



UNIVERSITÄT ZU LÜBECK  
INSTITUT FÜR TELEMATIK

# Übertragung herkömmlicher Netzwerkmechanismen auf DNA-Tile-basierte Nanonetzwerke Transfer of conventional network mechanisms to DNA- Tile-based nanonetworks

## Masterarbeit

im Rahmen des Studiengangs  
**Informatik**  
der Universität zu Lübeck

vorgelegt von  
**Andreas Waldner**

ausgegeben und betreut von  
**Dr. rer. nat. Florian-Lennert Lau**

Lübeck, den 17. Oktober 2023

Im Focus das Leben



# **Erklärung**

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Lübeck, den 17. Oktober 2023



## Kurzfassung

In der heutigen technologischen Landschaft zeichnet sich ein Trend zur kontinuierlichen Miniaturisierung ab, wobei wir schon seit einigen Jahren in die Nanoebene eintauchen. Obwohl Nanomaterialien und -technologien bereits in zahlreichen Bereichen Anwendung finden, birgt diese Dimension immer noch ein großes und unausgeschöpftes Potenzial. Diese Arbeit beschäftigt sich mit der Anwendung herkömmlicher Mechanismen aus Netzwerken und Kommunikationsprotokollen auf DNA-basierten Nanonetzwerken. Während konventionelle Netzwerke eine breite Palette von Mechanismen und Strategien bieten, besteht das Ziel dieser Arbeit darin, deren Anwendbarkeit und Relevanz im Gebiet der Nanonetzwerke zu erforschen. Um Simulation und Analyse dieser Mechanismen zu optimieren, wurde ein Python-Skript entwickelt, das Tilesets generieren oder modifizieren kann. Die Ergebnisse dieser Arbeit bieten wertvolle Einblicke in das Potenzial von DNA-basierten Nanonetzwerken. Auch werden Grenzen und Probleme dieser Technologie analysiert und aufgezeigt.

## Abstract

In today's technological landscape, a trend towards miniaturization is emerging, and we have been immersed in the nanoscale for several years now. Although nanomaterials and nanotechnologies already have applications in numerous fields, this dimension still holds great and untapped potential. This work focuses on the application of conventional mechanisms from networks and communication protocols to DNA-based nanonetworks. While conventional networks offer a wide range of mechanisms and strategies, the goal of this work is to explore their applicability and relevance in the field of nanonetworks. To optimize simulation and analysis of these mechanisms, a Python script was developed that can generate or modify tile sets. The results of this work provide valuable insights into the potential of DNA-based nanonetworks. Also, limitations and problems of this technology are analyzed and highlighted.



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
1.1. Motivation/Problemstellung . . . . .	1
1.2. Der wissenschaftliche Beitrag dieser Arbeit . . . . .	3
1.3. Struktur der Arbeit . . . . .	4
<b>2. Grundlagen</b>	<b>5</b>
2.1. Einführung in Nanonetzwerke . . . . .	5
2.2. Grundlagen der DNA . . . . .	6
2.3. Self-Assembly . . . . .	8
2.4. Tile-basierte Self-Assembly . . . . .	8
2.5. Nanogeräteklassen . . . . .	11
2.6. Tile-Assembly Modelle . . . . .	12
2.7. Error-Handling in Tile-basierten Self-Assembly Systemen . . . . .	19
2.8. DNA-basierte Nanonetzwerke . . . . .	21
2.9. Kommunikation in Nanonetzwerken . . . . .	24
2.9.1. Die anwendungsorientierten Schichten . . . . .	28
2.9.2. Die transportorientierten Schichten . . . . .	29
<b>3. Related Work</b>	<b>31</b>
3.1. Hop Count Routing: A Routing Algorithm for Resource Constraint, Identity-Free Medical Nanonetworks . . . . .	31
3.2. CORONA: A Coordinate and Routing system for Nanonetworks . . . . .	33
3.3. A Deployable Routing System for Nanonetworks . . . . .	35
<b>4. Konzeption</b>	<b>39</b>
4.1. Die Adressierung in Nanonetzwerken . . . . .	39
4.2. Das Übertragungsmedium in Nanonetzwerken . . . . .	40
4.3. Routing und Dialogaufbau in Nanonetzwerken . . . . .	40
4.4. Framing in Nanonetzwerken . . . . .	41
4.5. Fehlererkennung durch Prüfsummen in Nanonetzwerken . . . . .	43
4.6. Fehlerkorrektur durch Snaked-Proofreading in Nanonetzwerken . . . . .	44
4.7. Datenflusskontrolle in Nanonetzwerken . . . . .	46
4.8. Datenflusskontrolle durch Acknowledgements in Nanonetzwerken . . . . .	47
4.9. Datenflusskontrolle durch Prioritätslevel in Nanonetzwerken . . . . .	51
4.10. Nachrichtencodierung in Nanonetzwerken . . . . .	52
4.11. Self-Assemblies in ihrer Höhe anpassen . . . . .	57

4.12. Flags in Nanonetzwerken . . . . .	58
4.13. Prioritätslevel auf Basis von Flags . . . . .	61
4.14. Einordnung der vorgestellten Mechanismen im ISO/OSI-Modell . . . . .	61
4.15. Ein zusammenfassendes Beispiel für alle Anforderungen . . . . .	63
<b>5. Konstruktion des Skripts</b>	<b>67</b>
5.1. NetTAS . . . . .	67
5.2. Die Generierung von Tilesets im Skript . . . . .	69
5.3. Die Generierung von Prüfsummen in Nanonetzwerken . . . . .	70
5.4. Farbcode als Anforderung an Tilesets . . . . .	71
5.5. Das grafische Interface des Skripts . . . . .	72
5.6. Anforderungen für das korrekte Ausführen des Skripts . . . . .	75
5.6.1. Notwendige Anforderungen an Tilesets . . . . .	76
5.6.2. Empfehlenswerte Anforderungen an Tilesets . . . . .	77
5.6.3. Vorschläge für Tilesets . . . . .	79
5.7. Ablauf des Skripts . . . . .	80
5.8. Erstellung von Flag- und Prioritätstiles . . . . .	81
5.9. Proofreading im Skript . . . . .	82
<b>6. Simulationen</b>	<b>89</b>
6.1. Simulation und Evaluation der Gewichtungen zur Generierung von Tilesets	89
6.2. Die Tilesets . . . . .	93
6.3. Evaluation der Adressierung in DNA-Tile-basierten Tile-Assemblies . . . . .	97
6.4. Simulation und Evaluation von Acknowledgements . . . . .	97
6.5. Simulation und Evaluation von Prioritätsleveln . . . . .	99
6.6. Simulation und Evaluation von Flags . . . . .	101
6.7. Simulation und Evaluation von Prüfsummen . . . . .	103
6.8. Simulation und Evaluation von Snaked-Proofreading . . . . .	107
6.8.1. Snaked-Proofreading für das H-3-norm Tileset . . . . .	107
6.8.2. Snaked-Proofreading für die erstellten Tilesets . . . . .	111
6.8.3. Snaked-Proofreading für Acknowledgements . . . . .	113
6.8.4. Snaked-Proofreading für Flags . . . . .	114
6.8.5. Snaked-Proofreading für Prioritätslevel . . . . .	115
6.8.6. Snaked-Proofreading für Prüfsummen . . . . .	117
6.9. Evaluation Zusammenfassung . . . . .	118
<b>7. Zusammenfassung und Aussichten</b>	<b>119</b>
7.1. Aussicht für die Zukunft . . . . .	119
7.2. Zusammenfassung . . . . .	120
<b>Literaturverzeichnis</b>	<b>vii</b>
<b>A. Zugriff auf den Code</b>	<b>xiii</b>

# 1. Einleitung

Die Konzepte der Nanotechnologie haben die Grenzen der herkömmlichen Technologie überschritten und neue Horizonte der Wissenschaft und Technik eröffnet. Unter diesen Konzepten befindet sich das *Nanonetzwerk*, ein innovatives Netzwerk, das aus einer Sammlung von Nanomaschinen besteht, die miteinander interagieren und kommunizieren können. In diesem Kapitel soll eine Motivation für das Thema, der wissenschaftliche Beitrag dieser Arbeit und die Struktur der Arbeit vorgestellt werden.

## 1.1. Motivation/Problemstellung

Nanonetzwerke öffnen ein neues Fenster in die Welt der Mikro- und Nanotechnologie und zeigen großes Potenzial für Anwendungen in der Medizin, Umweltüberwachung, Materialwissenschaft und einigen anderen Bereichen [2]. Sie ermöglichen die Kommunikation und den Informationsaustausch auf der Nanoskala, was einen grundlegend neuen Ansatz für die Kommunikation und den Informationsaustausch darstellt. Wie klein die Nano-ebene ist, ist für einige Personen schwer einzuschätzen. Ein typischer Vergleich dafür ist die Breite eines Haares. Ein Haar ist je nach Beschaffenheit 0,05 bis 0,08 Millimeter breit. Das sind umgerechnet 50.000 bis 80.000 Nanometer. Vergleichsweise dazu beträgt die Breite eines DNA-Strangs 2,5 Nanometer [52, 53]. Diese Skala führt zu einer neuen Dimension von Netzwerken, die über die bisher bekannten Grenzen hinausgehen.

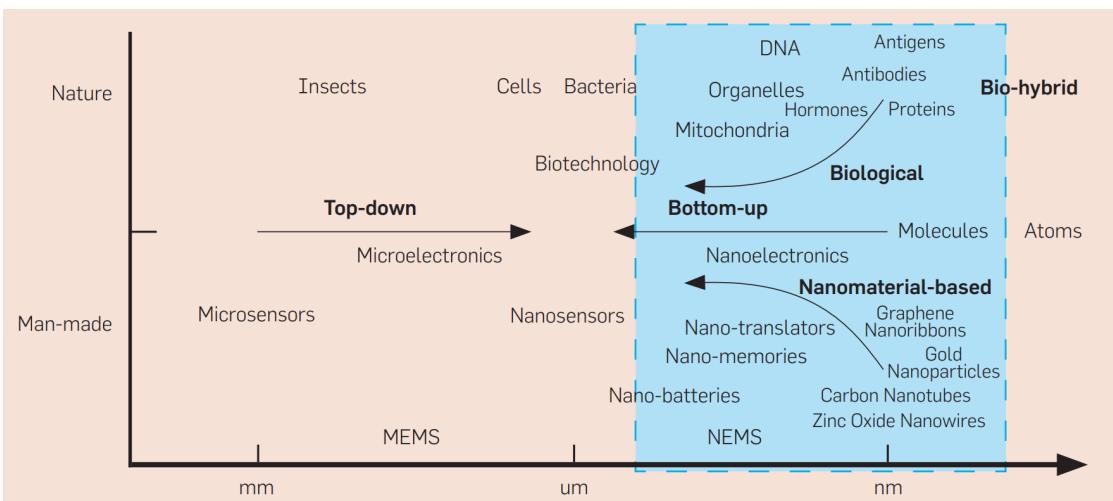
Nanotechnologien können eine Vielzahl von Strukturen und Partikeln umfassen, die sich in Funktion und Fähigkeiten erheblich unterscheiden können. Je nach Anwendung können verschiedene Strukturen verwendet werden, was zur Vielseitigkeit dieser Technologie beiträgt. Einige Beispiele dieser Strukturen sind in Abbildung 1.1 zu sehen.

Es ist zu beachten, dass in der Welt der Nanotechnologien eine Unterscheidung zwischen biologischen und menschengemachten Materialien gemacht wird. Biologische Nanomaschinen nutzen biologische Mechanismen und Prozesse, während menschengemachte Maschinen auf künstlichen Materialien und Herstellungsprozessen basieren.

Diese Arbeit konzentriert sich auf eine spezielle Art von Nanonetzwerken: auf die DNA-basierten Nanonetzwerke. Diese verwenden die Struktur und die Eigenschaften von DNA-Molekülen, um Netzwerke zu bilden und Funktionen auszuführen. Im weiteren Verlauf dieser Arbeit wird der Begriff „Nanonetzwerke“ speziell auf diese Art von Netzwerken bezogen, es sei denn, es wird anders angegeben.

## 1. Einleitung

---



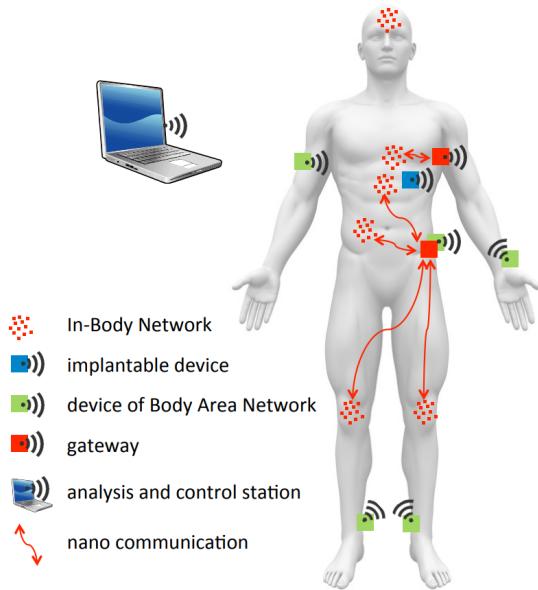
**Abbildung 1.1.:** Biologische und menschengemachte Strukturen, die entweder durch ein Top-Down oder Bottom-Up Ansatz für Nanotechnologien verwendet werden können.[2]

Die Bedeutung von DNA-basierten Nanonetzwerken kann nicht hoch genug eingeschätzt werden, da sie eine breite Palette von Anwendungen ermöglichen, von der gezielten Medikamentenabgabe bis zur Biosensorik und darüber hinaus. Indem das Verständnis und die Fähigkeiten in diesem Bereich erweitert werden, öffnen sich die Türen zu revolutionären Fortschritten in einer Vielzahl von Disziplinen.

Wie zuvor angedeutet, gibt es einige verschiedene Anwendungsgebiete für Nanotechnologien und im Besonderen für Nanonetzwerke. Das größte und möglicherweise wichtigste Anwendungsgebiet ist jedoch die Medizin [17, 28, 49, 55]. Nanopartikel im Spezifischen werden beispielsweise schon länger zur Krebsbekämpfung eingesetzt [44].

Es lassen sich auch andere mögliche Anwendungsgebiete für diese Technologie finden, wie beispielsweise in der Geologie [19], in der Agrartechnik [4, 5] oder in der Materialindustrie [32, 45]. Doch die Fähigkeit von Nanonetzwerken, auf einem deutlich kleineren Skalenbereich als herkömmliche Netzwerke zu agieren, verleiht ihnen ein hohes Potenzial für Anwendungen im Bereich der Humanmedizin.

Zwar werden einige dieser Technologien heute schon verwendet, doch das Potenzial ist noch lange nicht voll ausgeschöpft. So können diese bereits implementierten Technologien in ein *In-Body-Network* (IBN) zusammengefügt werden. Das theoretische Nanonetzwerk funktioniert mit einer Kontrollstation, die mit Geräten verbunden ist, die beispielsweise unter der Haut eines Menschen implantiert werden können. Diese Geräte werden wiederum dafür verwendet, um über den menschlichen Blutkreislauf miteinander und mit der externen Kontrollstation zu kommunizieren. Ein solches System kann nicht nur die Vitalwerte des Körpers überwachen, sondern diese Daten zur Analyse und Auswertung an die Kontrollstation geben. Wiederum kann die Kontrollstation anhand der ausgewerteten Daten Anweisungen an das In-Body-Network senden, das daraufhin entsprechende



**Abbildung 1.2.:** In-Body-Netzwerke kombiniert mit einem Body-Area-Network (BAN). Eine Kontrollstation, welche für die Analyse von Daten zuständig ist, kommuniziert über drahtlose Verbindungen mit im Körper implantierten Geräten. Diese Geräte können wiederum miteinander oder über ein Gateway mit In-Body-Networks kommunizieren.[12]

Medikamente an spezifischen Stellen des Körpers anwenden kann. Ein solches Netzwerk, wie es in Abbildung 1.2 dargestellt ist, kann mit heutigem Stand zwar noch nicht *in-vivo* (im lebendigen Organismus) realisiert werden, doch die theoretischen Modelle sind schon so weit, dass sie auf mathematischer Ebene funktionsfähig sind.

Jedoch gibt es bislang keine konkreten Ansätze, um die kleinschrittige Kommunikation mit DNA-Tiles und ihrer Self-Assembly zu regeln. Wie kann Information in Self-Assemblies abgebildet werden? Welche Mechanismen aus herkömmlichen Kommunikationsprotokollen lassen sich auf einer so kleinen Ebene übersetzen? Wie funktioniert Adressierung oder Fehlererkennung in einer Self-Assembly? Diese und andere Fragen haben dazu geführt, dass diese Arbeit sich kleinschrittiger und tiefer mit diesem Thema der Nanonetzwerke befasst.

## 1.2. Der wissenschaftliche Beitrag dieser Arbeit

Diese Arbeit soll eine umfangreiche Grundlage für die Kommunikation mit DNA-Tile-basierten Self-Assemblies schaffen. Die Ansätze und Ideen sollen dementsprechend möglichst allgemein gehalten werden, sodass sie für verschiedene Anwendungsgebiete angepasst werden können. Auch soll mit dieser Arbeit ein Gefühl übermittelt werden,

welche Mechanismen herkömmlicher Kommunikationsprotokolle mit Tile-basierter Self-Assembly implementierbar sind und welche nicht. Leser\*innen dieser Arbeit sollen am Ende Inspiration und Gedankenanstöße für eigene Forschung auf dem Gebiet erhalten.

### 1.3. Struktur der Arbeit

Die Arbeit ist in sieben Kapitel unterteilt. Nach der Einleitung in diesem Kapitel folgen mit Kapitel 2 die Grundlagen für DNA-basierte Nanonetzwerke. In den Grundlagen finden sich Informationen über DNA, verschiedene Tilebildungsverfahren von DNA-Strängen, Self-Assembly der Tiles und Assemblymodelle. Auch werden Grundlagen zu herkömmlichen Kommunikationsprotokollen geliefert. Nach den Grundlagen werden in Kapitel 3 wissenschaftliche Arbeiten näher betrachtet, da diese eine besondere Nähe zu dieser Arbeit besitzen.

Nach den Grundlagen werden in Kapitel 4 die Konzepte vorgestellt, die für die Umsetzung von Mechanismen aus herkömmlichen Kommunikationsprotokollen auf Nanoebene notwendig sind. Von Adressierung, Routing, Fehlererkennung, Fehlerkorrektur, Framing, Datenflusskontrolle, Nachrichtencodierung zu Flags, werden verschiedene Ideen modelliert und vorgestellt. Einige dieser Ideen wurden in einem Python Skript implementiert, das Tilesets für die Simulationsumgebung NetTAS verändert oder generiert. Das Skript und die Umsetzung der Mechanismen werden in Kapitel 5 vorgestellt.

Die betrachteten Tilesets und ihre Simulationsergebnisse werden im Kapitel 6 analysiert und ausgewertet. In Kapitel 7 wird abschließend eine Zusammenfassung der Arbeit geliefert.

Da eine Motivation und ein grober Überblick für die Arbeit geliefert wurde, kann mit dem detaillierteren Inhalt der Arbeit begonnen werden.



## 2. Grundlagen

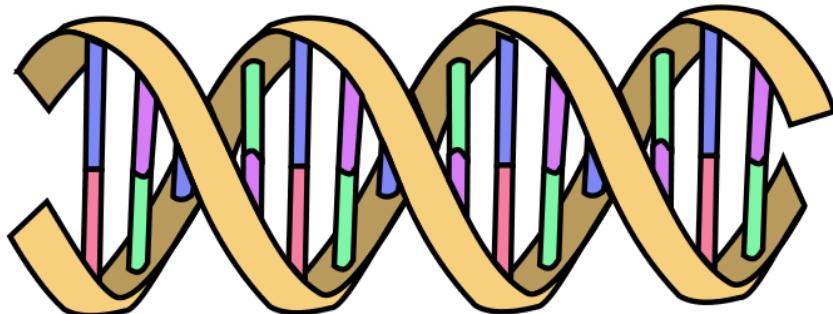
Dieses Kapitel liefert wichtige Grundlagen für die Arbeit. Zu Beginn werden Nanonetzwerke eingeführt und einige ihrer Herausforderungen erläutert. Anschließend folgt die Definition von DNA-Tile-basierten Nanonetzwerken mit der Konstruktion, Modellierung und Darstellung dieser Systeme. Zum Abschluss des Kapitels geht es um die Kommunikationsprotokollsichten, die in dieser Arbeit besonderes Augenmerk erhalten, da sie auf die DNA-Tile-basierte Nanonetzwerkebene abstrahiert werden sollen.

### 2.1. Einführung in Nanonetzwerke

Der Begriff *Nanonetzwerk* umfasst allgemein alle Strukturen, die Netzwerken ähneln und auf dem Skalenbereich der Nanoebene existieren. Auch wenn in dieser Arbeit nur DNA-Tile-basierte Nanonetzwerke betrachtet werden, ist es sinnvoll, zu Beginn allgemeinere Eigenschaften von Nanonetzwerken zu betrachten.

