrolleinheit.

Formal lässt sich die Turingmaschine durch ein 7-Tupel beschreiben:

Definition 1.3

Eine *(deterministische) Turingmaschine* ist ein 7-Tupel $(Z, \Sigma, \Gamma, \delta, z_0, \square, E)$, wobei die Komponenten folgende Bedeutung haben:

- Z ist die endliche *Menge von Zuständen*,
- Σ das *Eingabealphabet*,
- Γ das *Arbeitsalphabet* mit $\Sigma \subset \Gamma$,
- $\delta : Z \times \Gamma \to Z \times \Gamma \times \{-1, 0, 1\}$ die *Überführungsfunktion*,
- $z_0 \in Z$ der *Startzustand*,
- $\square \in \Gamma \setminus \Sigma$ das *Blanksymbol* und
- $E \subseteq Z$ die *Menge der Endzustände*.

Zu Beginn einer Berechnung befindet sich auf dem Band das Eingabewort und auf allen anderen Zellen das Blanksymbol. Der Lese-Schreib-Kopf befindet sich auf der ersten Zelle des Eingabewortes. Die Steuereinheit speichert den aktuellen Zustand, die Überführungsfunktion δ und die Menge der Endzustände E und wertet diese aus. Zu Beginn der Berechnung befindet sich die Steuereinheit im Startzustand z_0. Ein Arbeitsschritt der Turingmaschine besteht aus folgenden Aktionen:

- Das Zeichen des Bandes, auf dem der Lese-Schreib-Kopf steht, wird eingelesen.

- Aus dem eingelesenen Zeichen und dem aktuellen Zustand wird mittels der Überführungsfunktion δ das zu schreibende Zeichen, der nächste Zustand und die Bewegung des Kopfes bestimmt. Bei -1 bewegt sich der Kopf zur linken, bei 1 zur rechten Nachbarzelle und bei 0 bleibt er auf der aktuellen Zelle stehen.
- Der Lese-Schreib-Kopf schreibt das Ausgabezeichen in die aktuelle Zelle des Bandes und bewegt sich entsprechend der durch δ bestimmten Richtung. Die Steuereinheit wechselt in den durch δ festgelegten Zustand.

Sobald einer der Endzustände erreicht ist, endet die Berechnung der Maschine. Auf dem Band befindet sich dann das Ausgabewort.

Eine *Konfiguration* einer Turingmaschine ist ein Wort $k \in \Gamma^* \otimes Z \otimes \Gamma^*$. Die Konfiguration beschreibt den momentanen Zustand, in dem sich die Steuereinheit befindet, sowie das auf dem Arbeitsband stehende Wort und die Stellung des Kopfes. Für $k = \alpha z \beta$ ist z der Zustand, in dem sich die Maschine befindet, und $\alpha\beta$ der beschriebene Teil des Bandes, auf allen anderen Feldern des Bandes befindet sich das Blanksymbol $\square$. Der Lese-Schreib-Kopf steht auf dem ersten Buchstaben von β. Die Startkonfiguration zu Beginn einer Berechnung ist $z_0 x$ für eine Eingabe x.

Der Übergang von einer Konfiguration k_1 zur nächsten Konfiguration k_2 sei gegeben durch die zweistellige Relation $k_1 \models_{\mathfrak{M}} k_2$ für eine gegebene Turingmaschine $\mathfrak{M}$. Die reflexive und transitive Hülle sei $\models^*_{\mathfrak{M}}$.

Definition 1.4

Die von einer Turingmaschine $\mathfrak{M}$ *akzeptierte Sprache* ist definiert als

$$L(\mathfrak{M}) = \{x \in \Sigma^* \mid z_0 x \models^*_{\mathfrak{M}} \alpha z \beta \text{ für } \alpha, \beta \in \Gamma^* \text{ und } z \in E\} \ .$$

$L(\mathfrak{M})$ enthält somit alle Wörter über dem Eingabealphabet, für die $\mathfrak{M}$ nach einer beliebigen Anzahl von Rechenschritten einen Endzustand erreicht, das sich danach auf dem Band befindliche Ausgabewort $\alpha\beta$ hat hierbei keine Bedeutung.

Es gilt: Die durch Turingmaschinen akzeptierbaren Sprachen sind genau die Sprachen vom Typ 0. [32]

Laut der *Churchschen These* entspricht das Modell der Turingmaschine dem intuitiven Berechnungsbegriff. D.h. die Klasse der Funktionen, die durch Turingmaschinen berechenbar sind, stimmt überein mit der Klasse der im intuitiven Sinne berechenbaren Funktionen. Diese These kann nicht bewiesen werden, da der Begriff der intuitiven Berechenbarkeit keine formale Definition hat. Die These wurde 1936 von Alonzo Church aufgestellt und ist heute allgemein akzeptiert, da alle Versuche, den Berechenbarkeitsbegriff zu formalisieren (neben der Turingmaschine zum Beispiel WHILE- oder GOTO-Programme, μ-rekursive Funktionen oder das von Church entwickelte λ-Kalkül), genau dieselbe Klasse von berechenbaren Funktionen beschreiben und somit äquivalent sind. [31]

Das Modell der Turingmaschine kann auf mehrere Bänder erweitert werden:

Definition 1.5

Eine *(deterministische) Turingmaschine mit k Bändern* enthält die gleichen Komponenten wie die in Definition 1.3 definierte Turingmaschine mit einem Band, die Überführungsfunktion δ ist nun wie folgt spezifiziert:

$$\delta : Z \times \Gamma^k \to Z \times \Gamma^k \times \{-1, 0, 1\}^k$$

Es wird auf allen Bändern gleichzeitig gelesen und geschrieben, die Lese-Schreib-Köpfe auf den einzelnen Bändern können sich unabhängig voneinander bewegen.

Die Konfiguration einer Turingmaschine mit k Bändern wird als Tupel $(z, u_1, v_1, \ldots, u_k, v_k) \in Z \times (\Gamma^*)^{2k}$ angegeben. z sei dabei der aktuelle Zustand, $u_i v_i$ das Wort auf Band i, der Lese-Schreib-Kopf stehe auf der ersten Zelle von v_i. Zu Beginn befinde sich das Eingabewort auf dem ersten Band, alle anderen Bänder seien leer. Die Relation $\models_{\mathfrak{M}}$ und ihre reflexive und transitive Hülle $\models^*_{\mathfrak{M}}$ werden analog zur Einband-Turingmaschine definiert, ebenso wie die akzeptierte Sprache $L(\mathfrak{M})$.

Die Mehrband-Turingmaschine ist nicht berechnungsstärker als die Einband-Turingmaschine, diese kann eine Turingmaschine mit k Bändern in quadratischer Zeit (bezogen auf die Eingabegröße) simulieren. Sie wird verwendet, weil sie „handlicher“ bei der Lösung vieler Probleme ist. [37]

Der bei der Definition der Turingmaschine angegebene Begriff *deterministisch* besagt, dass jeder Rechenschritt der Turingmaschine durch den Zustand der Steuereinheit und das gelesene Zeichen genau festgelegt (*determiniert*) ist. Im Gegensatz dazu kann es beim Modell der *nichtdeterministischen Turingmaschine* in jedem Rechenschritt mehrere Möglichkeiten geben, wie sich die Maschine verhält. Die Arbeitsweise der nichtdeterministischen Turingmaschine wird also nicht durch eine Überführungs*funktion*, sondern eine Überführungs*relation* (beziehungsweise eine Abbildung in die Potenzmenge) festgelegt:

Definition 1.6

Eine *nichtdeterministische Turingmaschine mit k Bändern* enthält die gleichen Komponenten wie die deterministische Variante aus Definition 1.5, wobei δ eine Relation ist:

$$\delta \subseteq (Z \times \Gamma^k) \times (Z \times \Gamma^k \times \{-1, 0, 1\}^k)$$

Alternativ kann δ auch als Abbildung in die Potenzmenge definiert werden:

$$\delta : Z \times \Gamma^k \to \wp(Z \times \Gamma^k \times \{-1, 0, 1\}^k)$$

Der Begriff *Konfiguration* sowie die Relation $\models_{\mathfrak{M}}$ und ihre reflexive und transitive Hülle $\models^*_{\mathfrak{M}}$ werden analog verwendet.

Die von einer nichtdeterministischen Turingmaschine erkannte Sprache wird ebenfalls analog zum deterministischen Fall definiert:

Definition 1.7

Die von einer nichtdeterministischen Turingmaschine $\mathfrak{M}$ erkannte Sprache ist:

$$L(\mathfrak{M}) = \{x \in \Sigma^* \mid (z_0, \lambda, x, \lambda, \ldots, \lambda) \models^*_{\mathfrak{M}} (z, u_1, w_1, \ldots, u_k, w_k) \text{ mit } z \in E \text{ und } u_1, w_1, \ldots, u_k, w_k \in \Sigma^*\}$$

Meist werden Komplexitätsfunktionen mithilfe der O-Notation angegeben, um konstante Faktoren ignorieren zu können. Hierbei sei für eine Funktion $f : \mathbb{N} \to \mathbb{N}$

$$O(f(n)) = \{g : \mathbb{N} \to \mathbb{N} \mid \text{es gibt Konstanten } c \text{ und } n_0,. \text{ sodass für alle } n \geq n_0 \text{ gilt: } g(n) \leq c \cdot f(n) \ \}$$

Statt „$\in$“ wird dabei oft „$=$“ geschrieben, zum Beispiel bei $5n^3 + 12n^2 + 3 = O(n^3)$.

Als *Problem* im Sinne der Komplexitätstheorie wird die Akzeptierung von Sprachen betrachtet. Ein Problem besteht also darin, für ein Eingabewort, das dann als *Instanz des Problems* bezeichnet wird, zu erkennen, ob es Element einer bestimmten Sprache ist.

1.2 Komplexitätsmaße und -klassen

Die Komplexitätstheorie beschäftigt sich mit der Frage nach der „Schwierigkeit“ von algorithmisch lösbaren Problemen. Die wichtigsten Maße, diese Komplexität zu messen, sind die benötigte Zeit und der verbrauchte Speicherplatz. Die Aussagen zur Komplexität sollen möglichst unabhängig von konkreten Rechnerarchitekturen getroffen werden können. Das der Definition der Komplexitätsmaße zugrunde liegende formale Maschinenmodell sollte deshalb auf der einen Seite so abstrakt sein, dass Aussagen relativ einfach getroffen werden können, auf der anderen Seite sollten diese Aussagen möglichst nicht zu weit von wirklich vorhandenen Rechnerarchitekturen entfernt sein [30].

Das am häufigsten verwendete Modell ist das der in Abschnitt 1.1 vorgestellten Turingmaschine.

Als Grundlage zur Definition der Zeit- und Platzkomplexität dient eine Mehrband-Turingmaschine, die im Folgenden oft mit *TM* abgekürzt wird. Für die nichtdeterministische Variante wird die Abkürzung *NTM* verwendet.

Definition 1.8

Für eine Mehrband-Turingmaschine $\mathcal{M}$ wird für eine Eingabe x das Zeitkomplexitätsmaß $time_{\mathcal{M}}(x)$ wie folgt definiert:

$$time_{\mathcal{M}}(x) = \begin{cases} t\ , & \text{falls } \mathcal{M} \text{ bei Eingabe von } x \text{ nach genau } t \\ & \text{Schritten stoppt.} \\ 0\ , & \text{falls } \mathcal{M} \text{ bei Eingabe von } x \\ & \text{nicht stoppt.} \end{cases}$$

Dazu analog wird das Platzkomplexitätsmaß $space_{\mathcal{M}}(x)$ definiert:

$$space_{\mathcal{M}}(x) = \begin{cases} z\ , & \text{falls } \mathcal{M} \text{ bei Eingabe von } x \text{ nach endlich} \\ & \text{vielen Schritten stoppt und genau } z \text{ Zellen} \\ & \text{der Arbeitsbänder betreten hat.} \\ 0\ , & \text{falls } \mathcal{M} \text{ bei Eingabe von } x \\ & \text{nicht stoppt.} \end{cases}$$

Definition 1.9

Eine Turingmaschine $\mathcal{M}$ heißt *zeitbeschränkt durch eine Funktion* f, falls für alle x gilt: $time_{\mathcal{M}}(x) \leq f(|x|)$.
Analog heißt $\mathcal{M}$ *platzbeschränkt durch* g, falls für alle x gilt:
$space_{\mathcal{M}}(x) \leq g(|x|)$.

Mithilfe dieser Komplexitätsmaße lassen sich *(deterministische) Komplexitätsklassen* von Sprachen definieren:

Definition 1.10

Für ein festes Eingabealphabet Σ sei definiert:

$$\begin{aligned} Time(f) = \{L \subseteq \Sigma^* \mid & \text{Es gibt eine durch } f \text{ zeitbeschränkte} \\ & \text{TM } \mathcal{M} \text{ mit } L(\mathcal{M})\text{=L } \} \\ Space(g) = \{L \subseteq \Sigma^* \mid & \text{Es gibt eine durch } g \text{ platzbeschränkte} \\ & \text{TM } \mathcal{M} \text{ mit } L(\mathcal{M})\text{=L } \} \end{aligned}$$

Die Klassen $\mathcal{P}$ und $\mathcal{PSPACE}$ enthalten alle Sprachen mit polynomieller Zeit- beziehungsweise Platzschranke:

Definition 1.11

$$\mathcal{P} = \bigcup_{k \in \mathbb{N}} Time(n^k)$$
$$\mathcal{PSPACE} = \bigcup_{k \in \mathbb{N}} Space(n^k)$$

Die Definition der Zeit- und Platzkomplexitätsmaße für nichtdeterministische Turingmaschinen muss etwas abgewandelt werden, da es mehrere akzeptierende Berechnungen geben kann. Unter diesen wird die mit dem kürzesten Rechenweg beziehungsweise die mit der minimalen Anzahl der betretenen Felder ausgewählt.

Definition 1.12
Sei $\mathfrak{M}$ eine nichtdeterministische Mehrband-Turingmaschine.

$$ntime_{\mathfrak{M}}(x) = \begin{cases} min\{t \mid \text{Es gibt eine Berechnung von } \mathfrak{M} \text{ bei Eingabe von} \\ \qquad\qquad x, \text{ die nach genau } t \text{ Schritten stoppt. } \} \\ 0 \quad \text{, falls } \mathfrak{M} \text{ bei Eingabe von } x \text{ nicht stoppt.} \end{cases}$$

$$nspace_{\mathfrak{M}}(x) = \begin{cases} min\{z \mid \text{Es gibt eine Berechnung von } \mathfrak{M} \text{ bei Eingabe von} \\ \qquad\qquad x, \text{ die nach endlich vielen Schritten stoppt und} \\ \qquad\qquad \text{genau } z \text{ Zellen auf den Arbeitsbändern nutzt. } \} \\ 0 \quad \text{, falls } \mathfrak{M} \text{ bei Eingabe von } x \text{ nicht stoppt.} \end{cases}$$

Bie Begriffe der Zeit- und Platzbeschränktheit werden analog zum deterministischen Fall verwendet, die *nichtdeterministischen Komplexitätsklassen* sind dann:

Definition 1.13

$$NTime(f) = \{L \subseteq \Sigma^* \mid \text{Es gibt eine durch } f \text{ zeitbeschränkte NTM } \mathfrak{M} \text{ mit } L(\mathfrak{M})\text{=L } \}$$
$$NSpace(g) = \{L \subseteq \Sigma^* \mid \text{Es gibt eine durch } g \text{ platzbeschränkte NTM } \mathfrak{M} \text{ mit } L(\mathfrak{M})\text{=L } \}$$

Die nichtdeterministischen Klassen $\mathcal{NP}$ und $\mathcal{NPSPACE}$ werden analog zum deterministischen Fall definiert:

Definition 1.14

$$\mathcal{NP} = \bigcup_{k \in \mathbb{N}} NTime(n^k)$$
$$\mathcal{NPSPACE} = \bigcup_{k \in \mathbb{N}} NSpace(n^k)$$

Nach dem *Satz von Savitch* gilt $\mathcal{PSPACE} = \mathcal{NPSPACE}$, außerdem gelten folgende Inklusionen:

$$\mathcal{P} \subseteq \mathcal{NP} \subseteq \mathcal{PSPACE}$$

Die Vermutung, dass es sich hierbei um echte Inklusionen handelt, konnte bislang nicht gezeigt werden. Ob $\mathcal{P} = \mathcal{NP}$ oder $\mathcal{P} \neq \mathcal{NP}$ ist, ist die bekannteste offene Frage der Theoretischen Informatik.

Die zweite Inklusion folgt daraus, dass eine Turingmaschine in t Schritten nicht mehr als $O(t)$ Speicherzellen betreten kann. Somit ist $\mathcal{NP} \subseteq \mathcal{NPSPACE} = \mathcal{PSPACE}$. Auch hier konnte die Frage, ob $\mathcal{NP} = \mathcal{PSPACE}$ oder $\mathcal{NP} \neq \mathcal{PSPACE}$ gilt, noch nicht beantwortet werden.

1.3 Reduzierbarkeit und Vollständigkeit

Definition 1.15

Es seien $A \subseteq \Sigma_1^*$ und $B \subseteq \Sigma_2^*$ zwei Sprachen. A heißt *polynomiell reduzierbar auf* B, $A \leq_p B$, falls es eine totale und in polynomieller Zeit berechenbare Funktion $f : \Sigma_1^* \to \Sigma_2^*$ gibt, so dass für alle $x \in \Sigma_1^*$ gilt:

$$x \in A \Leftrightarrow f(x) \in B.$$

Polynomielle Reduzierbarkeit bedeutet also, dass sich ein Problem mithilfe einer Transformationsfunktion f in ein anderes überführen lässt. Dabei kann sich die Problemgröße höchstens polynomiell vergrößern, da die Transformationsfunktion polynomialzeitbeschränkt ist.

Ist B in einer der Komplexitätsklassen $\mathcal{P}$, $\mathcal{NP}$ oder $\mathcal{PSPACE}$ und ist $A \leq_p B$, so ist auch A in der jeweiligen Klasse.

Es lassen sich nun schwierigste Probleme innerhalb der Komplexitätsklassen definieren:

Definition 1.16

Eine Sprache A heißt $\mathcal{NP}$-*schwer*, falls sich jede Sprache $B \in \mathcal{NP}$ polynomiell auf A reduzieren lässt.
A heißt $\mathcal{NP}$-*vollständig*, falls sie $\mathcal{NP}$-schwer ist und $A \in \mathcal{NP}$ gilt.
Analog dazu lauten die Definitionen von $\mathcal{PSPACE}$-*schwer* und $\mathcal{PSPACE}$-*vollständig*.

Die Definition von $\mathcal{P}$-Vollständigkeit bezüglich der polynomiellen Reduktion ist nicht sinnvoll, da so alle nichttrivialen Sprachen in $\mathcal{P}$ vollständig wären (denn jede Sprache ist durch eine polynomiell zeitbeschränkte Turingmaschine erkennbar).

Würde nun von einer $\mathcal{NP}$-vollständigen Sprache gezeigt, dass sie in $\mathcal{P}$ läge, so wäre $\mathcal{P} = \mathcal{NP}$, denn jede Sprache in $\mathcal{NP}$ ließe sich dann polynomiell auf diese reduzieren.

Das gleiche gilt für die Frage nach $\mathcal{NP} \stackrel{?}{=} \mathcal{PSPACE}$.

1.4 Das Erfüllbarkeitsproblem SAT

Das Erfüllbarkeitsproblem für Boolesche Formeln *SAT* (*satisfiability*) war das erste, für das die $\mathcal{NP}$-Vollständigkeit nachgewiesen werden konnte, durch Stephen Cook im Jahr 1971. Da das in Abschnitt 1.5 vorgestellte $\mathcal{PSPACE}$-vollständige Problem der Quantifizierten Booleschen Formeln eine Verallgemeinerung des SAT-Problems darstellt, soll dieses zunächst näher betrachtet werden.

Definition 1.17

Es seien $x_1, \ldots, x_n$ *Boolesche Variablen*. Eine *Boolesche Formel* kann auf folgende Arten gebildet werden:

1. Die Konstanten 0 und 1 sind Boolesche Formeln.
2. x_i ist eine (atomare) Boolesche Formel ($i = 1, \ldots, n$).
3. Für zwei Boolesche Formeln ϕ und ρ sind
   - die *Disjunktion* $(\phi \vee \rho)$,
   - die *Konjunktion* $(\phi \wedge \rho)$ und
   - die *Negation* $(\neg\phi)$

   Boolesche Formeln.

Boolesche Variablen können die Werte 0 und 1 annehmen:

Definition 1.18

Eine *Belegung* β für eine Menge Boolescher Variablen $M = \{x_i \mid i = 1, \ldots, n\}$ ist eine Abbildung $\beta : M \to \{0, 1\}$.

Der Wert einer Booleschen Formel für eine Belegung β wird folgendermaßen definiert:

- $\text{Wert}_\beta(1) = 1$,
  $\text{Wert}_\beta(0) = 0$
- $\text{Wert}_\beta(x_i) = \begin{cases} 1 \text{ , falls } \beta(x_i) = 1 \\ 0 \text{ , falls } \beta(x_i) = 0 \end{cases}$
- $\text{Wert}_\beta(\phi \vee \rho) = \begin{cases} 1 \text{ , falls } \text{Wert}_\beta(\phi) = 1 \text{ oder } \text{Wert}_\beta(\rho) = 1 \\ 0 \text{ , sonst} \end{cases}$
- $\text{Wert}_\beta(\phi \wedge \rho) = \begin{cases} 1 \text{ , falls } \text{Wert}_\beta(\phi) = 1 \text{ und } \text{Wert}_\beta(\rho) = 1 \\ 0 \text{ , sonst} \end{cases}$
- $\text{Wert}_\beta(\neg\phi) = \begin{cases} 1 \text{ , falls } \text{Wert}_\beta(\phi) = 0 \\ 0 \text{ , falls } \text{Wert}_\beta(\phi) = 1 \end{cases}$

Definition 1.19

Eine Boolesche Formel ϕ mit Variablen $x_1, \ldots, x_n$ heißt genau dann *erfüllbar*, wenn es eine Belegung β gibt, sodass $\text{Wert}_\beta(\phi) = 1$ ist. Ein solches β heißt dann *erfüllende Belegung*, es gilt $\beta \models \phi$.

Im Folgenden sollen Boolesche Formeln betrachtet werden, die in *Konjunktiver Normalform* vorliegen:

Definition 1.20

Ein *Literal* x_i^ε ist eine Variable oder ihre Negation:

$$x_i^\varepsilon = \begin{cases} x_i \text{ , falls } \varepsilon = 1 \\ \neg x_i \text{ , falls } \varepsilon = 0 \end{cases}$$

Eine *Klausel* K ist eine Disjunktion von Literalen:

$$K = x_{i_1}^{\varepsilon_1} \vee \ldots \vee x_{i_k}^{\varepsilon_k} = \bigvee_{j=1}^{k} x_{i_j}^{\varepsilon_j}$$

Eine Boolesche Formel Φ in *Konjunktiver Normalform (KNF)* ist eine Konjunktion von Klauseln:

$$\Phi = K_1 \wedge \ldots \wedge K_m = \bigwedge_{l=1}^{m} K_l$$

Analog dazu kann auch die *Disjunktive Normalform* als Disjunktion von Literalkonjunktionen definiert werden.

Das Erfüllbarkeitsproblem SAT kann nun als Menge aller erfüllbaren Booleschen Formeln, welche in Konjunktiver Normalform vorliegen, definiert werden:

Problem 1.1 (SAT)
Gegeben sei eine Boolesche Formel Φ in Konjunktiver Normalform.
Gefragt ist, ob Φ erfüllbar ist.

$$SAT = \{\Phi \mid \Phi \text{ ist erfüllbare Boolesche Formel in KNF}\}$$

Satz 1.1 (Cook,1971) *SAT ist $\mathcal{NP}$-vollständig.*

Beweis: [Skizze] Es muss gezeigt werden, dass SAT $\in \mathcal{NP}$ ist und dass $L \leq_p$ SAT für alle $L \in \mathcal{NP}$.

$SAT \in \mathcal{NP}$: Eine NTM rät eine Belegung der Variablen und überprüft, ob die gegebene Formel bei dieser Belegung den Wert 1 hat. Diese Überprüfung ist in linearer Zeit möglich.

$L \leq_p SAT$ für alle $L \in \mathcal{NP}$: Die Arbeitsweise einer beliebigen NTM $\mathfrak{M}$ mit polynomieller Zeitschranke $p(n)$ soll durch eine Boolesche Formel F vollständig beschrieben werden, sodass für eine Eingabe $x = x_1 \ldots x_n$ gilt:

$$x \in L(\mathfrak{M}) \Leftrightarrow F \text{ ist erfüllbar.}$$

O.B.d.A. habe $\mathfrak{M}$ nur ein Band (eine Maschine mit mehreren Bändern lässt sich durch eine Einbandmaschine in quadratischer Zeit simulieren). Die Menge der Zustände sei $Z = (z_0, \ldots, z_r)$, z_r sei der einzige akzeptierende Zustand. Das Bandalphabet sei $\Gamma = (a_0 = \square, \ldots, a_\nu)$.
Es werden Boolesche Variablen wie folgt definiert:

- $q(i,k)$ für $0 \leq i \leq p(n)$ und $0 \leq k \leq r$.
  Die Belegung von $q(i,k)$ mit 1 bedeutet, dass sich $\mathfrak{M}$ nach i Schritten im Zustand z_k befindet.

- $h(i,j)$ für $0 \leq i \leq p(n)$ und $-p(n) \leq j \leq p(n)+1$.
  Die Belegung von $h(i,j)$ mit 1 bedeutet, dass sich der Lese-Schreib-Kopf nach i Schritten auf Position j befindet.

- $s(i,j,k)$ für $0 \leq i \leq p(n)$, $-p(n) \leq j \leq p(n)+1$ und $0 \leq k \leq \nu$.
  Die Belegung von $s(i,j,k)$ mit 1 bedeutet, dass sich nach i Schritten auf Bandposition j das Zeichen a_k befindet.

Es wird nun mithilfe dieser Variablen die Funktionsweise von $\mathfrak{M}$ durch eine Boolesche Formel F in Konjunktiver Normalform beschrieben. F setzt sich dabei aus sieben Teilformeln zusammen, die dann konjunktiv verknüpft werden:

- Zu jedem Zeitpunkt i gilt genau ein Zustand z_k:

$$F_1 = \bigwedge_{i=0}^{p(n)} \left[\left(\bigvee_{k=0}^{r} q(i,k) \right) \wedge \bigwedge_{\substack{k=0 \\ k'>k}}^{r} \left(\neg q(i,k) \vee \neg q(i,k') \right) \right]$$

  Der erste Teil der Formel sorgt dafür, dass mindestens ein Zustand gilt, aufgrund des zweiten Teiles können nicht zwei Zustände gleichzeitig gelten.

- Zu jedem Zeitpunkt i steht der Lese-Schreib-Kopf auf genau einem Feld j:

$$F_2 = \bigwedge_{i=0}^{p(n)} \left[\left(\bigvee_{j=-p(n)}^{p(n)+1} h(i,j) \right) \wedge \bigwedge_{\substack{j=-p(n) \\ j'>j}}^{p(n)} \left(\neg h(i,j) \vee \neg h(i,j') \right) \right]$$

- Zu jedem Zeitpunkt i steht auf jedem Feld j des Bandes genau ein Zeichen a_k:

$$F_3 = \bigwedge_{i=0}^{p(n)} \bigwedge_{j=-p(n)}^{p(n)+1} \left[\left(\bigvee_{k=0}^{\nu} s(i,j,k) \right) \wedge \bigwedge_{\substack{k=0 \\ k'>k}}^{\nu} \left(\neg s(i,j,k) \vee \neg s(i,j,k') \right) \right]$$

- Die Anfangskonfiguration muss beschrieben werden:

$$F_4 = q(0,0) \wedge h(0,1) \wedge \bigwedge_{j=1}^{n} s(0,j,k_j) \wedge \bigwedge_{j=-p(n)}^{0} s(0,j,0) \wedge \bigwedge_{j=n+1}^{p(n)+1} s(0,j,0)$$

  Dabei ist $x = a_{k_1} \ldots a_{k_n}$ das Eingabewort.

- Nach höchstens $p(n)$ Schritten ist der akzeptierende Zustand z_r erreicht:

$$F_5 = \bigvee_{i=0}^{p(n)} q(i,r)$$

- Die Veränderung von einer Konfiguration zur nächsten muss beschrieben werden:

$$F_6 = \bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=-p(n)}^{p(n)+1} \bigwedge_{k=0}^{r} \bigwedge_{l=0}^{\nu} \Bigg[\neg h(i,j) \vee \neg q(i,k) \vee \neg s(i,j,l) \quad \vee \bigvee_{(z_{k'},a_{l'},\Delta)\in\delta(z_k,a_l)} \big(h(i+1,j+\Delta) \wedge q(i+1,k') \wedge s(i+1,j,l')\big) \Bigg]$$

  Im letzten Teil der Formel findet sich auch der Nichtdeterminismus wieder. Dieser Teil der Formel ist noch nicht in Konjunktiver Normalform, durch Einführung zusätzlicher Variablen ist diese aber relativ einfach zu erhalten.

- Schließlich muss noch verhindert werden, dass sich die Beschriftung der Felder verändert, auf denen sich der Lese-Schreib-Kopf nicht befindet:

$$F_7 = \bigwedge_{i=0}^{p(n)-1} \bigwedge_{j=-p(n)}^{p(n)+1} \bigwedge_{l=0}^{\nu} [\neg s(i,j,l) \vee h(i,j) \vee s(i+1,j,l)]$$

Es werden $O(p(n)^2)$ Variablen benötigt und $O(p(n)^2)$ Klauseln erzeugt. Somit hat F polynomielle Länge bezüglich der Länge n des Eingabewortes x und kann auch in polynomieller Zeit berechnet werden.

Ein detaillierterer Beweis findet sich zum Beispiel in [12].

□

Schränkt man die Anzahl der Literale pro Klausel auf drei ein, so erhält man das immer noch $\mathcal{NP}$-vollständige Problem 3SAT:

Problem 1.2 (3SAT)

Gegeben sei eine Boolesche Formel Φ in Konjunktiver Normalform mit genau drei Literalen pro Klausel (*3KNF*).
Gefragt ist, ob Φ erfüllbar ist.

$$3SAT = \{\Phi \mid \Phi \text{ ist erfüllbare Boolesche Formel in 3KNF}\}$$

Satz 1.2 (SAT $\leq_p$ 3SAT) *SAT ist polynomiell reduzierbar auf 3SAT.*

Beweis: Es wird jede Klausel einzeln betrachtet und in eine Klausel mit drei Literalen umgewandelt. Dabei werden für jede Klausel eigene neue Variablen eingeführt.

Klauseln mit einem Literal x^ε werden umgeschrieben in

$$(x^\varepsilon \vee y_1^1 \vee y_2^1) \wedge (x^\varepsilon \vee y_1^0 \vee y_2^1) \wedge (x^\varepsilon \vee y_1^1 \vee y_2^0) \wedge (x^\varepsilon \vee y_1^0 \vee y_2^0).$$

Klauseln mit zwei Literalen $(x_1^{\varepsilon_1} \vee x_2^{\varepsilon_2})$ werden umgeschrieben in

$$(x_1^{\varepsilon_1} \vee x_2^{\varepsilon_2} \vee y^1) \wedge (x_1^{\varepsilon_1} \vee x_2^{\varepsilon_2} \vee y^0).$$

Klauseln mit mehr als drei Literalen $(x_1^{\varepsilon_1} \vee \ldots \vee x_k^{\varepsilon_k})$ werden umgeschrieben in

$$(x_1^{\varepsilon_1} \vee x_2^{\varepsilon_2} \vee y_1^1) \wedge (y_1^0 \vee x_3^{\varepsilon_3} \vee y_2^1) \wedge (y_2^0 \vee x_4^{\varepsilon_4} \vee y_3^1) \wedge \ldots \wedge (y_{k-3}^0 \vee x_{k-1}^{\varepsilon_{k-1}} \vee x_k^{\varepsilon_k})$$

Die Äquivalenz im Fall $k > 3$ ergibt sich folgendermaßen: Wird die ursprüngliche Klausel erfüllt, zum Beispiel durch die Belegung von $x_i^{\varepsilon_i}$, dann wählt man als Belegung $\beta(y_j) = 1$ für $j \leq i-2$ und $\beta(y_j) = 0$ für $j > i-2$. Die ersten $i-2$ der neuen Klauseln werden nun durch die y_j^1 erfüllt, die $(i-1)$-ste Klausel durch $x_i^{\varepsilon_i}$ und die restlichen Klauseln durch y_j^0.

Sind andererseits alle neuen Klauseln erfüllt, so sind drei Fälle zu unterscheiden: Haben bis zu einem $i < k-3$ alle y_i die Belegung 1 und y_{i+1} hat die Belegung 0, so kann die $(i+1)$-ste Klausel nur dadurch erfüllbar sein, dass $x_{i+2}^{\varepsilon_{i+2}} = 1$ ist,d.h die ursprüngliche Klausel ist erfüllbar. Haben alle y_j die Belegung 1, so muss $(x_{k-1}^{\varepsilon_{k-1}} \vee x_k^{\varepsilon_k})$ erfüllt sein und somit auch die ursprüngliche Klausel. Hat y_1 die Belegung 0, so muss $(x_1^{\varepsilon_1} \vee x_2^{\varepsilon_2})$ erfüllt sein und damit auch die gesamte ursprüngliche Formel.

□

Somit ist auch 3SAT $\mathcal{NP}$-vollständig.

1.5 Quantifizierte Boolesche Formeln

Definition 1.21

Eine *Quantifizierte Boolesche Formel* F ist ein Ausdruck der Form

$$F = Q_n x_n Q_{n-1} x_{n-1} \dots Q_1 x_1 \ \Phi,$$

wobei Φ eine Boolesche Formel mit den Variablen $x_1, \dots, x_n$ ist und $Q_i \in \{\exists, \forall\}$ Quantoren sind. Dabei heißt $\exists$ Existenz- und $\forall$ Allquantor.

Die Definition, wann eine Quantifizierte Boolesche Formel wahr ist, wird rekursiv gegeben:

Definition 1.22

Es sei $F = Q_n x_n Q_{n-1} x_{n-1} \dots Q_1 x_1 \ \Phi$ eine Quantifizierte Boolesche Formel.
Φ_0 und Φ_1 seien die Formeln, die man aus Φ durch Ersetzen der Variablen x_n durch Konstanten 0 beziehungsweise 1 erhält.
F_i $(i = 1, 2)$ sei die quantifizierte Formel $Q_{n-1} x_{n-1} \dots Q_1 x_1 \ \Phi_i$.

F heißt *wahr*, falls

$$\begin{array}{rrll} & n = 0 & \text{und} & \Phi \equiv 1 \\ \text{oder} & Q_n = \exists & \text{und} & F_0 \text{ oder } F_1 \text{ wahr sind} \\ \text{oder} & Q_n = \forall & \text{und} & F_0 \text{ und } F_1 \text{ wahr sind.} \end{array}$$

Im Fall $n = 0$ enthält F keine Variablen, sondern nur Boolesche Konstanten. $\Phi \equiv 1$ bedeute, dass $\text{Wert}_\beta \Phi = 1$ für jede Belegung β.

Problem 1.3 (QBF)

Gegeben sei eine Quantifizierte Boolesche Formel $F = Q_n x_n \cdots Q_1 x_1 \Phi$ mit Variablen $x_1, \dots, x_n$, Φ liege in Konjunktiver Normalform vor.
Gefragt ist, ob F wahr ist.

$$QBF = \{F = Q_n x_n \dots Q_1 x_1 \Phi \mid F \text{ ist eine wahre Quantifizierte Boolesche Formel mit } \Phi \text{ in KNF } \}.$$

Das Problem SAT kann als Spezialfall von QBF aufgefasst werden: Wenn Φ eine Boolesche Formel mit den Variablen $x_1, \dots, x_n$ ist, dann gilt:

$$\Phi \in SAT \Leftrightarrow \exists x_n \exists x_{n-1} \dots \exists x_1 \Phi \in QBF$$

Satz 1.3 (Stockmeyer, Meyer) *QBF ist $\mathcal{PSPACE}$-vollständig.*

Beweis: [Skizze] Der Beweis wird analog zum Beweis des Satzes von Cook (Satz 1.1) mithilfe der Simulation einer beliebigen deterministischen Turingmaschine mit polynomiell beschränktem Platzbedarf durch eine Quantifizierte Boolesche Formel geführt.

$QBF \in \mathcal{PSPACE}$: Für jede Belegung der n Variablen kann auf linearem Platz entschieden werden, ob Φ erfüllt ist. Dadurch kann durch eine rekursive Berechnung ebenfalls auf linearem Platz entschieden werden, ob F wahr ist.

$L \leq_p QBF$ für alle $L \in \mathcal{PSPACE}$: Sei $\mathfrak{M}$ eine deterministische Turingmaschine mit polynomieller Platzbeschränkung. Dann gibt es ein Polynom p, sodass $\mathfrak{M}$ höchstens $2^{p(n)}$ Konfigurationen zur Akzeptierung eines Wortes der Länge n benötigt. Daher können die Konfigurationen als Wörter über $\{0,1\}$ der Länge $p(n)$ dargestellt werden. O.B.d.A. habe $\mathfrak{M}$ eine einzige akzeptierende Konfiguration.

Es wird nun eine Boolesche Formel $F_i(a,b)$ definiert, die genau dann wahr ist, wenn $a = a_1 \dots a_{p(n)}$ und $b = b_1 \dots b_{p(n)}$ Konfigurationen von $\mathfrak{M}$ beschreiben und b von a aus in höchstens 2^i Schritten erreicht werden kann.

Es gilt nun: $x \in L(\mathfrak{M}) \Leftrightarrow F_{p(n)}(A,B) = 1$, wobei A die Startkonfiguration von $\mathfrak{M}$ bei Eingabe von x und B die akzeptierende Konfiguration ist.

Die Boolesche Formel $F_{i+1}(a,b)$ kann rekursiv definiert werden:

$$F_{i+1}(a,b) = \exists z(F_i(a,z) \wedge F_i(z,b)) \tag{1.1}$$

Hierbei stehe $\exists z$ für $\exists z_1 \dots \exists z_{p(n)}$. Es beschreibt z also die Konfiguration bei der Hälfte der Berechnung.

Die Gleichung 1.1 wird allerdings exponentiell groß. Dies kann durch folgende Konstruktion entgangen werden:

$$F_{i+1}(a,b) = \exists z \forall x \forall y[(x = a \wedge y = z) \vee (x = z \wedge y = b) \rightarrow F_i(x,y)] \tag{1.2}$$

Hierbei sind x und y wieder Wörter der Länge $p(n)$, es stehe $\forall x$ wiederum für $\forall x_1 \dots \forall x_{p(n)}$ Die Gleichheit zweier Wörter $x = a$ wird durch die Konjunktion $(x_1 = a_1) \wedge \dots \wedge (x_{p(n)} = a_{p(n)})$ beschrieben, die Gleichheit zweier Buchstaben $x_i = a_i$ durch $(x_i \wedge a_i) \vee (\neg x_i \wedge \neg a_i)$. Die Implikation $\psi_1 \rightarrow \psi_2$ wird beschrieben durch $\neg\psi_1 \vee \psi_2$. Nun tritt F_i nur einmal auf, die Größe der Formel wächst also nicht mehr exponentiell.

Die Formel $F_0(a,b)$ muss beschreiben, dass a und b gleich sind oder dass b in einem Schritt aus a folgt. Dies ist durch eine Boolesche Formel mit polynomiell vielen Literalen möglich.

Das Problem, dass die Quantoren auch innerhalb der Formel 1.2 zu finden sind, kann durch einfaches Vorziehen der Quantoren behoben werden, da die quantifizierten Variablen nicht außerhalb der Teilformel auftreten. Ein größeres Problem ist,

dass die Formel nicht in Konjunktiver Normalform vorliegt. Bei deren Berechnung entstünden exponentiell viele Klauseln. Allerdings kann die Disjunktive Normalform von F_{i+1} leicht berechnet werden, sie hat polynomielle Größe.

Die Menge der wahren Quantifizierten Booleschen Formeln, die in Disjunktiver Normalform vorliegen, kann auf das Komplement von QBF reduziert werden. Dies folgt aus der Anwendung der DeMorganschen Regeln:

$$\begin{aligned} & Q_n x_n \dots Q_1 x_1 \bigvee_{i=1}^{m} \bigwedge_{j=1}^{k} x_{ij}^{\varepsilon_{ij}} = 1 \text{ mit } x_{ij}^{\varepsilon_{ij}} \in \{x_1, \neg x_1, \dots, x_n, \neg x_n\} \\ \Leftrightarrow & \overline{Q}_n x_n \dots \overline{Q}_1 x_1 \bigwedge_{i=1}^{m} \bigvee_{j=1}^{k} x_{ij}^{\overline{\varepsilon}_{ij}} = 0 \text{ mit } \overline{Q}_i = \begin{cases} \exists, \text{ falls } Q_i = \forall \\ \forall, \text{ falls } Q_i = \exists \end{cases}, \\ & \overline{\varepsilon}_{ij} = \begin{cases} 0, \text{ falls } \varepsilon_{ij} = 1 \\ 1, \text{ falls } \varepsilon_{ij} = 0 \end{cases}. \end{aligned}$$

Es kann also jedes Problem aus $\mathcal{PSPACE}$ auf $\overline{\text{QBF}}$ polynomiell reduziert werden. Da $\mathcal{PSPACE}$ komplementabgeschlossen ist, ist auch QBF $\mathcal{PSPACE}$-vollständig.

□

Details des Beweises wie die Beschreibung der Disjunktiven Normalform können zum Beispiel bei Papadimitriou [28] gefunden werden, der erstmalige Nachweis der $\mathcal{PSPACE}$-Vollständigkeit von QBF erfolgte durch Meyer und Stockmeyer im Jahre 1973 [35].

Wie beim Erfüllbarkeitsproblem SAT kann die Formel auch auf drei Literale pro Klausel beschränkt werden:

Problem 1.4 (3QBF)

Gegeben sei eine Quantifizierte Boolesche Formel $F = Q_n x_n \dots Q_1 x_1 \Phi$ mit Variablen $x_1, \dots, x_n$, wobei Φ in Konjunktiver Normalform mit drei Literalen pro Klausel vorliege.

Gefragt ist wiederum, ob F wahr ist.

$$3QBF = \{F = Q_n x_n \dots Q_1 x_1 \Phi \mid F \text{ ist eine wahre Quantifizierte Boolesche Formel mit } \Phi \text{ in 3KNF } \}$$

Satz 1.4 (QBF $\leq_p$ 3QBF) *QBF ist polynomiell auf 3QBF reduzierbar.*

Beweis: Die Reduzierbarkeit folgt direkt aus der Reduzierbarkeit von SAT auf 3SAT. Neu auftretende Variablen werden mit Existenzquantoren rechts von den ursprünglichen Quantoren quantifiziert.

□

Damit ist auch 3QBF $\mathcal{PSPACE}$-vollständig.

1.6 Weitere PSPACE-vollständige Probleme

QBF war das erste Problem, für das die $\mathcal{PSPACE}$-Vollständigkeit gezeigt werden konnte. Es folgten eine Reihe weiterer Probleme, die hier kurz angegeben werden sollen, ohne auf die Beweise der Vollständigkeit einzugehen.

Viele $\mathcal{PSPACE}$-vollständige Probleme finden sich auf dem Gebiet der Automatentheorie, zum Beispiel das Problem, ob zwei gegebene nichtdeterministische endliche Automaten mit demselben Eingabealphabet die gleiche Sprachen erkennen, oder die Frage, ob ein gegebener linear beschränkter Automat ein bestimmtes Wort akzeptiert.

Auch auf dem Gebiet der Grammatiken gibt es eine Reihe von $\mathcal{PSPACE}$-vollständigen Problemen. Dazu gehören zum Beispiel die Frage, ob ein bestimmtes Wort von einer gegebenen kontextsensitiven Grammatik (Typ 1) erzeugt werden kann, oder die Frage, ob zwei gegebene kontextfreie Grammatiken (Typ 2) die gleiche Sprachen erzeugen. Ein Überblick über diese Probleme findet sich in [12].

Eine große Gruppe bilden verallgemeinerte Spiele für zwei Personen, genauer die Frage nach einer Gewinnstrategie. Diese Frage soll kurz am Geographie-Spiel erläutert werden. Der Name leitet sich aus dem Kinderspiel ab, bei dem abwechselnd geographische Namen genannt werden, wobei der Anfangsbuchstabe des neuen Namens der Endbuchstabe des letzten Namens sein soll; das Geographie-Spiel ist eine Verallgemeinerung: Gegeben sei ein gerichteter Graph mit einem definierten Startknoten. Spieler A markiert zu Beginn den Startknoten. Spieler A und B markieren nun abwechselnd jeweils einen Knoten, der direkter Nachfolger des zuletzt markierten Knotens ist und noch nicht markiert wurde. Gewonnen hat, wer einen Knoten markiert, sodass der Gegner keinen Knoten mehr markieren kann. Spieler A hat eine Gewinnstrategie, falls es eine Möglichkeit für ihn gibt, in jedem Zug einen Knoten so zu markieren, dass er unabhängig von der Handlung des Gegners am Ende immer gewinnt.

$$GEOGRAPHIE = \{X \mid X \text{ ist Kodierung eines gerichteten Graphen des Geographie-Spiels, bei dem Spieler A eine Gewinnstrategie hat. }\}$$

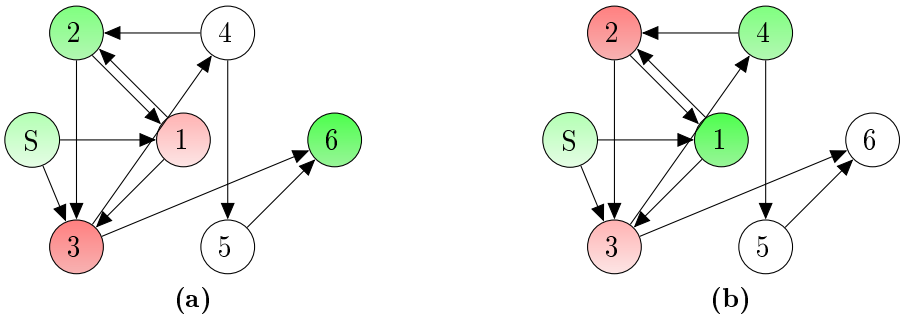


Abbildung 1.1: Instanz des Geographie-Spiels mit zwei möglichen Spielverläufen, es gewinnt jeweils Spieler A (grün markiert):
(a) $S \to 1 \to 2 \to 3 \to 6$, (b) $S \to 3 \to 4 \to 2 \to 1$.

GEOGRAPHIE ist $\mathcal{PSPACE}$-vollständig. Die Frage nach der Gewinnstrategie zeigt eine deutliche Ähnlichkeit zur Struktur Quantifizierter Boolescher Formeln: Gibt es einen Zug für Spieler A, sodass es für alle Züge des Gegners B einen Zug von A gibt, ..., sodass es für alle Züge von B einen Zug für A gibt und es danach keinen Zug für B mehr gibt?

Es gibt eine ganze Reihe solcher Spiele auf Graphen, die $\mathcal{PSPACE}$-vollständig sind. Für andere Spiele, zum Beispiel das verallgemeinerte GO-Spiel, ist nur bekannt, dass sie $\mathcal{PSPACE}$-schwer sind, jedoch nicht, ob sie auch in $\mathcal{PSPACE}$ liegen. Einen Überblick über viele dieser Spiele geben Garey und Johnson [12], viele Beweise der $\mathcal{PSPACE}$-Vollständigkeit oder -Schwere sind bei Reischuk [30] zu finden.

2 DNA-Computing

In diesem Kapitel sollen zunächst einige biologische Grundlagen erläutert werden: Es werden die Struktur und einige Eigenschaften des Erbmoleküls DNA beschrieben, ein Einblick in die Wirkungsweise enzymatischer Reaktionen gegeben und die im Labor zur Verfügung stehenden molekularbiologischen Methoden erklärt. Anschließend folgt ein kurzer Überblick über die Entwicklung des DNA-Computings. Am Schluss des Kapitels werden einige der zahlreichen theoretischen DNA-Computing-Modelle vorgestellt.

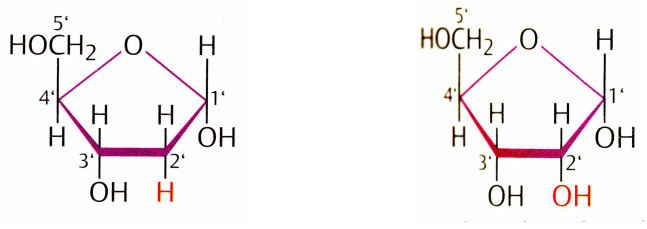
2.1 Biologische Grundlagen

2.1.1 Die DNA

DNA (Desoxyribonukleinsäure) ist das Material, mit dessen Hilfe die Erbinformation der Zelle gespeichert ist. Sie ist im Zellkern zu finden, woher sich auch der Name Nukleinsäure ableitet (*nukleus* lateinisch für Kern). DNA ist ein langkettiges Molekül (*Polymer*), dessen Grundbausteine (*Monomere*) kettenartig verbunden sind. Vier mögliche Monomere stehen zum Aufbau der DNA zur Verfügung, sie werden als Nukleotide bezeichnet. Ein Nukleotid besteht aus drei Bestandteilen: einem Zucker, einer Stickstoffbase und Phosphorsäure.

Der Zucker. Als Zuckerbaustein dient eine *Pentose* aus fünf Kohlenstoffatomen, die 2′-*Desoxyribose*, welche als Ringstruktur (*Haworthstruktur*) vorliegt (Abbildung 2.1). Die Kohlenstoffatome werden durchnummeriert; um sie von den Kohlen- und Stickstoffatomen der Base zu unterscheiden, werden sie mit einem Strich (') versehen. Der Name 2′-Desoxyribose weist darauf hin, dass im Unterschied zur normalen Ribose eine *Hydroxylgruppe* (-OH) am 2′-Kohlenstoffatom durch eine Wasserstoffgruppe (-H) ersetzt wurde. Ribose dient zum Aufbau der Nukleinsäure RNA (Ribonukleinsäure), einem der DNA ähnlichen Makromolekül.

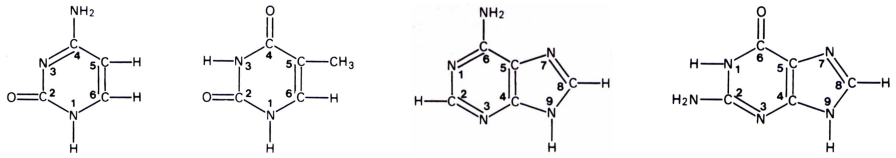
Die Base. Der zweite Baustein des Nukleotids ist eine stickstoffhaltige Base. Dabei unterscheidet man zwischen *Pyrimidinen*, die aus einem Ring bestehen, und *Purinen*, die sich aus zwei Ringen zusammensetzen. Pyrimidine sind *Cytosin* und *Thymin*, Purine sind *Adenin* und *Guanin* (Abbildung 2.2). Die Unterscheidung der Nukleotide erfolgt also aufgrund der Stickstoffbase. Wird der Zucker mit einer der



(a) **(b)**

Abbildung 2.1: Der Unterschied zwischen 2′-Desoxyribose (a) und Ribose (b) liegt in der Ersetzung der Hydroxylgruppe am 2′-Kohlenstoff durch eine Wasserstoffgruppe. Die Beschriftung der Kohlenstoffatome des Rings wird bei der Darstellung meist weggelassen. [4]

Basen verknüpft, so spricht man von einem *Nukleosid*. Die Bindung erfolgt über das 1′-Kohlenstoffatom des Zuckers und das 9-Stickstoffatom (bei Purinen) beziehungsweise das 1-Stickstoffatom (bei Pyrimidinen) der Base; es entsteht eine sogenannte *N-glykosidische Bindung* (Abbildung 2.3). Entsprechend der gebundenen Base heißen die vier Nukleoside Cytidin, Thymidin, Adenosin und Guanosin.



(a) **(b)** **(c)** **(d)**

Abbildung 2.2: Die Pyrimidine Cytosin (a) und Thymin (b) bestehen aus einem Ring mit sechs Kohlenstoff- und Stickstoffatomen, die Purine Adenin (c) und Guanin (d) aus zwei Ringen mit sechs beziehungsweise fünf Kohlenstoff- und Stickstoffatomen. [4]

Der Phosphorsäurebaustein. Ein *Nukleotid* entsteht aus einem Nukleosid durch Bindung einer Phosphorsäuregruppe am 5′-Kohlenstoffatom des Zuckers. Es können bis zu drei Phosphatgruppen gebunden werden, entsprechend spricht man dann von Nukleosidmono-, -di- oder -triphosphaten. Die einzelnen Phosphatgruppen werden durch die griechischen Buchstaben α, β und γ bezeichnet, wobei α dem Zucker am nächsten ist (Abbildung 2.4).

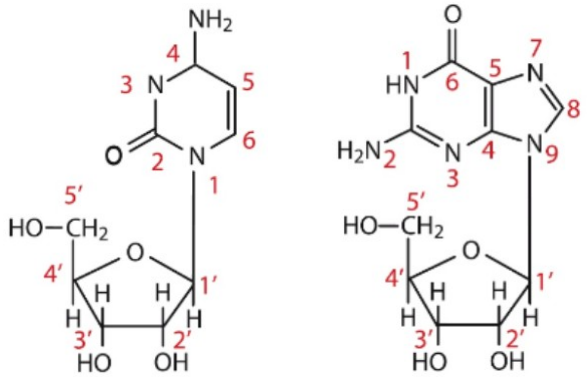


Abbildung 2.3: Die Bindung zwischen der Desoxyribose und einer Pyrimidinbase erfolgt zwischen dem 1′-C des Zuckers und dem 1-N der Base, hier am Beispiel des Cytidins dargestellt (a). Bei Purinbasen erfolgt die Bindung zum 9-N, als Beispiel hier Guanosin (b). [8]

Die vier Nukleotide, welche zu DNA polymerisieren können, heißen

2′-Desoxyadenosin-5'-triphosphat ($dATP$),
2′-Desoxyguanosin-5'-triphosphat ($dGTP$),
2′-Desoxycytidin-5'-triphosphat ($dCTP$) und
2′-Desoxythymidin-5'-triphosphat ($dTTP$),

je nachdem, welche der vier Basen gebunden ist.

Die Verknüpfung zweier Nukleotide erfolgt durch eine *Phosphodiesterbindung*. Dabei wird das α-Phosphat eines Nukleotids mit dem 3′-Kohlenstoff des anderen Nukleotids verknüpft; die beiden anderen Phosphatgruppen werden als Pyrophosphat abgespalten (Abbildung 2.5). Der Name Phosphodiesterbindung weist auf zwei Esterbindungen (C-O-C) hin, die zwischen den Zuckern und dem Phosphat entstehen.

Der so aufgebaute Strang besitzt zwei unterschiedliche Enden: das 3′-Ende mit einer freien Hydroxylgruppe, auch 3′-OH-Terminus genannt, und das 5′-Ende mit drei Phosphatgruppen, der 5′-P-Terminus. Dadurch besitzen DNA-Stränge eine Orientierung, sie können in $3' \rightarrow 5'$- sowie in $5' \rightarrow 3'$-Richtung betrachtet werden. [8]

Die Länge eines Polynukleotidstranges ist nicht begrenzt, in natürlichen Chromosomen können sie mehrere hundert Millionen Nukleotide lang sein. Auch in der Anordnung der vier verschiedenen Nukleotide gibt es keine Einschränkung, wodurch die zur Informationsspeicherung notwendige Variabilität erreicht wird. Ein Strang aus zehn Nukleotiden kann so bereits $4^{10} = 1\,048\,576$ verschiedene Nukleotidsequenzen haben.

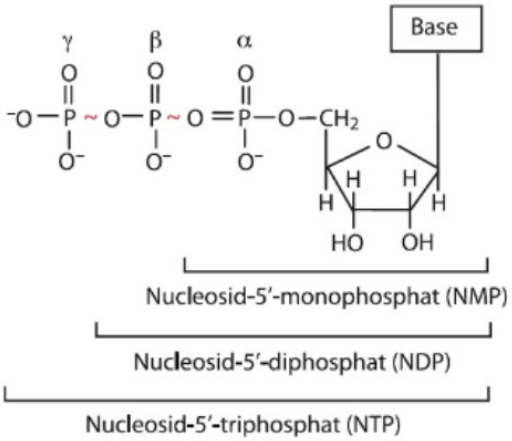


Abbildung 2.4: Ein Nukleosid kann bis zu drei Phosphatgruppen binden. [8]

Dass die Primärstruktur des DNA-Moleküls ein lineares Polymer ist, war seit Ende der 1930er Jahre bekannt, lange Zeit war aber unklar, in welcher Sekundärstruktur die DNA natürlicherweise in einer lebenden Zelle vorliegt. Einen ersten Hinweis lieferten Experimente von Erwin Chargaff, die nachwiesen, dass der Anteil der Pyrimidine gleich dem Anteil der Purine ist, und das unabhängig von den untersuchten DNA-Proben aus verschiedenen Geweben unterschiedlicher Organismen. Genauer zeigte Chargaff, dass der Anteil der Adenin- gleich dem der Thyminbausteine sowie der Anteil der Guanin- gleich dem der Cytosinbausteine ist. Den zweiten wichtigen Hinweis gaben Röntgenstrukturanalysen, die Rosalind Franklin 1952 mit einer Methode von Maurice Wilkins durchführte. Dabei wird kristallisierte DNA mit Röntgenstrahlen beschossen, die abhängig von der dreidimensionalen Struktur in bestimmten Winkeln abgelenkt werden. Das dabei entstehende Beugungsmuster zeigte, dass die DNA eine Helix ist. Aus diesen beiden und weiteren Hinweisen leiteten James Watson und Francis Crick im darauffolgenden Jahr am Cavendish Laboratory in Cambridge das nach ihnen benannte Modell der *Doppelhelix* ab (Abbildung 2.6). Zusammen mit Maurice Wilkins erhielten beide dafür 1962 den Nobelpreis für Medizin. (Rosalind Franklin, die den Preis wohl ebenso verdient hätte, war 1958 im Alter von nur 37 Jahren an Krebs gestorben.)

Die Doppelhelix besteht aus zwei Polynukleotiden, wobei die Stickstoffbasen im Inneren der Helix liegen und der Zucker sowie das Phosphat das Rückgrat bilden. Die Basen beider Stränge stehen über Wasserstoffbrücken in Verbindung. Diese Bindungen können sich nur zwischen einem Adenin- und einem Thyminrest sowie zwischen einem Guanin- und einem Cytosinrest ausbilden; das erklärt die von Chargaff entdeckten konstanten Basenverhältnisse. Wasserstoffbrückenbindungen sind schwache elektrostatische Anziehungen, die sich zwischen einem elektronegativen Atom - in

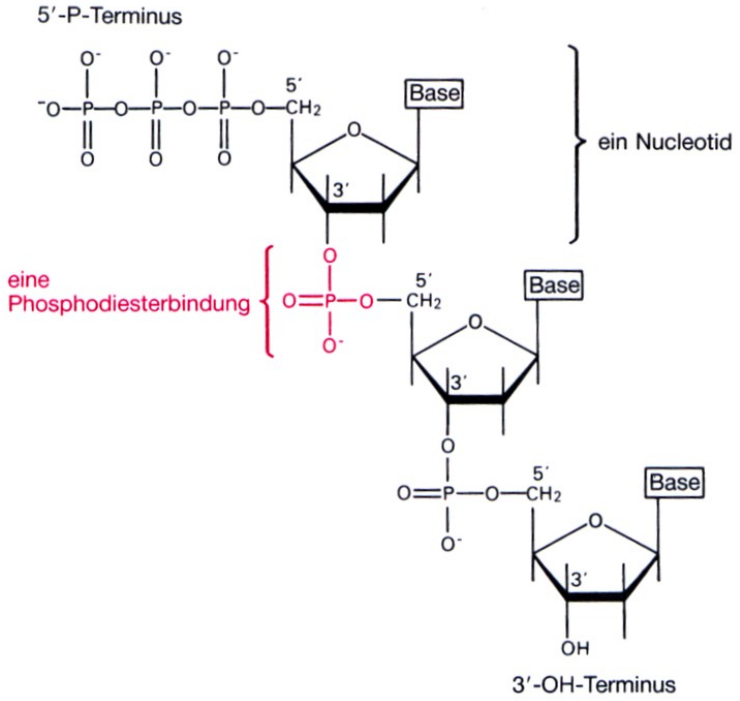


Abbildung 2.5: Ausbildung von Phosphodiesterbindungen. [4]

diesem Fall Sauerstoff oder Stickstoff - und Wasserstoff ausbilden. Zwischen Thymin und Adenin entstehen zwei, zwischen Cytosin und Guanin drei Brücken; diese schwachen Bindungen sind die einzigen, welche die beiden Stränge zusammenhalten (Abbildung 2.7).

Die Helix ist rechtsgewunden und hat eine Dicke von zwei Nanometern, eine Windung umfasst zehn Basenpaare und hat eine Länge von 3,4 Nanometern, der Abstand zwischen zwei benachbarten Basenpaaren beträgt also 0,34 Nanometer. Da sie nicht völlig regelmäßig ist, kann man zwei verschiedene Furchen unterscheiden, einer großen und einer kleinen. Dies ist von großer Bedeutung für die Wechselwirkung mit Proteinen. Die Stränge verlaufen nicht in der gleichen Richtung, sondern antiparallel, das heißt, eines der beiden Polynukleotide verläuft in $3' \rightarrow 5'$-Richtung, das andere in $5' \rightarrow 3'$-Richtung.