Nanonetzwerke dienen zur Kommunikation von Teilnehmern in einem Netzwerk auf Nanoebene. Auch wenn sie in ihren Aufgaben der Kommunikation zwischen Teilnehmern des Netzwerks herkömmlichen Netzwerken ähneln, gibt es bedeutende Unterschiede. Nanonetzwerke haben eine höhere Teilnehmeranzahl als die meisten herkömmlichen Netzwerke. Da sie auf einem kleinen Skalenbereich funktionieren, wird dieser Faktor vorwiegend durch eine hohe Menge an Teilnehmern ausgeglichen, um für den Skalenbereich große Distanzen überbrücken zu können. Auch weisen die meisten Nanonetzwerke eine hohe Heterogenität auf, da ein Nanogerät häufig nur eine spezifische Aufgabe übernehmen kann. Darüber hinaus existiert in den meisten Nanonetzwerken keine zentrale Kontrollinstanz, da diese auf Nanoebene schwer zu realisieren ist. So kann eine Steuerung wie im Beispiel aus Kapitel 1 von außen durch beispielsweise ein *Body-Area-Network* (BAN) geregelt werden.[8]

Da sich Nanonetzwerke so stark von herkömmlichen Netzwerken unterscheiden, ist es schwierig, ein herkömmliches *Kommunikationsprotokoll* in einem Nanonetzwerk unverändert anzuwenden. Kommunikationsprotokolle sind in der Informatik seit vielen Jahren von zentraler Bedeutung. Sie definieren die Regeln und Verfahren für die Übertragung von Daten zwischen verschiedenen Endgeräten und Netzwerken. Ein zentraler Bestandteil der Kommunikation von verteilten Geräten ist die *Synchronisation*. Für die Kommunikation von verteilten Systemen entwickelte Lamport im Jahr 1978 wichtige Grundlagen [24]. In der Informatik haben Mechanismen und Ansätze zur Synchronisation in herkömmlichen



**Abbildung 2.1.:** Ein farbcodiertes DNA Modell. Zu sehen ist das Zuckerphosphatrückgrat (gelb), das durch Basenpaare aus Adenin (A) und Thymin (T) oder Guanin (G) und Cytosin (C) verbunden ist.[51]

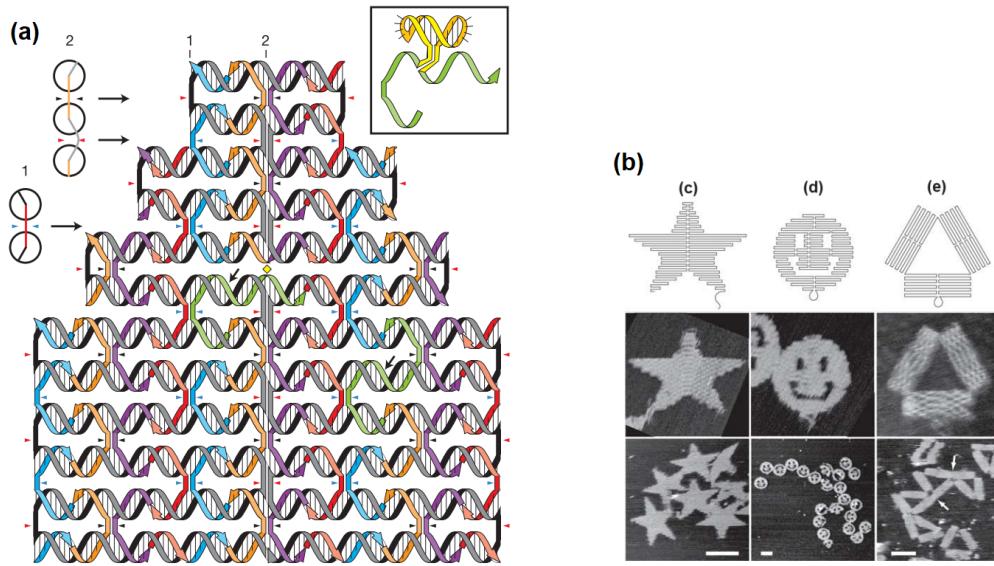
Netzwerken erhebliche Fortschritte gemacht, auch wenn sie immer noch nicht optimal sind. Auf Nanoebene ist die Synchronisation noch anspruchsvoller und muss meist über asynchrone Kommunikation oder andere Ansätze umgangen werden. Somit können einige Mechanismen zur Synchronisation eines Systems in Nanonetzwerken ausgeschlossen werden.[8]

In dieser Arbeit werden einige Kommunikationsprotokolle und -mechanismen auf die Nanoebene abstrahiert und analysiert. Bevor diese Mechanismen jedoch näher betrachtet und auf die Nanoebene übersetzt werden können, müssen im Folgenden einige Grundlagen aufgestellt werden.

## 2.2. Grundlagen der DNA

Die *Desoxyribonukleinsäure* (DNS oder DNA) ist ein Schlüsselbaustein in der Informationsstruktur lebender Organismen, da sie den genetischen Code enthält. Da die Bezeichnung DNA gebräuchlicher ist als die deutsche Abkürzung DNS, wird im Folgenden immer von DNA die Rede sein. Dieser Code besteht aus einer Sequenz der organischen Basen Adenin (A), Guanin (G), Cytosin (C) und Thymin (T). Die spezifische Anordnung dieser Basen bildet den genetischen Code, der die Produktion von Proteinen und folglich die biologischen Funktionen von Zellen und Organismen steuert. [3]

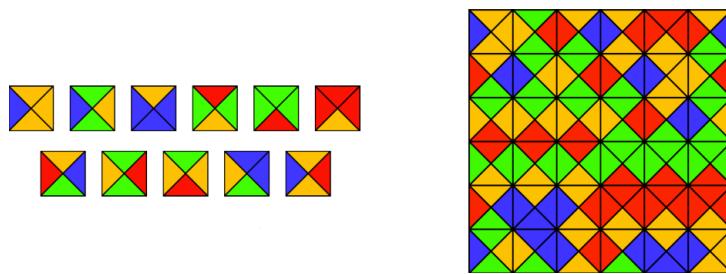
Die grundlegende Struktur der DNA wurde erstmals im Jahr 1953 von Watson und Crick als Doppelhelix-Struktur vorgestellt [50]. Diese Doppelhelix besteht aus zwei entgegengesetzten Strängen von Zucker- und Phosphat-Molekülen, die durch Basenpaare (A-T und G-C) miteinander verbunden sind, wie in Abbildung 2.1 dargestellt. Diese Struktur ist fundamental für die Fähigkeit der DNA zur Selbsterhaltung und Replikation. Durch sie wird in lebenden Organismen das Informationsmanagement ermöglicht.



**Abbildung 2.2.:** DNA Origami aus der Arbeit von Rothemund [42]. Dabei ist in (a) die Konstruktion eines langen DNA Strangs zu sehen, der durch Überkreuzungen und DNA Klammern in die gewollte Form gebracht werden kann. In (b) wird ein Machbarkeitsbeweis gegeben. Rothemund hat dafür verschiedene Strukturen erstellt. Von oben nach unten: Modell, Nahaufnahme einer Konstruktion und Aufnahme der Gesamtstruktur. Hier abgebildet sind die folgenden Formen: ein Stern (c), ein Smiley (d) und ein Dreieck aus Rechtecken (e)

Das Konzept der DNA-Replikation wurde erstmals im Jahr 1958 von Meselson und Stahl beschrieben. Dabei handelt es sich um den Prozess, bei dem ein DNA-Molekül in zwei identische Kopien geteilt wird [35]. Dieser Prozess beginnt mit der Denaturierung oder Trennung der Doppelhelix in zwei einzelne Strände. Jeder dieser Einzelstränge dient dann als Vorlage für die Synthese eines neuen, komplementären Strangs. Dabei erkennt und bindet ein Enzym namens DNA-Polymerase die komplementären Basen (A zu T und C zu G) an den Einzelstrang. Dies führt zur Bildung einer exakten Kopie des ursprünglichen DNA-Strangs.

Diese Grundkenntnisse über die Struktur und den Replikationsmechanismus der DNA sind essenziell für das Verständnis der molekularen Genetik und können in der Nanotechnologie insbesondere bei der DNA-basierten Datenspeicherung genutzt werden. Die Fähigkeit der DNA zur Selbsterhaltung und Replikation ermöglicht die Erstellung gäuer Kopien von Daten, was einen robusten und effizienten Mechanismus für die Datensicherung bietet.



**Abbildung 2.3.:** Links Wang Tiles, die sich rechts zu einer lückenlosen Struktur zusammenfügen lassen.[33]

## 2.3. Self-Assembly

Für diese Arbeit von besonderem Interesse ist die *DNA Self-Assembly*. Dabei wird DNA verwendet, um Strukturen zu bilden, die ohne weiteres äußeres Einwirken selbstständig und selbstorganisiert von den DNA-Strängen gebildet werden. Diese Technik bietet einen Bottom-Up Ansatz zur Bildung von komplexeren Strukturen auf der nanoskalaren Ebene.

Die Grundlagen für DNA Self-Assembly legten Seeman et al. bereits in den 1980er Jahren. Es entstand die Idee davon, die Eigenschaften der Basenpaare gezielt zu nutzen, um die Moleküle in einer bestimmten Art und Weise anzurichten.[43]

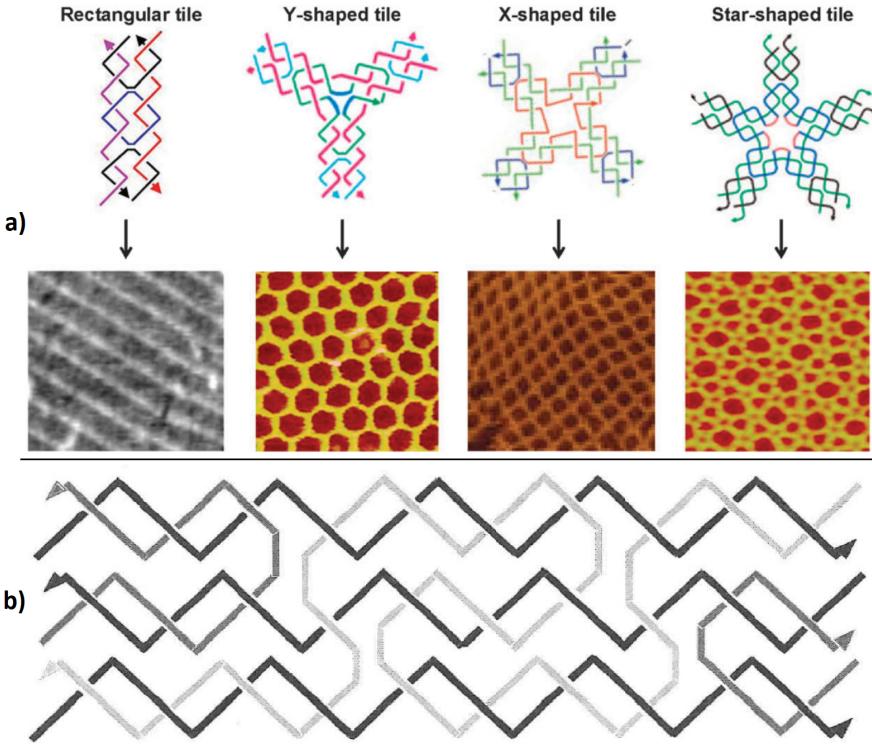
Ein weiterer Grundbaustein der DNA Self-Assembly ist die Arbeit von Chad A. Mirkin et al. aus dem Jahr 1996. In ihrer Arbeit „A DNA-based method for rationally assembling nanoparticles into macroscopic materials“ stellen die Forschenden eine Technik vor, durch die synthetisierte Goldnanopartikel durch DNA-Moleküle reversibel in spezifische Muster und Strukturen geordnet werden.[36]

Auf Basis dieser Grundbausteine wurden einige Ansätze zur DNA Self-Assembly entwickelt. Einer dieser Ansätze ist das *DNA-Origami*. Erstmals vorgestellt im Jahr 2006 von Paul Rothemund, zeigte diese Technik, dass durch DNA Self-Assembly beliebige Strukturen zuverlässig konstruiert werden können. Ein DNA-Origami Molekül, sowie einige von Rothemund erstellte Strukturen sind in Abbildung 2.2 zu sehen.[42]

Die von Rothemund vorgestellte Technik lässt sich ohne Probleme auch auf dreidimensionale Strukturen übertragen, wie die Arbeit von Ke et al. zeigt. [22]

## 2.4. Tile-basierte Self-Assembly

Basierend auf der zuvor beschriebenen Konstruktion von Nanostrukturen auf DNA-Basis, wird nun die *Tile-basierte Self-Assembly* vorgestellt. Die theoretische Basis dieser Art der Self-Assembly sind die nach ihrem Erfinder benannten: die *Wang Tiles*. Wang Tiles werden meist durch Quadrate mit vier farbigen Seiten dargestellt, wie in Abbildung 2.3 zu

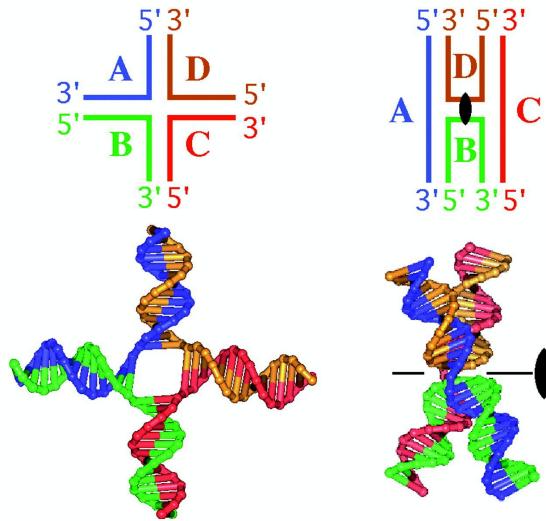


**Abbildung 2.4.:** Darstellung von zwei verschiedenen Tile-Strukturen. In a) ist das *double-crossover-tile* (DX-tile) dargestellt [41]. Von links nach rechts ist eine Rechteckform, eine Y-Form, eine X-Form und eine Sternform zu sehen. Unten sind Aufnahmen der Strukturen unter einem Mikroskop dargestellt. In b) ist das Modell eines Triple-Crossover-Tiles (TX-Tiles) aus der Arbeit von Winfree et al. abgebildet [23].

erkennen ist. Ein Tile aus einer Menge von Wang Tiles kann sich in einem  $\mathbb{Z}^2$  Bereich nur mit anderen Tiles verbinden, wenn die Farben mit allen verbundenen Tiles übereinstimmen. Dieses Verhalten kann mit einem Puzzle verglichen werden und lässt sich direkt auf die Bindung von DNA-Tiles übertragen.[48]

Dafür werden im Weiteren einige DNA-Tiles vorgestellt. Es gibt verschiedene Arten von DNA-Tiles. In dieser Arbeit werden im Folgenden DX-Tiles, TX-Tiles und Holliday Junctions betrachtet. Diese sollten für einen allgemeinen Überblick von DNA-Tiles ausreichen, der in dieser Arbeit benötigt wird.

Die *Double-Crossover-Tiles* (DX-Tiles) wurden 1993 von Fu und Seeman vorgestellt [16]. Dabei werden zwei Doppelstränge miteinander verwoben. Wie ganz links in Abbildung 2.4 a) zu sehen ist, besteht die so entstandene Struktur aus zwei äußeren Strängen (orange und violett) und zwei inneren Strängen (schwarz und blau). Diese Verwebung von DNA bildet eine stabile Struktur. Dabei ist zu erkennen, dass das DX-Tile in dieser Grundform mehrere offene Stränge auf beiden Seiten hat. Es lassen sich damit weitere Strukturen wie



**Abbildung 2.5.:** Darstellung der Holliday Junction. Oben die Schemata von zwei möglichen Verbindungen. Unten die daraus entstandenen Modelle.[13]

zum Beispiel die Y-Form, X-Form oder Sternform bilden. Auch ermöglichen die offenen Stränge die Verbindung von mehreren DX-Tiles, wodurch größere Strukturen durch Self-Assembly ermöglicht werden. Dies ist unten in Abbildung 2.4 a) in den Aufnahmen der gebildeten Strukturen zu erkennen.

Eine Erweiterung der DX-Tiles bilden die *Triple-Crossover-Tiles* (TX-Tiles). Vorgestellt wurden sie von Winfree et al. im Jahr 2000 [23]. Durch die Verwebung von drei Doppelsträngen ist diese Struktur noch stabiler. Außerdem ermöglicht das TX-Tile das Bilden von komplexeren Strukturen als das DX-Tile, da es einen größeren Abstand zwischen den offenen Enden aufweist. Ein solches TX-Tile ist in Abbildung 2.4 b) zu sehen.

Von besonderem Interesse für diese Arbeit sind die *Holliday Junctions*. Von Seeman et al. [20] in ihrer Arbeit aus dem Jahr 1983 das erste Mal in einem Machbarkeitsbeweis vorgestellt, stammt diese Verbindung von DNA-Strängen aus der Feder von Holliday [18]. Da diese Verbindung vier offene Enden in verschiedenen Richtungen hat, bietet sie sich gut zum Bilden von komplexeren Strukturen an. Wie in Abbildung 2.5 zu erkennen ist, besteht eine Holliday Junction aus vier Strängen, die jeweils mit zwei anderen Strängen verbunden sind. Dabei entstehen vier offene Enden, die sich je nach Sequenz der Basenpaare nur mit ausgewählten anderen Strängen verbinden können. Aus Abbildung 2.5 lässt sich ableiten, dass die Ausrichtung der vier offenen Enden variieren kann. Für diese Arbeit wird jedoch nur die Ausrichtung in vier unterschiedliche Richtungen relevant sein.

Da somit der Prozess zur Bildung von DNA-Tiles vorgestellt wurde, kann im Weiteren die Modellierung und mathematische Darstellung der Tiles besprochen werden.

## 2.5. Nanogeräteklassen

Um Modelle von Nanonetzwerken aus Nanogeräten und ihren Nachrichten erstellen zu können, müssen die verschiedenen Nanostrukturen zuerst definiert werden. Hierfür werden Definitionen aus der Arbeit von Büther et al. herangezogen [8]. Um Nanogeräte definieren zu können, müssen zuerst einige Kategorien festgelegt werden. Dazu zählen *Aktuatoren A*, eine Komponente für die *Kommunikation* mit anderen Geräten *C*, eine Komponente zur *Informationsverarbeitung I*, eine Komponente zur *Fortbewegung L*, ein *Speicher* für Daten und Programme *M*, eine *Energieversorgung P*, *Sensoren S* und *Zeitgeber T*.

Um spezifische Nanogeräte definieren zu können, muss zu Beginn eine allgemeine Definition anhand der Kategorien zu Nanostrukturen gegeben werden.

**Definition 1** Eine Nanostruktur  $\mathcal{N}_S = K_{opt}$  ist ein nanogroßes, künstliches Konstrukt, das konzipiert wurde, um eine spezifische Funktion in einer Umgebung  $\Gamma$  zu erfüllen. Eine Nanostruktur besteht aus null oder mehr optionalen Komponenten  $K_{opt}$ . Diese sind definiert als  $K_{opt} \subseteq \{A, C, I, L, M, P, S, T\}$ .

Eine einheitliche und klare Definition für *Nanogröße* findet sich in der Literatur nicht, da sie sich je nach Angabe zwischen einem Nanometer und wenigen Mikrometern bewegt. Sie wird hier jedoch trotzdem verwendet, unter der Annahme, dass für den jeweiligen Anwendungsfall die Maximalgröße einer Nanostruktur beachtet wird. Mit dieser allgemeinen Definition können im Weiteren Nanogeräte definiert werden.

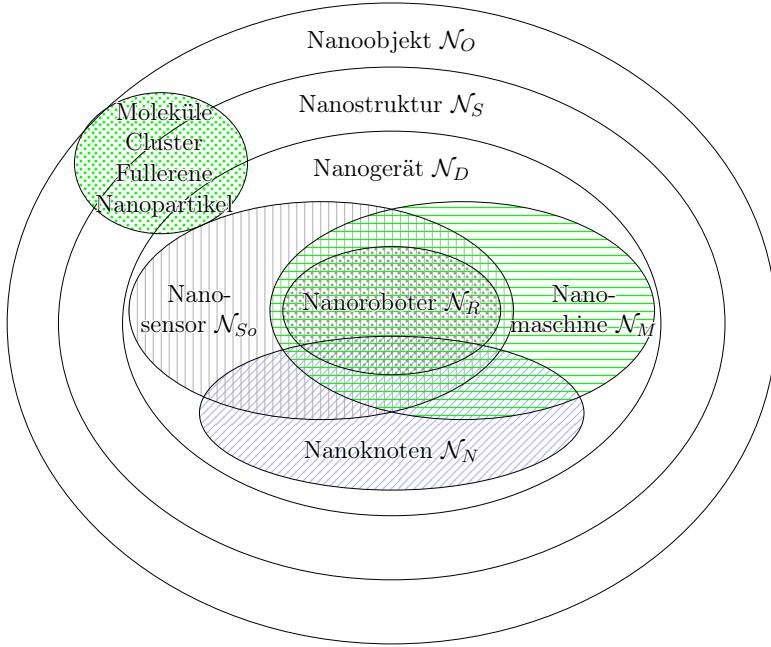
**Definition 2** Ein Nanogerät  $\mathcal{N}_D = K_{mand} \cup K_{opt}$  ist eine Nanostruktur. Es besteht aus einer Menge notwendiger Komponenten  $K_{mand} = \{P\}$  und einer Menge aus null oder mehr optionalen Komponenten  $K_{opt} = \{A, C, I, L, M, S, T\}$ .

Um ein Nanogerät von einer passiven Nanostruktur unterscheiden zu können, benötigt das Nanogerät eine autarke Energieversorgung *P*. Die restlichen Komponenten sind wie auch bei den Nanostrukturen optional. So können verschiedene Nanogeräte näher definiert werden.

**Definition 3** Eine Nanomaschine  $\mathcal{N}_M = K_{mand} \cup K_{opt}$  ist ein Nanogerät mit notwendigen Komponenten  $K_{mand} = \{A, P\}$  und einer Menge aus null oder mehr optionalen Komponenten  $K_{opt} \subseteq \{C, I, L, M, S, T\}$ , in einer Umgebung  $\Gamma$ .

**Definition 4** Ein Nanosensor  $\mathcal{N}_{Se} = K_{mand} \cup K_{opt}$  ist ein Nanogerät mit notwendigen Komponenten  $K_{mand} = \{P, S\}$  und einer Menge aus null oder mehr optionalen Komponenten  $K_{opt} \subseteq \{A, C, I, L, M, T\}$ , in einer Umgebung  $\Gamma$ .

**Definition 5** Ein Nanoknoten  $\mathcal{N}_N = K_{mand} \cup K_{opt}$  ist ein Nanogerät mit notwendigen Komponenten  $K_{mand} = \{P, C\}$  und einer Menge aus null oder mehr optionalen Komponenten  $K_{opt} \subseteq \{A, I, L, M, S, T\}$ , in einer Umgebung  $\Gamma$ .



**Abbildung 2.6.:** Venn-Diagramm verschiedener Nanostrukturen und ihren Überschneidungen.[26]

Nanomaschinen, Nanosensoren und Nanoknoten sind in dieser Definition Nanogeräte, die entweder Aktuatoren  $A$ , Sensoren  $S$  oder Kommunikationskomponenten  $C$  besitzen. Analog zu diesen Definitionen lassen sich im Folgenden auch Nanoroboter definieren.

**Definition 6** Ein Nanoroboter oder Nanobot  $\mathcal{N}_R = K_{mand} \cup K_{opt}$  ist ein programmierbares Nanogerät mit einem hohen Grad an Autonomie in einer Umgebung  $\Gamma$ . Es besteht aus einer Menge notwendiger Komponenten  $K_{mand} = \{A, I, M, P, S\}$  und einer Menge aus null oder mehr optionalen Komponenten  $K_{opt} \subseteq \{C, L, T\}$ .

Alle diese Nanostrukturen beziehungsweise Nanogeräte lassen sich mengentheoretisch mit ihren Überschneidungen in Abbildung 2.6 darstellen.

## 2.6. Tile-Assembly Modelle

Neben der Definition von Nanostrukturen müssen für diese Arbeit eindeutig definierte Modelle und Modellierungverfahren vorgestellt werden, um Nanonetzwerke modellieren zu können. Dafür werden in den folgenden Absätzen das *Abstract Tile-Assembly Model* (aTAM), das *Kinetic Tile-Assembly Model* (kTAM), das *Two-Handed Tile-Assembly Model* (2HAM) und das *Kinetic Two-Handed Tile-Assembly Model* (kTHAM) definiert. Um diese Modelle vorstellen zu können, müssen jedoch zu Beginn einige grundlegenden

Begriffe definiert werden. Alle folgenden Notationen und Definitionen basieren auf den Arbeiten von Lutz et al. [25] und Patitz [39].

**Definition 7** Ein  $n$ -dimensionales Tile  $t_n$  ist ein Objekt in  $\mathbb{Z}^n$  mit Einheitslänge und 90 Grad Winkeln. Eine Seite eines Tiles  $t_n$  ist durch einen Vektor  $u_i \in U_t \subseteq \mathbb{Z}^n$  definiert.  $U_t$  ist eine Menge von eindeutigen Richtungsvektoren. Der Gegenvektor  $-u_i$  beschreibt die gegenüberliegende Seite von  $u_i$  in derselben Dimension. Der Vektor  $u_i$  hat genau einen Eintrag, der ungleich 0 ist. Die Seiten eines Tiles sind durch folgende Relation definiert:

$$\text{Seite: } t_n \mapsto \underbrace{U_t \times U_t \times \cdots \times U_t}_n$$

In dieser Arbeit werden hauptsächlich zweidimensionale Tiles von Bedeutung sein. Daher wird im Folgenden immer von zweidimensionalen Tiles gesprochen, wenn die Dimension nicht angegeben ist.

In jedem *Tile Assembly System (TAS)* gibt es eindeutig definierte Regeln, durch die sich zwei Tiles miteinander verbinden können. Dafür wird Folgendes definiert:

**Definition 8** Ein Kleber oder Glue  $g \in G$ , wobei  $G$  die Menge der möglichen Kleber beschreibt, ist durch einen Bezeichner oder ein Label  $\mathcal{L}_g \in \Sigma^*$  definiert, wobei  $\Sigma$  ein Alphabet ist und  $s \in \mathbb{N}$  eine Kleberstärke mit den Funktionen:

$$\begin{aligned} \text{label} : G &\mapsto \Sigma^* \\ \text{strength} : G &\mapsto \mathbb{N} \end{aligned}$$

**Definition 9** Zwei Tiles  $t$  und  $t'$ , die durch  $v_t$  und  $v_{t'} \in \mathbb{Z}^n$  definierte Orte belegen, sind genau dann benachbart, wenn  $|v_t - v_{t'}| = 1$  und der resultierende Vektor  $e = v_t - v_{t'}$  genau ein Element ungleich 0 hat.

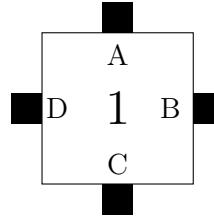
Ein Tile kann auf allen Seiten immer nur genau einen oder keinen Nachbarn haben.

**Definition 10** Die Temperatur  $\tau$  eines TAS beschreibt die minimale Stärke des Klebers  $s$  für eine stabile Verbindung.

Damit ein Tile in einem TAS eine stabile Verbindung aufbauen kann, muss die absolute Kleberstärke des Tiles mindestens der Temperatur des Systems entsprechen. Die absolute Kleberstärke ergibt sich aus der Summe aller Kleberstärken des Tiles mit den verbundenen anderen Tiles.

**Definition 11** Ein  $n$ -dimensionaler Tileytyp  $T_n$  ist eine Schablone für ein Tile  $t_n$ . Ein  $n$  dimensionaler Tileytyp wird durch einen Bezeichner  $\mathcal{L}_T \in \Sigma^*$  und einer Menge von Klebern  $g_i \in G$  definiert. Ein Kleber befindet sich an jeder Seite  $u_{t,i} \in U_t$  von  $t_n$ . Folgende Funktionen sind jedem Tileytyp zugeordnet:

$$\begin{aligned} \text{glue} : T_n \times U_t &\mapsto G \\ \text{strength} : g \in G &\mapsto \mathbb{N} \end{aligned}$$



**Abbildung 2.7.:** Mathematisches Schema eines DNA-Tiles. Die schwarzen Boxen stellen dabei die Stärke des Klebers auf den jeweiligen Seiten dar (hier 1). Die Beschriftungen (A, B, C, D) stellen die Farbe oder die Kleberbezeichner dar. Mit der „1“ wird das Tile beschriftet und gibt damit dem Tile einen Namen, um Beschreibungen und Analysen einfacher zu machen.

Eine so definierte Tileschablone kann wie in Abbildung 2.7 dargestellt werden. Hierbei bekommt jedes Tile eine eindeutige Beschriftung, in diesem Beispiel ist es die „1“. Auf den vier Seiten des Tile kann so mit einer weiteren Beschriftung die *Farbe*, der *Kleberbezeichner*, dargestellt werden. Der Kleberbezeichner wird durch die Sequenz der Basenpaare an den offenen Enden definiert. Dies ist in Abbildung 2.8 zu erkennen. Nur wenn der Kleberbezeichner von zwei Tiles gleich ist, können sich die Tiles im Zuge der Self-Assembly verbinden. Im Beispiel aus Abbildung 2.7 sind die Kleberbezeichner mit den Beschriftungen „A,B,C,D“ gegeben. Außerdem wird die Kleberstärke des Tiles durch ausgefüllte schwarze Boxen an den Seiten des Tiles dargestellt. Dabei entscheidet die Menge an schwarzen Boxen die Stärke des Klebers.

Die Verbindung zweier Tiles funktioniert wie folgt:

**Definition 12** Zwei Tiles  $t$  und  $t'$  binden korrekt bei Temperatur  $\tau$ , wenn folgende Bedingungen gelten:

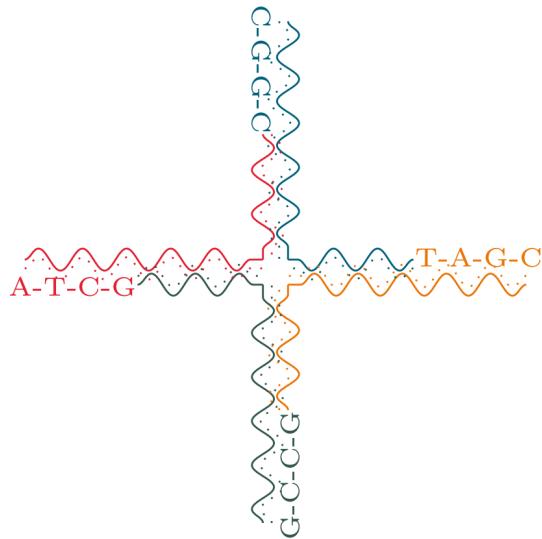
- (1)  $t$  und  $t'$  sind benachbart.
- (2)  $\exists u_i \in U_t : \text{label}(\text{glue}(t, u_i)) = \text{label}(\text{glue}(t', -u_i))$   
 $\wedge \text{strength}(\text{glue}(t, u_i)) \geq \tau \wedge \text{strength}(\text{glue}(t', -u_i)) \geq \tau$

Wenn eine der beiden Bedingungen (1) oder (2) verletzt wird, wird diese Verbindung Error oder false genannt.

Die Verbindung von mehreren Tiles wird *Assembly* genannt.

**Definition 13** Eine  $n$ -dimensionale Tile Assembly oder Assembly ist eine partielle Funktion  $\alpha : \mathbb{Z}^n \mapsto T$  mit  $T$  als Menge von  $n$ -dimensionalen Tiles. Eine Assembly  $\alpha$  ist  $\tau$ -stabil, wenn kein Tile  $t_n \in T$  aus der Assembly entfernt werden kann, ohne dafür Kleber der Stärke  $\tau \in \mathbb{N}$  entfernen zu müssen.

**Definition 14** Die Grenze einer Assembly  $\alpha$  ist eine Teilmenge von  $\alpha$ . Diese Menge enthält alle Tiles mit mindestens einem freien Nachbar-Tile.



**Abbildung 2.8.:** Biologisches Schema einer Holliday Junction, deren vier offenen Enden so gewählt wurden, dass sie sich oben und unten sowie rechts und links mit sich selbst verbinden kann.[27]

**Definition 15** Die Wachstumsfront einer  $n$ -dimensionalen Assembly  $\alpha$  ist eine Teilmenge von  $\mathbb{Z}^n$ . Eine Stelle ist nur dann Teil der Wachstumsfront, wenn folgende Bedingungen eintreffen:

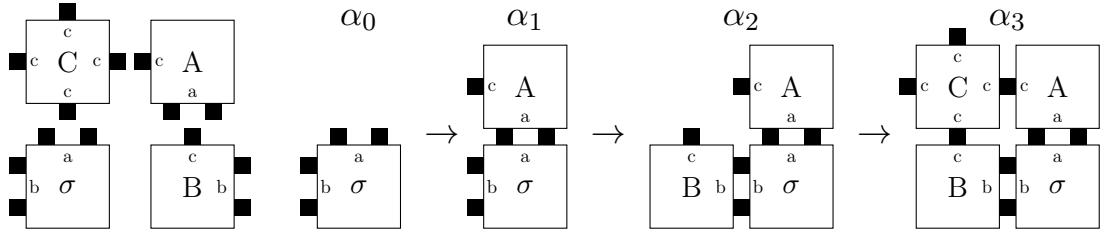
- (1) Die Stelle ist nicht belegt.
- (2) Die Stelle liegt an der Grenze der Assembly.
- (3) Die benachbarte Seite des Tiles an der Grenze hat eine minimale Kleberstärke 1.

In einer gegebenen Assembly ändern sich sowohl die Grenzlinie als auch die Wachstumsfront kontinuierlich durch die Prozesse des Hinzufügens und Entfernen von Tiles. Wenn eine Assembly zu einem Zeitpunkt keine offenen Enden mit einer Kleberstärke von größer oder gleich eins aufweist, dann existiert für diese Assembly keine Wachstumsfront. Trotz dieser dynamischen Veränderungen verfügt jede Assembly zu jedem Zeitpunkt über eine klar definierte Grenze.

Das erste Tile einer Assembly, bezeichnet als  $\alpha_0$  zum initialen Zeitpunkt  $t = 0$ , erhält die Bezeichnungen *Seed-Assembly* oder *Seed-Tile* und wird als  $\sigma$  gekennzeichnet. An der Wachstumsfront dieser Seed-Assembly kann die Anbindung weiterer Tiles auf einer nichtdeterministischen Basis erfolgen.

Im Folgenden werden die Modelle definiert, die in Kapitel 6 zur Simulation und Evaluation der Ergebnisse verwendet werden.

**Definition 16** Ein Tile Assembly Model (TAM) ist ein Tupel  $\mathcal{T}_\tau = (T, \sigma, \tau)$ , mit  $T$  als



**Abbildung 2.9.:** Beispiel Assembly von vier Tiles (links) in Sequenz bei Temperatur  $\tau = 2$ . Im ersten Schritt  $\alpha_0$  liegt nur die Seed-Assembly vor. In Schritten  $\alpha_1$  und  $\alpha_2$  binden sich nichtdeterministisch die Tiles „A“ und „B“ an der Seed-Assembly an. In Schritt  $\alpha_3$  bindet sich so das Tile „C“. Da bei Temperatur zwei keine stabile Verbindung von zwei „C“-Tiles möglich ist, ist  $\alpha_3$  das Resultat der Assembly.[27]

endliche Menge von Tiles (auch Tileset genannt),  $\sigma$  der Seed-Assembly und  $\tau \in \mathbb{N}$  als Temperatur des TAM.

**Definition 17**  $\mathcal{A}[\mathcal{T}]$  ist die Menge aller terminierten Assemblies, die als Ergebnis eines TAM in endlicher Zeit erreicht werden können. Eine Assembly  $\alpha \in \mathcal{A}[\mathcal{T}]$  gilt als terminiert ( $\alpha_\square$ ), wenn kein weiteres  $\tau$ -stabiles Tile in der Assembly hinzugefügt werden kann.

**Definition 18** Sei  $\alpha_i$  eine durch ein TAM definierte Assembly. Eine Assembly Sequenz ist eine Sequenz  $S = < \alpha_0, \alpha_1, \dots >$ , wobei  $\alpha_{i+1}$  die Assembly  $\alpha_i$  mit einem weiteren hinzugefügten Tile darstellt. Wenn die Sequenz  $s$  endlich ist, dann wird das letzte Element der Sequenz Ergebnis  $\alpha_\square$  oder terminiert genannt.

Aus diesen Definitionen kann so das erste Modell, das Abstract Tile Assembly Model (aTAM) definiert werden:

**Definition 19** Sei  $T$  eine Menge von Tiles, die nicht rotiert werden können. Sei  $\sigma$  eine Seed-Assembly. Dann ist das Tupel  $\mathcal{A} = (T, \sigma, \tau)$  ein Abstract Tile Assembly Model (aTAM) mit Temperatur  $\tau$ . In einem aTAM kann immer nur ein einzelnes Tile zu einem Zeitpunkt hinzugefügt werden. Des Weiteren können Tiles nicht wieder aus der Assembly entfernt werden.

Das aTAM ist das einfachste in dieser Arbeit betrachtete Modell der Tile-basierten Assembly. Als Beispiel für solch eine Assembly dient Abbildung 2.9. Zu sehen ist ein aTAM  $\mathcal{A} = (T, S, \tau)$  mit der Menge der Tiles  $T = \{\sigma, a, b, c\}$  und der Temperatur  $\tau = 2$ . Das Tileset ist links in der Abbildung dargestellt. Die Assembly Sequenz  $S = < \alpha_0, \alpha_1, \alpha_2, \alpha_3 >$  ist rechts zu erkennen. Da das Beispiel mit im aTAM durchgeführt wird, ist  $\alpha_3 = \alpha_\square$ , da bei einer Temperatur von zwei keine weiteren Tiles mit dem Bezeichner „C“ an der Wachstumsfront hinzugefügt werden können. Diese Verbindungen wären nicht  $\tau$ -stabil.

Durch die Annahme, dass sich keine Tiles wieder aus der Assembly entfernen können und Schritt für Schritt immer nur ein Tile hinzugefügt werden kann, werden einige Grenzfälle in aTAM nicht betrachtet. Im Beispiel aus Abbildung 2.9 könnte zum Beispiel durch Hinzufügen von drei „C“ Tiles, wieder eine stabile Verbindung gebildet werden. Diese Betrachtung gibt es in aTAM jedoch nicht. Dadurch gilt dieses Modellierungsverfahren als simpel, aber nicht realitätsnah. Ein Modellierungsverfahren, das näher an der Realität und näher am Verhalten von DNA im Vorgang der Self-Assembly liegt, ist das *Kinetic Tile-Assembly Model (kTAM)*. Durch diese Eigenschaft wird kTAM die größte Rolle bei der Simulation und Auswertung von Kommunikationsprotokollen in dieser Arbeit spielen. Im Gegensatz zu aTAM können sich in kTAM verbundene Tiles von der Assembly lösen. Außerdem ist es möglich, dass ein Tile sich an einer falschen Stelle bindet. Dies ist in aTAM nicht möglich. Um die Verbindung und Trennung von Tiles in kTAM darstellen zu können, muss jedoch erst einmal die *Forward Rate* und *Backward Rate* definiert werden.

**Definition 20** Sei  $\mathcal{T}_\tau$  ein TAM,  $\sigma$  ein nicht leere Seed-Assembly und  $T$  eine Menge an Tiles. Die Forward Rate  $r_f$ , mit welcher Tiles  $t$  in der Assembly hinzugefügt werden, wird wie folgt definiert:

$$r_f(t) = k_f e^{-G_{mc}}.$$

$k_f$  beschreibt hier das Timing des Systems,  $G_{mc}$  die benötigte Energie, um ein Tile zu binden (Binding Cost). Dabei gilt  $G_{mc} > 0$ .

**Definition 21** Analog zur Forward Rate ist die Backward Rate wie folgt definiert:

$$r_{r,b}(t) = k_f e^{-G_{se}}.$$

Sie beschreibt mit welcher Rate sich Verbindungen in einer Assembly wieder lösen.  $G_{se}$  beschreibt hierbei die benötigte Energie zum Lösen der Verbindung (Bond Breaking Cost). Durch  $b$  wird die Anzahl der Verbindungen des Tiles angegeben.

Daraus lässt sich im Weiteren das kTAM definieren:

**Definition 22** Sei  $\sigma$  eine nicht leere Seed-Assembly,  $T$  eine Tileset,  $r_{r,b}$  die Backward Rate und  $r_f$  Forward Rate. Dann ist ein Kinetic Tile Assembly Model wie folgt definiert:

$$\mathcal{K} = (T, \sigma, r_{r,b}, r_f)$$

Im kTAM leitet sich die Temperatur aus dem Verhältnis zwischen  $G_{mc}$  (Binding Cost) und  $G_{se}$  (Bond Breaking Cost) ab. Da die Temperatur selbst nicht explizit spezifiziert wird, wird sie implizit durch die zuvor definierten Forward und Backward Rates repräsentiert. In kTAM wird in jedem Schritt ein nicht deterministisch ausgewähltes Tile an einer nicht deterministisch festgelegten Stelle der Assembly hinzugefügt. Eine korrekte Positionierung ist dabei um den Faktor  $e^{G_{se}}$  wahrscheinlicher als eine falsche Positionierung. Außerdem ist bei falscher Positionierung die Wahrscheinlichkeit größer, dass durch die Backward Rate diese Verbindung wieder aufgelöst wird.

Ein weiteres realitätsnäheres Modell ist das *Two-Handed Tile-Assembly Model (2HAM)*. Im 2HAM gibt es keine Seed-Assembly. Solange die Temperaturbeschränkungen eingehalten werden, kann jedes Tile und jede Assembly in allen Schritten mit anderen Assemblies interagieren. Dadurch wird eine Art Potenzmenge aller möglichen Assemblies gebildet. Das Entfernen einer einzelnen Seed-Assembly, sowie das Verhalten beim Erzeugen einer Assembly macht dieses Modellierungsverfahren realitätsnäher. Für die formale Definition des 2HAM müssen zunächst einige Definitionen gegeben werden:

**Definition 23** Zwei Assemblies  $\alpha$  und  $\beta$  sind disjunkt, wenn für alle Stellen in  $\alpha$  und  $\beta$  gilt:  $\alpha \cap \beta = \emptyset$ . Eine Assembly besteht immer aus mindestens einem Tile.