Die wohl wichtigste dieser Eigenschaften ist die Basenpaarung: Dadurch, dass es zu jeder der vier nur eine komplementäre Base gibt, ist durch Festlegung eines Stranges schon der andere genau bestimmt. Dies ist die zentrale Grundlage für Vererbung, denn durch die komplementäre Basenpaarung wird eine Vervielfältigung und Übertragung von Information möglich.

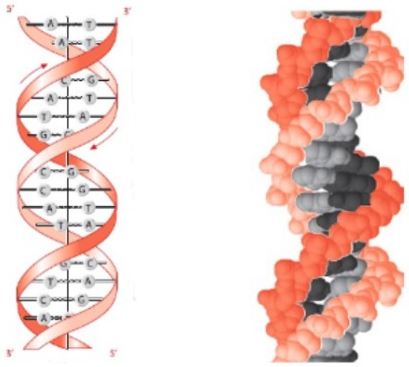
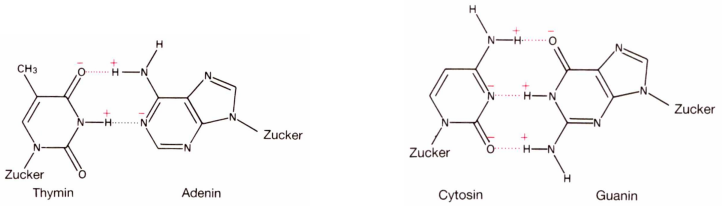


Abbildung 2.6: Die DNA liegt in der Zelle als Doppelhelix vor. (verändert nach [8])



(a) **(b)**

Abbildung 2.7: Zwischen Thymin und Adenin bilden sich zwei (a), zwischen Cytosin und Guanin drei Wasserstoffbrücken (b). [4]

Die hier beschriebene Sekundärstruktur ist die, die Watson und Crick entdeckten; sie wird heute B-DNA genannt. Sie ist die in der Zelle am häufigsten anzutreffende DNA-Struktur. Daneben wurden weitere Formen gefunden, die als A- und Z-DNA bezeichnet werden. Sie unterscheiden sich in der Anzahl der Basenpaare pro Windung, ihrem Abstand oder der Drehrichtung (Z-DNA ist linksgängig). Die unterschiedlichen Formen entstehen bei verschiedenen relativen Feuchtigkeiten - die B-Form zum Beispiel bei 92 % - oder bei bestimmten Sequenzfolgen (Z-DNA entsteht bei abwechselnd angeordneten Purin- und Pyrimidinnukleotiden). Unter Laborbedingungen wurden noch die weiteren Formen C-, D- und E-DNA erzeugt, die aber wahrscheinlich nie in natürlicher Umgebung auftreten.

Die Länge von DNA-Doppelsträngen wird in Basenpaaren (bp) angegeben, die von Einzelsträngen in Basen. Davon abgeleitete Einheiten der Länge sind Kilobasen (-paare) oder Megabasen (-paare) mit den Abkürzungen kb(p) und Mb(p). DNA-Moleküle aus bis zu 100 Basenpaaren werden als *Oligonukleotide* bezeichnet, bei längeren Molekülen spricht man von *Polynukleotiden*.

Neben DNA in linearer Form, in welcher bei allen höheren Lebewesen die Chromosomen im Zellkern vorliegen, gibt es in vielen Organismen auch ringförmige DNA. Einige Bakterienarten besitzen ein ringförmiges Chromosom, außerdem enthalten viele von ihnen auch kürzere ringförmige DNA-Stränge, die *Plasmide* genannt werden und zusätzliche genetische Informationen enthalten. Diese können zwischen zwei Bakterien ausgetauscht werden; auf diese Weise kann sich zum Beispiel Antibiotika-Resistenz ausbreiten. Plasmide werden in der Gentechnologie zur Übertragung von gewünschten DNA-Sequenzen in Organismen genutzt. [4]

2.1.2 Enzyme und ihre Wirkungsweise

Enzyme sind von herausragender Bedeutung für die Laborarbeit mit DNA, deshalb sollen in diesem Abschnitt einige grundlegende Begriffe und Wirkungsweisen geklärt werden.

Enzyme sind Biokatalysatoren, d.h. sie beschleunigen chemische Reaktionen in lebenden Organismen. Sie werden als *Katalysatoren des Lebens* bezeichnet, weil sie alle Prozesse, die innerhalb eines Organismus ablaufen, so beschleunigen, dass Leben überhaupt erst möglich wird. In den allermeisten Fällen handelt es sich bei Enzymen um Proteine; daneben gibt es allerdings auch einige Fälle, in denen RNA katalytisch wirksam wird. Wie alle Katalysatoren gehen auch Enzyme unverändert aus den Reaktionen hervor, die sie beschleunigen. Die meisten Enzyme wirken nur auf einen bestimmten Stoff und katalysieren nur eine bestimmte Reaktion, sie sind also sowohl substrat- als auch wirkungsspezifisch. Einige Enzyme wirken auch nur auf eine bestimmte Molekülgruppe von ansonsten unterschiedlichen Substraten, dies bezeichnet man als Gruppenspezifität. Daneben gibt es auch Enzyme, die nur wenig spezifisch sind und auf eine ganze Reihe von Substraten wirken.

Viele Enzyme benötigen sogenannte Kofaktoren, um wirken zu können. Dies können zum Beispiel Ionen wie Eisen, Kupfer oder Natrium, Vitamine oder daraus entstandene Derivate sein.

Die Aktivität von Enzymen ist temperaturabhängig, das Temperaturoptimum liegt bei menschlichen Enzymen um 37°C, bei einigen Bakterienarten in siedenden Quellen kann es allerdings auch bei bis zu 110°C liegen. Auch der pH-Wert beeinflusst die Enzymaktivität, bei extremen pH-Werten kommt es zur Denaturierung, d.h. zur Zerstörung der Struktur des Proteins. Die Optima bezüglich des pH-Wertes können sehr unterschiedlich sein, so hat zum Beispiel das im Magen wirkende Verdauungsenzym Pepsin ein Optimum von 2-3, das im Darm wirkende Trypsin dagegen von 7-8 (Abbildung 2.8).

Bei einer chemischen Reaktion muss zunächst Aktivierungsenergie aufgewendet werden, um einen energetisch ungünstigeren Übergangszustand zu erreichen. Erst

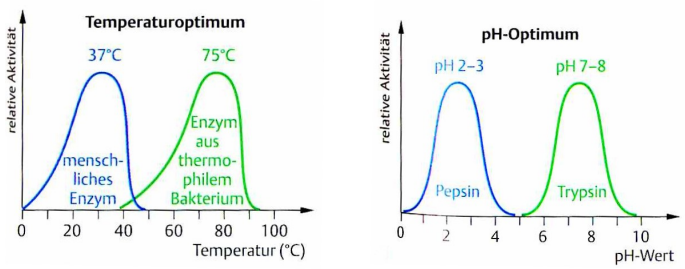


Abbildung 2.8: Die Aktivität von Enzymen ist abhängig von Temperatur und pH-Wert, sie kann zwischen verschiedenen Enzymen sehr unterschiedlich sein. [26]

danach kann die Reaktion ablaufen. Bei vielen Reaktionen ist diese benötigte Aktivierungsenergie so hoch, dass die Reaktion kaum stattfindet oder extrem langsam verläuft. Enzyme senken die Aktivierungsenergie und steigern so die Reaktionsgeschwindigkeit um einen Faktor bis zu 10^{22}, d.h. die Reaktionszeit kann von mehreren tausend Jahren auf 10^{-10} Sekunden verkürzt werden. Durch das Eingehen zahlreicher Wechselwirkungen mit dem Substrat stabilisiert das Enzym den Übergangszustand und senkt so die benötigte Energie (Abbildung 2.9).

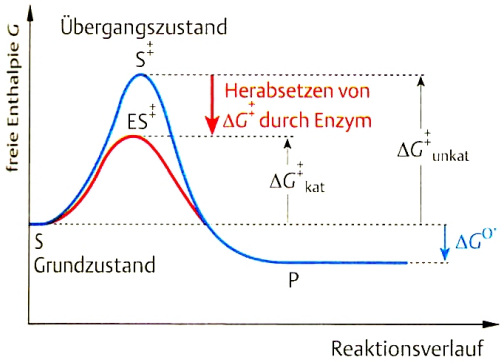


Abbildung 2.9: Enzyme wirken durch Absenken der Aktivierungsenergie $\Delta G^{\ddagger}$, sodass der Übergangszustand leichter erreicht wird und die Reaktion schneller ablaufen kann. Die Reaktion läuft nur ab, wenn insgesamt Energie frei wird, d.h. wenn die Reaktion *exergon* ist ($\Delta G^{0'} < 0$). Daran ändert auch der Einsatz des Enzyms nichts. [26]

Die Bindung und Umsetzung der Substrate erfolgt an einer bestimmten Stelle des Enzyms, dem *aktiven Zentrum*. Sowohl in diesem als auch im Substrat kommt es zu Konformationsänderungen, wodurch das Substrat optimal ausgerichtet wird. Dabei bildet das aktive Zentrum im Verlauf der Reaktion eine zum Übergangszustand kom-

plementäre Struktur und stabilisiert diesen dadurch. Meist liegt das aktive Zentrum in einer *hydrophoben* (wasserabweisenden) Tasche des Enzyms, wodurch sich von der Umgebung völlig unterscheidende Bedingungen entstehen. Wassermoleküle können durch ihre Polarität elektrostatische Anziehungskräfte zwischen geladenen Teilchen abschirmen, deshalb ist ihre Abwesenheit bei vielen Reaktionen notwendig.

Die Aktivität von Enzymen kann durch *Inhibitoren* gehemmt werden. Diese Hemmstoffe können auf verschiedene Arten wirken. Man unterscheidet zwischen irreversibler und reversibler Hemmung. Bei ersterer binden die Hemmstoffe, zum Beispiel Quecksilber, kovalent im aktiven Zentrum. Das Enzym ist dadurch zerstört und kann nicht weiter verwendet werden. Bei der reversiblen Hemmung erfolgt hingegen nur eine nichtkovalente Bindung, die mit physikalischen Methoden rückgängig gemacht werden kann. Man unterscheidet verschiedene Arten der reversiblen Hemmung, je nachdem, an welcher Stelle des Enzyms ein Inhibitor bindet. Erfolgt die Bindung ebenfalls im aktiven Zentrum, bezeichnet man dies als kompetitive Hemmung, da der Inhibitor und das Substrat um das aktive Zentrum konkurrieren. Eine Erhöhung der Substratkonzentration kann in diesem Fall den Umsatz steigern. Bindet der Inhibitor jedoch an einer anderen Stelle des Enzyms und verhindert dadurch den Substratumsatz (man unterscheidet hier noch die Fälle nicht- oder unkompetitive Hemmung), hat die Erhöhung der Substratkonzentration keinen Einfluss auf den Umsatz. [26]

2.1.3 Molekularbiologische Methoden

In diesem Abschnitt sollen die Methoden, die Molekularbiologen zur Manipulation von DNA im Labor zur Verfügung stehen, vorgestellt werden. Zusätzlich sollen die dabei möglicherweise entstehenden Probleme erläutert werden.

Gewinnen von DNA

DNA kann entweder aus Organismen isoliert oder künstlich synthetisiert werden. Zur künstlichen Synthese sind Geräte verfügbar, die Einzelstränge bis zu einer Länge von 100 Basen automatisch durch das schrittweise erfolgende Anfügen neuer Nukleotide herstellen können. Obwohl diese automatisierte Synthese als sehr zuverlässig gilt, können vereinzelt Stränge entstehen, in denen einzelne Nukleotide fehlen oder falsche Nukleotide eingebaut wurden.

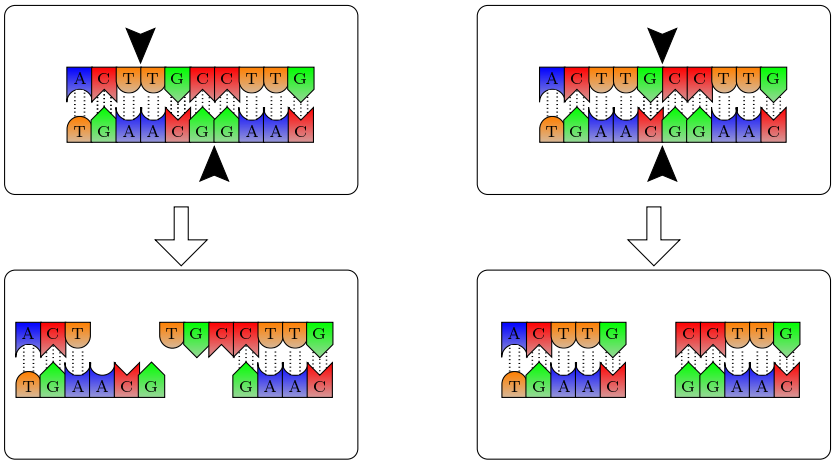
Bei der Isolation von DNA aus Organismen können durch Mutationen Unterschiede in der Sequenz der gewonnenen Stränge auftreten.

Schneiden von DNA

Zum Schneiden von DNA stehen spezielle Enzyme zur Verfügung, die *Nukleasen*. Diese spalten die Phosphodiesterbindungen zwischen Nukleotiden. Man unterteilt

Nukleasen in *Exo-* und *Endonukleasen*. Erstere bauen DNA-Stränge vom Ende her Nukleotid für Nukleotid ab, letztere schneiden die Stränge im Inneren. Nukleasen können sowohl einzel- als auch doppelstrangspezifisch sein.

Eine besondere Gruppe von doppelstrangschneidenden Enzymen und eines der wichtigsten molekularbiologischen Instrumente sind *Restriktionsendonukleasen*. Diese schneiden DNA-Stränge an sequenzspezifischen Stellen, zumeist an palindromischen Sequenzen. Einige dieser Enzyme hinterlassen *glatte Enden* (engl. *blunt ends*), andere *klebrige Enden* (engl. *sticky ends*), d.h. die entstehenden Doppelstränge haben einen einzelsträngigen 3'- oder 5'-Überhang (Abbildung 2.10).



(a) klebrige Enden **(b)** glatte Enden

Abbildung 2.10: Schneiden von DNA-Strängen mithilfe von Restriktionsendonukleasen, die klebrige (a) oder glatte Enden (b) erzeugen.

Zurzeit sind rund 4000 Restriktionsendonukleasen bekannt, von denen einige hundert kommerziell erhältlich sind. Die meisten dieser Enzyme haben eine Erkennungssequenz von vier bis acht Basen Länge, die oft palindromisch ist, und schneiden innerhalb dieser Erkennungssequenz. Daneben sind auch Enzyme bekannt, die außerhalb ihrer Erkennungssequenz, meist in einem bestimmten Abstand, schneiden.

Jedes Restriktionsenzym benötigt zum korrekten Arbeiten eine spezifische Temperatur und Bedingungen wie pH-Wert und Salzkonzentration. Deshalb muss bei der Arbeit darauf geachtet werden, diese Bedingungen durch Nutzung der richtigen Pufferlösung genau einzuhalten.

Daraus ergibt sich auch eine mögliche Fehlerquelle der Restriktionsspaltung. Sind die Bedingungen nicht optimal, so kann es zu unvollständiger oder unspezifischer, d.h. an anderen als der definierten Stelle stattfindender Spaltung kommen. Es ist auch auf eine ordentliche Durchmischung zu achten, um eine vollständige Spaltung

zu gewährleisten. Eine weitere Fehlerquelle bei Verwendung von DNA, die aus Organismen gewonnen wurde, ist, dass diese DNA oft methyliert ist. Dieses Anhängen von Methylgruppen ($-CH_3$) findet in einer lebenden Zelle auf eine spezifische Weise statt und dient der Unterscheidung von zelleigener und fremder (zum Beispiel durch Viren übertragener) DNA. Viele Restriktionsenzyme schneiden nicht an methylierten Stellen, sodass es zu einem unvollständigen Schneiden kommen kann. Abhilfe schafft die vorherige Entfernung der Methylgruppen oder die Verwendung von Restriktionsenzymen, die unabhängig von einer Methylierung schneiden. [25]

Neben den Restriktionsenzymen, welche doppelsträngige DNA schneiden, findet auch eine Endonuklease, die nur Einzelstränge spaltet, in der Molekularbiologie häufige Verwendung. Diese *S1-Nuklease* hinterlässt nach der Spaltung Mononukleotide oder kleine DNA-Fragmente aus wenigen Nukleotiden (*Oligonukleotide*). Sie wird beispielsweise genutzt, um klebrige Enden zu entfernen oder um Doppelstränge zu spalten, die nicht vollständig komplementär sind. [5]

Denaturieren und Hybridisieren

Unter normalen Bedingungen, d.h. Raumtemperatur und einem pH-Wert von 7, liegt DNA als Doppelstrang vor. Das Auftrennen in Einzelstränge bezeichnet man als *Denaturieren*, Synonyme sind *Schmelzen, Dissoziation* oder *Melting*. Durch Erhitzen auf über 90°C werden die Wasserstoffbrücken zwischen den komplementären Basen gelöst und so Doppelstränge in Einzelstränge getrennt. Die Denaturierung kann auch durch Wasserstoffbrücken brechende Agentien wie Harnstoff, Formamid oder Natriumhydroxid erreicht werden; diese Chemikalien wirken durch die Schaffung eines stark basischen Milieus.

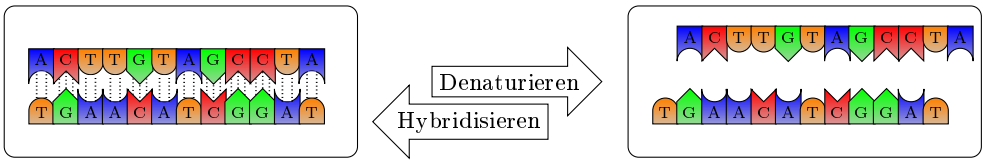


Abbildung 2.11: Beim Denaturieren werden DNA-Doppelstränge in Einzelstränge aufgetrennt, beim Hybridisieren entstehen aus komplementären Einzelsträngen Doppelstränge.

Die *Hybridisierung* ist das Gegenstück zur Denaturierung. Bei der Hybridisierung werden zwei Einzelstränge mit komplementärer Basenfolge zu einem Doppelstrang über Wasserstoffbrücken verbunden. Die Hybridisierung erfolgt durch langsames Absenken der Temperatur (mindestens über einen Zeitraum von vier Stunden, meist über Nacht). Falls das Absenken der Temperatur zu schnell erfolgt, werden eventuell

nicht alle möglichen Wasserstoffbrücken ausgebildet. Auch innerhalb eines einzelnen Stranges können zwischen komplementären Bereichen Wasserstoffbrücken ausgebildet werden, wodurch nichtlineare DNA-Strukturen entstehen können. Andere Bezeichnungen für die Hybridisierung sind *Renaturierung* oder *Annealing*. [17, 25]

Gelelektrophorese

Mithilfe der Gelelektrophorese kann ein Gemisch von DNA-Fragmenten entsprechend der Länge aufgetrennt werden. Dazu wird in einem Längenbereich von 0,5 bis 25 kb ein Agarose-Gel verwendet, bei kleineren Fragmentlängen ein Polyacrylamid-Gel. Agarose ist ein *Polysaccharid* (langkettiges Kohlenhydrat), das aus Algen gewonnen wird. Das DNA-Gemisch wird in eine Tasche des Gels gegeben, an dieses wird im Anschluss Spannung angelegt. Aufgrund ihrer negativen Ladung wandern die DNA-Stränge durch das Gel in Richtung des positven Pols. Durch die Siebstruktur des Gels wird es von großen DNA-Fragmenten langsamer durchlaufen als von kleineren. Bis zu einer bestimmten Fragmentgröße ist die Laufstrecke umgekehrt proportional zum Logarithmus der Fragmentlänge. Nach Zugabe eines Nukleinsäure-Farbstoffes, zum Beispiel Ethidiumbromid, können die DNA-Stränge als distinkte Banden unter UV-Licht auf dem Gel sichtbar gemacht werden. Um die Größen der Fragmente bestimmen zu können, lässt man in einer weiteren Spur des Gels einen Größenmarker mitlaufen, ein Gemisch von DNA-Fragmenten, von denen man bereits die Längen kennt. Nun kann man durch Vergleich der Lauflängen die Größe der unbekannten Fragmente ermitteln (Abbildung 2.12).

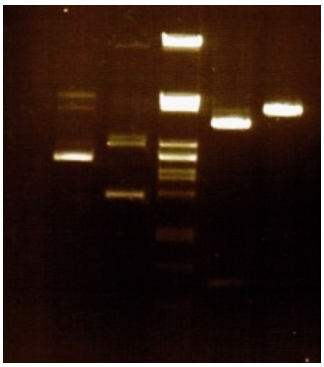


Abbildung 2.12: Beispiel einer Geleletrophorese: Auf der mittleren Spur ist ein Größenmarker aufgetragen, auf den anderen Spuren jeweils ein Gemisch von DNA-Fragmenten.

Bei größeren Fragmenten ist es nicht mehr möglich, sie mit dieser normalen Gelelektrophorese aufzutrennen, da sie sich nach einiger Zeit längs zum elektrischen Feld ausrichten und so das Gel ungehindert durchlaufen können. Die Zeit, die zur Ausrich-

tung gebraucht wird, ist bei Fragmenten, die größer als 20 kb sind, relativ ähnlich, weshalb sie nicht mehr als Banden zu unterscheiden sind. Durch eine regelmäßige Änderung der Richtung des elektrischen Feldes ist es allerdings möglich, auch diese Fragmente zu trennen. Man bezeichnet diese Methode als Pulsfeld-Gelelektrophorese, mit ihr ist eine Trennung bis 20000 kb Länge möglich, Verfeinerungen der Methode erlauben sogar die Trennung noch weit größerer Fragmente.

Das Herauslösen der Banden aus dem Gel bezeichnet man als *Elution*. Diese kann, auch in Abhängigkeit vom verwendeten Gel, auf verschiedene Weisen erfolgen: mit Hilfe von Glasmilch (einer Suspension kleiner Glaskügelchen, die DNA in der Gegenwart einer hohen Salzkonzentration bindet), durch Zentrifugieren oder als Elektroelution.

Eine mögliche Fehlerquelle bei der Gelelektrophorese ist der Verlust eines kleinen Teils der Stränge bei der Elution, der sich kaum vermeiden lässt. Außerdem können Stränge mit einer geringen Längendifferenz möglicherweise nicht unterschieden werden. [25]

Markieren und Separieren

Es ist möglich, während der Synthese DNA-Stränge zu markieren. Diese können dann aus einem Gemisch von markierten und unmarkierten DNA-Strängen separiert werden.

Eine Möglichkeit der Markierung ist die *Biotinylierung*. Dabei wird das Molekül Biotin (auch als Vitamin H beziehungsweise B_7 bekannt) kovalent mit dem 5'-Ende des DNA-Stranges verknüpft. Die Separation mit Biotin markierter Moleküle erfolgt mithilfe der starken Wechselwirkung zwischen Biotin und Avidin. Dazu wird Avidin auf einer Oberfläche kovalent gebunden, meist werden kleine magnetische Kugeln verwendet. Nach Hinzugabe der DNA-Stränge binden Avidin und Biotin. Die unmarkierten Stränge werden abgewaschen, es verbleiben nur die markierten Moleküle auf den Magnetkugeln, die dann isoliert werden. Durch Zugabe einer bestimmten Pufferlösung wird die Wechselwirkung zwischen Biotin und Avidin aufgehoben und die markierten Stränge können zurückgewonnen werden (Abbildung 2.13). Oft wird statt Avidin die aus Bakterien gewonnene Variante Streptavidin verwendet, die zwar etwas weniger stark mit Biotin bindet, dafür aber auch seltener Bindungen mit anderen Stoffen eingeht. [5]

Eine andere Möglichkeit der Markierung ist die kovalente Bindung des Moleküls *Acrydite* mit einem DNA-Molekül. Die Separierung von DNA-Strängen, die auf diese Weise markiert sind, erfolgt mithilfe der Gelelektrophorese. Das Acrydite verbindet sich mit dem Agarose-Gel, wodurch die markierten Stränge nach Anlegen von Span-

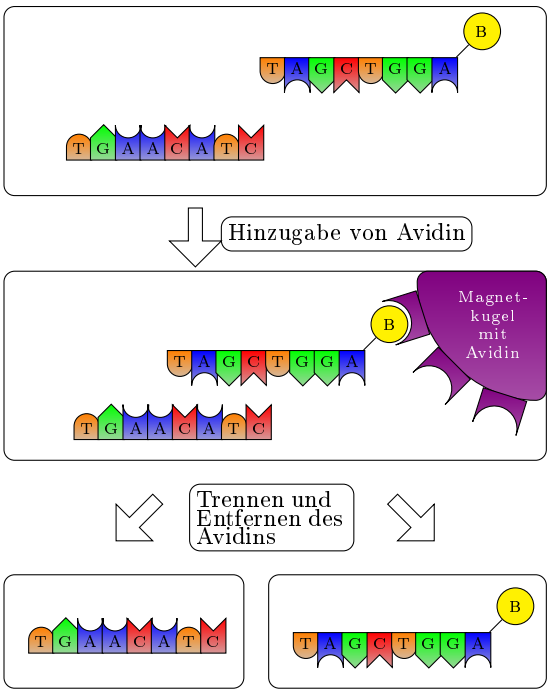


Abbildung 2.13: Separation mit Biotin (B) markierter DNA-Stränge.

nung im Gegensatz zu unmarkierten Strängen nicht durch das Gel wandern können. [2]

Ligieren

Ligasen sind Enzyme, die die Ausbildung von Phosphodiesterbindungen zwischen Nukleotiden katalysieren. Dies kann innerhalb eines Doppelstranges oder zwischen zwei DNA-Fragmenten erfolgen (siehe Abbildung 2.14). Dabei ist die Verknüpfung von Strängen mit komplementären klebrigen Enden sehr viel wahrscheinlicher als die von Strängen mit glatten Enden, da sich die Stränge in räumlicher Nähe zueinander befinden müssen, um ligiert zu werden. Dies kommt bei Strängen mit klebrigen Enden durch Ausbildung von Wasserstoffbrücken viel häufiger vor als bei Strängen mit glatten Enden. [25]

PCR

Die *Polymerase-Kettenreaktion* (PCR) ist eine Methode zum Kopieren von DNA-Strängen und dem Erzeugen von komplementären Strängen. Durchgeführt wird sie mithilfe des Enzyms Polymerase.

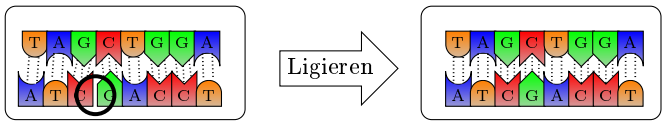


Abbildung 2.14: Ligasen knüpfen fehlende Phosphodiesterbindungen innerhalb von Doppelsträngen.

Zu Beginn werden alle Doppelstränge in Einzelstränge denaturiert. Nun muss am 3′-Ende jedes DNA-Strangs ein kurzes komplementäres DNA-Fragment,welches *Primer* genannt wird, gebunden werden. Dieses Fragment wird dann durch die Polymerase unter Nutzung von hinzugegebenen Nukleotiden dATP, dCTP, dGTP und dTTP verlängert. So entsteht der zum ursprünglichen Strang komplementäre Strang. Durch die Notwendigkeit der Primer können nur Stränge kopiert werden, von denen zumindest die Sequenz an den Enden bekannt ist.

In n Zyklen der PCR können 2^n Kopien erzeugt werden. Der Prozess der Vervielfältigung von DNA wird auch *Amplifikation* genannt.

Fehler bei der Amplifikation können beispielsweise dadurch auftreten, dass die Polymerase vorzeitig abbricht und dadurch ein verkürzter Strang entsteht. Dazu kann es bei langen zu kopierenden Strängen kommen, die PCR ist deshalb nur für Stränge bis 3 kb effizient. Verkürzte Stränge können auch dann auftreten, wenn die Primer auch zu einer Sequenz im Inneren der Stränge komplementär sind. Ein anderes Problem ist der Einbau von nichtkomplementären Nukleotiden. Bei der PCR wird die hitzebeständige Taq-Polymerase verwendet, die aus Bakterien stammt, welche in heißen Quellen leben. Sie besitzt im Gegensatz zu anderen Polymerasen keine Korrekturlesefunktion und hat daher eine höhere Fehlerrate, die bei etwa einem Fehler pro 9000 Nukleotiden liegt. Trotzdem wird diese Polymerase verwendet, da nur sie nicht bei den für die Trennung der Doppelstränge notwendigen Temperaturen denaturiert. [5]

Sequenzieren

Die Basenfolge von DNA-Strängen kann Nukleotid für Nukleotid gelesen werden, dafür stehen bereits automatisierte Verfahren zur Verfügung. Diese können Stränge bis zu einer Länge von 750 Nukleotiden sequenzieren. Stränge, die länger sind, müssen zuvor mit Restriktionsenzymen in sich überlappende Fragmente geschnitten werden, die dann einzeln sequenziert werden. Anschließend werden diese Fragmentsequenzen zusammengesetzt. [25]

Nachweis von DNA

Um zu testen, ob sich in einem Reagenzglas DNA-Stränge befinden, kann der Inhalt des Glases zum Beispiel auf ein Agarose-Gel gegeben werden und mit einem Nu-

kleinsäure-Farbstoff wie Ethidiumbromid gefärbt werden. Durch Vergleich der Stärke der entstehenden Banden mit einem ebenfalls aufgetragenen Größenmarker lässt sich auch die Konzentration der aufgetragenen DNA-Lösung bestimmen. Sehr kleine Mengen von DNA können durch Silberfärbung sichtbar gemacht werden, dies ist allerdings sehr viel aufwendiger als die Färbung mit Ethidiumbromid. [25]

Konzentrationsbestimmung

Die Konzentration einer DNA-Lösung lässt sich durch Messung der *optischen Dichte* bei UV-Licht einer Wellenlänge von 260 nm bestimmen. Nukleinsäuren absorbieren Licht dieser Wellenlänge; je größer die Konzentration einer DNA-Lösung, desto größer ist die optische Dichte. Mithilfe eines Umrechnungsfaktors, welcher von der Art der Nukleinsäure (einzel- oder doppelsträngig) sowie der Länge der Stränge abhängt, lässt sich die Konzentration in Mikrogramm pro Milliliter Lösung bestimmen.