**Definition 24** Der Zustand  $S$  eines Tilesets  $T$  ist eine Multimenge von Assemblies, für die Folgendes gilt:

1. Alle Assemblies  $\alpha \in S$  sind  $\tau$ -stabil. Es müssen  $\tau$  Kleber entfernt werden, damit sich die Assembly auflöst.
2. Alle Assemblies  $\alpha \in S$  können aus der Menge  $T$  durch korrekte Vereinigung gebildet werden.

Der Zustand  $S$  von  $T$  ist als Multimenge definiert, da eine Verbindung eines Tiles mit sich selbst möglich sein muss. So müssen einige Elemente mehrfach in der Menge vorkommen. Zwei Assemblies  $\alpha$  und  $\beta$  bilden eine *korrekte Verbindung*, wenn ihre Verbindung  $\tau$ -stabil ist und  $\alpha$  und  $\beta$  disjunkt sind.

**Definition 25** Ein Two-Handed Tile Assembly Model (2HAM) ist wie folgt definiert: Ein 2HAM ist ein Tupel  $\mathcal{H} = (T, S_0, \tau)$  mit dem Tilesset  $T$ , dem Startzustand  $S_0$  und der Temperatur  $\tau$ .

**Definition 26** Sei  $\mathcal{H} = (T, S_0, \tau)$  ein 2HAM. Die Assembly Sequenz eines 2HAM ist eine Sequenz von Zuständen  $S = < S_0, \dots, S_k >$  mit  $k = \{1, \dots, \infty\}$ .  $S_{i+1}$  wird aus  $S_i$  gebildet, indem alle möglichen Vereinigungen auf Assemblies  $s, s' \in S_i$  durchgeführt werden.

**Definition 27** Sei  $\mathcal{H} = (T, S_0, \tau)$  ein 2HAM. Ein Zustand  $S_i$  ist terminierend, wenn  $S_{i+1}$  nach allen möglichen Vereinigungen äquivalent zu  $S_i$  bleibt.

Alle Definitionen zu aTAM, kTAM und 2HAM bis hier stammen aus der Arbeit von Patitz [39] und der Arbeit von Lathrop et al.[25]. Alle jetzt folgenden Definitionen zum *Kinetic Two-Handed Tile-Assembly Model (kTHAM)* stammen aus der Arbeit von Kaussow in Zusammenarbeit mit Lau [21].

"Das Kinetic Two-Handed Tile-Assembly Model kombiniert die realitätsnahen Merkmale von 2HAM und kTAM. Analog zum 2HAM gibt es keine Seed-Assembly. Gleichzeitig, ähnlich wie im kTAM, können sich Verbindungen und Vereinigungen bilden, die nicht

$\tau$ -stabil sind, und bereits gebildete Verbindungen können in späteren Schritten wieder aufgelöst werden.

Dafür müssen wieder einige Definitionen aufgestellt werden.

**Definition 28** Eine korrekte Vereinigung von zwei Assemblies  $\alpha$  und  $\beta$  im kTHAM ist eine Vereinigung, in der  $\alpha$  und  $\beta$  disjunkt sind.

**Definition 29** Ein Zustand  $S = \{A_S, C_S\}$  eines Tilesets  $T$  ist ein Tupel von Multimengen von Assemblies und ein Vektor für ihre Nummer im kTHAM. Für das Tupel gilt Folgendes:

1. Alle Assemblies  $A_S$  können aus  $T$  durch korrekte Vereinigungen gebildet werden.
2. Alle Assemblies  $A_S$  sind einzigartig und ihre Häufigkeit des Auftretens ist durch den Vektor  $C_S$  definiert.
3. Die Zahl der Einträge in  $C_S$  ist gleich der Anzahl von Assemblies in  $A_S$ .

Die Forward Rate und Backward Rate sind analog zum kTAM.

**Definition 30** Ein Kinetic Two-Handed Tile-Assembly Model (kTHAM) ist ein Tupel  $\mathcal{K}_H = (T, S_0, r_f, r_{r,b})$  mit der Menge an Tiles  $T$ , dem Startzustand  $S_0$ , der Forward Rate  $r_f$  und der Backward Rate  $r_{r,b}$ .

**Definition 31** Eine Assembly Sequenz in kTHAM ist eine Sequenz  $S$  von Zuständen  $S = < S_0, \dots, S_k >$  mit  $k \in \{1, \dots, \infty\}$ . Ein Zustand  $S_{i+1}$  entsteht aus dem Zustand  $S_i$  durch Trennung oder korrekten Vereinigung von zwei Assemblies. Assemblies, die nur aus einem Tile bestehen, dürfen nicht weiter getrennt werden.

Das kTHAM ist durch diese Eigenschaften ein noch realitätsnäheres, jedoch rechenaufwendiges Modell. Dadurch, dass sowohl ganze Assemblies gebunden als auch wieder gelöst werden können, werden in kTHAM mehr Extremfälle betrachtet als in kTAM oder 2HAM.

Zur Behandlung von Extremfällen wird im Folgenden das Error-Handling betrachtet.

## 2.7. Error-Handling in Tile-basierten Self-Assembly Systemen

In dieser Sektion sollen mögliche Probleme und Fehler im Prozess der Tile-basierten Self-Assembly betrachtet werden. Dafür ist anzumerken, dass die mathematische Darstellung und Berechnung von Self-Assemblies einige Aspekte der Realität auslässt, um die Berechenbarkeit zu garantieren. So kann es im mathematischen Modell wie in Abbildung 2.7

Error	aTAM	kTAM	2HAM	kTHAM
Growth-Error	✓	✓	✓	✓
Facet-Error	✗	✓	✓	✓
Nucleation-Error	✗	✗	✓	✓

**Tabelle 2.1.:** Tabellarische Darstellung der vier Modellierungsverfahren von DNA-Tile-basierter Self-Assembly mit den drei typischen Errors. ✓ steht hierbei dafür, dass das TAM diesen Error modelliert, ✗ nicht.

nicht dazu kommen, dass das Tile rotiert. In diesem Betrachtungshorizont sind die Tiles einer Assembly immer fest ausgerichtet. Dies ist in der Natur nicht gegeben. Für realitätsnähere Betrachtungen müsste jedes Mal berechnet werden, in welcher Ausrichtung sich das Tile befindet. Außerdem müsste für die Evaluation der Ergebnisse beachtet werden, in welchem Kontext diese Assembly sich in der Realität zusammensetzen würde. Äußere Einflüsse könnten den Vorgang der Self-Assembly verändern.

In Self-Assembly Systemen können jedoch auch ohne diese weiteren Betrachtungen einige *Errors* entstehen, die im Folgenden vorgestellt werden sollen. Die Definitionen der Errors stammen alle aus der Arbeit von Lau et al. [27].

Es gibt drei typischerweise auftretende Errors in Self-Assembly Systemen. Nicht alle dieser Errors treten in jedem Modellierungsverfahren wie kTAM oder 2HAM gleichermaßen auf.

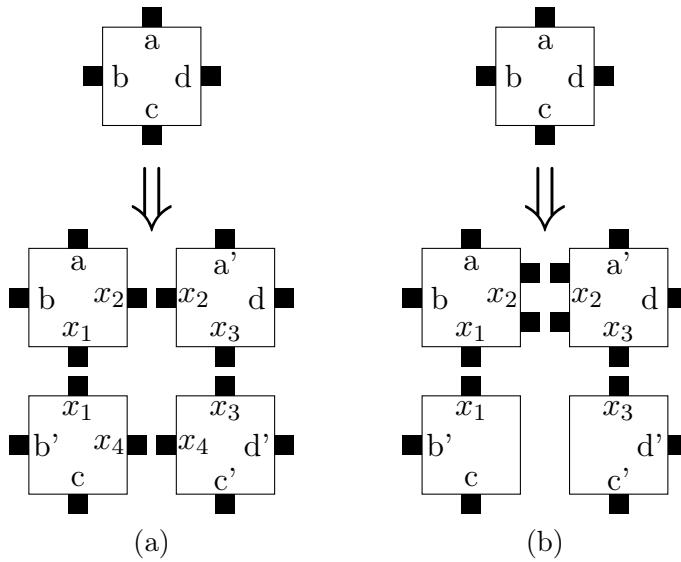
**Definition 32** Ein Growth-Error entsteht, wenn ein Tile sich in der Assembly an einer Stelle verbindet, an der mindestens einer seiner Kleber eine nicht korrekte Bindung mit einem benachbarten Kleber eingeht.

**Definition 33** Ein Facet-Error entsteht, wenn ein Tile sich zwar an einer korrekten Stelle mit korrekten Verbindungen zwischen Klebern verbindet, die Temperaturbeschränkungen jedoch nicht eingehalten wurden.

**Definition 34** Ein Nucleation-Error entsteht, wenn eine Assembly mit keinem oder einem anderen Tile als der Seed-Assembly startet.

In Tabelle 2.1 ist dargestellt, welche Errors von den vier zuvor definierten Modellierungsverfahren berücksichtigt werden.

Um eine Assembly nicht so anfällig für Growth- und Facet-Errors zu machen, kann ein Verfahren namens  $k \times k$ -Proofreading verwendet werden [10, 54]. Bei diesem Verfahren werden die Tiles aus der Menge der Tiles  $T$  und einer Assembly  $\alpha$  ausgetauscht. Dabei wird jedes einzelne Tile mit einem Block von  $k \times k$  mit  $k \in \mathbb{Z} \wedge k > 1$  Tiles ersetzt. Ein Beispiel für ein  $2 \times 2$ -Proofreading ist in Abbildung 2.10 (a) zu sehen. Dabei wird das obere Tile mit den vier Tiles darunter ausgetauscht. Bei der Ersetzung muss darauf geachtet werden, dass die logische Funktion des einzelnen Tiles erhalten bleibt. Durch



**Abbildung 2.10.:** Darstellung einer beispielhaften Tile-Ersetzung zur Implementierung präventiver Errorkorrektur. Dabei zeigt (a) das  $k \times k$ -Proofreading und (b) das Snaked-Proofreading.[27]

$k \times k$ -Proofreading wird ein Growth- oder Facet-Error nicht unmöglich. Es wird nur die Wahrscheinlichkeit reduziert, da mehrere Errors hintereinander passieren müssen.

Eine Erweiterung des  $k \times k$ -Proofreading ist das *Snaked-Proofreading* [10]. Dabei wird die innere Struktur des Blocks, der jedes Tile ersetzen soll, verändert. Der Ansatz des  $k \times k$ -Proofreading bleibt jedoch erhalten. Durch das Entfernen eines inneren Klebers im Inneren des Blocks werden Facet-Errors bei ersten Verbindungsversuchen eines neuen Blocks erheblich reduziert. Dafür muss die Kleberstärke der restlichen inneren Kleber möglicherweise angepasst werden. Dies ist in Abbildung 2.10 (b) zu erkennen. Beim Snaked-Proofreading muss der Block in einer festen Reihenfolge gebildet werden.

## 2.8. DNA-basierte Nanonetzwerke

Da alle Grundlagen für DNA-Tile-basierten Self-Assembly vorgestellt wurden, kann die folgende Definition aus [26] betrachtet werden:

**Definition 35** Sei  $\mathcal{T} = (T, \sigma, \tau)$  ein Tile Assembly System, wobei  $T$  eine endliche Menge an Tiles ist,  $\sigma$  eine Seed-Assembly und  $\tau \in \mathbb{N}^+$  die Temperatur des TAS. Ein DNA-basiertes Nanonetzwerk  $\mathcal{N}_\Phi$  ist ein Nanonetzwerk. Die Komponenten sind durch die Menge  $\mathcal{N}_\Phi = \mathcal{N}_{Se} \cup \mathcal{N}_R \cup \mathcal{M}_\Phi$  gegeben, wobei  $\mathcal{N}_{Se}$  eine Menge von DNA-basierten Nanosensoren und  $\mathcal{N}_R$  eine Menge von DNA-basierten Nanorobotern ist.  $\mathcal{M}_\Phi$  ist ein Tilesset für eine Menge von Nachrichtenmolekülen.

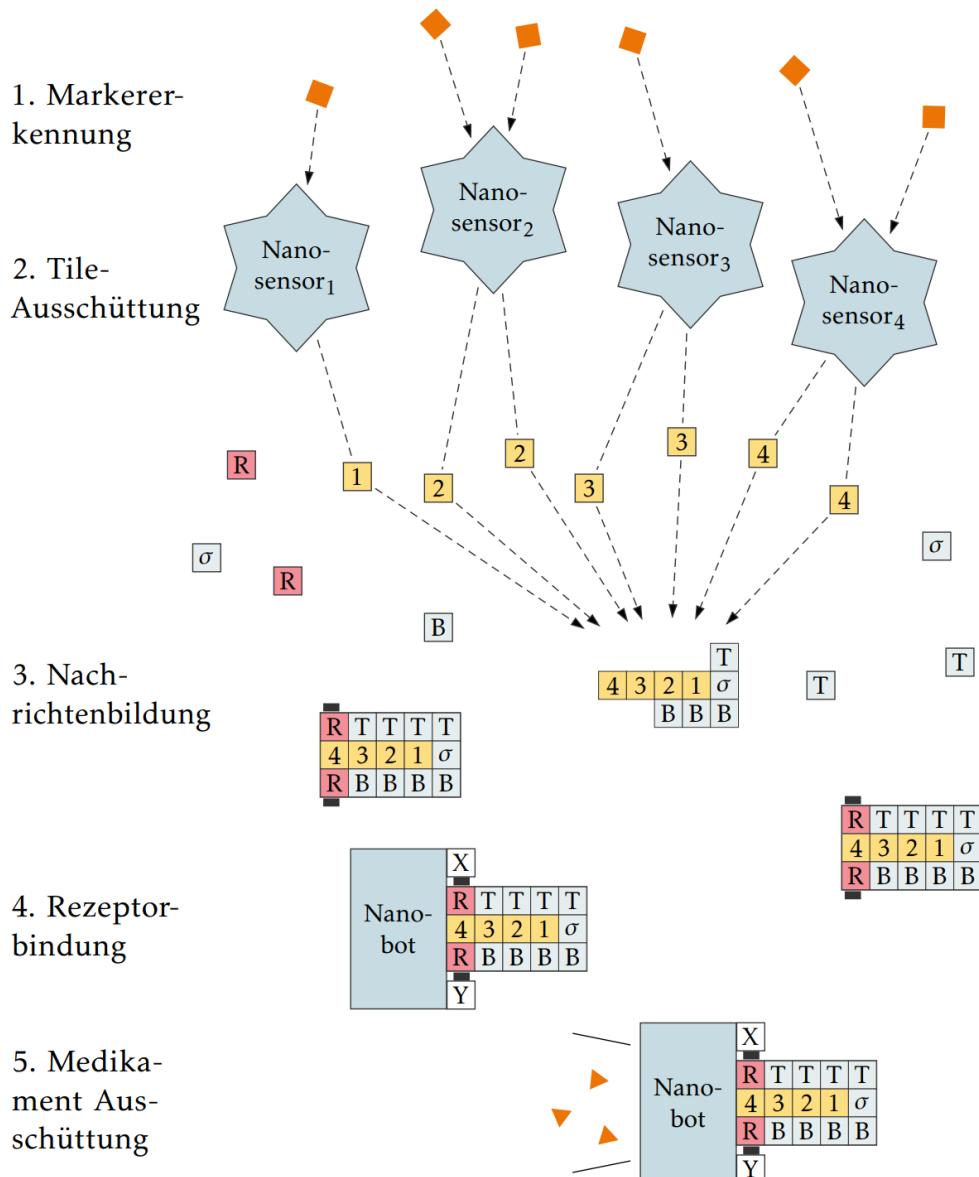
Bei dieser Definition von DNA-basierten Nanonetzwerken ist zu beachten, dass die Nanosensoren  $\mathcal{N}_{Se}$  und die Nanoroboter  $\mathcal{N}_R$  korrekt aus dem Tileset  $T$  gebildet werden können. Das muss aber nicht der Fall sein. In einigen Fällen kann es sinnvoller sein, die Sensoren und Roboter durch zum Beispiel DNA-Origami zu konstruieren. Diese Technik basiert nicht auf Tiles, sondern wie oben beschrieben auf langen DNA-Einzelsträngen und DNA-Klammern. Zusätzlich reduziert die eigenständige Konstruktion von Nanosensoren und Nanorobotern, beispielsweise mithilfe von DNA-Origami, die Komplexität des Tilesets  $T$ . Dementsprechend wird in dieser Arbeit der Fokus auf der Nachrichtenbildung und den dafür benötigten Tilesets liegen. Soweit nicht konkret von der Konstruktion von Nanosensoren oder Nanorobotern die Rede ist, wird im Folgenden bei jedem Tileset  $T$  für ein Nanonetzwerk  $\mathcal{N}_\Phi$  immer davon ausgegangen, dass sowohl die Nanosensoren  $\mathcal{N}_{Se}$  als auch die Nanoroboter  $\mathcal{N}_R$  konstruiert wurden und somit nicht im Tileset  $T$  betrachtet werden müssen.

Zur Terminologie muss gesagt werden, dass hier allgemein von *DNA-basierten Nanonetzwerken* gesprochen wird. Die Arbeit fokussiert sich jedoch auf die *DNA-Tile-basierten Nanonetzwerke*. Das bedeutet, dass beispielsweise keine konkreten Umsetzungen von DNA-Origami-basierten Mechanismen und Geräten geliefert werden. Manchmal ist es notwendig anzunehmen, dass ein solches Gerät existiert, jedoch soll der Fokus auf den DNA-Tiles liegen.

Des Weiteren werden noch zwei neue Bausteine für die Konstruktion und Auswertung eines Nachrichtenmoleküls in einem Nanonetzwerk benötigt: *Rezeptoren* und *Liganden*. Rezeptoren sind Teile eines Nanoroboters, die in der Lage sind, eine Verbindung mit Liganden einzugehen. Liganden sind spezielle Tiles, die sich nicht nur mit einem Nachrichtenmolekül, sondern auch mit Rezeptoren eines Nanoroboters verbinden können.

Ist das Nanonetzwerk konstruiert, läuft die Kommunikation in einem solchen Netzwerk allgemein wie folgt ab:

1. *Markererkennung*: Die Nanosensoren des Systems können durch spezifische Marker aktiviert werden. Die Art der Marker hängt von der Konstruktionsweise der Nanosensoren ab. Beispielsweise kann ein Nanosensor auch aus DNA-Origami gebildet werden. Durch eine Würfelform können im Inneren des Würfeln Tiles gespeichert werden. Bindet sich der Marker in Form eines DNA-Strangs, kann sich der Würfel auf einer Würfelseite öffnen.[11]
2. *Tile-Ausschüttung*: Hat ein Nanosensor einen Marker erkannt, so schüttet dieser die Tiles aus, die zur Kommunikation im Netzwerk verwendet werden.
3. *Nachrichtenbildung*: Die von den Sensoren freigesetzten Tiles verbinden sich durch Self-Assembly mit der bereits vorhandenen Seed-Assembly und den Liganden, um ein stabiles Nachrichtenmolekül zu bilden.
4. *Rezeptorbildung*: Wenn die Nachricht komplett konstruiert wurde und die Liganden sich am Rand des Nachrichtenmoleküls verbunden haben, kann sich das Nachrichtenmolekül mit diesen Liganden an den Rezeptoren eines Nanoroboters binden.



**Abbildung 2.11.:** Ein beispielhafter Ablauf eines Nanonetzwerks, das ein Medikament erst dann ausschüttet, wenn vier verschiedene Sensoren Marker registriert haben. Im ersten Schritt müssen Marker (hier orange) von den Nanosensoren erkannt werden, damit das System aktiviert wird. Darauf folgt der zweite Schritt, in dem die Tiles ausgeschüttet werden, die zur Nachrichtenbildung benötigt werden. Im dritten Schritt ist die Nachrichtenbildung dargestellt, die nur dann vollständig ist, wenn Tiles von allen vier Sensoren vorhanden sind. Im vierten Schritt binden sich die Liganden der Nachricht an die Rezeptoren eines Nanoroboters. Im fünften Schritt schüttet der Nanoroboter Medikamente aus, da er durch die Nachricht aktiviert wurde. [26]

5. *Nanoroboteraktivierung*: Wenn sich die Nachricht an den Rezeptoren des Nanoroboters gebunden haben, wird dieser aktiviert und erledigt die Aufgabe, für die der Nanoroboter konstruiert wurde.

Ein Beispiel für solch eine Kommunikation ist in Abbildung 2.11 gegeben. Hier soll ein Medikament ausgeschüttet werden, wenn vier verschiedene Sensoren ihre Marker erkennen. Dafür wird eine „AND“-Nachricht gebildet. Das Nachrichtenmolekül bildet sich nur komplett und bindet die Liganden, wenn die Tiles von allen vier Sensoren vorhanden sind. Diese von den Nanosensoren ausgeschütteten Tiles sind in diesem Beispiel mit „1“, „2“, „3“ und „4“ beschrieben. Die Liganden sind rot mit dem Buchstaben „R“ dargestellt, die Seed-Assembly durch das dafür typische  $\sigma$ . Ist das Nachrichtenmolekül vollständig konstruiert, so kann es sich an den Rezeptoren (hier mit  $X$  und  $Y$  dargestellt) eines Nanoroboter binden. Ist diese Verbindung fertig, wird der Nanoroboter aktiviert und lässt Medikamente aus, die dieser im Inneren gespeichert hatte.[26]

## 2.9. Kommunikation in Nanonetzwerken

Mit allen bis hier definierten Informationen können Nanonetzwerke erneut in Betrachtung gezogen werden. Diese Sektion wird sich auf die Kommunikation in Nanonetzwerken konzentrieren.