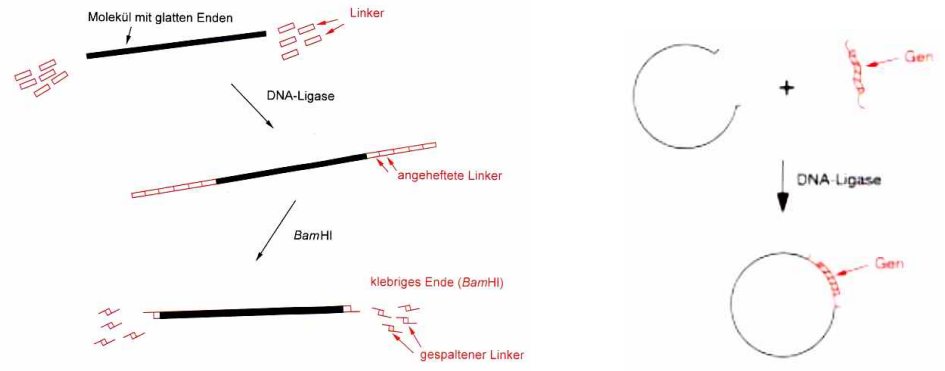
Die Konzentrationsmessung ist relativ ungenau, sie hängt auch vom pH-Wert, dem Salzgehalt sowie von Proteinen, die sich eventuell ebenfalls in der Probe befinden, ab.

Eine Alternative zur Bestimmung durch Messung der optischen Dichte ist das Auftragen auf ein Agarose-Gel. Durch die Dicke der entstehenden Bande lässt sich eine Aussage über die Konzentration treffen, allerdings ist auch diese Methode ungenau, zudem tritt hier zusätzlich das Problem der Rückgewinnung der Stränge auf. [25]

Integrieren von DNA in Plasmide

Häufig muss ein doppelsträngiges DNA-Fragment mit glatten Enden in ein Plasmid integriert werden, zum Beispiel, um es in einen Organismus zu übertragen.

Dazu wird das Plasmid zunächst mit einem Restriktionsenzym geschnitten. Dieses Enzym muss klebrige Enden erzeugen, seine Erkennungssequenz darf nur einmal in diesem Plasmid und nicht in dem einzufügenden DNA-Fragment auftreten. Zu dem DNA-Fragment wird eine große Anzahl von sogenannten *Linkern* (oder *Adaptern*) gegeben. Dies sind Oligonukleotide, die ebenfalls die Erkennungssequenz des Restriktionsenzyms enthalten. Mithilfe einer Ligase werden diese Linker an den Enden des DNA-Fragments gebunden. Nach dem Schneiden mit dem Restriktionsenzym besitzt das DNA-Fragment nun klebrige Enden, die komplementär zu denen des aufgeschnittenen Plasmids sind. Dadurch können sich zwischen den Enden des Fragments und denen des Plasmids Wasserstoffbrücken ausbilden; eine Ligase knüpft anschließend Phosphodiesterbindungen (siehe Abbildung 2.15). Durch eine Gelelektrophorese können die Plasmide, in die das DNA-Fragment integriert wurde, von den übrigen getrennt werden. [5]



(a) **(b)**

Abbildung 2.15: Anfügen eines Linkers an ein DNA-Fragment zur Herstellung klebriger Enden (a), wodurch das Fragment in ein Plasmid eingebaut werden kann (b). [5]

Reinigen von DNA

Für die meisten der hier geschilderten molekularbiologischen Verfahren ist die Verwendung spezieller Enzyme und Puffer, die für die optimalen Umgebungsbedingungen sorgen, notwendig. Nach Durchführung der Reaktion müssen diese entfernt werden. Eine häufige Methode ist die Phenol-Chloroform-Extraktion. Dabei wird die DNA-Lösung nacheinander mit Phenol und Chloroform ausgeschüttelt, wodurch Proteine und Puffersalze in einer separaten Phase denaturieren. Die DNA bleibt hingegen in der wässrigen Phase gelöst und kann mit einer Pipette entfernt werden. Anschließend werden mit einer Alkoholfällung die Reste von Phenol und Chloroform entfernt. Sollen nur verwendete Enzyme entfernt werden, so stellt die Filterung mit einer speziellen proteinbindenden Membran eine schnelle und ungefährliche Alternative dar (Phenol ist gesundheitsgefährdend). Bei allen Methoden kann es zum Verlust kleiner Mengen an DNA kommen. [25]

Bei allen hier beschriebenen Verfahren ist immer darauf zu achten, dass die DNA-Lösungen nicht mit Fremdstoffen verunreinigt werden, da dadurch die verwendeten Enzyme gehemmt oder unbrauchbar gemacht werden können. Auch kann es zum Beispiel zur Übertragung von Nukleasen von der menschlichen Haut kommen, die dann DNA ungewollt abbauen und so die Probe zerstören.

2.2 Entwicklung des DNA-Computings

Als Initiator des DNA-Computings gilt der Physiker und spätere Nobelpreisträger Richard Feynman, der in seiner 1959 am *California Institute of Technology* gehaltenen Rede *There's Plenty of Room at the Bottom* [11] die Möglichkeiten darstellte, die sich in der *submikroskopischen Welt* der Atome und Moleküle eröffnen. Als Beispiel nannte er die Speicherung und Verarbeitung von Informationen in biologischen Systemen mithilfe von DNA. Auch die sich durch die Effekte der Quantenmechanik ergebenden neuen Möglichkeiten auf der atomaren Ebene erkannte er und gab damit sowohl den Anstoß zum DNA-Computing als auch zu Quantencomputern. Durch diese Rede gilt Feynman heute als *Vater der Nanotechnologie*.

1994 gelang es Leonard Adleman zum ersten Mal, eine kleine Instanz eines $\mathcal{NP}$-vollständigen Problems mit laborpraktischen Methoden zu lösen [1]. Dies gilt als eigentlicher Beginn des DNA-Computings. Adleman löste eine Instanz des Hamiltonpfadproblems. Gegeben ist dabei ein Graph mit Knoten und gerichteten Kanten zwischen einigen dieser Knoten. Zwei der Knoten seien als Start- beziehungsweise Endknoten ausgezeichnet. Die Frage ist, ob es einen Weg gibt, der vom Start- zum Endknoten führt und dabei jeden Knoten genau einmal besucht. Die von Adleman betrachtete Instanz bestand aus sieben Knoten (Abbildung 2.16) und besitzt einen Hamiltonpfad. Zur Lösung waren sieben Tage Laborarbeit nötig.

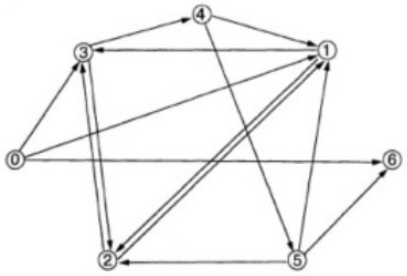


Abbildung 2.16: Gerichteter Graph mit einem Hamiltonpfad von 0 nach 6 ($0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6$). [1]

Nachdem es damit zum ersten Mal gelungen war, ein NP-vollständiges Problem zu lösen, gab es eine ganze Reihe weiterer laborpraktischer Experimente. Die bis heute größte Instanz eines NP-vollständigen Problems wurde 2002 von Braich et al. [3] praktisch gelöst, eine Instanz des Erfüllbarkeitsproblems SAT mit 20 Variablen.

Neben diesen Arbeiten, die sich mit der praktischen Lösung spezieller Probleme beschäftigten, verfolgten andere Autoren weitgehend getrennt davon ein anderes Ziel: Das Aufstellen von Modellen des DNA-Computings, die sowohl vom Medium DNA als auch von den molekularbiologischen Operationen mehr oder weniger stark ab-

strahieren, und die Untersuchung dieser Modelle hinsichtlich ihrer Berechnungsstärke [17].

2.3 DNA-Computing-Modelle

In diesem Abschnitt sollen einige der formalen DNA-Computing-Modelle vorgestellt werden, die in den letzten zwanzig Jahren entwickelt wurden. Diese nutzen Operationen, die mehr oder weniger stark von den molekularbiologisch zur Verfügung stehenden Methoden (siehe Abschnitt 2.1.3) abstrahieren. Ziel dieser theoretischen Modelle ist es, herauszufinden, was mit DNA-Computing berechnet werden kann und welcher Aufwand dafür notwendig ist. Es sollen dabei möglichst Modelle entwickelt werden, die ein im Sinne der Churchschen These *universelles* Rechnermodell beschreiben, also alles berechnen können, was mit Turingmaschinen berechenbar ist (siehe Abschnitt 1.1).

2.3.1 Filtering-Modelle

Die einfachsten Modelle sind die Filtering-Modelle, zu denen das in Abschnitt 2.2 beschriebene Adleman-Experiment gehört. Ausgehend von einem zu Beginn bereitstehenden DNA-Pool, der den gesamten Suchraum kodiert, werden Schritt für Schritt alle DNA-Stränge entfernt, die keine Lösung des Problems darstellen. Am Schluss erfolgt ein Test, ob DNA vorhanden ist oder nicht.

Die Operationen, die in diesen Modellen zur Verfügung stehen, erlauben keine Veränderung der Stränge, sondern nur das Herausfiltern von Strängen mit einer bestimmten Teilsequenz, das Vereinigen zweier Reagenzglasinhalte, das Kopieren eines Reagenzglasinhaltes sowie den Test auf Leerheit.

Filtering-Modelle sind platzbeschränkt universell, dies wird durch Transformation von platzbeschränkten WHILE-Programmen nachgewiesen. [17]

2.3.2 Stickersysteme

Stickersysteme nutzen die komplementäre Basenpaarung (*Watson-Crick-Komplementarität*, benannt nach den Entdeckern der Doppelhelixstruktur), um den Aufbau von Wörtern zu simulieren. Sie wurden zum ersten Mal im Jahr 1998 durch Kari, Păun et al. [19] beschrieben.

Betrachtet wird ein Alphabet V und eine symmetrische Relation $\rho \subseteq V \times V$, die Komplementaritätsrelation. Statt $(x_1, x_2) \in V^* \times V^*$ werden Paare von Strängen in der Form $\binom{x_1}{x_2}$ geschrieben.

Die Menge von komplementären Buchstaben sei $\left[\begin{smallmatrix} V \\ V \end{smallmatrix}\right]_\rho = \left\{ \left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right] \mid a, b \in V,\ (a, b) \in \rho \right\}$, die Menge aller komplementären Paare von Strängen wird als *Watson-Crick-Domain* $WK_\rho(V) = \left[\begin{smallmatrix} V \\ V \end{smallmatrix}\right]_\rho^*$ bezeichnet, seine Elemente als *Moleküle*.

Es sollen aber nicht nur vollständige Doppelstränge betrachtet werden, sondern auch solche mit ein- oder beidseitig überstehenden Enden oder Einzelstränge:

$$
\begin{aligned}
W_\rho(V) &= L_\rho(V) \cup R_\rho(V) \cup LR_\rho(V) \\
&\quad \text{mit} \\
L_\rho(V) &= \left(\binom{\lambda}{V^*} \cup \binom{V^*}{\lambda} \right) \left[{V \atop V} \right]_\rho^*, \\
R_\rho(V) &= \left[{V \atop V} \right]_\rho^* \left(\binom{\lambda}{V^*} \cup \binom{V^*}{\lambda} \right), \\
LR_\rho(V) &= \left(\binom{\lambda}{V^*} \cup \binom{V^*}{\lambda} \right) \left[{V \atop V} \right]_\rho^+ \left(\binom{\lambda}{V^*} \cup \binom{V^*}{\lambda} \right)
\end{aligned}
$$

Die Überhänge an den Seiten werden als *sticky ends* oder *klebrige Enden* bezeichnet, die Elemente von $W_\rho(V)$ heißen *Dominos*.

$L_\rho(V)$ ist die Menge der Dominos mit linkem Überhang, $R_\rho(V)$ die der Dominos mit rechtem Überhang und $LR_\rho(V)$ die Menge der Dominos mit beidseitigem Überhang und einem doppelsträngigen Teil dazwischen. Da der Überhang auch aus dem leeren Wort λ bestehen kann, ist die Watson-Crick-Domain $WK_\rho(V)$ in jeder der drei Mengen enthalten.

Es wird nun für ein Domino $x \in LR_\rho$, das also zumindest zum Teil doppelsträngig ist, und ein beliebiges Domino $y \in W_\rho$ eine (nichtkommutative) Operation $\mu(x, y)$ definiert, die beschreibt, in welchen Fällen x um y erweitert werden kann. Diese acht Fälle sind in Abbildung 2.17 dargestellt (für eine genaue Beschreibung der Operation μ siehe [29]). Die Operation μ ist also die Verallgemeinerung der molekularbiologischen Ligation.

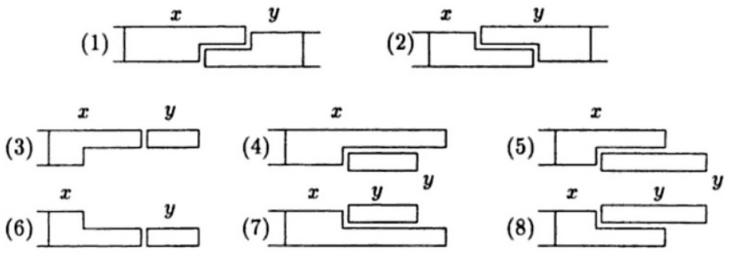


Abbildung 2.17: Die Funktion μ gibt an, in welchen Fällen ein Strang x um einen Strang y erweitert werden kann. [29]

Ein Stickersystem wird definiert als ein 4-Tupel $\gamma = (V, \rho, A, D)$, wobei
- V ein Alphabet,
- $\rho \subseteq V \times V$ die symmetrische Komplementaritätsrelation auf V,
- $A \subset LR_\rho(V)$ eine endliche Menge von Axiomen mit einem doppelsträngigen Anteil und
- $D \subset W_\rho(V) \times W_\rho(V)$ eine endliche Menge von Paaren von Dominos

ist.

Ein Domino y kann aus einem Domino x ($x, y \in LR_\rho(V)$) abgeleitet werden, $x \Rightarrow_\gamma y$, genau dann, wenn $y = \mu(u, \mu(x, v))$ für $(u, v) \in D$. D.h. y entsteht aus x, indem ein Paar von Dominos aus D links und rechts mit x ligiert wird.

Es sei $\Rightarrow_\gamma^*$ die reflexive und transitive Hülle der Ableitung $\Rightarrow_\gamma$.

Die von einem Stickersystem $\gamma = (V, \rho, A, D)$ erzeugte Sprache ist dann:

$$L(\gamma) = \left\{ w \in V^* \mid \left[\begin{smallmatrix} w \\ w' \end{smallmatrix}\right] \in WK_\rho(V) \text{ mit } x \Rightarrow_\gamma^* \left[\begin{smallmatrix} w \\ w' \end{smallmatrix}\right] \text{ für ein } x \in A \text{ und ein } w' \in V^* \right\}.$$

$L(\gamma)$ enthält somit alle Wörter w, die zusammen mit ihrem komplementären Strang w' als vollständiger Doppelstrang aus einem der Axiome $x \in A$ abgeleitet werden können.

Dies war das Grundmodell der Stickersysteme. Es gibt darauf aufbauend Modelle, die die Art der Ableitung einschränken, zum Beispiel durch Beschränkung der Länge der Überhänge, nur einseitig erfolgende Verlängerung oder Verlängerung ausschließlich mit Einzelsträngen. Zu jedem dieser Modelle kann die Klasse der erzeugbaren Sprachen betrachtet und eine Hierarchie dieser Sprachklassen aufgestellt werden. Allen diesen Sprachklassen gemeinsam ist, dass sie nur kontextsensitive Sprachen (Typ 1) enthalten, da bei Stickersystemen die Stränge nur verlängert werden können, ein Verkürzen jedoch nicht möglich ist. Die Stickersysteme stellen daher kein universelles Berechnungsmodell dar. [29]

2.3.3 Insertion-Deletion-Systeme

Die Idee der Insertion-Deletion-Systeme ist das gezielte Einfügen (*Insertion*) oder Entfernen (*Deletion*) bestimmter Sequenzen innerhalb von DNA-Strängen, abhängig von der Umgebung der entsprechenden Abschnitte (*Kontextsensitivität*). Sie sind wie Chomsky-Grammatiken ein Mechanismus zur Erzeugung von Sprachen. Insertion-Systeme wurden bereits 1981 durch Galiukschov im Rahmen der Theorie der Formalen Sprachen beschrieben, Insertion-Deletion-Systeme 1996 durch Kari und Thierrin [20]. Die Anwendung auf das DNA-Computing erfolgte durch Smith [33] ebenfalls im Jahr 1996. [29]

Ein Insertion-Deletion-System ist ein 4-Tupel $\gamma = (V, \Sigma, R, A)$, wobei

- V ein beliebiges Alphabet,
- $\Sigma \subseteq V$ das Alphabet der Terminalsymbole,
- R eine endliche Menge von Insertions- und Deletionsregeln und
- $A \subset V^*$ eine endliche Sprache von Axiomen ist.

Die Regeln in R sind Tripel der Form $(u, \alpha/\beta, v)$ mit $u, v \in V^*$ und $(\alpha, \beta) \in (V^+ \times \{\lambda\}) \cup (\{\lambda\} \times V^+)$.

Eine Regel der Form $(u, \lambda/\beta, v)$ ist eine Insertionsregel (siehe Abbildung 2.18); das Wort β wird in die Umgebung uv eingefügt ($uv \rightarrow u\beta v$).

Eine Regel der Form $(u, \alpha/\lambda, v)$ ist hingegen eine Deletionsregel; das Wort α wird aus der Umgebung uv entfernt ($u\alpha v \rightarrow uv$).

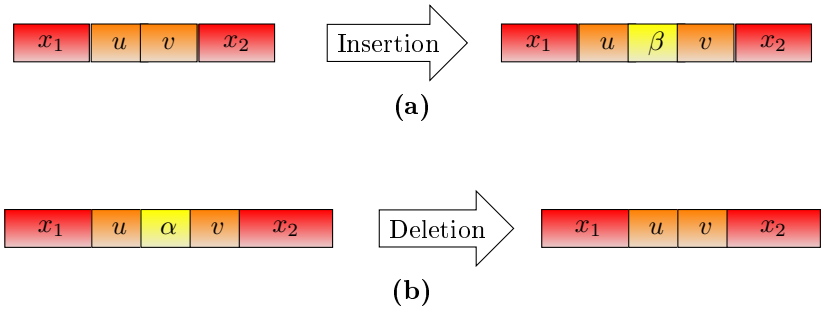


Abbildung 2.18: Anwendung der Insertionsregel $(u, \lambda/\beta, v)$ (a) oder der Deletionsregel $(u, \alpha/\lambda, v)$ (b).

Ein Ableitungsschritt von x nach y ($x, y \in V^*$) ist eine Relation $\vdash_\gamma$, die definiert ist durch:

$$x \vdash_\gamma y = \{(x, y) \mid x = x_1uvx_2 \wedge y = x_1u\beta vx_2 \wedge (u, \lambda/\beta, v) \in R \text{ oder } x = x_1u\alpha vx_2 \wedge y = x_1uvx_2 \wedge (u, \alpha/\lambda, v) \in R\}$$

Die reflexive und transitive Hülle dieser Relation sei $\vdash_\gamma^*$.

Die durch ein Insertion-Deletion-System γ erzeugte Sprache ist dann definiert durch

$$L(\gamma) = \{w \in \Sigma^* \mid \text{ es gibt ein } x \in A \text{ mit } (x \vdash_\gamma^* w)\}.$$

Der Universalitätsnachweis des Modells erfolgt durch Überführung einer Chomsky-Grammatik vom Typ 0 in ein Insertion-Deletion-System. Es ist hierbei bereits ein Insertion-Deletion-System universell, bei dem maximal Wörter der Länge 3 eingefügt oder entfernt werden und die Kontextregionen u und v im Fall der Insertion höchstens die Länge 2 haben sowie die Deletion kontextunabhängig stattfindet. [17]

2.3.4 Splicing-Systeme

Splicing-Systeme basieren auf einer Abstraktion der molekularbiologischen Rekombination von DNA-Strängen, bei der durch Schneiden von DNA-Strängen mit Restriktionsenzymen und anschließender Ligation der dabei entstandenden Fragmente neue DNA-Stränge entstehen können. Beschrieben wurden Splicing-Systeme zum ersten Mal durch Tom Head 1987 [14], sie sind damit das einzige DNA-Computing-Modell, dass bereits einige Jahre vor dem Adleman-Experiment veröffentlicht wurde. Sie werden nach ihrem Erfinder als *H-Systeme* bezeichnet.

Für ein beliebiges Alphabet V wird eine *Splicingregel* r als Wort über $V \cup \{\#, \$\}$ (mit $\#, \$ \notin V$) durch $r = u_1\#u_2\$u_3\#u_4$ für $u_1, \ldots, u_4 \in V^*$ definiert.

Ein Paar von Wörtern (x, y) über V wird in ein Wortpaar (z, w) mittels der Splicingregel $r = u_1\#u_2\$u_3\#u_4$ überführt , $(x, y) \vdash_r (z, w)$, genau dann, wenn gilt:

$$\begin{aligned} x &= x_1u_1u_2x_2 \\ y &= y_1u_3u_4y_2 \\ z &= x_1u_1u_4y_2 \\ w &= y_1u_3u_2x_2 \ . \end{aligned}$$

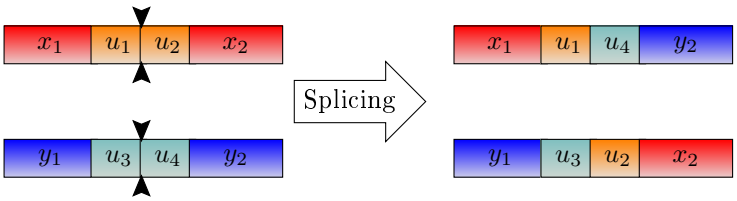


Abbildung 2.19: Durch Anwendung der Splicingregel $u_1\#u_2\$u_3\#u_4$ entstehen aus den Strängen $x_1u_1u_2x_2$ und $y_1u_3u_4y_2$ die neuen Stränge $x_1u_1u_4y_2$ und $y_1u_3u_2x_2$.

Für eine Sprache $L \subseteq V^*$ und eine Menge von Splicingregeln R wird $\sigma_R(L)$ wie folgt definiert:

$$\begin{aligned} \sigma_R(L) = \{w \in V^* \mid \text{Es gibt } z \in V^*, r \in R \text{ und } x, y \in L, \text{ sodass} \\ (x, y) \vdash_r (z, w) \text{ oder } (x, y) \vdash_r (w, z)\} \end{aligned}$$

Es sei

$$\begin{aligned} \sigma_R^0(L) &= L \\ \sigma_R^{i+1}(L) &= \sigma_R^i(L) \cup \sigma_R(\sigma_R^i(L)) \\ \sigma_R^*(L) &= \bigcup_{i \in \mathbb{N}} \sigma_R^i(L). \end{aligned}$$

Es ist $\sigma_R^*(L)$ also die Menge aller Wörter, die aus L durch beliebig oft durchgeführtes Splicing entstehen können.

Ein EH-System (erweitertes H-System) ist dann definiert als $\mu = (V, \Sigma, R, A)$, wobei

- V ein beliebiges Alphabet,
- Σ ein Terminalalphabet mit $\Sigma \cap V = \emptyset$,
- $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ eine Menge von Splicingregeln mit $\#, \$ \notin V$ und
- $A \subseteq V^*$ eine Menge von Axiomen ist.

Dabei bezieht sich „erweitert“ darauf, dass beim ursprünglich von Head aufgestellten H-System kein separates Terminalalphabet betrachtet wurde.

Die von μ erzeugte Sprache ist

$$L(\mu) = \{w \in \Sigma^* \mid w \in \sigma_R^*(A)\}$$

Es kann nun abhängig von der gewählten Menge an Axiomen A und der Menge an Splicingregeln R untersucht werden, welche Sprachen von einem Splicingsystem erzeugt werden können. Sind beide Mengen endlich, so können nur reguläre Sprachen erzeugt werden. Um alle Typ-0-Sprachen erzeugen zu können, muss mindestens eine der beiden Mengen unendlich viele Elemente enthalten: Ist die Menge der Axiome endlich, so muss die Menge der Splicingregeln unendlich groß sein; ist die Menge der Splicingregeln endlich, so darf die Menge der Axiome nicht kontextfrei sein. [29]

Das Modell der EH-Systeme ist also nur dann universell, wenn eine der Komponenten unendlich groß ist. Damit ein Modell praktisch umgesetzt werden kann, ist es natürlich notwendig, dass nur endlich viele Komponenten benötigt werden. Deshalb wurde das Modell vielfach erweitert, zum Beispiel durch Verwendung von nichtlinearen DNA-Strukturen oder zum Modell des Verteilten Spicings, bei dem mehrere Reagenzgläser verwendet werden, die jeweils durch ein endliches EH-System beschrieben werden und untereinander Zwischenprodukte auf eine definierte Weise austauschen. [17]

2.3.5 Weitere Modelle des *Biological Computings*

Neben diesen (und einigen anderen) Modellen des DNA-Computings gibt es weitere Modelle, die biologische Prozesse abstrahieren.

In-vivo-Modelle versuchen, Berechnungen mithilfe von lebenden Organismen durchzuführen. Zum einen gibt es Versuche, die Regulation der Aktivität von Genen zu nutzen, d.h. bestimmte Gene durch andere gezielt an- oder abzuschalten (zum Beispiel [10]). Ein anderes Untersuchungsobjekt ist die Genomumordnung (*Gene Assembly*) bei Ciliaten (einzelligen Organismen).

Ein Modell, das in den letzten Jahren populär wurde, ist das *Membrane-Computing*, welches vom Aufbau einer Zelle aus zahlreichen durch Membranen abgeschlossenen

Kompartimenten abstrahiert. Das Berechnungsmodell, *P-System* genannt nach seinem Erfinder Gheorghe Păun, nutzt eine Hierarchie von Membranen (in Abbildung 2.20 ist ein Beispiel dargestellt); diese können während der Berechnung zerstört werden oder neu entstehen. In jeder der von diesen Membranen begrenzten Regionen befinden sich Symbole in einer bestimmten Anzahl; diese sind die mathematische Entsprechnung von chemischen Molekülen. Chemische Reaktionen dieser Moleküle werden im Modell als Produktionsregeln beschrieben. Diese Regeln können Veränderungen der Symbole innerhalb der Regionen beschreiben oder den Austausch von Symbolen zwischen verschiedenen Regionen. Sie werden in allen Regionen gleichzeitig angewendet, die Berechnung endet, wenn in keiner Region eine der Regeln angewendet werden kann.

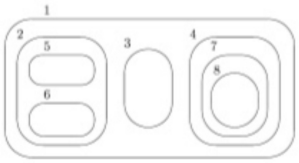


Abbildung 2.20: Beispiel einer Hierarchie von Membranen. [34]

In der Literatur werden zahlreiche Variationen dieses Grundmodells beschrieben, diese sind alle theoretischer Natur, es gibt noch keine praktischen Implementationen. [27]

3 DNA-Algorithmen für ein PSPACE-vollständiges Problem

In diesem Kapitel sollen Möglichkeiten gefunden werden, das PSPACE-vollständige Problem der Quantifizierten Booleschen Formeln (QBF) mithilfe des DNA-Computings zu lösen.

In Abschnitt 3.1 werden zunächst die benötigten DNA-Operationen vorgestellt. Danach werden zwei mögliche Lösungsalgorithmen beschrieben: der aus der Literatur bekannte Brute-Force-Algorithmus von Alexander Wolpert und Evgeny Dantsin [38] (Abschnitt 3.2) und ein Algorithmus, der auf Breitensuche basiert (Abschnitt 3.3). Es wird jeweils die Idee der Algorithmen erklärt und an Beispielen veranschaulicht, anschließend wird eine Möglichkeit der praktischen Implementation mithilfe der DNA-Operationen aus Abschnitt 3.1 angegeben und die dafür notwendige Zeit sowie der Bedarf an DNA-Strängen diskutiert.

Prinzipiell gibt es zwei Vorgehensweisen, um zu entscheiden, ob eine Quantifizierte Boolesche Formel wahr ist: durch Durchsuchen des Raumes aller Belegungen oder durch Vereinfachung und Auflösen (*Resolution*) der Formel.

Die im Folgenden vorgestellten Algorithmen beruhen auf dem Durchsuchen des Raumes der Belegungen. Beiden Algorithmen gemein ist die Aufteilung in zwei Phasen: Zunächst wird das Problem ohne die Quantifizierungen betrachtet und es werden alle erfüllenden Belegungen, kodiert in DNA-Strängen, gefunden. In der zweiten Phase werden die Quantifizierungen schrittweise abgearbeitet.

Die erste Phase entspricht dem bekannten SAT-Problem (Abschnitt 1.5). Für dieses gibt es in der Literatur bereits eine große Anzahl von theoretischen sowie einige praktische Arbeiten zur Lösung mithilfe des DNA-Computings.

Die bis heute größte praktische Lösung einer Instanz des 3SAT-Problems mit Hilfe des DNA-Computings stammt von Braich et al. [3] aus dem Jahr 2002. Sie betrachteten eine Boolesche Formel in 3KNF mit 20 Variablen und 24 Klauseln. Ihr Ansatz war ein Brute-Force-Verfahren: Alle 2^{20} möglichen Belegungen der 20 Variablen wurden zu Beginn durch DNA-Stränge kodiert, anschließend wurden Klausel für Klausel alle Stränge herausgefiltert, die diese Klausel nicht erfüllten. Am Ende bleiben nur die Stränge übrig, die für eine Belegung kodieren, welche alle Klauseln und somit die Formel erfüllt. Bei diesem Algorithmus werden also alle Blätter des Suchbaumes durchsucht (siehe Abbildung 3.1). Dieses Verfahren dient als Grundlage für den in

Abschnitt 3.2 vorgestellten Algorithmus.

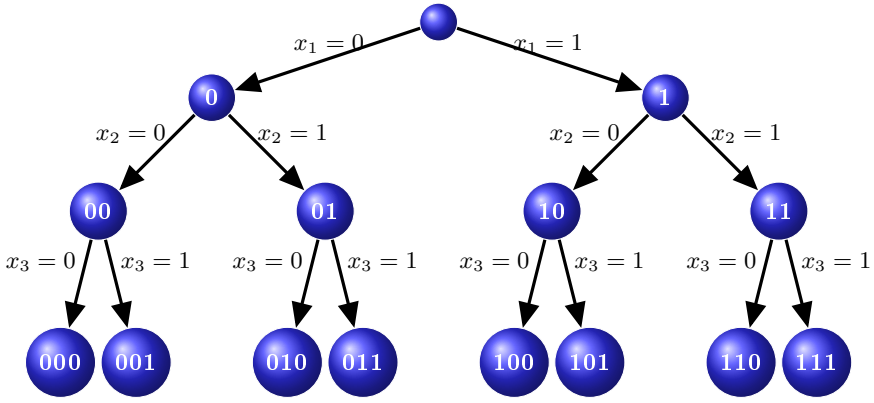


Abbildung 3.1: Suchbaum des SAT-Problems für drei Variablen.

Ein anderer Ansatz wurde von Yoshida und Suyama [41] gewählt. Anstatt alle Stränge zu Beginn zu erzeugen, wird eine Ebene des Suchbaumes nach der anderen betrachtet, die DNA-Stränge werden dabei schrittweise um die hinzukommende Variable verlängert. In jedem Schritt werden dann nur die Klauseln betrachtet, die bereits in den Strängen kodierte Variablen enthalten. Die Stränge, die für Belegungen kodieren, die diese Klauseln nicht erfüllen, werden jeweils entfernt. Dadurch werden alle nachfolgenden Knoten im Suchbaum nicht mehr betrachtet werden. Yoshida und Suyama lösten eine Instanz mit vier Variablen und 16 Klauseln mit diesem Verfahren und fanden durch Simulationen Abschätzungen für die Anzahl der benötigten DNA-Stränge (siehe Abschnitt 3.3.5). Auf ihrem Verfahren beruht der in Abschnitt 3.3 vorgestellte Algorithmus.

Neben diesen gibt es eine ganze Reihe weiterer Vorschläge zur Lösung des SAT-Problems mithilfe von DNA-Computing, zum Beispiel die Implementation randomisierter Verfahren [23, 6], die Nutzung von Micro-Arrays [21], die Arbeit mit DNA, die auf Oberflächen fest gebunden ist [22], die Umsetzung mit Genexpression und -regulation in lebenden Organismen [10] und viele weitere Vorschläge [36, 40].

3.1 Benötigte DNA-Operationen

Basierend auf den im Abschnitt 2.1.3 beschriebenen laborpraktischen Methoden werden Operationen definiert, mit denen die Algorithmen beschrieben werden. Diese Operationen abstrahieren von den zahlreichen Fehlern, die bei der praktischen Durchführung im Labor auftreten können.

Alle DNA-Einzelstränge werden in $5' \rightarrow 3'$-Richtung angegeben. Zu einem Einzelstrang X sei $\overline{X}$ der Strang mit der komplementären Basenfolge. Beispielsweise ist der

komplementäre Strang zu $ACCGT$ dann $ACGGT$; das $5'$-Ende des einen Stranges liegt aufgrund der antiparallelen Strangpaarung dem $3'$-Ende des anderen Stranges gegenüber. Ein markierter Strang wird durch ${}^*X$ dargestellt.

Folgende Operationen werden in den DNA-Algorithmen verwendet:

- $S \leftarrow \text{Synthese}(X_1, \ldots, X_n)$
  Für jedes Wort $X_i (i = 1, \ldots, n)$ über dem Alphabet $\{A, C, G, T\}$ wird der Einzelstrang $5'$-X_i-$3'$ in mehreren Kopien synthetisiert und zu S gegeben. Für $S \leftarrow \text{Synthese}({}^*X_1, \ldots, {}^*X_n)$ werden die Stränge zusätzlich dazu markiert.

- $(S_1, \ldots, S_n) \leftarrow S$
  Kopien des gesamten Inhalts von S werden in S_1 bis S_n gegeben.

- $(S_1, S_2) \leftarrow \text{Filtern}(S)$
  S sei ein Gemisch von markierten und unmarkierten Strängen. Alle markierten DNA-Stränge aus S werden in S_1 gegeben, alle unmarkierten in S_2.

- Hybridisieren(S)
  Ausbildung von Doppelbindungen zwischen komplementären Strängen in S.

- Denaturieren(S)
  Aufbrechen aller Doppelbindungen zwischen Strängen in S.

- Ligieren(S)
  Fehlende Bindungen innerhalb von Doppelsträngen werden geknüpft.

- $(S_1, S_2) \leftarrow \text{Gelelektrophorese}(S, n)$
  Die Stränge in S werden durch Gelelektrophorese ihrer Größe nach aufgetrennt. Alle Stränge mit einer Länge von n Basen beziehungsweise Basenpaaren (je nachdem, ob es sich um Einzel- oder Doppelstränge handelt) werden in S_1 gegeben, alle Stränge anderer Länge in S_2.

- Leeren(S)
  Der Inhalt von S wird entfernt, danach ist S leer.

- Leerheitstest(S)
  Es wird getestet, ob sich DNA-Stränge in S befinden.

- Schneiden$(S, X\#Y)$
  Bei allen Doppelsträngen, die die Teilsequenz XY auf einem der beiden Einzelstränge enthalten, wird zwischen X und Y geschnitten, dabei werden klebrige Enden erzeugt.

- Einzelstrangspaltung(S)
  Alle DNA-Stränge in S, die einzelsträngig vorliegen, werden zu Mononukleotiden abgebaut. Nicht vollständig komplementäre Doppelstränge werden ebenfalls an ihren einzelsträngigen Teilstücken abgebaut.

Als aus den vorangegangenen Operationen zusammengesetzte Operationen werden definiert:

- Verlängern(S, X, n, L)
  Alle Stränge aus S, die die Länge n und an ihrem $3'$-OH-Terminus die Sequenz L haben, werden an diesem Ende um die Sequenz X verlängert. Am Ende befinden sich nur die verlängerten Stränge in S. Die dafür nötige Operationsfolge ist in Algorithmus 3.1 angegeben, eine schematische Darstellung ist in Abbildung 3.2 zu sehen.

- $(S_1, S_2) \leftarrow$ Separieren(S, X)
  Alle Stränge aus S, die die Teilsequenz X enthalten, werden in S_1 gegeben, alle anderen in S_2. Die Abfolge der Operationen ist in Algorithmus 3.2 angegeben, Abbildung 3.3 zeigt eine schematische Darstellung des Ablaufs.

Algorithmus 3.1 (Verlängern)
Verlängern(S, X, n, L)

1. $(S_1, S_2) \leftarrow$ Gelelektrophorese(S, n)
2. $S_1 \leftarrow$ Synthese(X)
3. $S_1 \leftarrow$ Synthese($^*\overline{X_1}\ \overline{L}$), wobei X_1 Teilsequenz von X ist.
4. Hybridisieren(S_1)
5. Ligieren(S_1)
6. Denaturieren(S_1)
7. $(S_3, S_2) \leftarrow$ Filtern(S_1)
8. Leeren(S)
9. $(S, S_4) \leftarrow$ Gelelektrophorese($S_2, n + |X|$)
10. Leeren(S_1, S_2, S_3, S_4)
11. Rückgabe von S

Algorithmus 3.2 (Separieren)
$(S_1, S_2) \leftarrow$ **Separieren**(S, X)

1. $S \leftarrow$ Synthese$(^*\overline{X})$
2. Hybridisieren(S)
3. $(S_3, S_2) \leftarrow$ Filtern(S)
4. Denaturieren(S_3)
5. $(S_4, S_1) \leftarrow$ Filtern(S_3)
6. Leeren(S_3, S_4)

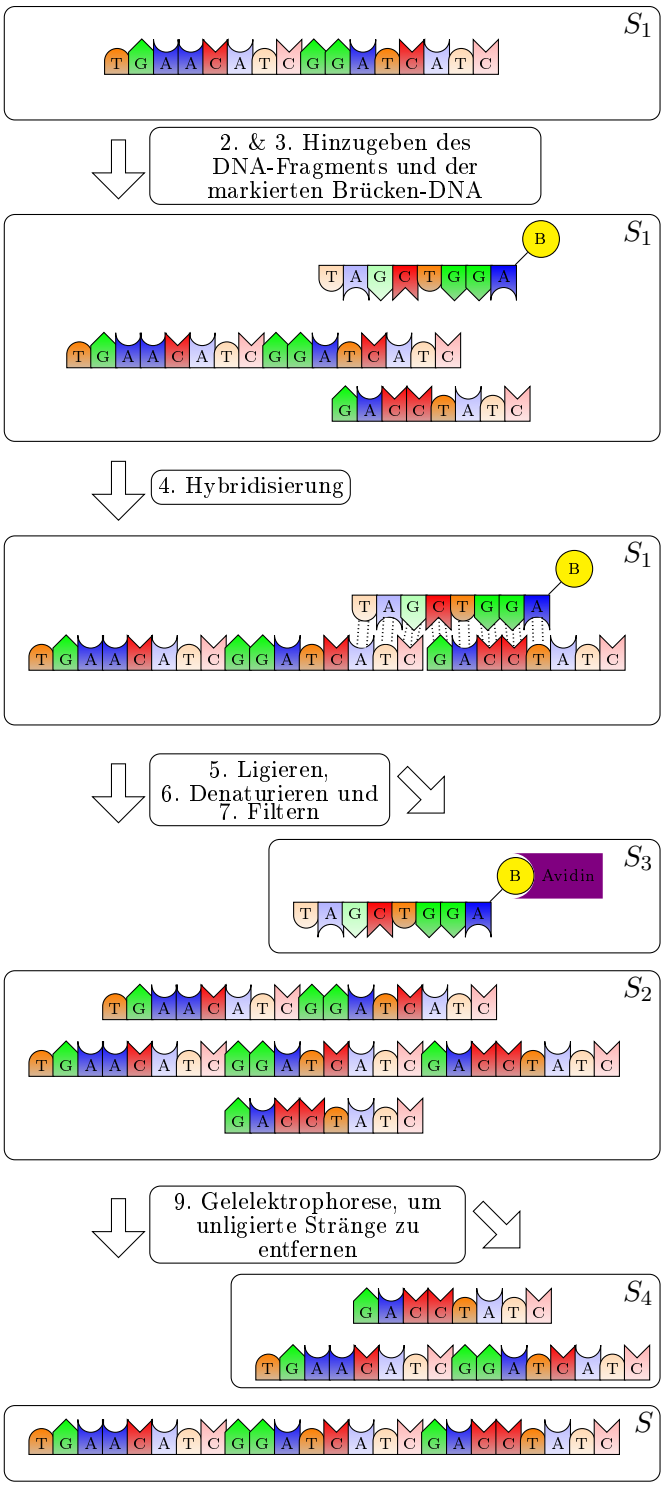


Abbildung 3.2: Verlängern eines DNA-Stranges.

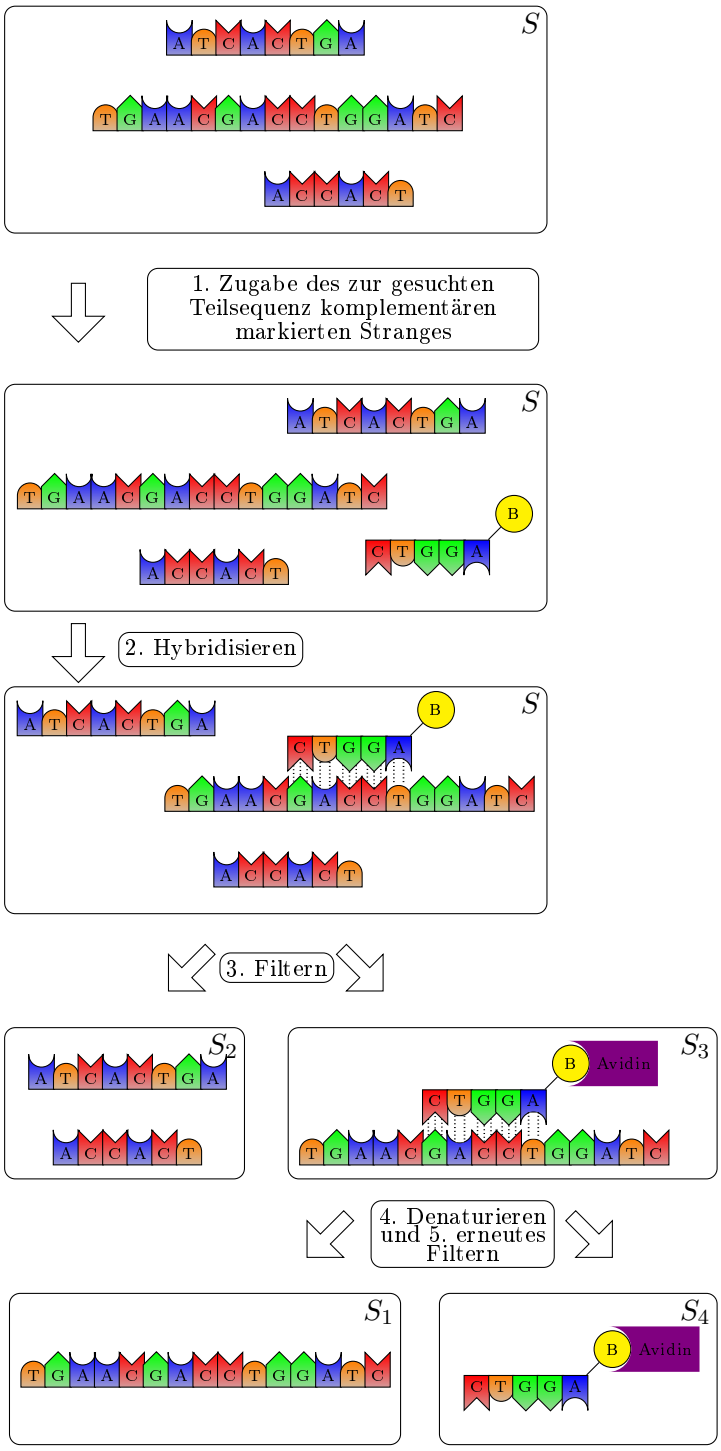


Abbildung 3.3: Separieren von Strängen, die eine bestimmte Teilsequenz enthalten.

3.2 Brute-Force-Algorithmus für 3QBF

3.2.1 Idee

Der Idee des Algorithmus stammt weitgehend von Evgeny Dantsin und Alexander Wolpert [38].

Gegeben sei eine Quantifizierte Boolesche Formel

$$F = Q_n x_n Q_{n-1} x_{n-1} \dots Q_1 x_1 \Phi$$

mit Quantoren $Q_i \in \{\exists, \forall\}$ und einer Booleschen Formel Φ, welche in Konjunktiver Normalform mit drei Literalen pro Klausel (3KNF) vorliegt. Die Anzahl der Klauseln sei m.

Der Algorithmus besteht aus zwei Phasen: In der ersten werden aus allen 2^n möglichen Belegungen diejenigen herausgesucht, welche Φ ohne die Quantifizierungen erfüllen. In der zweiten Phase werden dann aus diesen Schritt für Schritt diejenigen herausgefiltert, die F erfüllen, wobei die Quantoren von rechts nach links abgearbeitet werden.

Erste Phase: Aus der Menge aller 2^n Wörter $a_1 \cdots a_n \in \{0,1\}^{(n)}$ werden nacheinander für alle m Klauseln diejenigen Wörter aussortiert, welche für eine Belegung β mit $\beta(x_i) = a_i$ kodieren, die diese Klausel nicht erfüllt.

Am Ende bleiben dann nur die Wörter übrig, die für Belegungen kodieren, welche alle m Klauseln erfüllen.

Zweite Phase: Beginnend mit der am weitesten rechts stehenden wird für alle Quantifizierungen $Q_i x_i$, $i = 1, \dots, n-1$, folgendes gemacht:

1. Sind keine Wörter vorhanden, ist die Formel *falsch* und es kann abgebrochen werden.
2. Falls $Q_i = \exists$, wird bei allen Wörtern der Teil entfernt, der für $\beta(x_i)$ kodiert.
3. Falls $Q_i = \forall$, werden die Wörter in zwei Mengen aufgeteilt, je nachdem, ob a_i 0 oder 1 ist.
   a) In beiden Mengen wird bei allen Wörtern der Teil entfernt, der für $\beta(x_i)$ kodiert.
   b) Es wird der Durchschnitt beider Mengen gebildet.

Für Q_n wird folgendes gemacht:

Falls $Q_n = \exists$ und noch Wörter vorhanden sind, so ist die Formel wahr. Falls $Q_n = \forall$, werden die Stränge wieder in zwei Mengen geteilt, je nachdem, ob a_n 0 oder

1 ist. Falls sich in beiden Teilen Stränge befinden, ist die Formel wahr, ansonsten ist sie falsch.

Algorithmus 3.3 (Brute-Force-Algorithmus)
Eingabe: Quantifizierte Boolesche Formel $F = Q_n x_n \dots Q_1 x_1 \Phi$
mit $\Phi = \bigwedge_{j=1}^{m} K_j$ in 3KNF

Ausgabe: *F ist wahr* oder *F ist falsch.*

1. $S := \{a_1 a_2 \dots a_n \mid a_i \in \{0,1\},\ i = 1, \dots, n\}$
2. Für $j = 1, \dots, m$:
   $S := S \setminus \{a_1 a_2 \cdots a_n \mid \beta \not\models K_j,\ \beta(x_i) = a_i,\ i = 1, \dots, n\}$
3. Für $i = 1, \dots, n-1$:
   a) Falls $S = \emptyset$: Ausgabe *F ist falsch*
   b) Falls $Q_i = \exists$: $S := \{a_{i+1} \dots a_n \mid 0a_{i+1} \cdots a_n \in S \vee 1a_{i+1} \cdots a_n \in S\}$
   c) Falls $Q_i = \forall$: $S_0 := \{a_{i+1} \cdots a_n \mid 0a_{i+1} \cdots a_n \in S\}$
   $S_1 := \{a_{i+1} \cdots \mid 1a_{i+1} \cdots a_n \in S\}$
   $S := S_0 \cap S_1$
4. Falls $Q_n = \exists$: Falls $S = \emptyset$: Ausgabe *F ist falsch*
5. Falls $Q_n = \forall$: $S_0 := \{0 \mid 0 \in S\}$
   $S_1 := \{1 \mid 1 \in S\}$
   Falls $S_0 = \emptyset \vee S_1 = \emptyset$: Ausgabe *F ist falsch*
6. Ausgabe *F ist wahr*

Der Algorithmus beginnt also an den Blättern des Suchbaumes (siehe Abbildung 3.1) und findet zunächst alle erfüllenden Belegungen von Φ. Ausgehend von diesen wird der Suchbaum von unten nach oben durchlaufen. Bei existenzquantifizierten Variablen muss nur ein Nachfolgerknoten die Formel erfüllen, bei allquantifizierten Variablen beide, deshalb wird in diesem Fall der Durchschnitt $S = S_0 \cap S_1$ gebildet.

3.2.2 Beispiele

Die Idee des Brute-Force-Algorithmus soll an zwei Beispielen veranschaulicht werden, je eines, bei dem sich die betrachtete Formel als *wahr* beziehungsweise *falsch* herausstellen wird.

Beispiel I

Die folgende Formel F_1 sei gegeben:

$$F_1 = \exists x_4 \forall x_3 \forall x_2 \exists x_1 \ (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_1 \vee x_2 \vee x_4) \\ \wedge (\neg x_2 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3)$$

1. Die Ausgangsmenge S besteht aus den Wörtern, die für eine der $2^4 = 16$ Belegungen kodieren:

$$S = \{0000,\ 0001,\ 0010,\ 0011,\ 0100,\ 0101,\ 0110,\ 0111, \\ 1000,\ 1001,\ 1010,\ 1011,\ 1100,\ 1101,\ 1110,\ 1111\}$$

2. Es werden jetzt nacheinander für jede Klausel die Wörter aus S entfernt, die für Belegungen kodieren, welche diese Klausel nicht erfüllen:

$$\begin{aligned}
\text{Erste Klausel: } S = \{&0000,\ 0001,\ 0010,\ 0011,\ 0100,\ 0101,\ 0110,\ 0111, \\
&\cancel{1000},\ 1001,\ 1010,\ 1011,\ \cancel{1100},\ 1101,\ 1110,\ 1111\} \\
\text{Zweite Klausel: } S = \{&0000,\ 0001,\ 0010,\ 0011,\ 0100,\ 0101,\ 0110,\ 0111, \\
&1001,\ 1010,\ 1011,\ 1101,\ \cancel{1110},\ \cancel{1111}\} \\
\text{Dritte Klausel: } S = \{&\cancel{0000},\ 0001,\ \cancel{0010},\ 0011,\ 0100,\ 0101,\ 0110,\ 0111, \\
&1001,\ 1010,\ 1011,\ 1101,\ \} \\
\text{Vierte Klausel: } S = \{&\ 0001,\ 0011,\ \cancel{0100},\ 0101,\ 0110,\ 0111, \\
&1001,\ 1010,\ 1011,\ 1101,\ \} \\
\text{Fünfte Klausel: } S = \{&\ 0001,\ 0011,\ 0101,\ 0110,\ 0111, \\
&1001,\ 1010,\ 1011,\ \cancel{1101},\ \} \\
\text{Sechste Klausel: } S = \{&\ \cancel{0001},\ 0011,\ 0101,\ 0110,\ 0111, \\
&1001,\ 1010,\ 1011\ \}
\end{aligned}$$

Am Ende der ersten Phase enthält S alle sieben Wörter, die für eine erfüllende Belegung der unquantifizierten Formel kodieren:

$$S = \{0011,\ 0101,\ 0110,\ 0111,\ 1001,\ 1010,\ 1011\}$$

3. Die Quantoren werden nun von rechts nach links abgearbeitet:
Erster Quantor $\exists x_1$:

- Die Kodierung für $\beta(x_1)$ wird entfernt:

$$S = \{011,\ 101,\ 110,\ 111,\ 001,\ 010\}$$

Zweiter Quantor $\forall x_2$:

- Die Wörter werden auf zwei Mengen aufgeteilt, die Kodierung für die Belegung der zweiten Variablen wird anschließend entfernt:
  - $S_0 = \{11,\ \ 01,\ \ 10\}$
  - $S_1 = \{11,\ \ 01,\ \ 10\}$
- Es werden nur die Wörter behalten, die in beiden Mengen auftreten:

$$S = S_0 \cap S_1 = \{11,\ \ 01,\ \ 10\}$$

Dritter Quantor $\forall x_3$:

- Die Wörter werden verteilt, die Kodierung für die Belegung der dritten Variablen wird entfernt:
  - $S_0 = \{1\}$
  - $S_1 = \{1,\ \ 0\}$
- Es wird wieder nur das Wort behalten, das in beiden Mengen auftritt:

$$S = S_0 \cap S_1 = \{1\}$$

4.-6. Vierter Quantor $\exists x_4$:

- Da ein Wort vorhanden ist, ist die quantifizierte Formel F_1 wahr.

Beispiel II

Gegeben sei die Formel F_2

$$\begin{aligned} F_2 = \forall x_4 \forall x_3 \exists x_2 \forall x_1 \, (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3 \vee x_4) \\ \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \end{aligned}$$

1. Die Ausgangsmenge bilden wiederum alle $2^4 = 16$ Wörter:

$$\begin{aligned} S = \{&0000,\ \ 0001,\ \ 0010,\ \ 0011,\ \ 0100,\ \ 0101,\ \ 0110,\ \ 0111 \\ &1000,\ \ 1001,\ \ 1010,\ \ 1011\ \ 1100,\ \ 1101,\ \ 1110,\ \ 1111\} \end{aligned}$$

2. Für jede Klausel werden die Belegungen entfernt, die diese Klausel nicht erfüllen:

$$\begin{aligned}
\text{Erste Klausel: } S = \{&0000,\ 0001,\ \cancel{0010},\ \cancel{0011},\ 0100,\ 0101,\ 0110,\ 0111,\\
&1000,\ 1001,\ 1010,\ 1011,\ 1100,\ 1101,\ 1110,\ 1111\}\\
\text{Zweite Klausel: } S = \{&0000,\ 0001,\ \ \ \ \ \ \ \ 0100,\ 0101,\ \cancel{0110},\ 0111,\\
&1000,\ 1001,\ 1010,\ 1011,\ 1100,\ 1101,\ 1110,\ 1111\}\\
\text{Dritte Klausel: } S = \{&0000,\ 0001,\ \ \ \ \ \ \ \ 0100,\ 0101,\ \ \ \ \ \ 0111,\\
&1000,\ 1001,\ \cancel{1010},\ \cancel{1011},\ 1100,\ 1101,\ 1110,\ 1111\}\\
\text{Vierte Klausel: } S = \{&0000,\ 0001,\ \ \ \ \ \ \ \ 0100,\ 0101,\ \ \ \ \ \ \cancel{0111},\\
&1000,\ 1001,\ \ \ \ \ \ \ \ 1100,\ 1101,\ 1110,\ 1111\}
\end{aligned}$$

Zehn Wörter kodieren also für erfüllende Belegungen der unquantifizierten Formel:

$$S = \{0000,\ 0001,\ 0100,\ 0101,\ 1000,\ 1001,\ 1100,\ 1101,\ 1110,\ 1111\}$$

3. Die Quantoren werden nun von rechts nach links abgearbeitet.
Erster Quantor $\forall x_1$:

- Die Wörter werden auf zwei Mengen aufgeteilt, die Kodierung für die Belegung der ersten Variablen wird entfernt:
  - $S_0 = \{000,\ 001,\ 100,\ 101\}$
  - $S_1 = \{000,\ 001,\ 100,\ 101,\ 110,\ 111\}$
- Es werden nur die Wörter behalten, die in beiden Mengen auftreten:
  $S = S_0 \cap S_1 = \{000,\ 001,\ 100,\ 101\}$

Zweiter Quantor $\exists x_2$:

- Die Kodierung für die Belegung der zweiten Variablen wird entfernt:
  $S = \{00,\ 01\}$

Dritter Quantor $\forall x_3$:

- Die Wörter werden verteilt, die Kodierung für die Belegung der dritten Variablen wird entfernt:
  - $S_0 = \{0,\ 1\}$
  - $S_1 = \emptyset$
- Es werden nur die Wörter behalten, die in beiden Teilen auftreten.

  $$S = S_0 \cap S_1 = \emptyset$$

  Da S leer ist, bricht der Algorithmus vor dem vierten Durchlauf der Schleife mit dem Ergebnis ab, dass die quantifizierte Formel falsch ist.

3.2.3 Implementation

Erste Phase

Die Implementation der ersten Phase basiert auf einem Vorschlag von Braich et al. [3] zur Lösung des Erfüllbarkeitsproblems 3SAT. Dieser Vorschlag wurde für eine Boolesche Formel mit 20 Variablen und 24 Klauseln praktisch durchgeführt und ist damit die bislang größte Implementation.

1. Für jede der n Variablen werden zwei distinkte einzelsträngige DNA-Fragmente synthetisiert: X_i^W, $i = 1, \ldots, n$, repräsentiert die Belegung der Variablen x_i mit 1, X_i^F die Belegung mit 0. Die Wahl der Sequenzen dieser Stränge wird in Abschnitt 4.1 genauer beschrieben. Man erhält $2n$ verschiedene Stränge. Aus diesen werden nun alle 2^n Stränge synthetisiert, welche alle möglichen Belegungen der n Variablen repräsentieren. Zwischen den Fragmenten der Variablen befinde sich jeweils ein Verbindungsfragment („*Linker*“) L. Die Stränge haben nun die Form

$$5'\text{-}LX_1^{Z_1}LX_2^{Z_2}L \ldots LX_n^{Z_n}L\text{-}3' \quad \text{mit } Z_i \in \{W, F\}.$$

Diese Stränge bilden den Ausgangspool S. Die Länge der einzelnen Fragmente X_i^W und X_i^F sei für alle i gleich, die Länge eines dieser Fragmente wird daher immer mit $\left|X_1^W\right|$ angegeben. Alle Stränge in S haben somit die Länge $|L| + n \cdot \left|X_1^W L\right|$.

2. Für $i = 1, \ldots, m$ wird Folgendes gemacht:
Für die *i-te* Klausel $(x_{i1}^{\varepsilon_{i1}} \vee x_{i2}^{\varepsilon_{i2}} \vee x_{i3}^{\varepsilon_{i3}})$ werden die Fragmente

$${}^*\overline{L}\,\overline{X}_{i1}^{Z_{i1}},\ {}^*\overline{L}\,\overline{X}_{i2}^{Z_{i2}},\ {}^*\overline{L}\,\overline{X}_{i3}^{Z_{i3}}, \text{ wobei } X_{ik}^{Z_{ik}} = \begin{cases} X_j^W & \text{, falls } x_{ik}^{\varepsilon_{ik}} = x_j^1 \\ X_j^F & \text{, falls } x_{ik}^{\varepsilon_{ik}} = x_j^0 \end{cases} (k = 1, 2, 3) \text{ ist,}$$

zu S hinzugegeben und die zueinander komplementären Stränge werden hybridisiert. Alle Stränge, die die *i-te* Klausel erfüllen, sind mit mindestens einem der markierten Fragmente ${}^*\overline{L}\ \overline{X}_{ik}^{Z_{ik}}$ über Wasserstoffbrücken verbunden. Nun werden alle markierten Stränge herausgefiltert, die unmarkierten werden verworfen. Anschließend erfolgt die Denaturierung der (partiellen) Doppelstränge in Einzelstränge und ein erneutes Herausfiltern der markierten Stränge. Die unmarkierten Stränge bilden jetzt die neue Menge S aller die Klauseln $1, \ldots, i$ erfüllenden Stränge.

Im Anschluss wird getestet, ob die Menge S noch Stränge enthält. Ist das der Fall, beginnt die nächste Iteration; ansonsten wird mit dem Ergebnis abgebrochen, dass die Formel *falsch* ist.

Nach der *m-ten* Iteration enthält die Menge S alle Stränge, die alle Klauseln erfüllen.

Algorithmus 3.4 (Erste Phase des Brute-Force-Algorithmus)

1. $S \leftarrow \text{Synthese}(\{LX_1^{Z_1}L \cdots LX_n^{Z_n}L \mid Z_1, \ldots, Z_n \in \{W, F\}\})$
2. Für $i = 1, \ldots, m$:
   - $(S_1, S_2) \leftarrow \text{Separieren}(S, X_{i1}^{Z_{i1}})$
   - $(S_1, S_3) \leftarrow \text{Separieren}(S_2, X_{i2}^{Z_{i2}})$
   - $(S_1, S_4) \leftarrow \text{Separieren}(S_3, X_{i3}^{Z_{i3}})$
   - Leeren(S)
   - $S \leftarrow S_1$
   - Falls S leer: Ausgabe *F ist falsch*
   - Leeren(S_1, S_2, S_3, S_4)

Zweite Phase

1. In der zweiten Phase wird für $i = 1, \ldots, n-1$ eine Schleife durchlaufen:

1a) Ist die *i-te* Quantifizierung ein Existenzquantor, werden Fragmente $\overline{L}\,\overline{X}_i^W$ und $\overline{L}\,\overline{X}_i^F$ hinzugegeben und hybridisiert. Nun werden alle Stränge mit Restriktionsenzymen geschnitten, welche die Erkennungssequenz $X_i^F\#L$ oder $X_i^W\#L$ besitzen. Die vorherige Hybridisierung mit den komplementären Fragmenten ist nötig, da Restriktionsenzyme nur Doppelstränge schneiden. Anschließend werden die Stränge denaturiert und mithilfe der Gelelektrophorese der Länge nach aufgetrennt. Alle Stränge der Länge $(n-i)\cdot\left|X_1^W L\right| + |L|$ bilden die neue Menge S, mit der die nächste Iteration durchlaufen wird.