Die meisten Kommunikationsmethoden auf Nanoebene basieren auf *Diffusion*. Dies bezeichnet die zufällige und passive Verteilung von Partikeln in einem Medium. Zu den typischen Kommunikationsmethoden, die auf diesem Prinzip aufbauen, gehören das Messen der Partikelanzahl und die Bestimmung von Konzentrationen bestimmter Partikel. Dabei wird ein spezifischer Schwellenwert für die Partikelanzahl oder die Konzentration festgelegt, ab dem ein Ereignis als erreicht gilt. Wenn der Messwert den Schwellenwert überschreitet, kann dies darauf hinweisen, dass das Ereignis eingetreten ist. Es gibt jedoch Verfahren, die mehr Informationen übertragen können, aber dafür komplexere Partikelstrukturen und Messverfahren erfordern, wie das Messen von Partikeltypen und Partikelanordnungen. In solchen Fällen wird jeder Partikeltyp oder Partikelanordnung einer spezifischen Information zugeordnet. Sobald ein solcher Partikeltyp oder eine solche Anordnung durch Messungen erkannt wird, kann die entsprechende Information extrahiert werden.

In dieser Arbeit sind Tiles jedoch weiterhin die relevanteste Kommunikationsmethode. Tiles übertragen weniger Information in einem Molekül im Vergleich zum Erfassen von Partikeltypen oder -anordnungen. Allerdings ist die Bioinvasivität von Tiles deutlich geringer als bei Verfahren, die regelmäßige Messungen erfordern.[14]

Um Kommunikation mit Tiles ermöglichen zu können, müssen einige mathematische und logische Probleme algorithmisch mit Tiles gelöst werden können. Beispiele für einige mathematische Probleme sind in Tabelle 2.2 zu sehen. Diese Probleme können alle durch Self-Assembly von Tiles gelöst werden. Abbildung 2.12 zeigt beispielhaft ein Tileset sowie

Problem	Signatur	Beschreibung
ADD	$\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$	Integer-Addition
EQ	$\mathbb{Z} \times \mathbb{Z} \rightarrow \{0, 1\}$	Integer-Vergleich
GEQ	$\mathbb{Z} \times \mathbb{Z} \rightarrow \{0, 1\}$	Integer-Vergleich $\geq$
LEQ	$\mathbb{Z} \times \mathbb{Z} \rightarrow \{0, 1\}$	Integer-Vergleich $\leq$
SUB	$\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$	Integer-Subtraktion
MULT	$\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$	Integer-Multiplikation
DIV	$\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$	Integer-Division

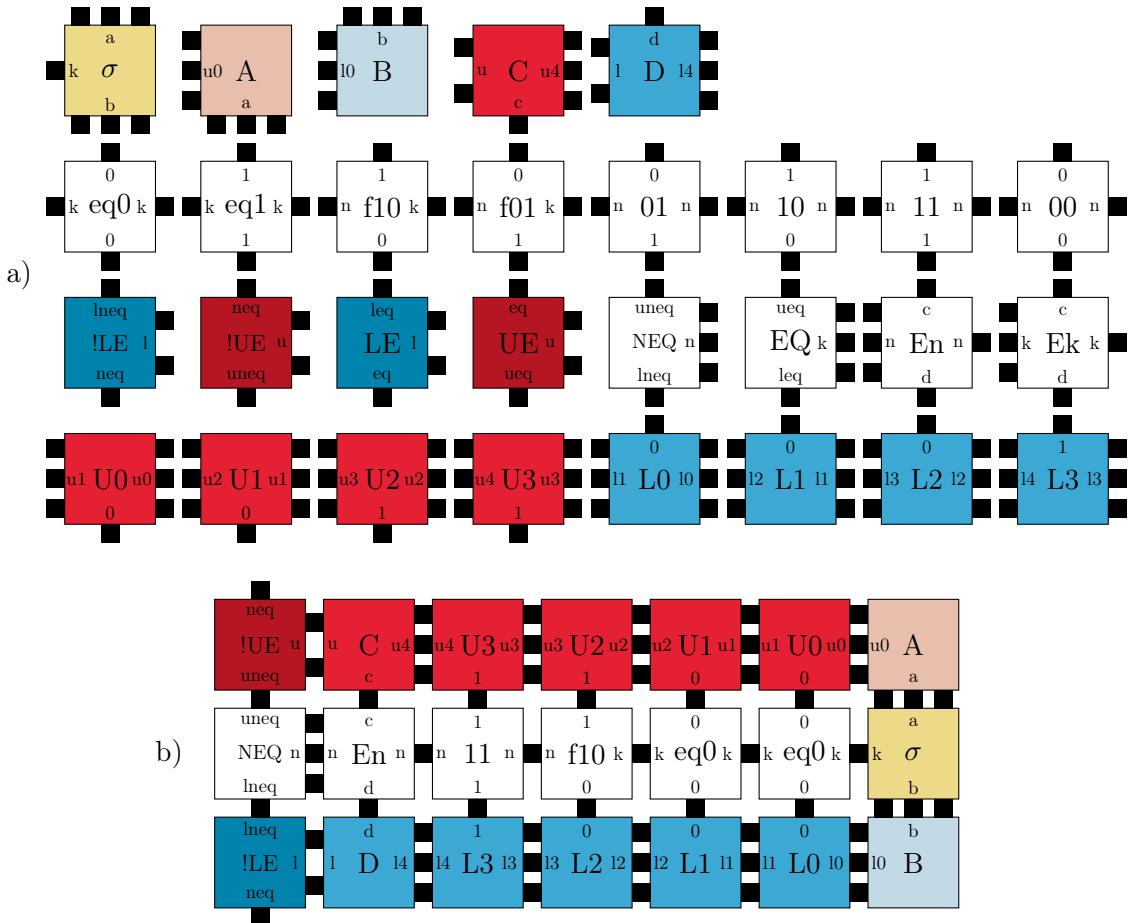
**Tabelle 2.2.:** Beispielhafte formale Definition für einige Probleme, die in Nanonetzwerken von Interesse sind. Dabei wird davon ausgegangen, dass die ganzen Zahlen binär als  $\{0, 1\}^k, k \in \mathbb{N}^+$  dargestellt werden.[26]

die resultierende Assembly für das Problem *EQ*, das den Äquivalenzvergleich zweier 4-Bit Integer darstellt. Dabei muss für die korrekte Konstruktion eine Temperatur von  $\tau = 3$  angenommen werden.

Das Tileset und die Self-Assembly folgen dabei dem Farbschema, das im Kapitel 5 definiert und benötigt wird [7]. Es wird jedoch in allen Abbildungen verwendet, da es eine einheitliche Darstellung für Assemblies bietet und neben dem später vorgestellten Nutzen auch so etwas mehr Übersicht bietet und es einfacher macht, die Abbildungen zu beschreiben. Das gelbfarbige Tiles ist dabei die Seed-Assembly  $\sigma$ . Die roten sowie blauen Tiles markieren die nördliche sowie südliche Grenze der Assembly. Dabei stellen das dunkelrote sowie dunkelblaue Tile die Liganden der Assembly dar. Das hellrote sowie hellblaue Tile stellt wiederum den „Start“ der Grenze nördlich beziehungsweise südlich der Seed-Assembly dar.

In der Abbildung 2.12 werden die zwei zu vergleichenden Binärzahlen in den inneren Kleberbezeichnern der Tiles  $U_0, U_1, U_2$  und  $U_3$  für die erste Zahl und  $L_0, L_1, L_2$  und  $L_3$  für die zweite Zahl dargestellt. In diesem Beispiel werden die Binärzahlen 1100 und 1000 auf Äquivalenz geprüft. Durch die Assembly der Tiles wird die Äquivalenz vom *Least Significant Bit* (LSB) zum *Most Significant Bit* (MSB) geprüft. An den Ziffernstellen von  $U_0$  und  $L_0$  sowie von  $U_1$  und  $L_1$  ist die Berechnung noch äquivalent und es bindet sich an beiden Stellen das Tile *eq0*. Da der südliche Kleber von  $U_2$  das Label „1“ und der nördliche Kleber von  $L_2$  das Label „0“ hat, kommt es hier zur Verbindung des Tiles *f10*. Der weitere Aufbau des Moleküls folgt so nicht mehr durch weitere *eq0*- oder *eq1*-Tiles, da sich der horizontale Kleber in der Berechnung von „k“ auf „n“ ändert. Somit bindet sich das Tile *neq*. Dieses bindet nur die Liganden, die ein „neq“ in den Kleberbezeichnern weitergeben. Eine Konstruktion, wie sie in diesem Beispiel gegeben ist, funktioniert nur in aTAM fehlerfrei. Für kTAM, 2HAM und kTHAM könnte es sich anbieten, Snaked-Proofreading auf der Assembly zu verwenden, um Growth- oder Faceterrors zu minimieren.

Genau wie für das hier vorgestellte Äquivalenzproblem können mit dem richtigen Tileset bei einer festgelegten Temperatur  $\tau$  einige mathematische und logische Probleme gelöst



**Abbildung 2.12.:** Darstellung von a) dem Tileset und b) der resultierende Self-Assembly für das Problem *EQ*, dem Äquivalenzvergleich zweier Integer aus Tabelle 2.2. In diesem Beispiel für 4-Bit Zahlen, lässt sich diese Konstruktion jedoch analog für beliebig lange Binärzahlen wiederholen. Das Farbschema bietet eine bessere Übersicht, wird jedoch erst im Kapitel 5 notwendig. Die Binärzahlen in der Self-Assembly sind in den südlichen Kleberbezeichnern der Tiles  $U0$ ,  $U1$ ,  $U2$  und  $U3$  sowie in den nördlichen Kleberbezeichnern der Tiles  $L0$ ,  $L1$ ,  $L2$  und  $L3$  dargestellt. Die zentralen Tiles werden zur Berechnung des Problems benötigt. Von der Seed-Assembly  $\sigma$  wird horizontal ein „ $k$ “ in den Kleberbezeichnern weiter gegeben. Sobald eine Ziffernstelle nicht äquivalent ist, wird statt einem „ $k$ “ ein „ $n$ “ weitergegeben. Da in diesem Beispiel die Binärzahlen 1100 (rot) und 1000 (blau) auf Äquivalenz getestet werden, ist das Ergebnis *neq* (not equal).

werden. In Tabelle 2.2 sind nur einige interessante Probleme definiert. Eine ausführlichere Aufzählung von Problemen, die in Nanonetzwerken von Interesse sein können, findet sich in der Arbeit von Lau [26].

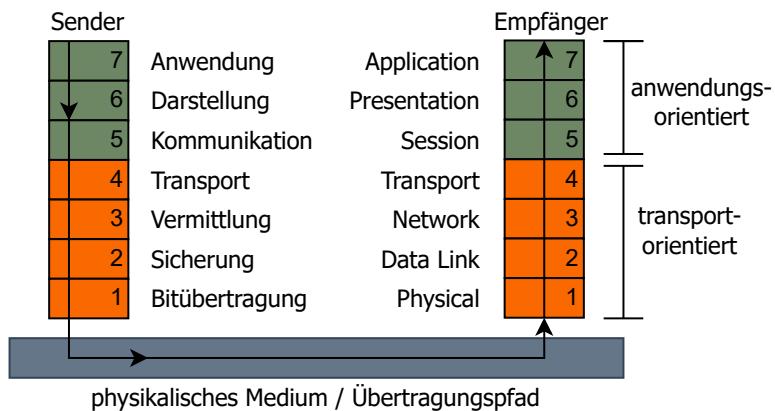
Einige Routing-Protokolle lassen sich mittels Tilesets und den damit implementierten mathematischen und logischen Problemen abstrahieren. Das so vorhandene Potenzial wurde in unterschiedlichen wissenschaftlichen Arbeiten vorgestellt. Beispielsweise bedarf die in [9] beschriebene Implementierung des Hop-Count-Routings der Grundoperationen *ADD*, *GEO*, *LEQ* und *EQ*. Dies veranschaulicht die Anwendbarkeit dieser Methodik und die Möglichkeiten der Problemreduktion. Einen weiteren Ansatz zeigt [47], in dem das Hop-Count-Routing mit noch weniger Grundoperationen umgesetzt wird. In [31] hingegen wird ein komplexeres Routing-Protokoll vorgestellt, das eine größere Anzahl an Operationen benötigt. Alle drei Protokolle sind im Related Work Kapitel 3 genauer vorgestellt.

Diese Arbeit fokussiert sich jedoch nicht auf die konkreten mathematischen oder logischen Operationen. Eher werden Mechanismen aus herkömmlichen Kommunikationsprotokollen betrachtet und auf Nanoebene übersetzt. Dafür wichtig ist die *Internationale Organisation für Normung* (ISO). Diese hat ein universelles Rahmenwerk für Kommunikationsprotokolle etabliert, das als das *Open Systems Interconnection* (OSI)-Modell bekannt ist. Dieses Modell besteht aus sieben Schichten, die in zwei übergeordnete Kategorien unterteilt sind: die *anwendungsorientierten Schichten* (Schichten 5-7) und die *transportorientierten Schichten* (Schichten 1-4). [15]

Bei einer Kommunikation zwischen einem Sender und einem Empfänger, die ein dem ISO/OSI-Modell folgendes Kommunikationsprotokoll verwenden, werden diese sieben Schichten sowohl auf der Sender- als auch auf der Empfängerseite durchlaufen. Die sieben Schichten des ISO/OSI-Modells sind wie folgt:

1. Bitübertragungsschicht
2. Sicherungsschicht
3. Vermittlungsschicht
4. Transportschicht
5. Kommunikationsschicht
6. Darstellungsschicht
7. Anwendungsschicht

Jede Schicht des ISO/OSI-Modells bietet spezifische Dienste an und zusammen ermöglichen sie die Kommunikation und den Datenaustausch zwischen unterschiedlichen Systemen. Die Struktur und der Datenfluss durch diese Schichten sind in Abbildung 2.13 dargestellt. Das ISO/OSI-Modell ist ein weit verbreiteter Standard für Netzwerkprotokolle. Während nicht alle Kommunikationsprotokolle strikt dem ISO/OSI-Modell folgen,



**Abbildung 2.13.:** Darstellung des ISO/OSI-Modells, das eine standardisierte Kommunikation zwischen einem Sender und einem Empfänger definiert. Beim Sender werden die Schichten des Modells von Anwendung, Darstellung und Kommunikation erst anwendungsorientiert durchlaufen. Danach folgen die Schichten von Transport, Vermittlung, Sicherung und Bitübertragung im transportorientierten Teil. Die Nachricht wird dann über das Medium an den Empfänger geschickt und in genau umgedrehter Reihenfolge entgegengenommen. Angelehnt an [38].

verwenden zahlreiche Protokolle einige der im Modell definierten Prinzipien. Dementsprechend werden in den folgenden Untersektionen die Schichten und ihre Funktion für Nanonetzwerke vorgestellt. Dabei lassen sich alle folgenden Informationen zum ISO/OSI-Modell aus dem Buch von Tanenbaum entnehmen. [46]

### 2.9.1. Die anwendungsorientierten Schichten

In dieser Untersektion sollen die anwendungsorientierten Schichten des ISO/OSI-Modells näher betrachtet werden.

Dabei ist die Anwendungsschicht im ISO/OSI-Modell die Schicht, die am nächsten an der Endbenutzer-Interaktion liegt. Sie ermöglicht es Anwendungen, auf das Netzwerk zuzugreifen und Dienste zu nutzen. Diese Schicht stellt Funktionen wie Identitätsüberprüfung, Zugangskontrolle und Ressourcenallokation zur Verfügung.

Die Darstellungsschicht im ISO/OSI-Modell kümmert sich im Wesentlichen um die Umwandlung der Daten in eine Form, die von der Anwendungsschicht verstanden werden kann. Es geht darum, wie Daten dargestellt, verschlüsselt und komprimiert werden.

Die Kommunikationsschicht spielt im ISO/OSI-Modell eine entscheidende Rolle, da sie den Datentransfer zwischen verschiedenen Netzwerken koordiniert. Sie ist zuständig für verschiedene Funktionen, wie das Routing und die Netzwerkadressierung.

Die genaue Implementierung der Mechanismen aus den anwendungsorientierten Schichten wird im Kapitel 4 vorgestellt. Hier gibt es jedoch einige grundlegende Mechanismen, die betrachtet werden können.

Da die Anwendungsschicht durch die Benutzernähe meist eine hohe Komplexität aufweist, kann es sinnvoll sein, die Mechanismen dieser Schicht auf Mikro- statt auf Nanoebene durchzuführen. Am Beispiel eines In-Body-Netzwerkes kann die Aufgabe der Datenverarbeitung und der Ressourcenallokation in einem Body-Area-Netzwerk auf Mikroebene stattfinden, während das In-Body-Netzwerk sich um die niedrigeren Schichten des Modells kümmert. Dass sich unterschiedliche Netzwerktypen und Geräte um unterschiedliche Schichten des ISO/OSI-Modells kümmern, ist auch in herkömmlichen Netzwerken üblich.[38]

Einige Mechanismen der Darstellungsschicht sind analog in der Implementierung von Tile-basierter Self-Assembly enthalten. Ein DNA-Strang ist eine Sequenz der zwei verschiedenen Basenpaare Adenin/Thymin und Guanin/Cytosin. So kann in einem offenen DNA-Strang, ähnlich zu einer binären Zahl, Information gespeichert und gelesen werden. Die Information, die am Ende in der Anwendungsschicht ausgelesen und genutzt werden soll, kann so in der Darstellungsschicht in die offenen DNA-Enden „programmiert“ beziehungsweise codiert werden.

Auch einige Mechanismen der Kommunikationsschicht sind inhärent in Tile-basierten Self-Assemblies vorhanden. So können beispielsweise die offenen DNA-Enden der Liganden mit Adressen gleichgesetzt werden. Denn nur bestimmte Liganden können sich mit bestimmten Rezeptoren des Nachrichtenempfängers verbinden.

Ein möglicher physikalischer Prozess zur Fortbewegung ist die zuvor vorgestellte Diffusion, die beispielsweise in In-Body-Netzwerken durch den Blutfluss erzeugt wird. Andere Optionen könnten die Fortbewegung durch Flagellen oder durch molekulare Motoren sein [26]. Wie die Mechanismen dieser Schichten in DNA-Tile-basierten Nanonetzwerken funktionieren, wird im Kapitel 4 diskutiert.

### 2.9.2. Die transportorientierten Schichten

Diese Untersektion beschäftigt sich mit den transportorientierten Schichten des ISO/OSI-Modells. Sie werden in herkömmlichen Kommunikationssystemen meist von unterschiedlichen Netzwerken oder Geräten implementiert. Dementsprechend kann für ein DNA-Tile-basiertes Nanonetzwerk nur schwer definiert werden, auf welcher Schicht der Mechanismus konkret umgesetzt wird.

Die Transportschicht des ISO/OSI-Modells ist für die End-to-End-Kommunikation und die Kontrolle des Datenflusses zwischen zwei Systemen verantwortlich.

Die Vermittlungsschicht, auch bekannt als die Datensicherungsschicht, ist im ISO/OSI-Modell dafür verantwortlich, die Kommunikation zwischen Geräten in einem Netzwerk zu

## 2. Grundlagen

---

kontrollieren und zu ermöglichen. Sie kümmert sich um Aufgaben wie Framing, physische Adressierung, Flusskontrolle und Fehlerkontrolle.

In der traditionellen Netzwerktechnologie stellt die Sicherungsschicht (Link Layer oder Data Link Layer im ISO/OSI-Modell) sicher, dass Daten sicher und fehlerfrei von einem Knoten zum anderen übertragen werden. Sie ist verantwortlich für die Erkennung und möglicherweise auch für die Korrektur von Fehlern, die auf der physischen Ebene auftreten können.

Die Bitübertragungsschicht, auch bekannt als die physikalische Schicht im ISO/OSI-Modell, ist für die Übertragung von rohen Bitströmen über das physische Medium zuständig. Sie befasst sich mit den technischen Aspekten der Übertragungsmedien wie Verbindungsaufbau, Verbindungstrennung, Modulation und Bitrate.

Wie zuvor angemerkt, verschwimmen die vier transportorientierten Schichten in DNA-Tile-basierten Nanonetzwerken etwas. Mechanismen zur Fehlerkorrektur sind mit dem  $k \times k$ -Proofreading und dem Snaked-Proofreading in der Sektion 2.7 vorgestellt worden. Mechanismen wie die Datenflusskontrolle, Framing, Fehlererkennung oder Modulation, werden im Kapitel 4 näher betrachtet.

Dieses Kapitel hat umfassende Grundlagen von Nanonetzwerken, DNA, Tilebildung, Self-Assembly, Assembly-Modellen, Error-Handling und Kommunikationsmechanismen gebildet und geliefert. Im folgenden Kapitel werden drei Arbeiten näher betrachtet, da diese besondere Nähe zu dieser Arbeit besitzen.