1b) Falls es sich um einen Allquantor handelt, werden zunächst markierte komplementäre Stränge ${}^*\overline{L}\,\overline{X}_i^F$ hinzugegeben und mit den passenden Strängen hybridisiert. Die entstandenen Doppelstränge werden dann herausgefiltert und in ein neues Reagenzglas S_0 übertragen. Zu den restlichen Strängen werden $\overline{L}\,\overline{X}_i^W$-Stränge gegeben und hybridisiert. In Reagenzglas S_0 befinden sich nun nur Stränge mit der Belegung 0 für x_i, im zweiten Reagenzglas S_1 nur solche mit der Belegung 1.

In Reagenzglas S_0 werden alle Stränge nach Zugabe von Restriktionsenzymen mit der Erkennungssequenz $X_i^F\#L$ geschnitten, in S_1 mit $X_i^W\#L$. Nach der Denaturierung erfolgt für alle Stränge jedes der beiden Reagenzgläser eine Auftrennung der

Länge nach und alle Stränge der Länge $(n-i)\cdot\left|X_1^W L\right|+|L|$ werden behalten, alle anderen verworfen.

Zu den DNA-Strängen des Reagenzglases S_0 werden jetzt die komplementären Fragmente ${}^*\overline{L}\ \overline{X}_n^Z$ und $\overline{L}\ \overline{X}_j^Z$ mit $j=i+1,\ldots,n-1$ und $Z\in\{W,F\}$ gegeben und hybridisiert. Die einzelnen Fragmente werden mit Ligasen verbunden. Nach einer Denaturierung und anschließender Auftrennung mit Gelelektrophorese werden die markierten Stränge der Länge $(n-i)\cdot\left|X_1^W L\right|+|L|$ separiert. Dies sind die komplementären Stränge zu den Strängen aus S_0; sie werden nun in S_1 gegeben. Nach der Hybridisierung der zueinander komplementären Stränge erfolgt eine Spaltung aller Einzelstränge. Dadurch werden auch unvollständig hybridisierte Doppelstränge in kleinere Fragmente gespaltet. Anschließend werden die Wasserstoffbrücken durch Denaturierung gelöst und alle markierten Einzelstränge herausgefiltert. Die unmarkierten Stränge werden durch eine Gelelektrophorese nach ihrer Länge getrennt, alle Stränge der Länge $(n-i)\cdot\left|X_1^W L\right|+|L|$ sind die Stränge, die vollständig hybridisiert waren und damit sowohl in S_0 als auch in S_1 auftraten, und bilden die neue Menge S.

1c) Am Ende jeder Iteration wird getestet, ob überhaupt noch DNA-Stränge vorhanden sind. Wenn nicht, wird der Algorithmus mit dem Ergebnis abgebrochen, dass die Formel *falsch* ist.

2. Falls $Q_n=\forall$ ist, erfolgt eine erneute Trennung der Stränge in S_0 und S_1 und der Test, ob in beiden Reagenzgläsern DNA-Stränge vorhanden sind. Wenn sich in mindestens einem der beiden Gläser keine DNA befindet, so ist die Formel *falsch*.

3. Die Formel ist *wahr*, wenn der letzte Quantor ein Existenzquantor ist und nach der letzten Iteration noch Stränge vorhanden sind, oder wenn der letzte Quantor ein Allquantor ist und sich nach der letzten Trennung der Stränge in beiden Reagenzgläsern noch Stränge befinden.

Algorithmus 3.5 (Zweite Phase des Brute-Force-Algorithmus)

1. Für $i=1,\ldots,n-1$:
   a) Falls $Q_i=\exists$:
      - $S\leftarrow$ Synthese$(\overline{L}\ \overline{X}_i^W,\overline{L}\ \overline{X}_i^F)$
      - Hybridisieren(S)
      - Schneiden$(S,X_i^W\#L)$, Schneiden$(S,X_i^F\#L)$
      - Denaturieren(S)
      - $(S_1,S_2)\leftarrow$ Gelelektrophorese$(S,|L|+(n-i)\cdot\left|X_1^W L\right|)$

- Leeren(S)
- $S \leftarrow S_1$
- Leeren(S_1, S_2)

b) Falls $Q_i = \forall$:

- $S \leftarrow$ Synthese($^*\overline{L}\ \overline{X}_i^F$)
- Hybridisieren(S)
- $(S_0, S_1) \leftarrow$ Filtern(S)
- Leeren(S)
- $S_1 \leftarrow$ Synthese($^*\overline{L}\ \overline{X}_i^W$)
- Hybridisieren(S_1)
- Schneiden($S_0, X_i^F \# L$), Schneiden($S_1, X_i^W \# L$)
- Denaturieren(S_0), Denaturieren(S_1)
- $(S_2, S_4) \leftarrow$ Gelelektrophorese($S_0, |L| + (n-i) \cdot \left|X_1^W L\right|$)
- $(S_3, S_4) \leftarrow$ Gelelektrophorese($S_1, |L| + (n-i) \cdot \left|X_1^W L\right|$)
- Leeren(S_0, S_1, S_4)
- $S_2 \leftarrow$ Synthese($\{\overline{L}\ \overline{X}_j^Z \mid Z \in \{W, F\}, j \in \{i+1, \ldots n-1\}\}$)
- $S_2 \leftarrow$ Synthese($^*\overline{L}\ \overline{X}_n^W, {}^*\overline{L}\ \overline{X}_n^F$)
- Hybridisieren(S_2)
- Ligieren(S_2)
- Denaturieren(S_2)
- $(S_0, S_1) \leftarrow$ Gelelektrophorese($S_2, |L| + (n-i) \cdot \left|X_1^W L\right|$)
- $(S_3, S_2) \leftarrow$ Filtern(S_0)
- Leeren(S_0, S_1, S_2)
- Hybridisieren(S_3)
- Einzelstrangspaltung(S_3)
- Denaturieren(S_3)
- $(S_0, S_1) \leftarrow$ Filtern(S_3)
- $(S, S_0) \leftarrow$ Gelelektrophorese($S_1, |L| + (n-i) \cdot \left|X_1^W L\right|$)
- Leeren(S_0, S_1, S_3)

c) Falls S leer: Ausgabe *F ist falsch*

2. Falls $Q_n = \forall$:
   - $(S_0, S_1) \leftarrow \text{Separieren}(S, X_n^W)$
   - Falls S_0 oder S_1 leer: Ausgabe *F ist falsch*
3. Ausgabe *F ist wahr*

3.2.4 Laufzeit und Anzahl benötigter Stränge

In der ersten Phase sind für die Synthese der Stränge $O(n)$ und für den Test der Erfüllbarkeit der unquantifizierten Formel Φ $O(m)$ Schritte durchzuführen. In der zweiten Phase werden werden n Iterationen durchgeführt, es sind also $O(n)$ Schritte nötig. Insgesamt ergibt sich eine lineare Laufzeit $O(n + m)$, wenn man davon ausgeht, dass jede DNA-Operation in konstanter Zeit durchgeführt werden kann. Die Anzahl der benötigten Stränge ist mit $O(2^n)$ exponentiell, weil zu Beginn alle möglichen Belegungen erzeugt werden müssen. Hierin liegt der große Nachteil des Brute-Force-Ansatzes. Nach einer Abschätzung von Hartmanis in [13] wird bereits bei einer Variablengröße von $n = 200$ DNA mit einer Masse benötigt, die die der Erde übersteigt. Die Abschätzung wurde für das Adleman-Experiment gemacht, ist hier aber genauso gültig: Die Länge der DNA-Sequenzen bei 2^{200} Strängen ist mindestens $log_4 200$. Das Gewicht einer Base wird mit 10^{-25} kg nach unten abgeschätzt. Es wird also mindestens DNA mit einer Masse von

$$2^{200} \cdot log_4 200 \cdot 10^{-25}\text{kg} \geq 3 \cdot 10^{25}\text{kg}$$

benötigt. Die Masse der Erde liegt hingegen „nur“ bei $5,974 \cdot 10^{24}$ kg. Deshalb ist dieser Ansatz nur für Variablenanzahlen bis $n = 60$ brauchbar. Diese Problemgrößen können jedoch von heutigen digitalen Rechnern problemlos gelöst werden.

3.3 Algorithmus mit Breitensuche

3.3.1 Idee

Dieser Algorithmus basiert auf einem Verfahren in [41] zur Lösung des Erfüllbarkeitsproblems 3SAT. Ausgangspunkt ist nicht mehr die Menge aller 2^n möglichen Wörter; die Lösungen werden hingegen Schritt für Schritt aufgebaut, beginnend mit den vier Wörtern 00, 01, 10 und 11.

Gegeben sei wie zuvor eine Quantifizierte Boolesche Formel

$$F = Q_n x_n Q_{n-1} x_{n-1} \ldots Q_1 x_1 \Phi,$$

Φ liege wieder in Konjunktiver Normalform mit drei Literalen pro Klausel vor. O.B.d.A. trete keine der Variablen oder ihre Negation mehr als einmal pro Klausel auf.

Zunächst werden die Variablen so umbenannt, dass die Anzahl ihres Auftretens (oder ihrer Negation) nicht abnimmt. x_1 ist also die Variable, die am häufigsten, und x_n die, die am seltensten vorkommt. Zusätzlich werden die Literale innerhalb einer Klausel aufsteigend geordnet. Die Reihenfolge der Variablenindizes innerhalb des Quantorenblocks ist nun nicht mehr geordnet; man erhält

$$\hat{F} = Q_n x_{i_n} Q_{n-1} x_{i_{n-1}} \cdots Q_1 x_{i_1} \hat{\Phi},$$

wobei $\hat{\Phi}$ aus Φ durch Ersetzen von $x_j (j = 1, \ldots, n)$, durch x_{i_j} und aufsteigende Ordnung der Literale innerhalb einer Klausel entsteht.

M sei im Folgenden die Menge der Klauseln von $\hat{\Phi}$.

Durch die aufsteigende Ordnung der Literale innerhalb der Klauseln und dadurch, dass jede Variable oder ihre Negation nur einmal pro Klausel auftreten, hat das letzte Literal jeder Klausel einen Index größer oder gleich drei.

Es werden die folgenden Mengen definiert:

$$\begin{aligned} M_k &= \left\{ \left(x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l} \right) \mid (x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l} \vee x_k^1) \in M, j < l < k \right\} \\ \overline{M}_k &= \left\{ \left(x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l} \right) \mid (x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l} \vee x_k^0) \in M, j < l < k \right\} \end{aligned} \tag{3.1}$$

Nun lässt sich $\hat{\Phi}$ in der folgenden Form darstellen:

$$\hat{\Phi} = \bigwedge_{k=3}^{n} \left(\bigwedge_{\varphi \in M_k} (\varphi \vee x_k^1) \wedge \bigwedge_{\varphi \in \overline{M}_k} (\varphi \vee x_k^0) \right) \tag{3.2}$$

Definiert man

$$\Phi_k^0 = \bigwedge_{\varphi \in M_k} \varphi \text{ und } \Phi_k^1 = \bigwedge_{\varphi \in \overline{M}_k} \varphi,$$

so ergibt sich (durch Anwendung des Distributivgesetzes)

$$\hat{\Phi} = \bigwedge_{k=3}^{n} \left(\Phi_k^0 \vee x_k^1 \right) \wedge \left(\Phi_k^1 \vee x_k^0 \right). \tag{3.3}$$

Gestartet wird bei diesem Algorithmus mit den vier Wörtern, die für die Belegungen der ersten beiden Variablen x_1 und x_2 kodieren. Die Startmenge ist

$$S = \{a_1a_2 \mid a_1, a_2 \in \{0,1\}\} = \{00, 01, 10, 11\}.$$

Für $k = 3, \ldots, n$ wird eine Schleife durchlaufen:

Es werden zwei Mengen S_0 und S_1 gebildet. Dabei enthält S_0 alle Wörter aus S, die Φ_k^0 erfüllen, und S_1 alle Wörter, die Φ_k^1 erfüllen. Falls Φ_k^0 oder Φ_k^1 aus keiner Klausel bestehen, dann enthalten S_0 beziehungsweise S_1 alle Wörter aus S.

An jedes Wort aus S_0 wird nun $a_k = 0$ angehängt; an jedes aus S_1 $a_k = 1$. Danach werden beide Mengen zu $S = (S_0 \otimes \{0\}) \cup (S_1 \otimes \{1\})$ vereinigt. Falls $S = \emptyset$ ist, kann an dieser Stelle mit dem Ergebnis abgebrochen werden, dass die Formel falsch ist. Ansonsten erfolgt die nächste Iteration.

Nach der *n-ten* Iteration enthält die Menge S alle Wörter $a_1a_2 \ldots a_n$, die für Belegungen kodieren, welche die unquantifizierte Formel erfüllen. Nun werden analog zum Brute-Force-Algorithmus die Quantifizierungen von rechts nach links abgearbeitet, beginnend mit $Q_1x_{i_1}$. Dabei wird jedoch nicht immer der erste Teil des Wortes entfernt, sondern unter Umständen ein Teil im Inneren des Wortes, da die quantifizierten Variablen jetzt nicht mehr absteigend geordnet sind.

Algorithmus 3.6 (Algorithmus mit Breitensuche)

Eingabe: Quantifizierte Boolesche Formel $\hat{F} = Q_n x_{i_n} \ldots Q_1 x_{i_1} \hat{\Phi}$
mit $\hat{\Phi} = \bigwedge_{k=3}^{n} \left(\Phi_k^0 \vee x_k^1\right) \wedge \left(\Phi_k^1 \vee x_k^0\right)$

Ausgabe: *F ist wahr* oder *F ist falsch.*

1. $S := \{00, 01, 10, 11\}$
2. Für $k = 3, \ldots, n$:
   a) $S_0 := \{a_1 \ldots a_{k-1} \mid a_1 \ldots a_{k-1} \in S,\ \beta \models \Phi_k^0,\ \beta(x_i) = a_i,\ i = 1, \ldots, k-1\}$
      $S_1 := \{a_1 \ldots a_{k-1} \mid a_1 \ldots a_{k-1} \in S,\ \beta \models \Phi_k^1,\ \beta(x_i) = a_i,\ i = 1, \ldots, k-1\}$
   b) $S = (S_0 \otimes \{0\}) \cup (S_1 \otimes \{1\})$
   c) Falls $S = \emptyset$: Rückgabe *F ist falsch*
3. Für $j = 1, \ldots, n$:
   a) Falls $S = \emptyset$: Rückgabe *F ist falsch*

b) Falls $Q_j = \exists$:
$S := \{a_1 \dots a_{i_j-1} a_{i_j+1} \dots a_n \mid \quad a_1 \dots a_{i_j-1} 0 a_{i_j+1} \dots a_n \in S$
$\vee a_1 \dots a_{i_j-1} 1 a_{i_j+1} \dots a_n \in S\}$

c) Falls $Q_j = \forall$:
$S_0 := \{a_1 \dots a_{i_j-1} a_{i_j+1} \dots a_n \mid a_1 \dots a_{i_j-1} 0 a_{i_j+1} \dots a_n \in S\}$
$S_1 := \{a_1 \dots a_{i_j-1} a_{i_j+1} \dots a_n \mid a_1 \dots a_{i_j-1} 1 a_{i_j+1} \dots a_n \in S\}$
$S := S_0 \cap S_1$

4. Falls $Q_n = \exists$:
Falls $S = \emptyset$: Rückgabe *F ist falsch*

5. Falls $Q_n = \forall$:
Falls $0 \notin S$ oder $1 \notin S$: Rückgabe *F ist falsch*

6. Rückgabe *F ist wahr*

3.3.2 Korrektheit

Lemma 3.1 *Eine Belegung β erfüllt die Formel $\left(\Phi^0 \vee x_k^1\right) \wedge \left(\Phi^1 \vee x_k^0\right)$ genau dann, wenn β die Formel Φ^0 erfüllt und $\beta(x_k) = 0$ ist oder wenn β die Formel Φ^1 erfüllt und $\beta(x_k) = 1$ ist.*

$$\beta \models \left(\left(\Phi^0 \vee x_k^1\right) \wedge \left(\Phi^1 \vee x_k^0\right)\right) \Leftrightarrow \left(\beta \models \Phi^0 \wedge \beta(x_k) = 0\right) \vee \left(\beta \models \Phi^1 \wedge \beta(x_k) = 1\right)$$

Beweis:
„$\Rightarrow$“: Nach Anwendung des Distributivgesetzes ergeben sich vier Fälle. Zwei davon ergeben sofort die Aussage, $\beta \models (x_k^1 \wedge x_k^0)$ ist eine Kontradiktion. Es bleibt der Fall, dass β die Formel $\Phi^0 \wedge \Phi^1$ erfüllt. Da $\beta(x_k)$ nur 0 oder 1 sein kann und β sowohl Φ^0 als auch Φ^1 erfüllt, folgt auch hier die Aussage.
„$\Leftarrow$“: Die Aussage folgt sofort, da in beiden Fällen der rechten Seite beide durch „$\wedge$“ verknüpfte Formeln der linken Seite erfüllt sind.

Der formale Beweis lautet:

$$
\begin{array}{lll}
\text{„}\Rightarrow\text{“} & & \beta \models \big((\Phi^0 \vee x_k^1) \wedge (\Phi^1 \vee x_k^0)\big) \\
& \Rightarrow & \beta \models \big(\Phi^0 \wedge \Phi^1\big) \vee \big(\beta \models \Phi^0 \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \Phi^1 \wedge \beta(x_k) = 1\big) \\
& \Rightarrow & \big(\beta \models \big(\Phi^0 \wedge \Phi^1\big) \wedge (\beta(x_k) = 0 \vee \beta(x_k) = 1)\big) \\
& & \vee \big(\beta \models \Phi^0 \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \Phi^1 \wedge \beta(x_k) = 1\big) \\
& \Rightarrow & \big(\big(\beta \models \big(\Phi^0 \wedge \Phi^1\big) \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \big(\Phi^0 \wedge \Phi^1\big) \wedge \beta(x_k) = 1\big)\big) \\
& & \vee \big(\beta \models \Phi^0 \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \Phi^1 \wedge \beta(x_k) = 1\big) \\
& \Rightarrow & \vee \big(\beta \models \Phi^0 \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \Phi^1 \wedge \beta(x_k) = 1\big) \\
\\
\text{„}\Leftarrow\text{“} & & \big(\beta \models \Phi^0 \wedge \beta(x_k) = 0\big) \vee \big(\beta \models \Phi^1 \wedge \beta(x_k) = 1\big) \\
& \Rightarrow & \big(\beta \models \Phi^0 \vee \beta \models \Phi^1\big) \wedge \big(\beta \models \Phi^0 \vee \beta(x_k) = 1\big) \\
& & \wedge \big(\beta \models \Phi^1 \vee \beta(x_k) = 0\big) \\
& \Rightarrow & \beta \models \big((\Phi^0 \vee x_k^1) \wedge (\Phi^1 \vee x_k^0)\big)
\end{array}
$$

□

Der Algorithmus startet mit allen vier Wörtern $a_1 a_2 \in \{0,1\}^{(2)}$ und ordnet sie den Formeln Φ_3^0 oder Φ_3^1 zu, wenn sie diese erfüllen. Im ersten Fall werden die Wörter um $a_3 = 0$, im zweiten Fall um $a_3 = 1$ erweitert. Nach Lemma 3.1 ist die Menge all dieser Wörter genau die Menge der Wörter, die die Formel

$$\big(\Phi_3^0 \vee x_3^1\big) \wedge \big(\Phi_3^1 \vee x_3^0\big)$$

erfüllen.

Während der *j-ten* Iteration ($j = 4, \ldots, n$) werden aus der Menge S aller Wörter, die bereits die Teilformel

$$\bigwedge_{k=3}^{j-1} \big(\Phi_k^0 \vee x_k^1\big) \wedge \big(\Phi_k^1 \vee x_k^0\big)$$

erfüllen, die Mengen S_0 und S_1 gebildet, die aus allen Wörtern bestehen, die Φ_j^0 beziehungsweise Φ_j^1 erfüllen. Alle Wörter aus S_0 werden um $a_j = 0$ erweitert, alle Wörter aus S_1 um $a_j = 1$. Nach Lemma 3.1 enthält die durch Vereinigung gebildete Menge $S = (S_0 \otimes \{0\}) \cup (S_1 \otimes \{1\})$ genau die Wörter, die

$$\big(\Phi_j^0 \vee x_3^1\big) \wedge \big(\Phi_j^1 \vee x_3^0\big)$$

und somit auch

$$\bigwedge_{k=3}^{j} \big(\Phi_k^0 \vee x_k^1\big) \wedge \big(\Phi_k^1 \vee x_k^0\big)$$

erfüllen. Nach der *n-ten* Iteration enthält S somit alle Wörter $a_1 \cdots a_n$, die Formel (3.3) erfüllen.

3.3.3 Beispiele

Um die Idee des auf Breitensuche basierenden Algorithmus zu demonstrieren, werden wiederum die Beispiele aus dem vorherigen Abschnitt betrachtet.

Beispiel I

$$F_1 = \exists x_4 \forall x_3 \forall x_2 \exists x_1 \, (\neg x_1 \vee x_3 \vee x_4) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee \neg x_3) \ \wedge \ (x_1 \vee x_2 \vee x_4) \\ \wedge (\neg x_2 \vee x_3 \vee x_4) \ \wedge \ (\neg x_1 \vee \neg x_2 \vee \neg x_4) \ \wedge \ (x_1 \vee x_2 \vee x_3)$$

Zunächst werden die Variablen nach der Häufigkeit ihres Auftretens geordnet. Dies ist in diesem Beispiel schon der Fall: x_1 und x_2 treten jeweils fünfmal auf, x_3 und x_4 je viermal. Innerhalb der Klauseln werden die Literale aufsteigend geordnet; auch dies ist hier schon der Fall.

Gestartet wird mit der Menge $S = \{00,\ 01,\ 10,\ 11\}$.
Nun werden die Klauseln ausgewählt, in denen die Variable x_3 oder ihre Negation $\neg x_3$ als letztes Literal vorkommen:

- $x_3 : \ (x_1 \vee x_2 \vee x_3)$
- $\neg x_3 : \ (\neg x_1 \vee \neg x_2 \vee \neg x_3)$

Aus jeder Klausel wird x_3 beziehungsweise $\neg x_3$ entfernt:

- $\Phi_3^0 = x_1 \vee x_2$
- $\Phi_3^1 = \neg x_1 \vee \neg x_2$

Nun werden die Wörter den Formeln zugeordnet, die sie erfüllen:

- $S_0 = \{11,\ 10,\ 01\}$
- $S_1 = \{00,\ 10,\ 01\}$

An das Ende der Wörter wird im ersten Fall eine 0 angehängt, im zweiten Fall eine 1; man erhält die Menge $S = \{110,\ 100,\ 010,\ 001,\ 101,\ 011\}$.
Während der nächsten Iteration werden die Klauseln ausgewählt, deren letztes Literal x_4 oder $\neg x_4$ ist, diese werden entfernt:

- $\Phi_4^0 = (\neg x_1 \vee x_3) \wedge (x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$
- $\Phi_4^1 = (\neg x_1 \vee \neg x_2)$

Jetzt werden die Wörter aus der vorherigen Iteration den jeweiligen Formeln zugeordnet, sofern sie sie erfüllen:

- $S_0 = \{101,\ 011\}$
- $S_1 = \{100,\ 010,\ 001,\ 101,\ 011\}$

An das Ende der Wörter werden eine 0 beziehungsweise eine 1 angehängt. Man erhält die sieben Wörter, welche die unquantifizierte Formel erfüllen:

$$S = \{1010,\ 0110,\ 1001,\ 0101,\ 0011,\ 1010,\ 0111\}.$$

Mit diesen wird dann wie im Brute-Force-Algorithmus weitergearbeitet.

Beispiel II

$$F_2 = \forall x_4 \forall x_3 \exists x_2 \forall x_1 \, (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3 \vee x_4) \\ \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_3 \vee \neg x_4)$$

Die Variablen werden nach der Häufigkeit ihres Auftretens geordnet: x_1 und x_3 treten jeweils viermal auf, x_2 und x_4 je dreimal. x_2 und x_3 werden also umbenannt:

$$\forall x_4 \forall x_2 \exists x_3 \forall x_1 \, (x_1 \vee x_3 \vee \neg x_2) \wedge (x_1 \vee \neg x_2 \vee x_4) \\ \wedge (\neg x_1 \vee x_3 \vee \neg x_2) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$$

Nun werden die Literale innerhalb der Klauseln aufsteigend geordnet:

$$\forall x_4 \forall x_2 \exists x_3 \forall x_1 \, (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_4) \\ \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_4)$$

Gestartet wird mit der Menge $S = \{00,\ 01,\ 10,\ 11\}$.
Es werden zuerst alle Klauseln ausgewählt, in denen x_3 oder $\neg x_3$ auftreten, diese werden entfernt:

- $\Phi_3^0 = (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$
- $\Phi_3^1 = 1$

Nun werden die Wörter den Formeln zugeordnet, die sie erfüllen. Die leere Formel wird von allen Wörtern erfüllt.

- $S_0 = \{00,\ 10\}$
- $S_1 = \{00,\ 01,\ 10,\ 11\}$

Im ersten Fall wird eine 0 angehängt, im zweiten Fall eine 1; es entsteht die Menge

$$S = \{000,\ 100,\ 001,\ 011,\ 101,\ 111\}.$$

Jetzt werden die Klauseln mit x_4 oder $\neg x_4$ ausgewählt:

- $\Phi_4^0 = (x_1 \vee \neg x_2)$
- $\Phi_4^1 = (x_1 \vee \neg x_2)$

Die Wörter werden zugeordnet ...

- $S_0 = \{000,\ 100,\ 001,\ 101,\ 111\}$
- $S_1 = \{000,\ 100,\ 001,\ 101,\ 111\}$

... und 0 beziehungsweise 1 angehängt. Es entstehen zehn Wörter:

$$S = \{0000,\ 1000,\ 0010,\ 1010,\ 1110,\ 0001,\ 1001,\ 0011,\ 1011,\ 1111\}.$$

Mit diesen wird jetzt wie im Brute-Force-Algorithmus weitergearbeitet, wobei aber hier die Reihenfolge der Variablen vertauscht ist.

3.3.4 Implementation

Erste Phase

Die Implementation der ersten Phase ist die Lösung des unquantifizierten 3SAT-Problems und erfolgt wie in [41] beschrieben. Im Gegensatz zum Brute-Force-Algorithmus ist die zusätzliche Operation zur Verlängerung von DNA-Strängen notwendig.

Gestartet wird mit vier Arten von Strängen:

$$LX_1^W LX_2^W L,\ LX_1^W LX_2^F L,\ LX_1^F LX_2^W L,\ LX_1^F LX_2^F L$$

. Sie bilden den Ausgangspool S.

Für $k = 3, \ldots, n$ wird jetzt Folgendes gemacht:

Zunächst werden die Stränge aus S auf zwei Reagenzgläser S_0 und S_1 aufgeteilt. In S_0 werden analag zum Brute-Force-Algorithmus die Stränge herausgefiltert, die Φ_k^0 erfüllen. Es wird also eine Schleife durchlaufen und Schritt für Schritt werden die Stränge herausgefiltert und verworfen, die eine der Klauseln aus M_k nicht erfüllen.

Das gleiche findet in S_1 für alle Klauseln aus Φ_k^1 statt.

An alle Stränge aus S_0 wird $X_k^F L$ angehängt, an alle Stränge aus S_1 $X_k^W L$. Nun werden S_0 und S_1 zu S vereinigt.

Zum Schluss wird getestet, ob S leer ist; in diesem Fall kann mit dem Ergebnis abgebrochen werden, dass F falsch ist.

Algorithmus 3.7 (Erste Phase der Breitensuche-Algorithmus)

1. $S \leftarrow \{LX_1^{Z_1} LX_2^{Z_2} L \mid Z_1, Z_2 \in \{W, F\}\}$

2. Für $k = 3, \ldots, n$:
   - $S_0 \leftarrow S,\ S_1 \leftarrow S$
   - Für jede Klausel $(x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l}) \in M_k$:
     - $(S_2, S_3) \leftarrow$ Separieren$(S_0, X_j^{Z_j})$
     - $(S_2, S_4) \leftarrow$ Separieren$(S_3, X_l^{Z_l})$

       wobei Z_j(bzw. Z_l) $= \begin{cases} W & \text{, falls } \varepsilon_j\text{(bzw. } \varepsilon_l) = 1 \\ F & \text{, falls } \varepsilon_j\text{(bzw. } \varepsilon_l) = 0 \end{cases}$
     - Leeren(S_0)
     - $S_0 \leftarrow S_2$
   - Leeren(S_2, S_3, S_4)
   - Für jede Klausel $(x_j^{\varepsilon_j} \vee x_l^{\varepsilon_l}) \in \overline{M}_k$:
     - $(S_2, S_3) \leftarrow$ Separieren$(S_1, X_j^{Z_j})$
     - $(S_2, S_4) \leftarrow$ Separieren$(S_2, X_l^{Z_l})$

       wobei Z_j(bzw. Z_l) $= \begin{cases} W & \text{, falls } \varepsilon_j\text{(bzw. } \varepsilon_l) = 1 \\ F & \text{, falls } \varepsilon_j\text{(bzw. } \varepsilon_l) = 0 \end{cases}$
     - Leeren(S_1)
     - $S_1 \leftarrow S_2$
   - Leeren(S_2, S_3, S_4)
   - Verlängern$(S_0, X_k^F L, |L| + (k-1) \cdot \left|X_1^W L\right|, L)$
   - Verlängern$(S_1, X_k^W L, |L| + (k-1) \cdot \left|X_1^W L\right|, L)$
   - Leeren(S)
   - $S \leftarrow S_0,\ S \leftarrow S_1$
   - Leeren(S_0, S_1)
   - Falls S leer: Ausgabe *F ist falsch*

Zweite Phase

Der Unterschied zur Implementation des Brute-Force-Algorithmus ist, dass die Kodierungen der Variablen in den DNA-Strängen nicht mehr in der Reihenfolge der Quantifizierungen auftreten. Sie können also nicht vom Ende der Stränge abgeschnitten werden, sondern müssen unter Umständen aus der Mitte der Stränge entfernt werden. Würde man mit zwei Restriktionsenzymen mit den Erkennungssequenzen $L\#X_i^{Z_i}$ und $X_i^{Z_i}\#L$ schneiden, so zerfiele der DNA-Strang in zwei Teile.

Es sollen zwei mögliche Algorithmen vorgestellt werden, die einen Ausweg bilden: Die Änderung der Reihenfolge der Variablenkodierungen oder die Verwendung ringförmiger DNA.

Änderung der Reihenfolge. Die Reihenfolge der Variablenkodierungen innerhalb des Stranges soll so geändert werden, dass sie der Reihenfolge der Variablenindizes innerhalb des Quantorenblocks entspricht.