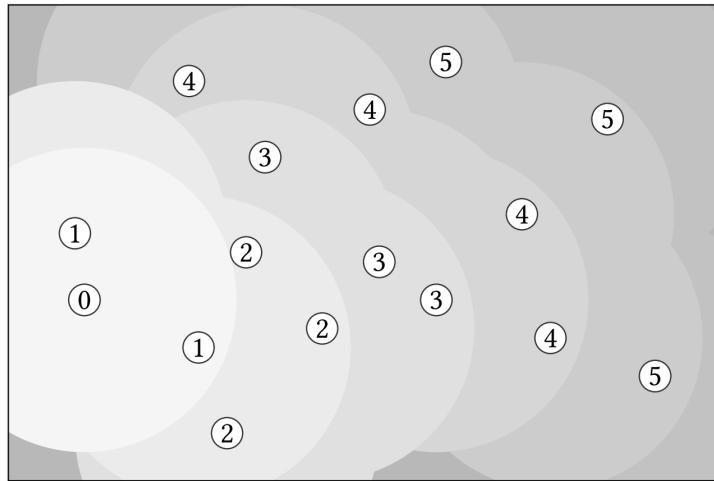
## 3. Related Work

Dieses Kapitel betrachtet drei wissenschaftliche Publikationen, die dieser Arbeit nahe stehen. Dabei handelt es sich um die Arbeiten *A Deployable Routing System for Nanonetworks* von Liaskos et al., *CORONA: A Coordinate and Routing system for Nanonetworks* von Tsoliaridou et al. und *Hop Count Routing: A Routing Algorithm for Resource Constrained, Identity-Free Medical Nanonetworks* von Büther et al., die alle verschiedene Ansätze für Routing und Koordination von Nanonetzwerken liefern [9, 31, 47]. Alle drei Arbeiten wurden kurz im Kapitel 2 vorgestellt. In den folgenden Sktionen sollen diese Arbeiten jedoch noch näher betrachtet und vorgestellt werden, da sie von besonderem Interesse für diese Arbeit sind.

### 3.1. Hop Count Routing: A Routing Alogrithm for Resource Constraint, Identity-Free Medical Nanonetworks

Die zuerst betrachtete Arbeit wurde im Jahr 2018 von Florian Büther, Immo Traupe und Sebastian Ebers verfasst [9]. Sie wurde mit besonderem Augenmerk auf *In-Body-Networks* (IBN) und *Body-Area-Networks* (BAN) verfasst. Die Nanogeräte in einem solchen Netzwerk unterliegen aufgrund des medizinischen Anwendungsfalls einigen Annahmen. So wird angenommen, dass ein Nanogerät in einem solchen Netzwerk verschiedene Komponenten besitzt: Eine Energieversorgung, eine einfache Recheneinheit, Sensoren und Aktuatoren, die eine aktive Kommunikation über elektromagnetische Terahertzwellen ermöglichen. Da sich diese Nanogeräte jedoch im menschlichen Körper befinden, können Terahertzwellen nur auf einer Entfernung von ungefähr zwei Millimetern zuverlässig erkannt werden. Aus diesem Grund müssen die Nanogeräte ein Netzwerk aus vielen identischen Einheiten bilden. So kann der Hop-Count genutzt werden, um ein System zu entwickeln, das sowohl in Bezug auf Energie als auch Berechnung effizient ist. Das in dieser Arbeit vorgestellte Hop-Count-Routing ist Gateway-orientiert. Es gibt ein Gateway auf Mikroebene, das Daten in das Nanonetzwerk sendet oder empfängt. Dabei gibt es zwei Phasen, die zum Routing benötigt werden: die Propagierungsphase und die Routingphase.

In der Propagierungsphase erhalten alle Nanoknoten im System zu Beginn den Hop-Count  $\infty$ . Das Gateway sendet eine Nachricht in das Netzwerk mit dem eigenen Hop-Count 0. Alle Empfängerknoten mit Hop-Count  $n_r$  und einer beliebigen Nachricht mit Hop-Count  $n_s$  verfahren in der Propagierungsphase wie folgt: Wenn  $n_r > n_s + 1$  gilt, dann



**Abbildung 3.1.:** Grafische Darstellung eines gatewayorientierten Hop-Counts nach der Propagierungsphase. Dabei wird die Reichweite einzelner Knoten durch verschiedene Helligkeitsstufen dargestellt. Die Zahl, mit welcher ein Knoten bezeichnet ist, stellt dabei die Anzahl der Hops zwischen Gateway und Knoten dar.

aktualisiere den eigenen Hop-Count auf  $n_s + 1$  und sende den aktualisierten Wert in einer neuen Nachricht im Broadcast weiter. Wenn  $n_r \leq n_s + 1$  gilt, dann mache nichts weiter. Da zu Beginn alle Knoten den Hop-Count  $\infty$  besitzen, wird jeder Knoten im Netzwerk mindestens einmal aktualisiert und sendet den Hop-Count weiter. Das Ergebnis einer solchen Propagierungsphase ist in Abbildung 3.1 zu erkennen. Haben so alle Knoten den minimalen Hop-Count zum Gateway berechnet, so kann die Routingphase durchgeführt werden.

Während der Routingphase erfolgt eine Unterscheidung zwischen zwei Arten von Nachrichten. Die erste Art umfasst Nachrichten, die von einem Nanoknoten zum Gateway gesendet werden. In diesem Fall muss die Nachricht per Broadcast mit dem aktuellen Hop-Count versehen werden. Alle Knoten überprüfen, wenn sie eine Nachricht empfangen, ob ihr eigener Hop-Count kleiner ist als der in der Nachricht angegebene Hop-Count. Wenn dies der Fall ist, leitet der Knoten die Nachricht mit seinem eigenen Hop-Count weiter. Andernfalls wird nichts weiter unternommen. Die zweite Art von Nachrichten besteht aus Nachrichten, die vom Gateway an einen spezifischen Nanoknoten gesendet werden. In diesem Fall erfolgt eine ähnliche Überprüfung, bei der abgefragt wird, ob der eigene Hop-Count größer ist als der empfangene Hop-Count.

Bei diesem Hop-Count-Routing werden eine große Anzahl von Nanoknoten in jeder Übertragung beteiligt. Eine Optimierung, die diese Überlastung des Netzes verhindern soll, ist die *destruktive Erfassung*. Dabei wird durch Änderung des Hop-Counts sichergestellt, dass alle Knoten nur einmal die Nachricht übertragen. Das führt dazu, dass Nachrichtenübertragungen in einem solchen Netzwerk nur noch sequentiell abläuft und nach einer Nachricht eine neue Propagierungsphase notwendig ist. Dieser Ansatz kann auch da-

zu führen, dass Übertragungen von Nanoknoten zum Gateway nicht ankommen, da die Knoten um das Gateway herum durch eine Nachricht zuvor genutzt wurden. Eine mögliche Lösung dieses Problems wäre, mehrere Gateways in einem System zu nutzen und alle Knoten den Hop-Count zu jedem Gateway speichern zu lassen. Hier würden aber erhöhte Speicher Kosten und weitere Flags in der Nachricht anfallen.

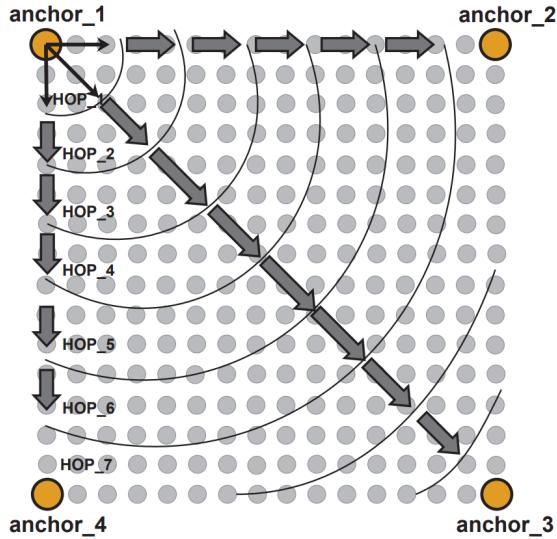
Bei der Auswertung des naiven und destruktiven Ansatzes für das Hop-Count-Routing kommen die Autoren der Arbeit zu dem Schluss, dass beide Ansätze schnell sind. Auch wenn der destruktive Ansatz eine geringere Nachrichtenmenge benötigt, ist der naive Ansatz ebenfalls einfach und schnell. Der worst-case von  $\mathcal{O}(n^2)$  ist in realistischen Implementierungen unwahrscheinlich. Die Laufzeit des Algorithmus ist in der Simulation gut, jedoch wurden einige Annahmen gemacht, die in der Praxis erneut betrachtet werden müssen. In den Simulationen wird angenommen, dass die Nanoknoten statisch immer an derselben Stelle liegen. In einem medizinischen Anwendungsfällen, in welchem dieses Netzwerk im menschlichen Körper liegt, ist dies wahrscheinlich nicht der Fall. Außerdem wird beim Modell des Nanoknotens nicht betrachtet, wie viel Zeit die Übertragung und Berechnung im Knoten selbst benötigt.

Abschließend ist zu dieser Arbeit zu sagen, dass ein interessanter und schneller Hop-Count-Routing Algorithmus geliefert wird, der in Hinsicht auf medizinische Body-Area-Networks entwickelt wird. Da einige vereinfachte Annahmen im Modell enthalten sind, muss sich ein solcher Ansatz jedoch noch in einem praxisnäheren Experiment bewähren.

Da sich die Arbeit auf Routing mit Terahertzwellen fokussiert, lässt sich der Algorithmus nicht auf DNA-Tiles und Self-Assemblies übertragen. Das Prinzip eines Body-Area-Networks, das die komplexeren Aufgaben des Netzwerkes übernimmt, ist jedoch interessant in Hinsicht auf die in dieser Masterarbeit betrachteten Nanonetzwerke. Ein Netzwerk auf Mikroebene, das über Gateways Tiles in ein Nanonetzwerk sendet und so mit Nanogeräten und Nanorobotern kommuniziert, stellt ein realistisches Anwendungsbeispiel für DNA-basierte Nanonetzwerke dar.

## 3.2. CORONA: A Coordinate and Routing system for Nanonetworks

In dieser Arbeit wird das koordinatenbasierte Routing-System CORONA vorgestellt. Das aus dem Jahr 2015 stammende Paper wurde von Ageliki Tsoliaridou, Christos Liaskos, Sotiris Ioannidis und Andreas Pitsillides verfasst [47]. Das Routing-System basiert auf zwei Annahmen: Das Nanonetzwerk, in welchem es eingesetzt wird, ist vertrauenswürdig und benötigt keine weiteren Sicherheitsmechanismen und die Knoten befinden sich im zweidimensionalen Raum. Auch wird in dieser Arbeit davon ausgegangen, dass ein Knoten in diesem Netzwerk eine Nanomaschine ist, die aus einer Energiequelle, einem Speicher, einer Antenne und einer Recheneinheit besteht. Dabei können alle Knoten autonom einfache Operationen ausführen und über kurze Strecken kommunizieren.



**Abbildung 3.2.:** Darstellung eines rechteckigen Gebiets in einem Nanonetzwerk. Die vier Eckpunkte des Gebiets bilden die Ankerpunkte und definieren über die Entfernung zu den einzelnen Knoten über Hops die eindeutigen Positionen jedes Nanoknotens im Gebiet. Abbildung aus der Arbeit von Tsoliaridou et al. [47]

Das System basiert auf der Idee, dass Nanonetzwerke in den meisten Anwendungsfällen vorwiegend datenzentriert funktionieren. Ein Ansatz aus einer früheren Arbeit, der einen wiederholten Broadcast aller Nachrichten durchführt, soll in dieser Arbeit durch ein Unicast-basiertes System ersetzt werden [30]. Die Arbeit legt den Fokus bei Nanomaschinen auf *Software-Defined Metamaterials* (SDM), einer neuen Klasse von künstlichen Materialien mit programmierbaren elektromagnetischen Eigenschaften [29]. Mit diesen Grundlagen werden in einem rechteckigen Gebiet Nanoknoten platziert. Die vier Ecken des Rechtecks werden dabei als Ankerpunkte definiert. Auf Basis von Standard-Triangulation können so allen Nanoknoten in diesem Gebiet eindeutige Koordinaten gegeben werden, indem die Entfernung über Hops zwischen dem Knoten und den Ankerpunkten definiert wird. Dieser Vorgang wird während der Einrichtungsphase einmalig durchgeführt. Das System ist in Abbildung 3.2 beispielhaft dargestellt.

Da sich das System im zweidimensionalen Raum befindet und die vier Ankerpunkte an den Eckpunkten des Gebietes liegen, reichen die zwei nächsten Ankerpunkte, um durch Triangulation die genaue Position eines Knotens zu definieren. Jede andere mögliche Position wäre außerhalb des Gebietes. Bei der Übertragung eines Pakets von einem Senderknoten zu einem Empfängerknoten entscheidet der Senderknoten basierend auf der geringsten Anzahl von Hops, welches Ankerpunktpaar für das Routing verwendet werden soll. Die Übertragung einer Nachricht von einem Senderknoten  $S$  mit den Koordinaten  $(s_1, s_2, s_3, s_4)$  zu einem Empfängerknoten  $R$  mit den Koordinaten  $(r_1, r_2, r_3, r_4)$  läuft so wie folgt ab: Der Senderknoten wählt das passende Ankerpaar  $(\text{anchor}_i, \text{anchor}_j)$  anhand

von  $s_k, k \in \{1, 2, 3, 4\}$  aus. Von diesem Ankerpaar wird auch die Position des Empfängers definiert. Das zu übertragende Paket wird wie bei einem Broadcast versendet, jedoch gibt es einen Mechanismus, durch welchen der Broadcast durch das gesamte System verhindert wird. Ein Knoten T mit den Koordinaten  $(t_1, t_2, t_3, t_4)$  muss beim Erhalten des Paketes einen Vergleich durchführen:

$$(t_i \in [s_i, r_i]) \& \& (t_j \in [s_j, r_j])$$

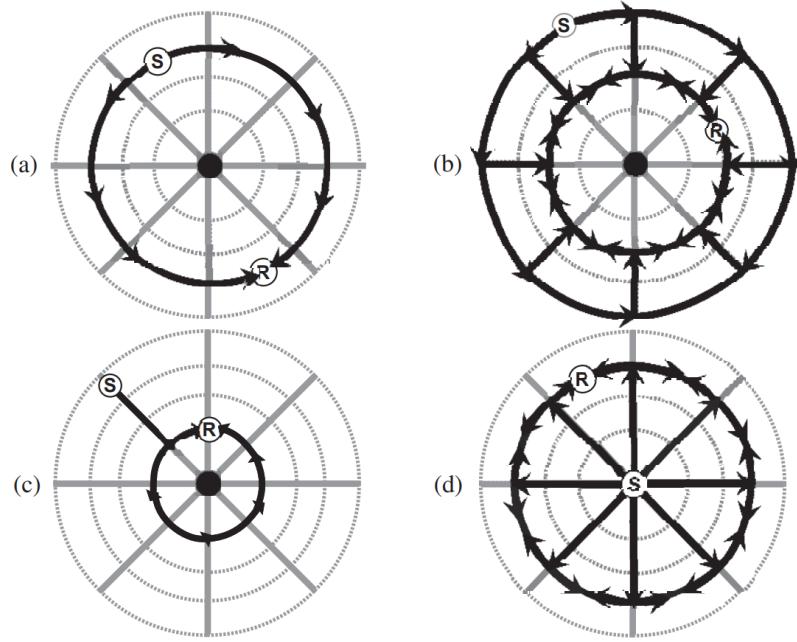
Dabei wird im Endeffekt nur überprüft, ob der Knoten T innerhalb des Gebietes ist, das von den Ankerpunkten aufgespannt wird. Ist dies nicht der Fall, so wird das Paket nicht weitergeleitet. Wenn so die Ankerpunkte für eine Übertragung zwischen S und R gut gewählt sind, kann die Übertragung durchgeführt werden, ohne dabei das gesamte System mit einem Broadcast zu fluten. In den Simulationen der Arbeit konnte gezeigt werden, dass CORONA im Vergleich zu alternativen Lösungen kürzere Routenpfade bietet.

Schlussfolgernd lässt sich zu CORONA sagen, dass es gute Verbesserungen und Optimierungen für das Routing in Nanonetzwerken geliefert hat. Jedoch funktioniert dieser Ansatz bislang nur im zweidimensionalen Raum und lässt sich in Bezug auf diese Masterarbeit schwer übertragen. Ein bedeutender Grund dafür ist, dass sich diese Arbeit mit SDMs statt DNA-Tiles befasst. Deshalb ist das Routing nicht in DNA-Tile-basierten Nanonetzwerken anwendbar.

### 3.3. A Deployable Routing System for Nanonetworks

Diese Arbeit wurde im Jahr 2015 von Christos Liaskos, Ageliki Tsoliariadou, Andreas Pitsillides, Ian F. Akyildiz, Nikolaos V. Kantartzis, Antonios X. Lalas, Xenofontas Dimitropoulos, Sotiris Ioannidis Maria Kafesaki und C.M. Soukoulis verfasst [29]. Das *Deployable Routing System* (DEROUS) für ad-hoc-Nanonetzwerk nimmt genau wie das CORONA auch SDMs an. DEROUS basiert auf dem Konzept eines Beacons, um den herum dynamisch kreisförmige und radiale Routenpfade geformt werden. Wie auch CORONA setzt es auf eine zweidimensionale Topologie. Es wird angenommen, dass die Nanoknoten Nachrichten drahtlos in zwei unterschiedlichen Modi übertragen können: Im Low-Power-Modus mit geringer Reichweite und niedrigem Energieverbrauch sowie im normalen Modus mit größerem Radius und höherem Energieverbrauch.

Der Vorgang bei DEROUS ist in zwei Phasen unterteilt: die Einsatzphase und die Datenroutingphase. Die Einsatzphase ist eine kurze Phase zum Initialisieren des Systems. Dabei wird extern ein Nanoknoten als Beacon-Point festgelegt. Dieser sendet ein Paket im Broadcast zuerst im Low-Power Modus aus. Dabei wird durch Hop-Count und ein spezifisches Flag die Entfernung jedes Knotens zum Beacon lokal bei jedem Knoten festgehalten. Der gleiche Vorgang wird danach noch einmal im normalen Modus durchgeführt. Alle Knoten speichern dadurch die Hop-Count-Entfernung von beiden Modi. Ist die Einsatzphase abgeschlossen, können sowohl im Low-Power als auch im normalen Modus Nachrichten zwischen einem Sender und Empfänger gesendet werden. Dabei



**Abbildung 3.3.:** Darstellung von vier (a-d) verschiedenen Routing Szenarien nach dem DEROUS-Protokoll. Die schwarzen Pfeile stellen dabei die Wege dar, die ein verschicktes Paket zwischen Sender S und Empfänger R nehmen kann. [29]

wird lediglich die Entfernung von Sender und Empfänger zum Beacon-Point benötigt. In der Datenroutingphase sendet der Sender  $S$  ein Paket mit seiner Entfernung und der Entfernung des Empfängers zum Empfänger  $R$ . Dies wird wieder mit einem Broadcast gemacht, jedoch leitet ein Nanoknoten  $T$  nur die Nachricht weiter, wenn eine der folgenden Voraussetzungen eingehalten ist:

- (1)  $HC_S \leq HC_T \leq HC_R \wedge HC_S \leq HC_R$
- (2)  $HC_R < HC_T < HC_S, \wedge HC_R < HC_S$

Dabei ist  $HC_X$  die Entfernung beziehungsweise der Hop-Count des Beacon zum Knoten  $X$ . Diese Bedingungen besagen, dass alle Knoten zwischen Sender  $S$  und Empfänger  $R$ , die auf dem Ring um den Beacon liegen, die Nachricht weiterleiten. Dies ist in Abbildung 3.3 zu erkennen.

Im Vergleich zu CORONA und anderen Routing-Systemen auf Nanoebene schneidet DEROUS gut ab. Es verringert die Anzahl von wiederholten Übertragungen und hat eine hohe Rate erfolgreicher Peer-to-Peer Übertragungen. Auch schneidet es im Vergleich zu anderen Protokollen gut ab, wenn es um die Flutungsrate geht. Dabei geht es darum, wie viel im Durchschnitt geflutet wird. Diese Rate ist in DEROUS abhängig davon, wo der Beacon gesetzt ist, jedoch sind auch worst case Szenarien noch besser als einige andere Ansätze.

Zusammenfassend lässt sich sagen, dass DEROUS einen vielversprechendes Ansatz für ad-hoc Nanonetzwerke bietet. Dabei ist es sowohl für Low-Power-Anwendungen als auch für Anwendungen mit höherem Energiebedarf geeignet. Obwohl die Betrachtung von DEROUS, ähnlich wie bei CORONA, momentan auf den zweidimensionalen Raum beschränkt ist, kann DEROUS mit relativ geringem Aufwand auf einen dreidimensionalen Raum erweitert werden. In diesem Fall würde die Kommunikation nicht radial oder kreisförmig, sondern sphärisch ablaufen.

Aus allen drei vorgestellten Arbeiten lässt sich ableiten, dass Kommunikationsprotokolle für DNA-basierten Nanonetzwerken in keiner Arbeit konkret betrachtet werden. Die meisten Arbeiten für Nanonetzwerke konzentrieren sich auf andere Materialien oder Mechanismen. Diese können als Inspiration für DNA-Tiles verwendet werden, jedoch lässt sich wegen der Eigenschaften der benutzen Materialien keines der Routing Protokolle komplett auf Nanonetzwerke mit DNA-basierter Self-Assembly übersetzen. Nach der Betrachtung der verwandten Arbeiten und den zuvor gelegten Grundlagen kann im Folgenden das Konzept vorgestellt werden.



## 4. Konzeption

In diesem Kapitel wird ein Konzept vorgestellt, das alle in dieser Arbeit besprochenen Mechanismen und Anforderungen herkömmlicher Kommunikationsprotokolle umfasst. Die Anforderungen werden zunächst in drei Hauptkategorien eingeteilt:

- Anforderungen, die inhärent in der Ausführung von Self-Assembly durch DNA-Tiles sind,
- Anforderungen, die speziell für die Kommunikation mit DNA-Tile-basierten Systemen definiert werden müssen,
- Anforderungen, die auf der Nanoebene nicht umgesetzt werden sollten, da sie sich besser für eine Implementierung auf einer anderen Ebene eignen.

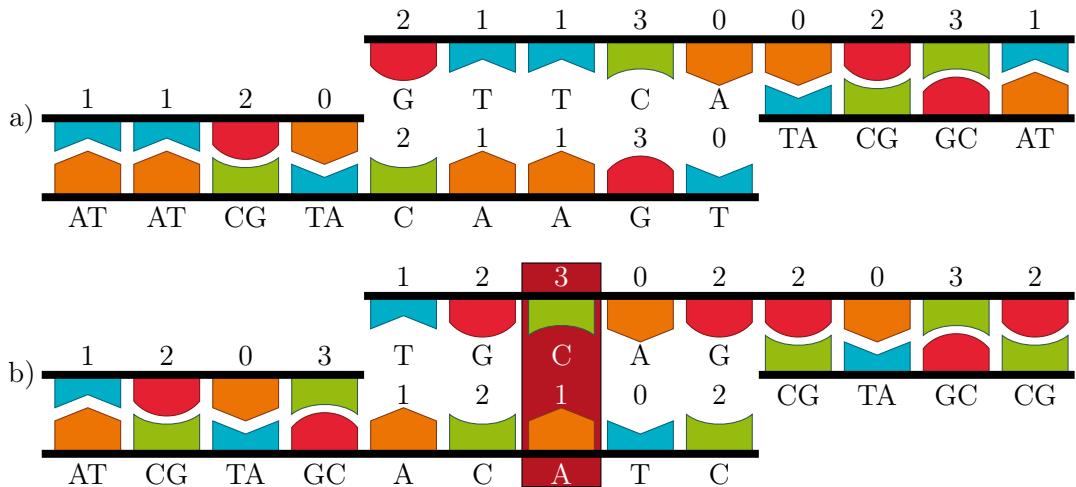
In den folgenden Sektionen werden diese Arten von Anforderungen besprochen und vorgestellt. Dazu gehören die Adressierung, das Übertragungsmedium, Routing, Dialogaufbau, Fehlererkennung, Fehlerkorrektur, Framing, Datenflusskontrolle, Nachrichtencodierung und Flags. Abschließend werden einige dieser Mechanismen in einem Beispiel zusammengefasst und im ISO/OSI-Modell eingeordnet.

### 4.1. Die Adressierung in Nanonetzwerken

Das zuerst betrachtete Konzept ist in DNA-Tile basierten Systemen inhärent. Die Adressierung von Nanogeräten muss nicht weiter implementiert werden. Kommunikation zwischen zwei Geräten funktioniert durch Verbindung der Liganden des Nachrichtenmoleküls an den Rezeptoren des Empfängergerätes. Dieser Mechanismus kann für die Adressierung genutzt werden. Wie in Abbildung 4.1 zu erkennen ist, können die Basenpaare für die Adressierung codiert werden. Im gegebenen Beispiel könnte die Codierung beispielsweise wie folgt aus:

$$\begin{aligned} TA &= \text{Empfänger } T = \text{ Sender } A = 0 = 00 \\ AT &= \text{Empfänger } A = \text{ Sender } T = 1 = 01 \\ CG &= \text{Empfänger } C = \text{ Sender } G = 2 = 10 \\ GC &= \text{Empfänger } G = \text{ Sender } C = 3 = 11 \end{aligned}$$

Die offenen Enden der DNA-Stränge können auf diese Weise als Adressen codiert werden. In diesem Beispiel in a) kann so die Adresse 03112 dargestellt werden. In b) verbinden sich



**Abbildung 4.1.:** Konzeptionelle Darstellung von offenen DNA-Strängen und der möglichen Codierung der Basenpaare zur Adressierung. In a) verbinden sich die Stränge mit der Adresse 03112, in b) verbinden sich die Stränge nicht, da verschiedene Adressen vorhanden sind (20121 und 20321).

die offenen Enden nicht, da Adenin und Cytosin kein Basenpaar bilden können. Somit sind auch die Adressen unterschiedlich (20121 und 20321). Um Adressierung in DNA-Tile-basierten Nanonetzwerken zu realisieren, muss somit kein weiterer Mechanismus entworfen werden.

## 4.2. Das Übertragungsmedium in Nanonetzwerken

Eine speziell in funkbasierten Netzwerken bedeutsame Anforderung betrifft die Auswahl des Übertragungsmediums. In DNA-Tile-basierten Nanonetzwerken ist das Übertragungsmedium im Anwendungsgebiet definiert. Da in dieser Arbeit besonderes Augenmerk auf die medizinische Anwendung gelegt wird, kann das Übertragungsmedium hier als der Blutkreislauf eines Menschen gesehen werden. Durch den Blutstrom und Diffusion bewegen sich so alle Geräte im System, was eine Anforderung für die Nanogeräte direkt abdeckt. Die Nanogeräte benötigen so keine externe Fortbewegungsmethode wie Nanomotoren oder ähnliche Ansätze. Andere Übertragungsmedien in anderen Anwendungsfällen sind jedoch denkbar.

## 4.3. Routing und Dialogaufbau in Nanonetzwerken

Zwei weitere Anforderungen von herkömmlichen Kommunikationsprotokollen ist das Routing und der Dialogaufbau zwischen einzelnen oder mehreren Geräten. Beide sind in

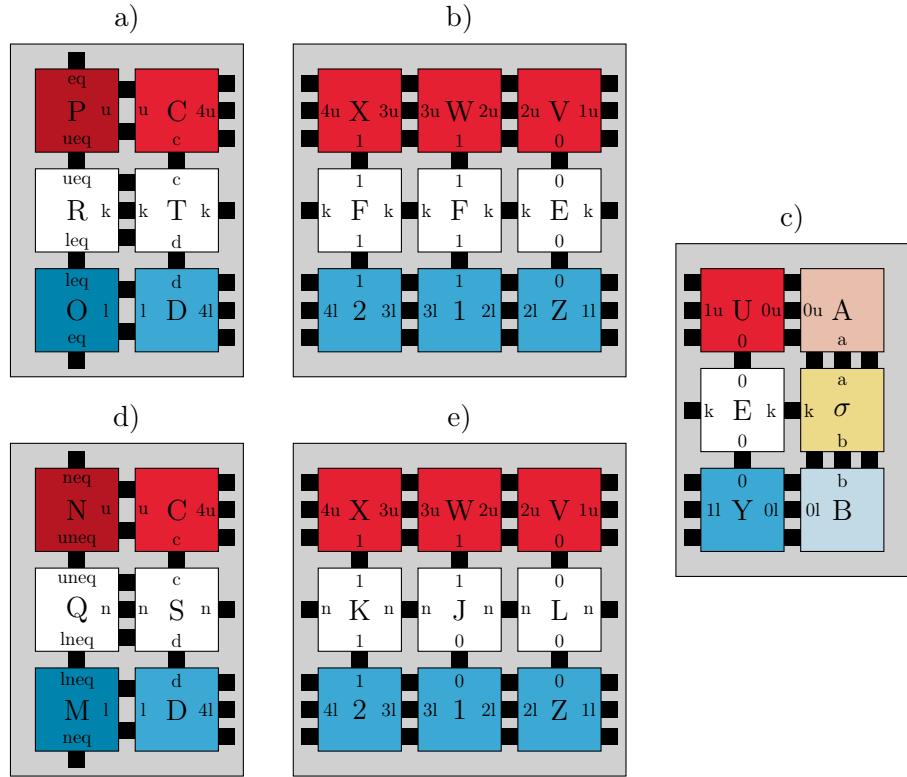
DNA-Tile-basierten Nanonetzwerken von niedriger Priorität. Auf einem so kleinen Skalenbereich ist es schwierig, einen Dialog zwischen Teilnehmern des Systems aufzubauen. Es ist in einem solchen Kontext sinnvoll, die Nachrichten und die Kommunikation auf ein Minimum zu beschränken. Bedeutend ist dabei die schnelle Übertragung von Informationen mit geringem Overhead. Obwohl das Routing eine zentrale Anforderung in herkömmlichen Kommunikationsprotokollen ist, hat es in DNA-Tile-basierten Nanonetzwerken eine geringere Relevanz, da die meisten Szenarien ohne Routing auskommen. Da sich alle Nachrichten in einem solchen System zufällig oder durch den Blutkreislauf durch das gesamte System bewegen, muss die Nachricht nicht von anderen Teilnehmern geroutet werden. Da auch die Adressierung, wie vorgestellt wurde, durch Liganden und Rezeptoren abgebildet ist, muss kein Routing vollzogen werden. Hierbei unterscheidet sich ein Nanonetzwerk basierend auf DNA-Tiles und Self-Assembly stark von anderen Nanonetzwerken. Die im Kapitel 3 vorgestellten Netzwerke benötigen beispielsweise alle eine Form des Routings, um Kommunikation zu ermöglichen.

## 4.4. Framing in Nanonetzwerken

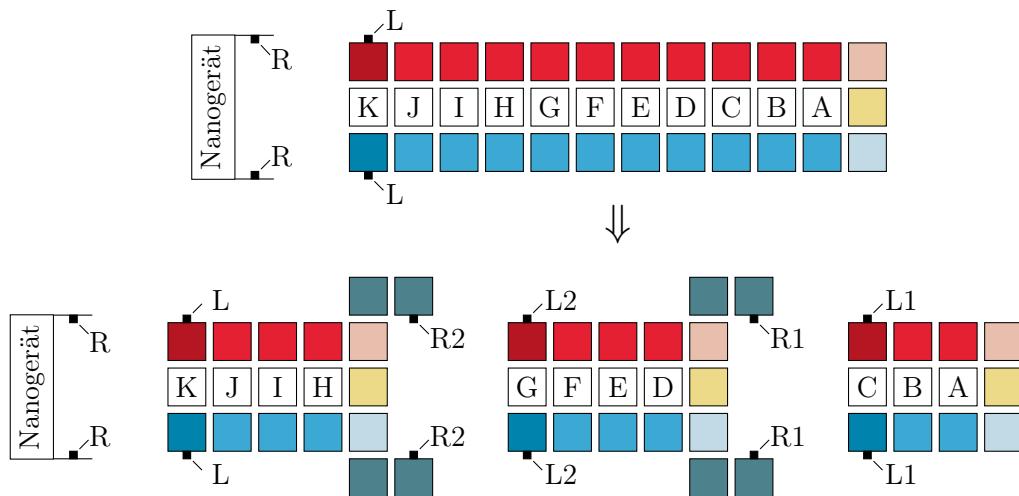
Ein Verfahren, um die Paketgröße von Nachrichten zu reduzieren und somit größere Datenmengen effizient verarbeiten zu können, ist das *Framing*. Framing hat in herkömmlichen Kommunikationsprotokollen großen Nutzen, in Nanonetzwerken ist Framing im Verhalten der Tile-basierten Self-Assembly enthalten. Mit der Seed-Assembly und den möglicherweise vorhandenen Tiles nördlich und südlich der Seed-Assembly kann der Framefang definiert werden. Die Liganden im Tileset bilden das Ende des Frames. Zusätzlich implementiertes Framing innerhalb einer Self-Assembly ist jedoch oft nicht sinnvoll. Obwohl das Aufteilen eines Nachrichtenmoleküls in mehrere kleinere Moleküle interessant und sinnvoll erscheinen mag, werden Tilesets so designt, dass sie sich kontrolliert und spezifisch bilden.

Um das Problem klarer zu machen, findet sich ein Beispiel in Abbildung 4.2. Dafür wird eine Self-Assembly für den binären Äquivalenzvergleich verwendet. Dabei wird die Information, ob die Binärzahlen äquivalent sind, vom Seedtile bis zum linken Ende des Moleküls weitergegeben. Solange die beiden Binärzahlen im inneren Kleber der nördlichen und südlichen Grenze des Moleküls äquivalent sind, wird ein  $k$  im horizontal im Molekül weitergegeben. Sobald sich zwei Binärstellen unterscheiden, wird statt dem  $k$  ein  $n$  weitergegeben. In Abbildung 4.2 ist a), b) und c) das korrekt gebundene Molekül. Jedoch kann sich auch d) und e) gemeinsam an den Rezeptoren eines potenziellen Empfängers binden. Das Tileset ist auf Temperatur drei ausgelegt, dementsprechend kann sich auch c) an e) binden, obwohl der innere Kleber mit  $k$  und  $n$  unterschiedlich ist. Somit würde sich das Molekül mit einem Growth-Error falsch binden. Nur in Self-Assemblies, die keine Information horizontal in der Self-Assembly weitergeben, könnte Framing angewendet werden.

Framing kann also für speziell dafür definierte Tilesets angewandt werden. Soll Framing



**Abbildung 4.2.:** Beispielhafte Aufteilung eines Nachrichtenmoleküls in drei Teile. Die grauen Boxen verdeutlichen dabei, welche Tile-Assemblies ein Frame bilden. Dabei kann an den Kleberbezeichnern erkannt werden, dass die Frames a), b) und c) das korrekte Moleköl bilden. Jedoch können sich Frame a) oder d) gleichermaßen an den Rezeptoren eines Empfängers binden. Im Fall d) würde sich e) weiter binden und bei Temperatur drei auch Frame c). Dies würde jedoch zu einem Growth-Error zwischen den Tiles *E* und *L* führen. Auch ist in dieser Abbildung durch die Farbkodierung zu erkennen, dass durch die Tiles *A,B,P* und *0* bereits eine Art Frame vorhanden ist.



**Abbildung 4.3.:** Skizzierte Darstellung der in dieser Arbeit vorgestellten Idee zur Verkleinerung von Self-Assemblies. Oben ist eine große Self-Assembly dargestellt, die sich mit den Liganden L an den Rezeptoren R eines beliebigen Nanogerätes binden kann. Unten ist die Aufteilung der Self-Assembly auf drei fast gleich große Self-Assemblies dargestellt. Das ursprüngliche Tileset muss dabei so definiert sein, dass das Problem aus Abbildung 4.2 nicht auftreten kann.

in Nanonetzwerken angewendet werden, so bietet es sich an, mehrere kleinere Tilesets zu bilden, die sich untereinander an den Liganden verbinden können. Die Idee wird in Abbildung 4.3 skizziert, jedoch wird sie in dieser Arbeit nicht weiter behandelt, da die Umsetzung eines solchen Prozesses weitere Definitionen benötigt, die den Rahmen dieser Arbeit überschreiten würden.

Bisher benötigen die vorgestellten Anforderungen keine zusätzlichen Mechanismen für ihre Implementierung in DNA-Tile-basierten Nanonetzwerken. Lediglich beim Framing wurde eine Idee präsentiert, die weiteres Eingreifen erfordert. Im Gegensatz dazu müssen die folgenden Anforderungen implementiert werden, da sie nicht bereits durch die inhärenten Mechanismen der DNA-Tile-basierten Tile-Assembly abgedeckt sind.

## 4.5. Fehlererkennung durch Prüfsummen in Nanonetzwerken

Im Folgenden soll die Anforderung der Fehlererkennung betrachtet werden. Prüfsummen sind ein typisches Instrument der Umsetzung von Fehlererkennung. Sie finden sich in einigen Headern von herkömmlichen Kommunikationsprotokollen wieder. Bei Prüfsummen handelt es sich um ein einfaches Verfahren, das überprüfen soll, ob eine Nachricht richtig übertragen wurde. Mit ihnen kann beim Empfänger durch eine Berechnung festgestellt werden, ob die empfangene Datei unverfälscht übertragen wurde. Auf der Nanoeben-

ne stellt der Prüfsummemechanismus eine Herausforderung dar, insbesondere wegen des nicht-deterministischen Verhaltens von Self-Assemblies. Für eine erfolgreiche Bildung der korrekten Prüfsumme während des Self-Assembly-Prozesses muss für jede Kombination möglicher Tiles ein spezifisches *Prüfsummentile* existieren.

Trotz dieser Komplexität gibt es Anwendungsfälle, in denen dieser Mechanismus sinnvoll sein könnte. Ein Beispiel ist die Übertragung eines spezifischen Nachrichtenmoleküls. Selbst wenn dieses Molekül nicht-deterministisch durch Self-Assembly erstellt wird, wäre ein Prüfsummentile hilfreich, wenn im Voraus bekannt ist, welches Molekül gebildet werden soll.

Ein Beispiel dafür ist ein System, in welchem eine Nachricht zwischen einem einzelnen Sender und Empfänger verschickt werden muss. Dabei kann vor der Self-Assembly festgelegt werden, welches Tile für die korrekte Prüfsumme gebunden werden muss. Dies ist beispielhaft in Abbildung 4.4 a) und b) dargestellt. Dadurch, dass das Ergebnis der Self-Assembly bekannt ist, braucht es nur ein weiteres Tile, das als Prüfsumme dient. Dafür muss das Prüfsummentile zum neuen Liganden gemacht werden.

Jedoch gibt es Situationen in Nanonetzwerken, in welchen das Molekül nicht deterministisch gebildet wird. Ein solches Szenario tritt beispielsweise auf, wenn sich das Nachrichtenmolekül aus unterschiedlichen Tilesets zusammensetzt und die Quellen dieser Tilesets keine Informationen über die Tiles anderer Quellen besitzen.

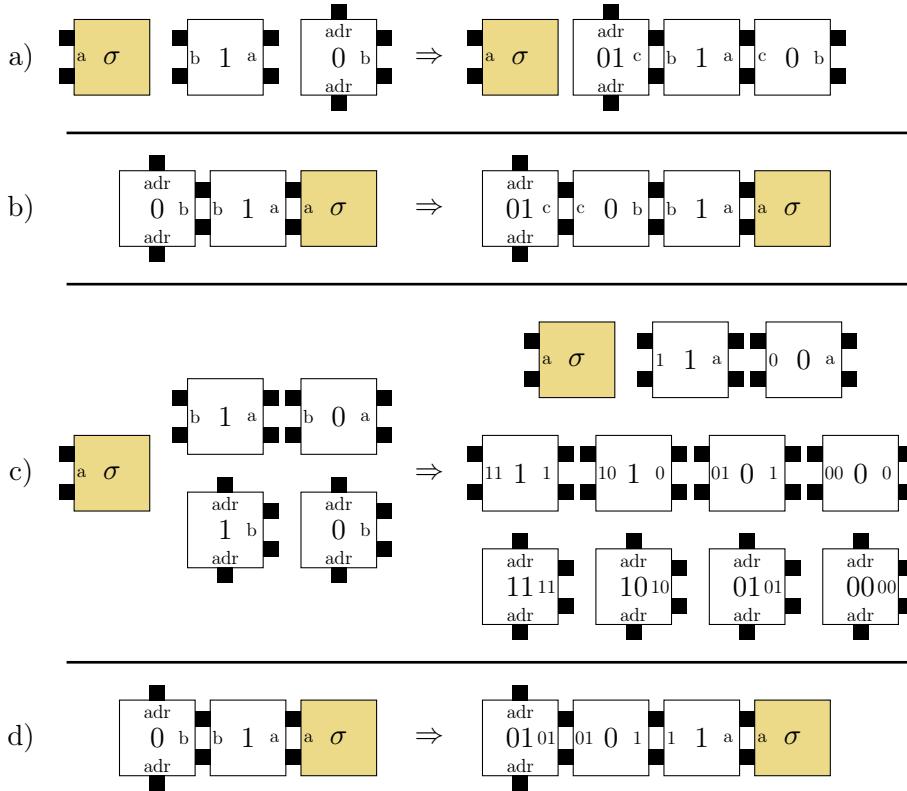
Das Problem mit Nichtdeterminismus und Prüfsummen ist in Abbildung 4.4 c) und d) dargestellt. Es wird deutlich, dass zwar die Größe der Self-Assembly konstant bleibt, das Tileset jedoch stark anwächst. Die Größe des Tilesets hängt dabei von der Struktur des Nachrichtenmoleküls ab. Für Moleküle, die diesem Beispiel folgen, wächst das Tileset beispielsweise exponentiell an. Allgemein lässt sich das wie folgt darstellen: Für eine Zahl im  $x$ -ten Basissystem, mit der Anzahl der Ziffern  $y$  gilt:

$$\begin{aligned} & xy + 1 \text{ Tiles im Tileset ohne Prüfsumme} \\ & 1 + x^y + \sum_{n=1}^y x^n \text{ Tiles im Tileset mit Prüfsumme} \end{aligned}$$

Da bereits kurze Nachrichten ein starkes Anwachsen des Tilesets verursachen, hängt die Verwendung von Prüfsummentiles vom jeweiligen Tileset und Anwendungsgebiet ab.