S enthalte alle Stränge, die $\hat{\Phi}$ erfüllen, die Stränge haben die Form

$$LX_1^{Z_1}LX_2^{Z_2}L\cdots LX_n^{Z_n} \text{ mit } Z_k \in \{W,F\} \text{ für } k=1,\ldots,n.$$

Die gewünschte Reihenfolge ist aber die der Variablenindizes des Quantorenblocks:

$$LX_{i_1}^{Z_{i_1}}LX_{i_2}^{Z_{i_2}}\cdots LX_{i_n}^{Z_{i_n}}.$$

Um diese zu erreichen, werden zunächst Schritt für Schritt die passenden DNA-Stücke $LX_{i_1}^{Z_{i_1}}$, $LX_{i_2}^{Z_{i_2}}$, …, $LX_{i_n}^{Z_{i_n}}$ an die alten Stränge angefügt. Anschließend wird dann der alte Teil der Stränge entfernt.

Es wird also eine Schleife durchlaufen für $k=1,\ldots,n$: Alle Stränge aus S, die die Teilsequenz $X_{i_k}^W$ enthalten werden in ein Reagenzglas S_W gegeben, alle Stränge mit $X_{i_k}^F$ in S_F.

Die Stränge in S_W werden um $X_{i_k}^W L$, die in S_F um $X_{i_k}^F L$ verlängert. Anschließend werden alle zu S vereinigt.

Es entstehen so Stränge doppelter Länge. Jetzt muss der ursprüngliche Teil der Stränge vom neu entstandenen Teil getrennt werden. Dies erfolgt mithilfe zweier Restriktionsenzyme, die die Erkennungssequenzen $L\#X_{i_1}^W$ und $L\#X_{i_1}^F$ haben. Dadurch wird zum einen zwischen altem und neuem Teil geschnitten, zum anderen auch innerhalb des alten Stranges. Es entstehen also aus jedem Strang drei Fragmente. Die neu synthetisierten Stränge werden nun durch eine Gelelektrophorese von den ursprünglichen Strängen, welche in zwei Teile zerfallen und dadurch kürzer sind, separiert und können wie im Brute-Force-Algorithmus in die zweite Phase übergehen. Zwar fehlt den neuen Strängen am $5'$-Ende ein Linker L, doch hat dies im weiteren Verlauf keinerlei Bedeutung. In den Abbildungen 3.4 und 3.5 ist eine schematische Darstellung des Ablaufs dieser Zwischenphase gegeben.

Algorithmus 3.8 (Zwischenphase des Breitensuchealgorithmus)

1. Für $k=1,\ldots,n$:
   - $(S_W, S_F) \leftarrow \text{Separieren}(S, X_{i_k}^W)$
   - $\text{Leeren}(S)$
   - $\text{Verlängern}(S_W, X_{i_k}^W L, |L| + (n+k-1)\cdot\left|X_1^W L\right|, L)$

- Verlängern$(S_F, X_{i_k}^F L, |L| + (n+k-1) \cdot \left|X_1^W L\right|, L)$
- $S \leftarrow S_W,\ S \leftarrow S_F$
- Leeren(S_W, S_F)

2. Schneiden$(S, L\#X_{i_1}^W)$
3. Schneiden$(S, L\#X_{i_1}^F)$
4. $(S_1, S_2) \leftarrow$ Gelelektrophorese$(S, n \cdot \left|X_1^W L\right|)$
5. Leeren(S, S_2)
6. $S \leftarrow S_1$
7. Leeren(S_1)

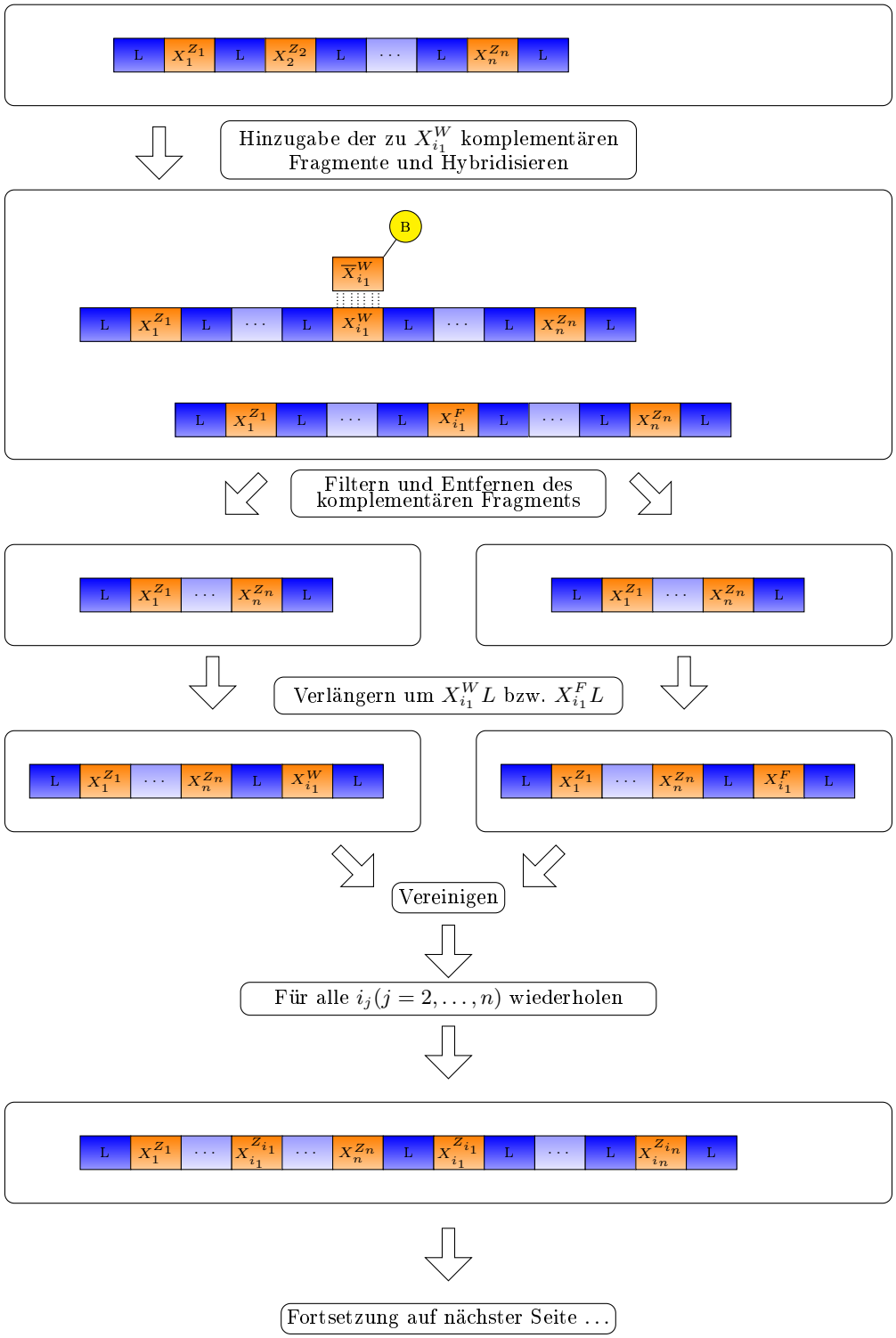


Abbildung 3.4: Schematische Darstellung der Zwischenphase.

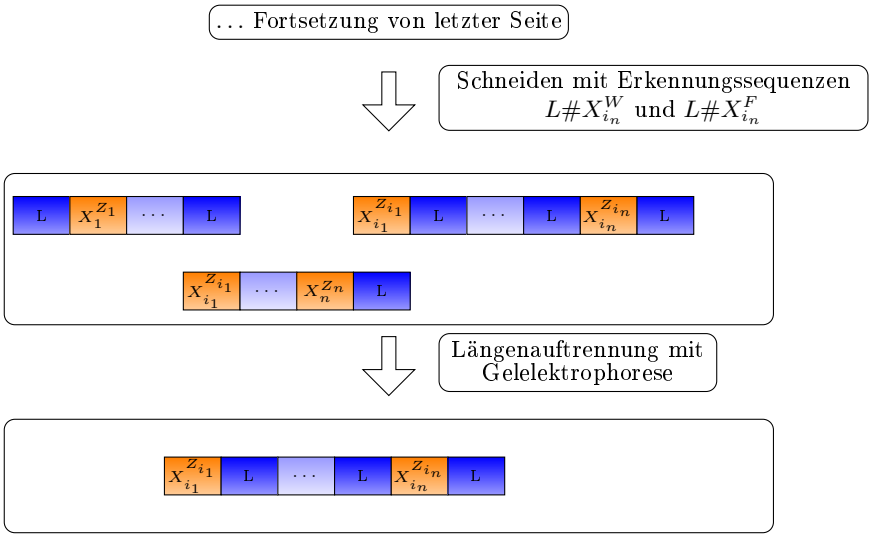


Abbildung 3.5: Schematische Darstellung der Zwischenphase (Fortsetzung).

Verwendung ringförmiger DNA. Statt lineare DNA-Stränge zu verwenden, kann auch mit ringförmiger DNA gearbeitet werden. Diese tritt natürlicherweise als Bakteriengenom oder als Plasmide auf (siehe Abschnitt 2.1.1). Die Arbeit mit Plasmiden zur Lösung von $\mathcal{NP}$-vollständigen Problemen wurde unter anderem von Liu et al. [23] und Head et al. [16, 15] vorgeschlagen. Aus ringförmiger DNA können Teile herausgeschnitten werden, ohne dass dabei der gewünschte Strang zerfällt.

Zunächst müssen die in der ersten Phase erhaltenen einzelsträngigen DNA-Moleküle in Doppelstränge umgewandelt werden, um sie in Plasmide einbauen zu können. Dies geschieht durch die Hinzugabe der Fragmente $\overline{L}$, $\overline{X}_i^W$ und $\overline{X}_i^F$ $(i = 1, \ldots, n)$ in einer großen Anzahl, anschließend erfolgt die Hybridisierung und Ligierung. Alle Doppelstränge der richtigen Länge erhält man nach einer Gelelektrophorese.

Das Plasmid, in welches die Stränge eingebaut werden sollen, muss über eine Erkennungssequenz für ein Restriktionsenzym verfügen, die nur ein einziges Mal in diesem Plasmid und nicht in den einzubauenden DNA-Strängen auftritt. An dieser Stelle wird das Plasmid aufgeschnitten. Um die DNA-Stränge nun in das Plasmid einzubauen, werden ihnen an den Enden sogenannte Adapter angefügt. Dies sind Oligonukleotide, die ebenfalls die Erkennungssequenz für das Restriktionsenzym enthalten. Die Adapter werden in großer Zahl zu den DNA-Strängen gegeben und mit diesen ligiert. Anschließend wird mit dem Restriktionsenzym geschnitten, es entstehen dadurch klebrige Enden, die komplementär zu den Enden der aufgeschnittenen Plasmide sind. Nach Zugabe einer großen Anzahl von Plasmiden werden die Stränge durch eine Ligation in die Plasmide integriert. Nach einer Gelelekrophorese erhält

man die Plasmide der richtigen Größen (siehe auch Abbildung 2.15). In Algorithmus 3.9 ist die Abfolge der Operationen dargestellt, die benötigt wird, um Stränge aus einem Reagenzglas S, die die Länge $|X|$ haben, in Plasmide einzubauen. Hierbei bezeichne Ad die Sequenz des Adapters, $Ad_1\#Ad_2$ sei die Erkennungssequenz innerhalb des Adapters und des Plasmids für das Restriktionsenzym.

Algorithmus 3.9 (Einbau in Plasmide)
Plasmideinbau$(S, |X|)$

1. $S \leftarrow$ Synthese$(\overline{L}, \{\overline{X}_i^W, \overline{X}_i^F \mid i = 1, \ldots, n\})$
2. Hybridisieren(S)
3. Ligieren(S)
4. $(S_1, S_2) \leftarrow$ Gelelektrophorese$(S, |X|)$
5. $S_1 \leftarrow$ Synthese$(Ad, \overline{Ad})$
6. Ligieren(S_1)
7. $S_1 \leftarrow \{\text{Plasmide}\}$
8. Schneiden$(S_1, Ad_1\#Ad_2)$, wobei $Ad = Ad_1Ad_2$
9. Ligieren(S_1)
10. Leeren(S)
11. $(S, S_2) \leftarrow$Gelelektrophorese$(S_1, |X| + |\text{Plasmid}| + |Ad|)$
12. Leeren(S_1, S_2)

Nun können einzelne Teile aus den Strängen herausgeschnitten werden, ohne dass diese zerfallen. Es wird die zusätzliche Anforderung an die beteiligten Restriktionsenzyme mit den Erkennungssequenzen $X_{i_k}^Z\#L$ und $L\#X_{i_k}^Z$, die benötigt werden, um $X_{i_k}^Z$ herauszuschneiden, gestellt, dass dabei zueinander komplementäre klebrige Enden entstehen. Dadurch können nach dem Herausschneiden beide Enden wieder zusammengefügt werden, sodass wieder ein ringförmiges Plasmid vorliegt. Da sicherlich nicht alle Plasmide geschnitten werden, wird erneut eine Gelelektrophorese durchgeführt, um die nun um $\left|X_{i_k}^Z L\right|$ kürzeren Plasmide zu erhalten. Im Falle einer existenzquantifizierten Variable kann nach einem Test auf Leerheit in die nächste Iteration übergegangen werden.

Bei allquantifizierten Variablen ist das Verfahren um einiges komplizierter: Zunächst müssen die Stränge auf zwei Reagenzgläser S_W und S_F aufgeteilt werden,

je nachdem, ob sie $X_{i_k}^W$ oder $X_{i_k}^F$ enthalten. In beiden Reagenzgläsern findet nun die Verdopplung, der Einbau in die Plasmide und das Herausschneiden von $X_{i_k}^W L$ beziehungsweise $X_{i_k}^F L$ statt. In die nächste Iteration dürfen nur Stränge übergehen, die in beiden Reagenzgläsern zu finden sind. Dies kann durch die folgenden Operationen gewährleistet werden: Die kodierenden Stränge werden zunächst wieder aus den Plasmiden herausgeschnitten und in Einzelstränge denaturiert. Aus Reagenzglas S_W werden alle Stränge separiert, die die Sequenzen $X_{i_n}^W$ oder $X_{i_n}^F$ enthalten; aus Reagenzglas S_F werden diejenigen Stränge separiert, die die dazu komplementären Sequenzen $\overline{X}_{i_n}^W$ oder $\overline{X}_{i_n}^F$ enthalten. Es wurde hier aufgrund der Sequenzen $X_{i_n}^Z$ und $\overline{X}_{i_n}^Z$ separiert, weil diese bis zum Schluss vorhanden sind, es könnte auch durch die Wahl der Sequenzen L und $\overline{L}$ separiert werden. Beide separierten Mengen werden zusammengegeben und hybridisiert. Nach einer Einzelstrangspaltung und einer Gelelektrophorese bleiben nun nur diejenigen Stränge übrig, die vollständig hybridisierten und somit in beiden Reagenzgläsern S_W und S_F vorlagen. Diese können wieder in die Plasmide eingebaut werden und (nach einem Leerheitstest) in die nächste Iteration übergehen.

Algorithmus 3.10 (Breitensuchealgorithmus mit Plasmiden)

1. Plasmideinbau$(S, n \cdot \left|X_1^W L\right| + |L|)$
2. Für $k = 1, \ldots, n-1$:
   a) Falls $Q_k = \exists$:
      - Schneiden$(S, X_{i_k}^W \# L)$, Schneiden$(S, L \# X_{i_k}^W)$
      - Schneiden$(S, X_{i_k}^F \# L)$, Schneiden$(S, L \# X_{i_k}^F)$
      - Ligieren(S)
      - $(S_1, S_2) \leftarrow$ Gelelektrophorese$(S, |\text{Plasm.}| + |Ad| + (n-k) \cdot \left|X_1^W L\right| + |L|)$
      - Leeren(S)
      - $S \leftarrow S_1$
      - Leeren(S_1, S_2)
   b) Falls $Q_k = \forall$:
      - Schneiden$(S, Ad_1 \# Ad_2)$
      - Denaturieren(S)
      - $(S_W, S_1) \leftarrow$ Separieren$(S, X_{i_k}^W)$
      - $(S_F, S_2) \leftarrow$ Separieren$(S_1, X_{i_k}^F)$
      - Leeren(S, S_1, S_2)

- Plasmideinbau$(S_W, (n-k+1)\cdot\left|X_1^W L\right|+|L|)$
- Plasmideinbau$(S_F, (n-k+1)\cdot\left|X_1^W L\right|+|L|)$
- Schneiden$(S_W, X_{i_k}^W \# L)$, Schneiden$(S_W, L\# X_{i_k}^W)$
- Schneiden$(S_F, X_{i_k}^F \# L)$, Schneiden$(S_F, L\# X_{i_k}^F)$
- Ligieren(S_W), Ligieren(S_F)
- $(S_1, S_3) \leftarrow$Gelelektrophorese$(S_W, |\text{Pl.}|+|Ad|+(n-k)\cdot\left|X_1^W L\right|+|L|)$
- $(S_2, S_3) \leftarrow$Gelelektrophorese$(S_F, |\text{Pl.}|+|Ad|+(n-k)\cdot\left|X_1^W L\right|+|L|)$
- Leeren(S_W, S_F)
- $S_W \leftarrow S_1$, $S_F \leftarrow S_2$
- Leeren(S_1, S_2, S_3)
- Schneiden$(S_W, A_1 \# A_2)$, Schneiden$(S_F, Ad_1 \# Ad_2)$
- Denaturieren(S_W, S_F)
- $(S, S_2) \leftarrow$ Separieren$(S_W, \{X_{i_n}^W, X_{i_n}^F\})$
- $(S, S_2) \leftarrow$ Separieren$(S_F, \{\overline{X}_{i_n}^W, \overline{X}_{i_n}^F\})$
- Hybridisieren(S)
- Einzelstrangspaltung(S)
- Plasmideinbau$(S, (n-k)\cdot\left|X_1^W\right|+|L|)$

c) Falls S leer: Ausgabe *F ist falsch*

3. Falls $Q_n = \forall$:
   - Schneiden$(S, Ad_1 \# Ad_2)$
   - Denaturieren(S)
   - $(S_W, S_1) \leftarrow$ Separieren$(S, X_{i_k}^W)$
   - $(S_F, S_2) \leftarrow$ Separieren$(S_1, X_{i_k}^F)$
   - Falls S_W oder S_F leer: Ausgabe *F ist falsch*
4. Ausgabe *F ist wahr*

Folgen zwei Allquantoren aufeinander, so kann auf den zwischenzeitlichen Einbau in Plasmide verzichtet werden, das gleiche gilt, falls der erste Quantor ein Allquantor ist.

3.3.5 Laufzeit und Anzahl benötigter Stränge

Die Laufzeit hängt wie im Brute-Force-Algorithmus linear von der Anzahl der Variablen n und der Klauseln m ab.

Im Gegensatz zum Brute-Force-Algorithmus wird nicht mehr der gesamte Raum der Belegungen durchsucht, die Zahl der benötigten Stränge ist also kleiner als $O(2^n)$. Sie ist nur abhängig von der ersten Phase, da in der zweiten Phase keine Stränge erzeugt werden, im Gegenteil, die Zahl der Stränge wird ständig verkleinert. Deshalb gelten die Ergebnisse, die Yoshida und Suyama in [41] vorstellten. Das Ordnen der Variablen nach der Häufigkeit ihres Auftretens vor Beginn des Algorithmus hat zum Ziel, möglichst viele der Klauseln zu einem frühen Zeitpunkt zu betrachten. Dadurch können mehr Stränge frühzeitig verworfen werden, was ein starkes Zunehmen der DNA-Menge verhindert.

Yoshida und Suyama führten eine Simulation ihres Algorithmus durch, um die Anzahl der benötigten DNA-Stränge im Durchschnitts- und im schlechtesten Fall zu bestimmen. Sie betrachteten zufällig gewählte Instanzen des 3SAT-Problems mit n zwischen 20 und 31. Sie wählten ein Verhältnis von m/n zwischen 4,3 und 4,7 (in Schritten von 0,05), da dies den in der Praxis schwierigsten Instanzen entsprechen würde. Für jedes n und jedes m wurden 2000 Instanzen betrachtet. Als Ergebnis erhielten sie im schlechtesten Fall einen Bedarf an Strängen von $2^{0{,}58n}$, im Durchschnittsfall von $2^{0{,}36n}$. Falls diese Schranken auch für größere Variablenzahlen gelten würden, so vermuten sie, dass ihr Algorithmus auch für große Probleme praktikabel wäre, für $n = 100$ würde beispielsweise die Masse der benötigten DNA gegenüber dem Brute-Force-Algorithmus von mehreren Tonnen auf wenige Milligramm sinken: Die Rechnung von Hartmanis (als untere Schranke) [13] unter Berücksichtigung, dass jeder Strang mit rund 1000 Kopien vorliegt, ergibt

$$2^{100} \cdot \log_4 100 \cdot 1000 \cdot 10^{-25} \text{ kg} \approx 400\,000 \text{ t} \qquad \text{(Brute-Force-Algorithmus)}$$

$$2^{0{,}58 \cdot 100} \cdot \log_4 100 \cdot 1000 \cdot 10^{-25} \text{ kg} \approx 100 \text{ mg} \qquad \text{(Breitensuche schlechtester Fall)}$$

$$2^{0{,}36 \cdot 100} \cdot \log_4 100 \cdot 1000 \cdot 10^{-25} \text{ kg} \approx 20 \text{ ng} \qquad \text{(Breitensuche im Durchschnitt)}$$

Wang et al. schlagen in [36] ein ähnliches Verfahren zur Lösung des SAT-Problems vor und führen ebenfalls Simulationen durch mit einer Variablenzahl n zwischen 5 und 50, einem Verhältnis von Klauseln zu Variabeln m/n von 1 bis 50 und 1000 getesteten Instanzen für jedes n. Sie kommen zu dem Ergebnis, dass der Faktor im Exponenten sowohl im schlechtesten als auch im Durchschnittsfall mit steigendem n sinkt bis auf 0,48 im schlechtesten und 0,42 im Durchschnittsfall für $n = 50$. Wie zu erwarten war, ist der Durchschnittswert bei kleinerem Quotienten m/n, also wenigen Klauseln im Verhältnis zu der Anzahl der Variablen, höher als bei größerem Quotienten, da mehr Belegungen als nicht erfüllend verworfen werden können. Es wird in [36] angenommen, dass diese maximale Stranganzahl von $2^{0{,}48n}$ auch für größere Variablenzahlen gilt oder weiter abnimmt.

Diese Werte wurden für das unquantifizierte Problem ermittelt. Sie sind auf den quantifizierten Fall übertragbar, weil die Anzahl der Stränge nur von der ersten Phase abhängt, in der die Quantifizierungen nicht betrachtet werden. Allerdings müssen, damit eine quantifizierte Formel *wahr* sein kann, mindestens $2^{|\forall|}$ Stränge nach der ersten Phase vorhanden sein, hierbei stehe $|\forall|$ für die Anzahl der Allquantoren der betrachteten Formel. Eine hohe Anzahl an Allquantoren bedeutet damit einen hohen Bedarf an Strängen, wenn sich die Formel als *wahr* herausstellt. Deshalb wird in diesem Fall das Verhältnis m/n deutlich unter den in den gerade beschriebenen Simulationen liegen müssen.

Natürlich sind dies nur Aussagen über die benötigte Masse, die Auswirkung von Fehlern und Ungenauigkeiten, die zwangsläufig im Labor auftreten, haben sicherlich einen großen Einfluss auf die praktische Durchführbarkeit. Deren Auswirkung soll im nächsten Kapitel diskutiert werden.

4 Diskussion der praktischen Umsetzbarkeit

4.1 Wahl der Stränge

Bei der Wahl der DNA-Sequenzen für die Kodierungen der Literale ist darauf zu achten, dass es zu möglichst wenigen Hybridisierungen zwischen den einzelnen Strängen und innerhalb der Stränge kommen kann, da dies zu falschen Berechnungen führen würde. Die Stränge sollten auch einen einheitlichen Schmelzpunkt haben, um frühzeitiges oder unvollständiges Denaturieren einzelner Stränge zu vermeiden. Braich et al. [3, 2] schlagen eine Reihe von Maßnahmen vor, um dies zu verhindern. Die Stränge sollten nur die Basen A, T und C enthalten, um dem Auftreten von unerwünschten Sekundärstrukturen entgegenzutreten. Es sollten keine Folgen von mehr als vier gleichen Basen nacheinander auftreten, da dies sowohl zur Entstehung ungewöhnlicher Sekundärstrukturen als auch zu einer Veränderung des Schmelzpunktes führen kann. Die Länge der komplementären Sequenzen zwischen jeweils zwei Strängen wird begrenzt, ebenso zwischen der komplementären Sequenz eines Stranges und allen anderen Strängen. Auch auf die Möglichkeit, dass ein Strang zu sich selbst komplementär ist, muss geachtet werden. Die Überprüfung aller dieser Kriterien erfordert quadratische Zeit in der Größe der Variablenanzahl.

Die Länge der Sequenzen ist natürlich abhängig von der Variablenzahl. Braich et al. [3] und Yoshida und Suyama [41] wählen eine Länge von 15 Basen pro Literal für eine Problemgröße von $n = 20$ beziehungsweise $n = 4$. Die Strangsequenzen müssen nicht für jede Berechnung neu festgelegt werden, es reicht vielmehr, dies bei Festlegung einer maximalen Variablenzahl ein einziges Mal zu tun.

Bei der Wahl der Sequenzen ist zusätzlich zu beachten, dass es jeweils zwischen dem Ende der Sequenz und der Linkersequenz eine Schnittstelle für ein Restriktionsenzym geben muss. Diese Schnittstelle muss für jede Sequenz einzigartig sein, darf also nicht in mehreren Sequenzen auftreten. Eine ausführlichere Diskussion der Einschränkungen durch die Wahl der Restriktionsenzyme findet sich in Abschnitt 4.2.1.

Nachdem die Sequenzen für die X_i^W und X_i^F nach diesen Regeln ausgewählt wurden, werden sie mithilfe von automatisierten Syntheseverfahren hergestellt.

Bei dem in Abschnitt 3.2 beschriebenen Brute-Force-Verfahren ist es notwendig, zu Beginn alle 2^n Kodierungen für alle möglichen Belegungen der Variablen als DNA-

Stränge zu synthetisieren. Diese Synthese erfolgt schrittweise: Zu Beginn werden die Stränge $LX_n^W L$ und $LX_n^F L$ synthetisiert. Beide Mengen werden zusammengegeben und nach einer Durchmischung wieder in zwei Mengen geteilt. In der ersten Menge wird die Synthese mit LX_{n-1}^W fortgesetzt, in der zweiten Menge mit LX_{n-1}^F. Wiederum werden beide Mengen vereinigt, gemischt und in zwei Mengen geteilt. Dieser Prozess wird bis zur ersten Variable fortgesetzt. Zum Schluss enthält man ein Reagenzglas mit allen 2^n Arten von Strängen $LX_1^{Z_1} X_2^{Z_2} \ldots X_{n-1}^{Z_{n-1}} X_n^{Z_n}$.

Jeder Strang wird in einer Vielzahl von Kopien bereitgestellt, weshalb der Verlust eines Teils der Stränge bei Durchführung der Algorithmen vertretbar ist. So beginnen Braich et al. beispielsweise mit einer DNA-Menge von 500 pmol (Picomol), was $3{\cdot}10^{14}$ Strängen entspricht. Da ihr Ausgangspool 2^{20} verschiedene Stränge umfasst, liegt jeder Strang in ungefähr 100 Millionen Kopien vor. Yoshida und Suyama beginnen sogar mit einer Menge von 25 pmol für jeden der vier verschiedenen Stränge, mit denen gestartet wird, dies entspricht $15 \cdot 10^{12}$ Kopien pro Strang.

4.2 Probleme bei molekularbiologischem Arbeiten

Beim Arbeiten im Labor ist es nicht zu vermeiden, dass Fehler auftreten. Einige Fehlerquellen wurden bei der Vorstellung der molekularbiologischen Methoden in Abschnitt 2.1.3 bereits angesprochen. Auch durch sorgfältiges Arbeiten kann nicht verhindert werden, dass ein Teil der Stränge verloren geht. Ein kleiner Teil der DNA-Lösung bleibt beispielsweise in Pipettenspitzen zurück oder kann nicht aus dem Gel der Elektrophorese zurückgewonnen werden. Ein weiteres Problem entsteht durch die Arbeit mit vielen unterschiedlichen Enzymen, da diese teilweise starke Unterschiede in den benötigten Reaktionsbedingungen wie zum Beispiel Temperatur und pH-Wert aufweisen. Daher müssen für jedes Enzym die Bedingungen angepasst werden und die Proben anschließend von den dafür verwendeten Pufferlösungen gereinigt werden, wodurch es zum Verlust oder zur Zerstörung von einzelnen Strängen kommen kann.

Keines der verwendeten Enzyme arbeitet fehlerfrei. Zur Verringerung eines Teils der Fehler sind in den vorgestellten Algorithmen Kontrollen eingebaut worden: Um unerwünschte Ligationen zu vermeiden, findet nach Anwendung dieser Operation immer eine Überprüfung der Stranglänge mittels Gelelektrophorese statt; nur mit Strängen der gewünschten Länge wird weitergearbeitet. Dies findet auch nach dem Schneiden mit Restriktionsenzymen statt, ungeschnittene oder zu oft geschnittene Stränge können so aussortiert werden. Die fehleranfälligste Operation ist die Separation markierter Stränge, in Abschnitt 4.3 wird beschrieben, wie diese Operation verbessert werden kann.

Durch eine stetige Verbesserung und Automatisierung der Methoden ist es in Zukunft vielleicht möglich, einige der auftretenden Fehler zu verringern. Neben diesen allgemeinen Problemen bei der Arbeit im Labor treten weitere Probleme auf, die im

Folgenden beschrieben werden.

4.2.1 Verfügbarkeit geeigneter Restriktionsenzyme

In den vorgestellten Verfahren wird davon ausgegangen, dass es Restriktionsenzyme mit frei zu definierender Erkennungssequenz gibt. Dies ist in der Praxis nicht der Fall. Es sind zwar rund 4000 Restriktionsenzyme bekannt, von diesen sind auch mehrere hundert kommerziell erhältlich, die meisten besitzen aber relativ kleine Erkennungssequenzen von vier bis acht Basenpaaren und viele dieser Enzyme haben gleiche Erkennungssequenzen. Daher gibt es eine starke Einschränkung in der Verfügbarkeit von Restriktionsenzymen mit einer vorgegebenen Erkennungssequenz.

In den letzten Jahren gab es einige Versuche, Nukleasen künstlich herzustellen, welche an frei wählbaren Erkennungssequenzen schneiden [9, 39, 24]. Diese haben aber bis jetzt eine sehr viel kleinere Effizienz als die natürlichen Restriktionsenzyme und sind deshalb noch nicht einsetzbar.

Aus diesem Grund ist die Operation des Schneidens die stärkste Abstraktion von den molekularbiologisch verfügbaren Methoden, die in den vorgestellten Algorithmen auftritt. In diesen werden $2 \cdot n$ verschiedene Restriktionsenzyme benötigt. Solange nicht Restriktionsenzyme zur Verfügung stehen, die größere Erkennungssequenzen haben, wird eine Umsetzung mit einer großen Anzahl von Variablen wohl nicht möglich sein.

4.2.2 Zeitaufwand

Die benötigte Anzahl der Operationen ist zwar linear, bezogen auf die Anzahl der Variablen und der Klauseln; das Ausführen vieler dieser Operationen ist jedoch sehr zeitaufwendig. So benötigten Braich et al. [3] über zwei Wochen Laborarbeit, um eine Instanz des SAT-Problems mit zwanzig Variablen zu lösen. Ein Ausweg wäre die Automatisierung dieser Operationen. Für das SAT-Problem wurde dies durch Johnson im Jahr 2007 [18] getan: Durch einen automatisierten DNA-Computer konnte ein Problem mit zehn Variablen in 28 Stunden gelöst werden. Für ein Problem mit zwanzig Variablen wären circa zwei Tage notwendig, dies ist eine deutliche Beschleunigung gegenüber der manuellen Durchführung. Durch weitere Verbesserungen der Automatisierung, so nimmt Johnson an, wäre die Lösung einer Instanz mit zwanzig Variablen in zwei Stunden möglich [18].

4.2.3 Größe der Stränge

Bei steigender Variablenzahl nimmt die Länge der zu synthetisierenden Stränge zu, zumal auch die Kodierung jeder der Variablen länger werden muss. Längere Stränge sind aber empfindlicher beim Umgang im Labor, es kann leichter zu einem Bruch kommen. Zwar liegen DNA-Moleküle in der Natur oft in großen Längen von mehreren

hundert Millionen Basenpaaren vor, allerdings sind diese zum einen doppelsträngig, sodass Brüche, die in einem der Stränge auftreten, repariert werden können, zum anderen sind diese Moleküle stark kondensiert – die menschliche DNA hat beispielsweise eine Länge von rund zwei Metern und findet in einem Zellkern von wenigen Mikrometern Durchmesser Platz – und werden durch Hilfsproteine, zum Beispiel Histone, gestützt und geschützt. Bei den vorgestellten Algorithmen wird hingegen die meiste Zeit mit einzelsträngiger DNA gearbeitet. Bei dieser können außerdem ungewöhnliche Sekundärstrukturen entstehen, wenn viele Wasserstoffbrücken zwischen verschiedenen Teilen der Stränge ausgebildet werden. Dadurch kann es beispielsweise zu Problemen bei der Operation der Hybridisierung kommen. Insbesondere bei dem vorgestellten Verfahren zur Wiederherstellung der ursprünglichen Reihenfolge der Variablen in Algorithmus 3.8 kann die Größe der Stränge zum Problem werden, da hier zwischenzeitlich Stränge mit doppelter Länge vorliegen.

Die Arbeit mit einzelsträngiger DNA ist aber Voraussetzung, um in der zweiten Phase der Algorithmen durch Nutzung der komplementären Basenpaarung die Stränge zu finden, die in beiden Reagenzgläsern vorliegen. Deshalb ist eine Umgehung dieses Problems nur schwer möglich.

4.3 Verringerung des Fehlers bei der Separation

Die Operation des Herausfilterns von markierten DNA-Strängen, die auf der starken Affinität zwischen Avidin und Biotin beruht, ist relativ fehleranfällig. So können auf der einen Seite Stränge separiert werden, die die Teilsequenz nicht enthalten (falsch positive Auswahl), zum anderen können Stränge, die die Sequenz enthalten, nicht separiert werden (falsch negative Auswahl). Diese Fehler werden desto größer, je mehr Separationsschritte durchgeführt werden. Selbst wenn die Wahrscheinlichkeit, dass ein Strang richtig separiert wird, bei 0,99 liegt, sinkt die Wahrscheinlichkeit auf $0{,}99^{100} = 0.37$, wenn 100 Separationen nacheinander durchgeführt werden.

Im Brute-Force-Algorithmus werden $3 \cdot m$ Separationen bei m Klauseln durchgeführt, im Breitensuche-Algorithmus $2 \cdot m + 2 \cdot n$ Separationen. Eine Anzahl von 100 Separationen ist also schon bei einer mittleren Anzahl von Variablen und Klauseln durchzuführen.

Chen und Winfree [7] schlagen zur Verkleinerung dieses Fehlers die wiederholte Durchführung der Separation nach folgendem Schema vor: Es werden für jede Separation nacheinander k Separationen durchgeführt, am Ende werden alle Stränge, die mehr als $k/2$ mal ausgewählt worden, in das Reagenzglas für die Stränge gegeben, die die Teilsequenz enthalten. Alle Stränge, die weniger als $k/2$ mal ausgewählt wurden, werden in das Reagenzglas für die Stränge ohne die gesuchte Teilsequenz gegeben. Abbildung 4.1 zeigt das Verfahren für vier Wiederholungen. Für gerade k können alle Stränge, die genau $k/2$ mal ausgewählt wurden, sowohl dem ersten als auch dem

zweiten Reagenzglas zugeordnet werden.

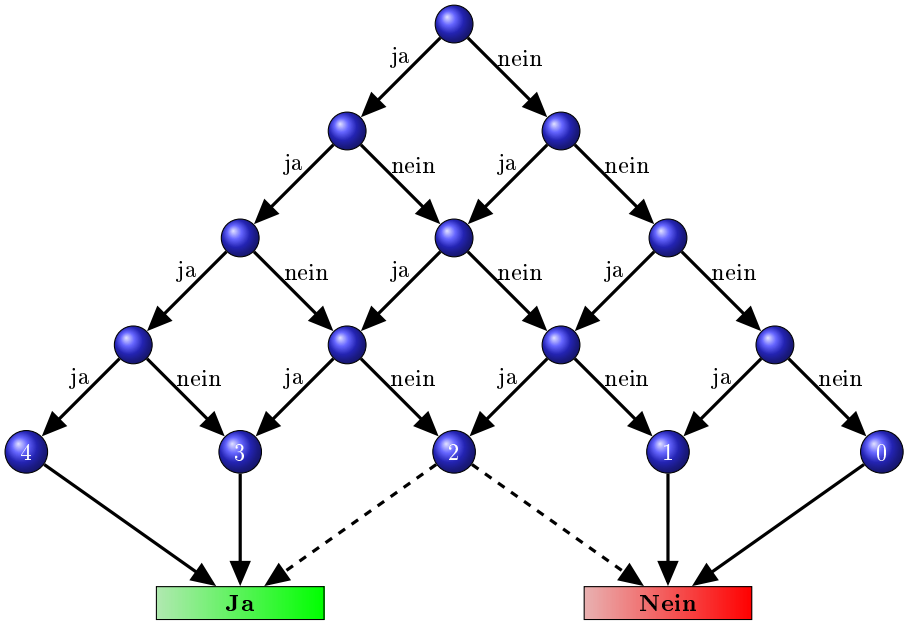


Abbildung 4.1: Schema der mehrfachen Separation für $k = 4$.

Einige Separationen sind hierbei überflüssig, wenn ein Strang schon mehr als $k/2$ mal ausgewählt wurde oder gar nicht mehr so oft ausgewählt werden kann. Als Schema ergibt sich die in Abbildung 4.2 dargestellte Raute.

In Abhängigkeit von der Anzahl der Wiederholungen k verringert sich der auftretende Fehler auf

$$P = \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{k}{i} p^i (1-p)^{k-i},$$

wobei p der Fehler bei einer einzigen Separation ist. Mithilfe der Stirlingschen Ungleichung und einer Reihenentwicklung ergibt sich eine obere Grenze von

$$P < \frac{(4p(1-p))^{k/2}}{1-2p},$$

wobei angenommen wurde, dass $p > 0{,}5$ ist. [7]

Bereits für $k = 4$ ergibt sich eine Verbesserung von $p = 0{,}01$ auf $P < 0{,}002$ und somit eine Wahrscheinlichkeit von $(1-0{,}002)^{100} = 0{,}82$, dass ein Strang nach hundert durchgeführten Separationsprozessen richtig zugeordnet wurde.

Das Herausfiltern von Strängen, die mit dem Molekül Acrydite markiert wurden, erfolgt mithilfe der Gelelektrophorese und ist weniger fehleranfällig. Markierte Strän-

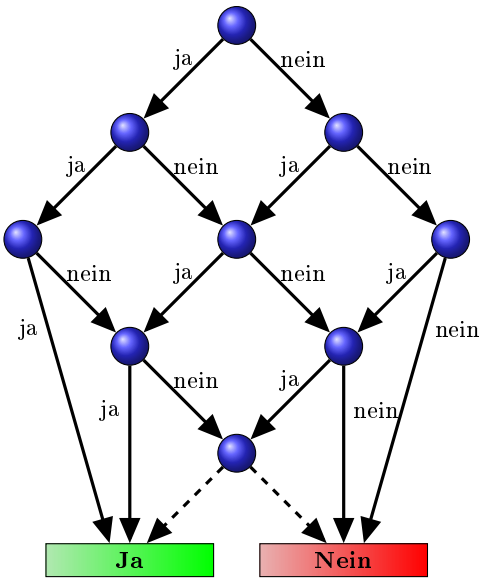


Abbildung 4.2: Vereinfachtes Schema der mehrfachen Separation für $k = 4$.

ge können nicht durch das Gel wandern und verbleiben dadurch an der aufgetragenen Stelle. Probleme können hierbei vor allem bei der Elution aus dem Gel auftreten, da einige Stränge zurückbleiben können. Außerdem ist es schwierig, die nichtmarkierten Stränge, die durch das Gel wandern, zurückzugewinnen. Da dies an einigen Stellen der vorgestellten Algorithmen aber notwendig ist (im Gegensatz zum Algorithmus von Braich et al. [3], die diese Art der Markierung verwenden), ist die Verwendung der Biotin-Avidin-Separation trotz der Fehleranfälligkeit vorzuziehen. Außerdem ist es in den vorgestellten Algorithmen auch erforderlich, markierte Stränge nach ihrer Größe aufzutrennen, dies ist bei mit Acrydite markierten Strängen nicht möglich.

4.4 Bedarf an DNA-Strängen

Während die bisher diskutierten Grenzen der molekularbiologischen Verfahren eventuell durch deren Verbesserung zukünftig aufgehoben oder zumindest verschoben werden könnten, stellt der DNA-Bedarf eine prinzipielle Beschränkung der Umsetzbarkeit dar. Wie bereits im vorherigen Kapitel ausgeführt, benötigen sowohl der Brute-Force-Algorithmus als auch der auf Breitensuche basierende Algorithmus eine exponentielle Anzahl von DNA-Strängen. Dies wird bei allen Algorithmen der Fall sein, die auf dem Durchsuchen des Raumes der Belegungen basieren, da im Fall, dass die gegebene Quantifizierte Formel wahr ist, mindestens $2^{|\forall|}$ verschiedene DNA-Stränge benötigt werden. Hierbei ist $|\forall|$ die Anzahl der Allquantoren der Formel.

Die Anzahl der Allquantoren könnte damit ein entscheidendes Kriterium für die Verifizierung einer quantifizierten Formel als *falsch* sein. Liegt die Anzahl der Stränge, die für unterschiedliche Belegungen kodieren, nach der ersten Phase unter $2^{|\forall|}$, so ist die Formel *falsch*. Leider sind die molekularbiologischen Methoden zur Bestimmung der Konzentration relativ ungenau. Aus dieser Konzentration ließe sich die Masse der vorliegenden DNA bestimmen. Ein Nukleotid hat eine Masse von rund $5 \cdot 10^{-25}$ kg, dies ergibt sich aus dem Molekulargewicht von 330 g/mol und einer Teilchenzahl von $6{,}022 \cdot 10^{23}$ pro Mol [25]. Es ließe sich also die Anzahl der Nukleotide und damit die Anzahl an Strängen bestimmen, da die Länge der Stränge bekannt ist. Es ist allerdings nicht klar, wieviele gleichartige Stränge nach dem Durchlaufen der ersten Phase noch vorhanden sind, sodass die Anzahl verschiedenartiger Stränge nur geschätzt werden kann. Durch diese Unsicherheiten ist es deshalb nur schwer möglich, die Stranganzahl als entscheidendes Abbruchkriterium einzusetzen.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurden Möglichkeiten vorgestellt, wie das $\mathcal{PSPACE}$-vollständige Problem der Quantifizierten Booleschen Formeln mithilfe des DNA-Computings in polynomieller Zeit gelöst werden könnte. Für den aus der Literatur bekannten Algorithmus von Dantsin und Wolpert, der auf einer Brute-Force-Strategie beruht, wurde eine mögliche Implementation angegeben. Basierend auf einem Breitensuche-Algorithmus für das SAT-Problem von Yoshida und Suyama wurden ein Algorithmus und zwei mögliche Implementationen erarbeitet, die für eine Reduzierung der benötigten Menge an DNA gegenüber dem Brute-Force-Ansatz sorgen. Dadurch könnte es möglich werden, Probleminstanzen mit einer höheren Variablenzahl zu lösen. Dieser Vorteil wird zwar durch eine größere Zahl an durchzuführenden Operationen erkauft, diese ist aber immer noch linear in Bezug auf die Anzahl der Variablen und Klauseln. Es wäre interessant, zu sehen, ob sich diese Implementationen auch praktisch umsetzen ließen. Insbesondere die praktische Durchführung der zweiten Phase mit der Ausnutzung der Komplementarität der DNA-Stränge durch die Operation der Hybridisierung wäre eine Neuerung, nachdem die prinzipielle Durchführbarkeit der jeweiligen ersten Phase durch die Versuche von Braich [3] und Yoshida und Suyama [41] zumindest für kleine Problemgrößen gezeigt werden konnte.

Beim Vergleich zwischen den Varianten des Breitensuche-Algorithmus liegt der Nachteil des Verfahrens zur Änderung der Variablenreihenfolge in der zeitweiligen Erzeugung von Strängen doppelter Länge, was eine potenzielle Fehlerquelle darstellt. Von Nachteil bei der Verwendung der Plasmide könnte das häufige Einfügen und Herausschneiden der DNA-Sequenzen sein, das für jede allquantifizierte Variable stattfinden muss. Dadurch kann es zum Verlust eines Teils der Stränge und dadurch zur Verfälschung des Ergebnisses kommen. Hier könnte versucht werden, Verfahren zu entwickeln, die diese Fehler verkleinern.

Die vorgestellten Algorithmen beschränken sich zwar auf das Problem 3QBF, ihre Erweiterung auf eine beliebige Anzahl von Literalen pro Klausel wäre aber relativ problemlos möglich. Es könnte auch versucht werden, als Grundlage zur Lösung des SAT-Problems andere Algorithmen – bereits vorhandene oder noch zu entwickelnde – zu nutzen, welche die Menge an benötigter DNA weiter reduzieren.

Zwei Grenzen für die praktische Umsetzung werden aber deutlich: Zum einen auftretende Fehler und Einschränkungen der verfügbaren molekularbiologischen Methoden. Diese könnten durch Weiterentwicklung und Automatisierung in der Zukunft verringert werden. Zum anderen prinzipielle Grenzen durch die Verlagerung

des exponentiellen Bedarfs von der Zeit bei herkömmlichen Lösungsmethoden auf den Raum, also die benötigte Menge an DNA. Dieses Problem haben die vorgestellten Algorithmen gemein mit den zahlreichen aus der Literatur bekannten Möglichkeiten zur Lösung $\mathcal{NP}$-vollständiger Probleme. Es stellt sich somit die Frage, ob DNA-Computing überhaupt zur Lösung $\mathcal{NP}$- und $\mathcal{PSPACE}$-vollständiger Probleme geeignet ist und jemals die Leistung herkömmlicher Computer hinsichtlich dieser Probleme übertreffen kann. Doch auch wenn dies nicht möglich sein sollte, so könnte es andere Anwendungsbereiche für das DNA-Computing geben, in denen herkömmliche Computer nicht eingesetzt werden können. Denkbar wäre beispielsweise der Einsatz *in vivo*, also im lebenden Organismus, zur Reparatur geschädigter Zellen, der in der Zukunft möglich sein könnte. Durch das Zusammenwirken von Informatik und Molekularbiologie kann es in beiden Bereichen Impulse zu neuen Ideen geben; auf der einen Seite die Entwicklung neuer Berechnungsmodelle und algorithmischer Strategien, auf der anderen Seite tiefere Einblicke in die Funktionsweisen biochemischer und biologischer Prozesse und Möglichkeiten, in diese gezielt einzugreifen. Das *Biological Computing* mit dem Teilgebiet DNA-Computing entwickelt sich rasant in viele verschiedene Richtungen, sodass, was heute noch als nicht durchführbar gilt, in der Zukunft Realität werden könnte.

Glossar

In diesem Glossar sollen einige der in der Arbeit am häufigsten auftauchenden biologischen Begriffe kurz erklärt werden.

3′-Ende/ 5′-Ende

DNA-Stränge besitzen unterschiedliche Enden, je nachdem, welches der beiden Kohlenstoffatome, über die Nukleotide verbunden sind, frei bleibt. Am freien 3′-Ende befindet sich eine Hydroxylgruppe (-OH), am freien 5′-Ende eine Phosphatgruppe. Durch diese unterschiedlichen Enden haben die Stränge eine Orientierung. → S. 30

Biotin

Molekül, mit dessen Hilfe DNA am 5′-Ende markiert werden kann. Durch die starke Affinität (eine der stärksten nichtkovalenten Bindungen) zu dem Molekül Streptavidin können markierte Stränge von unmarkierten getrennt werden. → S. 39

Denaturieren

Aufbrechen von Wasserstoffbrücken durch Hitze, aus einem doppelsträngigen DNA-Molekül entstehen dadurch zwei Einzelstränge. → S. 37
Bei Proteinen bezeichnet Denaturierung die Zerstörung der Sekundärstruktur durch Hitze oder chemische Reagenzien, wodurch das Protein inaktiviert wird. → S. 33

Doppelhelix

Sekundärstruktur, in der die DNA natürlicherweise vorliegt. Zwei DNA-Stränge mit komplementärer Basenpaarung sind über Wasserstoffbrücken zwischen ihren Basen verbunden, Zucker und Phosphat bilden das Außengerüst. Die Helix ist um die eigenen Achse gewunden; die Stränge liegen antiparallel vor, d.h. das 3'-Ende des einen Stranges liegt gegenüber dem 5'-Ende des anderen Stranges und umgekehrt. → S. 32

Gelelektrophorese
Molekularbiologische Technik zur Auftrennung von DNA-Strängen nach ihrer Größe. Durch Anlegen einer Spannung kann die negativ geladene DNA durch ein Gel wandern; je kleiner die Stränge sind, desto größer ist die Geschwindigkeit. → S. 38

Hybridisieren
Knüpfen von Wasserstoffbrücken zwischen komplementären DNA-Strängen, welches durch langsames Abkühlen einer denaturierten DNA-Lösung entsteht. → S. 37

Komplementäre Basenpaarung
Zwischen jeweils zwei Basen können Wasserstoffbrücken ausgebildet werden: Zwischen Adenin und Thymin entstehen zwei, zwischen Guanin und Cytosin drei Wasserstoffbrücken. Zwei komplementäre DNA-Stränge können sich dadurch zu einer Doppelhelixstruktur verbinden. → S. 32

Kovalente Bindung
Starke chemische Bindung zwischen zwei Atomen, sie entsteht durch Ausbildung gemeinsamer Paare von Valenzelektronen (Elektronen der äußeren Schale).

Ligase
Enzym zum Knüpfen von Phosphodiesterbindungen zwischen zwei Nukleotiden; sie kann Brüche in Doppelsträngen reparieren oder zwei Stränge verbinden. → S. 40

Nuklease
Enzym, welches durch Aufbrechen der Phosphodiesterbindung zwischen Nukleotiden DNA-Stränge spalten kann. Man unterscheidet Exo- und Endonukleasen, je nachdem, ob die DNA vom Ende her oder innerhalb eines Stranges gespalten wird, sowie zwischen einzelstrang- und doppelstrangspaltenden Nukleasen. Eine in der Molekularbiologie häufig verwendete einzelstrangspaltende Nuklease ist die aus einem Pilz gewonnene S1-Nuklease. Von großer Bedeutung sind auch Restriktionsendonukleasen. → S. 35

Nukleotid
Baustein der DNA, die genaue Bezeichnung ist 2'-Desoxynukleotid; es besteht aus einem Zucker, einem Phosphatrest und einer von vier möglichen Basen (Adenin, Cytosin, Guanin oder Thymin). → S. 30

Oligonukleotid
Kurzer DNA-Strang, der aus bis zu 100 Nukleotiden zusammengesetzt ist; *oligo* griechisch für *wenige*.

Phosphodiesterbindung
Bindung zwischen zwei Nukleotiden, sie dient dem Aufbau eines DNA-Stranges und wird durch die Enzyme Ligase oder Polymerase geknüpft. → S. 31

Plasmid
Ringförmiges DNA-Molekül, dass in vielen Bakterienarten zusätzlich zur chromosomalen DNA auftritt. Plasmide werden in der Gentechnologie zum Transfer von gewünschten Genen in Organismen benutzt. → S. 32

Polymerase
Enzym zur Synthese von DNA; sie kann an einem DNA-Einzelstrang den komplementären Strang synthetisieren. → S. 40

Polynukleotid
Linearer DNA-Strang, der aus vielen Nukleotiden zusammengesetzt ist; *poly* griechisch für *viele*.

Restriktionsenzym
Spezielle Art von Nukleasen, die innerhalb eines DNA-Doppelstranges an einer definierten Stelle schneiden können. Es sind mehrere Tausend dieser Enzyme bekannt, sie werden aus vielen Bakterienarten isoliert und haben meist Erkennungssequenzen mit einer Länge von vier, sechs oder acht Basenpaaren. → S. 35

Wasserstoffbrücke
Schwache elektrostatische Bindung, entsteht zwischen einem elektronegativen Atom wie Sauerstoff oder Stickstoff und einem partiell positiv geladenen Wasserstoffatom. Wasserstoffbrückenbindungen ist rund fünfmal schwächer als kovalente Bindungen, haben aber große Bedeutung für die Entstehung der dreidimensionalen Strukturen von Makromolekülen wie Nukleinsäuren und Proteinen. → S. 32

Literaturverzeichnis

[1] Adleman, Leonard M.: Molecular Computation of Solutions to Combinatorial Problems. In: *Science* 266 (1994), Nr. 5187, S. 1021–1024

[2] Braich, Ravinderijt S. ; Johnson, Cliff ; Rothemund, Paul W. K. ; Hwang, Darryl ; Chelyapov, Nickolas ; Adleman, Leonard M.: Solution of a Satisfiability Problem on a Gel-Based DNA Computer. In: *LECTURE NOTES IN COMPUTER SCIENCE* 2054 (2000), S. 27–42

[3] Braich, Ravinderjit S. ; Chelyapov, Nickolas ; Johnson, Cliff ; Rothemund, Paul W. K. ; Adleman, Leonard: Solution of a 20-Variable 3-SAT Problem on a DNA Computer. In: *Science* 296 (2002), S. 499–502

[4] Brown, Terence A.: *Moderne Genetik.* 2. Auflage. Spektrum Akademischer Verlag, 1999

[5] Brown, Terence A.: *Gentechnologie für Einsteiger.* 5. Auflage. Elsevier Spektrum Akademischer Verlag, 2007

[6] Chen, Kevin ; Ramachandran, Vijay: A space-efficient randomized DNA algorithm for k-SAT. In: Anne Condon, Grzegorz R. (Hrsg.): *DNA Computing, 6th International Workshop on DNA-based Computer* Bd. 2054, Springer, 2000 (Lecture Notes in Computer Science), S. 199–208

[7] Chen, Kevin ; Winfree, Erik: Error Correction in DNA Computing: Misclassification and Strand Loss. In: Winfree, Erik (Hrsg.) ; Gifford, David K. (Hrsg.): *DNA Based Computers V*, American Mathematical Society, 2000 (DIMACS), S. 49–63

[8] Christen, Phillip ; Jaussi, Rolf: *Biochemie.* Springer, 2005

[9] Eisenschmidt, Kristin ; Lanio, Thomas ; Simoncsits, András ; Jeltsch, Albert ; Pingoud, Vera ; Wende, Wolfgang ; Pingoud, Alfred: Developing a programmed restriction endonuclease for highly specific DNA cleavage. In: *Nucleic Acids Research* 33 (2005), Nr. 22, S. 7039–7047

[10] Eng, Tony L.: On solving 3CNF-satisfiability with an in vivo algorithm. In: *BioSystems* 52 (1999), S. 135–141

[11] Feynman, Richard P.: There's Plenty of Room at the Bottom. In: *Engineering and Science Magazine* 23:5 (1960), S. 22–36

[12] Garey, Michael R. ; Johnson, David S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979

[13] Hartmanis, Juris: On the Weight of Computations. In: *Bulletin of the European Association for Theoretical Computer Science* 55 (1995), S. 136–138

[14] Head, Tom: Formal language theory and DNA: An analysis of the generative capacity of specific recombinant behaviors. In: *Bulletin of Mathematical Biology* 49 (1987), Nr. 6, S. 737–759

[15] Head, Tom: Circular suggestions for DNA computing. In: *Pattern Formation in Biology, Vision and Dynamics* (2000), S. 325–335

[16] Head, Tom ; Rozenberg, Grzegorz ; Bladergroen, Reno S. ; Breek, Cornelis K. ; Lommerse, Piet H. ; Spaink, Herman P.: Computing with DNA by operating on plasmids. In: *Biosystems* 57 (2000), Nr. 2, S. 87–93

[17] Hinze, Thomas ; Sturm, Monika: *Rechnen mit DNA. Eine Einführung in Theorie und Praxis.* Oldenburg Wissenschaftsverlag, 2004

[18] Johnson, Clifford R.: Automating the DNA computer: solving n-Variable 3-SAT problems. In: *Natural Computing* 7 (2008), Nr. 3, S. 239–253

[19] Kari, Lila ; Păun, Gheorghe ; Rozenberg, Grzegorz ; Salomaa, Arto ; Yu, Sheng: DNA computing, sticker systems, and universality. In: *Acta Informatica* 35 (1998), Nr. 5, S. 401–420

[20] Kari, Lila ; Thierrin, Gabriel: Contextual insertion/deletion and computability. In: *Information and Computation* 131 (1996), Nr. 1, S. 47–61

[21] Lin, Che-Hsin ; Cheng, Hsiao-Ping ; Yang, Chang-Biau ; Yang, Chia-Ning: Solving satifiability problems using a novel microarray-based DNA computer. In: *BioSystems* 90 (2006), S. 242–252

[22] Liu, Qinghua ; Wang, Liman ; Frutos, Anthony G. ; Condon, Anne E. ; Corn, Robert M. ; Smith, Lloyd M.: DNA computing on surfaces. In: *Nature* 403 (2000), S. 175–179

[23] Liu, Wenbin ; Gao, Lin ; Zhang, Qiang ; Xu, Guandong ; Zhu, Xiangou ; Liu, Xiangrong ; Xu, Jin: A Random Walk DNA Algorithm for the 3-SAT Problem. In: *Current Nanoscience* 1 (2005), S. 85–90

[24] Mancin, Fabrizio ; Scrimin, Paolo ; Tecilla, Paolo ; Tonellato, Umberto: Artificial metallonucleases. In: *Chemical Communications* 2005 (2005), Nr. 20, S. 2540–2548

[25] Mülhardt, Cornel: *Der Experimentator: Molekularbiologie / Genomics.* 6. Auflage. Spektrum Akademischer Verlag, 2009

[26] Munk, Katharina (Hrsg.): *Taschenlehrbuch Biologie: Biochemie - Zellbiologie.* Georg Thieme Verlag, 2008

[27] Nagy, Naya ; Akl, Selim G.: Aspects of biomolecular computing. In: *Parallel processing letters* 17 (2007), Nr. 2, S. 185–211

[28] Papadimitriou, Christos H.: *Computational Complexity.* Addison-Wesley, 1994

[29] Paun, Gheorghe ; Rozenberg, Grzegorz ; Salomaa, Arto: *DNA Computing: New Computing Paradigms.* Springer, 1998

[30] Reischuk, Karl R.: *Einführung in die Komplexitätstheorie.* B. G. Teubner Stuttgart, 1990

[31] Schöning, Uwe: *Ideen der Informatik: Grundlegende Modelle und Konzepte.* Oldenburg Wissenschaftsverlag, 2002

[32] Schöning, Uwe: *Theoretische Informatik - kurzgefasst.* 5. Auflage. Spektrum Akademischer Verlag, 2008

[33] Smith, Warren D.: DNA computers in vitro and in vivo. In: Lipton, Richard J. (Hrsg.) ; Baum, Eric B. (Hrsg.): *DNA Based Computers*, American Mathematical Society, 1996 (DIMACS), S. 121–186

[34] Sosík, Petr ; Rodríguez-Patón, Alfonso: Membrane computing and complexity theory: A characterization of PSPACE. In: *Journal of Computer and System Sciences* 73 (2007), Nr. 1, S. 137–152

[35] Stockmeyer, Larry J. ; Meyer, Albert R.: Word problems requiring exponential time (Preliminary Report). In: *Proceedings of the fifth annual ACM symposium on Theory of computing* ACM New York, NY, USA, 1973, S. 1–9

[36] Wang, Xiaolong ; Bao, Zhenmin ; Hu, Jingjie ; Wang, Shi ; Zhan, Aibin: Solving the SAT problem using a DNA computing algorithm based on ligase chain reaction. In: *BioSystems* 91 (2008), S. 117–125

[37] Wegener, Ingo: *Theoretische Informatik: Eine algorithmenorientierte Einführung.* 2. Auflage. B.G. Teubner Stuttgart, 1999

[38] Wolpert, Alexander ; Dantsin, Evgeny: A robust DNA computation model that captures PSPACE. In: *International Journal of Foundations of Computer Science* 14 (2003), Nr. 5, S. 933–951

[39] Yamamoto, Yoji ; Mori, Massao ; Aiba, Yuichiro ; Tomita, Takafumi ; Chen, Wen ; Zhou, Jing-Min ; Uehara, Akihiko ; Ren, Yi ; Kitamura, Yoshihito ; Komiyama, Makoto: Chemical modification of Ce (IV)/EDTA-based artificial restriction DNA cutter for versatile manipulation of double-stranded DNA. In: *Nucleic Acids Research* 35 (2007), Nr. 7, S. e53

[40] Yang, Chia-Ning ; Yang, Chang-Biau: A DNA solution of SAT problem by a modified sticker model. In: *BioSystems* 81 (2005), S. 1–9

[41] Yoshida, Hiroshi ; Suyama, Akira: Solution to 3-SAT by Breadth First Search. In: Winfree, Erik (Hrsg.) ; Gifford, David K. (Hrsg.): *DNA Based Computers V*, American Mathematical Society, 2000 (DIMACS), S. 9–22