## 4.6. Fehlerkorrektur durch Snaked-Proofreading in Nanonetzwerken

In der letzten Sektion wurde die Fehlererkennung behandelt. Diese Sektion widmet sich der Fehlerkorrektur. Die Fehlerkorrektur in Self-Assemblies ist im Kapitel 2 vorgestellt worden. Die Fehlerkorrektur in Kommunikationsprotokollen kümmert sich um die Sicherheit im Nachrichtenaustausch. Dabei sollen beim Übertragen von Daten Fehler erkannt



**Abbildung 4.4.:** Beispiel für die Implementierung von Prüfsummen durch Self-Assembly. In a) ist links ein Tileset gegeben für eine Self-Assembly von drei Tiles. Rechts in a) ist das modifizierte Tileset angegeben mit einem Prüfsummentile. In b) sind die Self-Assemblies der beiden Tilesets aus a) dargestellt. In c) ist links das Tileset aus a) angegeben, aber für den Fall, dass die Tiles nicht deterministisch sind, sondern durch die Tiles eine zweistellige Binärzahl codiert werden kann. Rechts in c) ist dann erneut das modifizierte Tileset angegeben für alle möglichen Prüfsummentiles 00, 01, 10 und 11. In d) sind analog zu b) die Self-Assemblies der Tilesets aus c) dargestellt. Diese Abbildung soll verdeutlichen, dass Prüfsummen bereits in kleinen Tilesets zu großem Wachstum führt. Die Größe der Self-Assembly bei Prüfsummen wiederum bleibt konstant.

und direkt korrigiert werden. Die Fehlererkennung wurde in der vorherigen Sektion als sehr situativ dargestellt, weshalb für diese Sektion angenommen wird, dass keine Fehlererkennung durchgeführt werden kann. Somit muss auf Fehlerprävention statt reaktiver Fehlerkorrektur gesetzt werden, die in Form des  $k \times k$ - oder erweitert durch das Snaked-Proofreading in Kapitel 2 bereits vorgestellt wurde.

In dieser Arbeit liegt der Fokus auf dem Snaked-Proofreading. Da Snaked-Proofreading eine Erweiterung von  $k \times k$ -Proofreading ist, lässt sich dieses daraus ableiten. Dabei ist anzumerken, dass das Snaked-Proofreading in Kapitel 2 in einer sehr allgemeinen Form vorgestellt wurde. Um es auf ein Tileset anzuwenden, sind positionelle Informationen über Tiles erforderlich. Abhängig von der Position des Tiles in der Self-Assembly müssen die inneren Kleber im Snaked-Proofreading unterschiedlich platziert werden. Der genaue Vorgang dafür wird im folgenden Kapitel 5 mit der Konstruktion des Skripts beleuchtet.

### 4.7. Datenflusskontrolle in Nanonetzwerken

Eine weitere Anforderung ist die Datenflusskontrolle. In herkömmlichen Kommunikationsprotokollen stellt sie einen essenziellen Bestandteil dar, der aus verschiedenen Gründen zur Anwendung kommt. Die Datenflusskontrolle umfasst dabei die folgenden Punkte:

1. *Vermeidung von Überlastung des Netzwerkes*: Ein schneller Sender kann beispielsweise einen langsamen Empfänger mit Daten überschwemmen. Dies muss verhindert werden, indem solche Geräte aufeinander angepasst werden.
2. *Effizienz*: Durch Analyse und Kontrolle der Datenmenge und Bandbreite kann das Netzwerk effizienter genutzt werden.
3. *Fehlerbehandlung*: Bei verlorener Kommunikation oder anderen Übertragungsfehlern sollten Mechanismen zur Neuübertragung implementiert sein, um die Datenintegrität zu gewährleisten.
4. *Gerechtigkeit*: Es muss sichergestellt werden, dass alle Geräte in einem System einen gerechten Zugang zum Medium haben.

Für die Datenflusskontrolle in Nanonetzwerken ist die Lebensdauer von DNA in verschiedenen Medien von Bedeutung. Forschung auf diesem Gebiet gibt es speziell in der forensischen Medizin. Die Lebensdauer von DNA hängt von verschiedenen Faktoren wie Umgebung, Temperatur oder Medium ab. Mehr Informationen dazu gibt es beispielsweise in diesen Arbeiten [1, 34, 37, 40]. Für diese Arbeit wird jedoch angenommen, dass in jedem eingesetzten Medium ein Mechanismus existiert, der die DNA-Moleküle und im spezifischen die Nachrichtenmoleküle nach einer Zeitspanne  $t_{deg}$  auflöst oder anderweitig aus dem System entfernt.

Datenflusskontrolle in Nanonetzwerken und ihre Anforderungen kann durch verschiedene Mechanismen und Aspekte erreicht werden. In den folgenden Sektionen werden alle vier

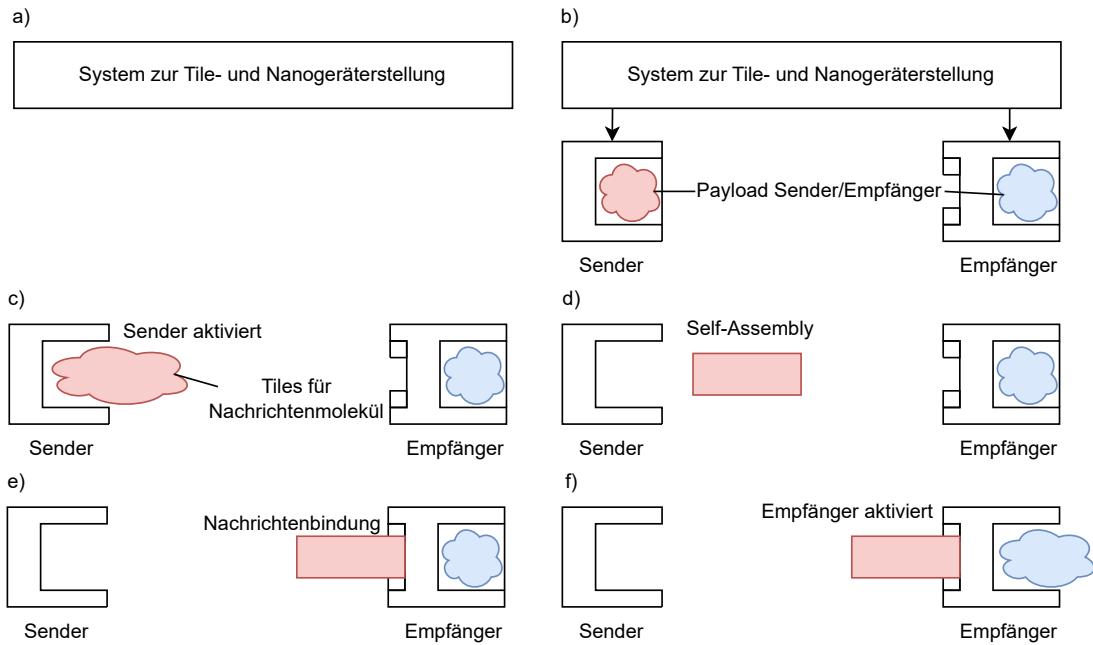
Anforderungen durch zwei Mechanismen abgedeckt.

Bevor jedoch näher auf die einzelnen Aspekte eingegangen wird, muss eine Unterscheidung zwischen zwei Fällen in Nanonetzwerken gemacht werden. Die folgenden Mechanismen ergeben nur Sinn, wenn sowohl Sender als auch Empfänger der Nachricht mehrere Nachrichten verschicken und empfangen können. Ist dies nicht der Fall und die Kommunikation basiert auf einem System wie in Abbildung 4.5, so müssen einige der folgenden Schritte nicht beachtet werden. Ein solches System geht davon aus, dass potenzielle Sender- und Empfänger-Nanogeräte von einem anderen, möglicherweise auf der Mikroebene angesiedelten, Gerät erstellt werden. Das Sendergerät kann eine Aktion ausführen, wenn sich das richtige Nachrichtenmolekül an den Rezeptoren bindet. Der Sender kann ein bestimmtes Nachrichtenmolekül freilassen je nach gewähltem Auslöser. So könnte das Gerät zum Beispiel anhand der Konzentration eines ausgewählten Stoffes die Nachricht freilassen.

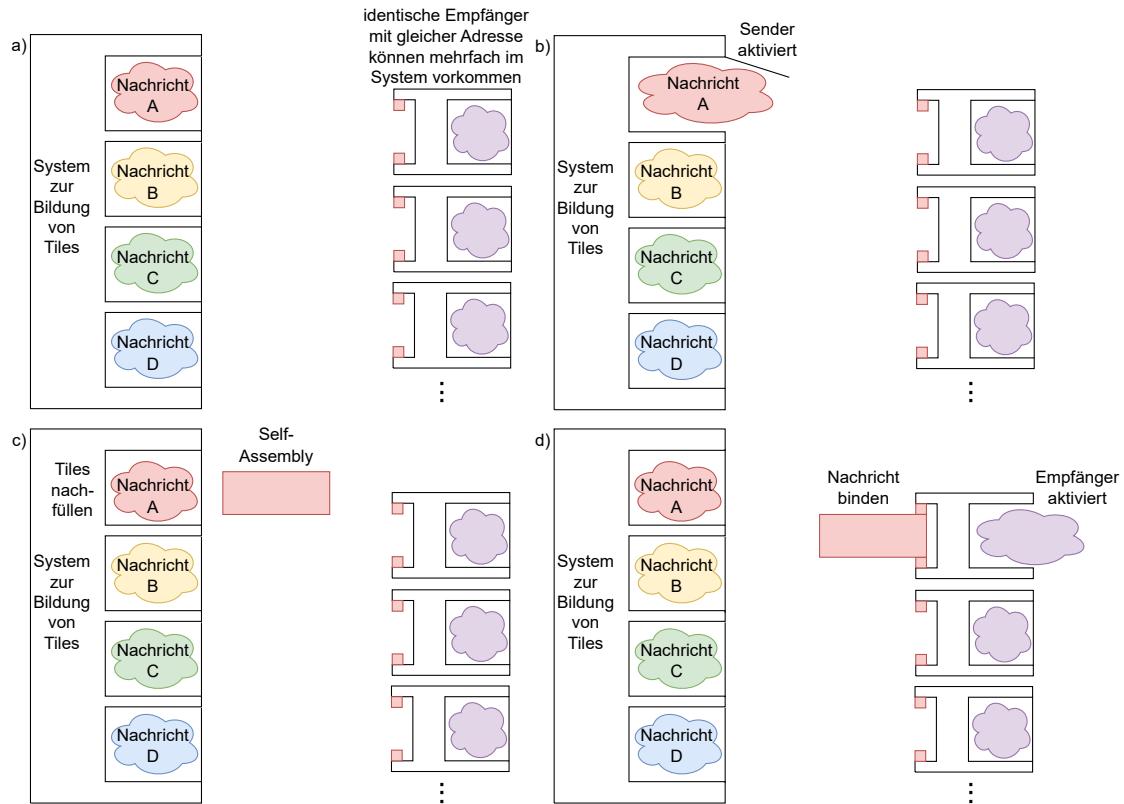
In diesem Fall wird immer nur maximal eine Nachricht zwischen einem Sender und einem Empfänger verschickt und es muss im Gesamtsystem nur auf Folgendes geachtet werden: Alle Nanogeräte, die gleichzeitig im System sind, brauchen einzigartige Rezeptoren, sodass jede Nachricht eindeutig adressiert werden kann. Dafür reicht alleine die Lebensdauer der DNA Tiles  $t_{deg}$ . Es muss sichergestellt werden, dass eine verwendete Adresse erst nach frühestens  $t_{deg}$  Zeiteinheiten wieder freigegeben wird. Damit werden in diesem Fall die Anforderungen durch Überlastungsbekämpfung und Effizienz obsolet, da in diesen Blickpunkten auf die „Einweg-Kommunikation“ nicht weiter eingegriffen werden kann. Der andere Fall, der zu betrachten ist, wird in Abbildung 4.6 vorgestellt. Dabei kann ein Sendergerät beliebig viele Nachrichten an beliebige Adressen versenden. Und ein Empfängergerät kann entweder beliebig oft im System vorkommen oder beliebig viele Nachrichten empfangen. In diesem Fall ist sowohl die Überlastungsbekämpfung als auch die Effizienz nicht durch das System gegeben, sondern kann durch Mechanismen verbessert werden. Diese werden im Folgenden betrachtet.

## 4.8. Datenflusskontrolle durch Acknowledgements in Nanonetzwerken

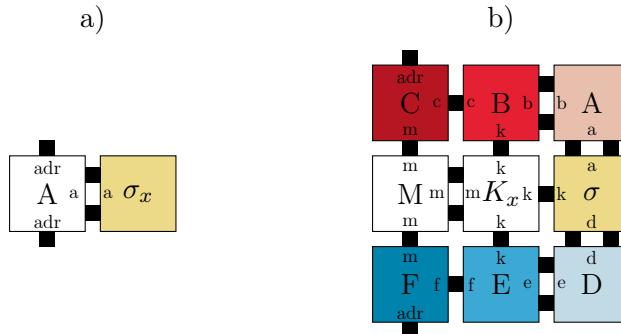
Der erste vorgestellte Mechanismus behandelt gleich drei der betrachteten Anforderungen an Datenflusskontrolle: die Vermeidung der Überlastung des Systems, die Effizienz und die Fehlerbehandlung. Für Acknowledgements sind einige Faktoren interessant. Neben der Lebensdauer eines DNA-Moleküls im System  $t_{deg}$  sind hier noch zwei weitere Angaben relevant. Die durchschnittliche Dauer zur Bildung eines Nachrichtenmoleküls durch Self-Assembly  $t_{sa}(\tau, n)$ . Diese ist abhängig von Temperatur  $\tau$  des Systems und der Größe  $n$  des Moleküls. Die Größe des Moleküls lässt sich dabei aus der Menge der Tiles bestimmen, die zur Konstruktion des gesamten Nachrichtenmoleküls verbunden werden. Der zweite wichtige Parameter ist die durchschnittliche Dauer des Bindens eines kompletten Moleküls nach Self-Assembly mit dem Empfänger des Nachrichtenmoleküls  $t_{bind}$ .



**Abbildung 4.5.:** Skizzierte Darstellung einer Kommunikation von Nanogeräten, die nach dem Senden einer Nachricht aufgelöst oder aus dem System entfernt werden können. Dafür liegt in a) ein System zur Erstellung von Nanogeräten und Tiles vor. Dieses erstellt in b) den Sender und den Empfänger für den Kommunikationsschritt. Auch die Payloads beider Akteure werden durch das System gebildet. In c) werden die Tiles für die Self-Assembly durch die Aktivierung des Senders freigegeben. Diese Aktivierung kann durch unterschiedliche Mechanismen erfolgen. Das Binden von Liganden an Rezeptoren des Gerätes oder die Registrierung von Markern sind mögliche Mechanismen dafür. In d) wird das Nachrichtenmolekül durch Self-Assembly gebildet und in e) beim Empfänger gebunden. Dieser lässt daraufhin die Payload frei. Der Inhalt der Payload kann je nach Anwendungsfall unterschiedlich aussehen.



**Abbildung 4.6.:** Skizzierte Darstellung einer Kommunikation von Nanogeräten, die erneutes Senden unterschiedlicher Nachrichten ermöglicht. In a) ist das System vorgestellt, welches das wiederholte Senden von Nachrichten ermöglicht. In b) wird das Nanogerät aktiviert, sodass eine Tileset freigeschaltet wird. Die Aktivierung kann durch unterschiedliche Mechanismen erfolgen. Eine Möglichkeit ist die Markererkennung oder die Bindung von Liganden an Rezeptoren. In c) kommt es zur Self-Assembly. Auch wird gezeigt, dass in so einem System die gleiche Nachricht erneut gesendet werden kann. In d) bindet sich die Nachricht beim Empfänger und dieser wird aktiviert. Damit kann beispielsweise die Payload des Empfängers freigeschaltet werden.



**Abbildung 4.7.:** Darstellung von möglichen Acknowledgement Molekülen. In a) der Molekülhöhe eins und in b) der Molekülhöhe drei. Andere Größen und Formen für Acknowledgements sind denkbar und werden im Kapitel 6 näher betrachtet. Hier sind zwei Beispiele zur Veranschaulichung dargestellt.

Um die Überlastung zu verringern, werden weitere Tilesets in den Prozess der Kommunikation verwendet. Ein Empfänger lässt beim Binden des Nachrichtenmoleküls ein solches Tileset frei. In Abbildung 4.7 sind zwei mögliche Self-Assemblies zu erkennen. Diese Assemblies stellen Acknowledgements dar, die schnellstmöglich gebildet und beim Sender gebunden werden, um den Erhalt der initialen Nachricht zu bestätigen. Dafür kann das Acknowledgement für ein möglichst kleines Molekül wie in Abbildung 4.7 a) nur aus einem Seed-Tile und einem Tile für die Liganden bestehen, der die Addressierung bestimmt. Eine andere Möglichkeit ist, dass es aus neun Tiles wie in Abbildung 4.7 b) besteht.

Das Acknowledgement ist in diesen beiden Assemblies durch  $\sigma_x$  und  $K_x$  mit einem Index  $x$  dargestellt. Dieser beschreibt die Notwendigkeit von eindeutigen Acknowledgements für jeden Kommunikationsvorgang. Es muss verhindert werden, dass ein Sender ein Acknowledgement einer vergangenen Nachricht verspätet erhält und sie für das Acknowledgement des zuletzt geschickten Nachrichtenmoleküls hält. Das geschieht nur unter der Annahme, dass  $t_{deg}$  groß genug gewählt ist. Somit können Assemblies und Tiles über mehrere Kommunikationsvorgänge nicht aus dem System entfernt werden. Somit benötigt jedes Acknowledgement einen eindeutigen Bezeichner, der je nach Kontext unterschiedlich festgelegt werden kann. Die Assemblies in Abbildung 4.7 sind dabei nur Beispiele für Acknowledgements. In b) ist das Molekül zwar größer, jedoch kann dies je nach Anwendungsfall notwendig sein. Ein anderer Ansatz für ein minimales Acknowledgement wie in a) könnte ein einzelnes Tile sein, welches sowohl Ligand als auch Seed-Assembly ist.

Es kann im Folgenden angenommen werden, dass ein Mechanismus im System existiert, der Acknowledgements nach dem Empfang der Nachricht freilässt. Dieses Nachrichtenmolekül kann sich dann beim Sender binden und gibt so die Bestätigung der erfolgreichen Übertragung. Auf dieser Basis kann Datenflusskontrolle durchgeführt werden.

Ein Sender lässt das Tileset für ein Nachrichtenmolekül frei. Für eine beliebige Über-

Priorität	Beispielhafter Nachrichtentyp	Erlaubte Übertragung pro $t_{deg}$ Zyklus
1	Notfallbenachrichtigung	$\infty$
2	Anweisung zur Medikamentausschüttung	5
3	spezifische Werte anfragen	3
4	allgemeiner Check	1

**Tabelle 4.1.:** Beispielhafte Einteilung von Nachrichtentypen in verschiedene Prioritätslevel für ein medizinischen System. Allen Prioritätsleveln wird aus Gerechtigkeitsgründen eine erlaubte Menge an Übertragungen pro Zyklus zugeteilt. Ein Zyklus stellt dabei die Zeitspanne  $t_{deg}$  dar, welche die durchschnittliche Dauer des Verfalls oder Entfernung eines Tiles im Medium darstellt.

tragung und deren Dauer  $t_{trans}$  gilt: Sei ein Nachrichtenmolekül der Größe  $n$ , bei einer Systemtemperatur  $\tau$  und einem je nach Anwendung gewähltem Zeitpolster  $t_w$  gegeben, kann folgendes betrachtet werden:

$$\text{Neuübertragung} \begin{cases} \text{ja,} & \text{wenn } \neg ACK \wedge t_{trans} > 2 * (t_{sa}(\tau, n) + t_{bind}) + t_w \\ \text{nein,} & \text{sonst} \end{cases}$$

Dabei wird betrachtet, unter welchen Umständen es zu einer Neuübertragung kommt. Wenn innerhalb des durchschnittlichen Übertragungszeitraums zuzüglich eines Zeitpolsters kein Acknowledgement empfangen wird, erfolgt eine Neuübertragung. Das Zeitpolster  $t_w$  muss dabei so gewählt sein, dass nach Möglichkeit keine False-Positives für eine Neuübertragung entstehen. Dabei wird angenommen, dass das Sendermolekül größer ist als das Acknowledgement. Ist dies nicht der Fall, so muss  $t_{sa}$  für das größere der beiden Moleküle betrachtet werden.

Durch diesen Mechanismus wird verhindert, dass ein langsamer Empfänger von einem schnellen Sender überlastet wird. Eine Optimierung der Zeitfenster steigert zudem die Effizienz, indem sie schnelle Neuübertragungen ermöglicht. Auch werden verlorene Nachrichten durch einen solchen Mechanismus erkannt und können durch die Neuübertragung wiederholt werden. Damit sind mit der Fehlerbehandlung drei der betrachteten Anforderungen an die Datenflusskontrolle vorgestellt. Im Folgenden soll der Aspekt der Gerechtigkeit im Medium betrachtet werden.

## 4.9. Datenflusskontrolle durch Prioritätslevel in Nanonetzwerken

Die Gerechtigkeit ist ein weiterer Faktor der Datenflusskontrolle. Dabei soll sichergestellt werden, dass alle Agierenden im Netzwerk regelmäßigen Zugriff auf das Medium erhalten. In einem Nanonetzwerk ist es jedoch schwierig, ohne Overhead einen Überblick

dariüber zu gewinnen, wer welche Nachrichten verschickt hat. Eine Lösung wäre es, durch ein zentrales Steuermedium verschiedene *Prioritätslevel* für verschiedene Nachrichtentypen festzulegen. Dies ist beispielhaft in Tabelle 4.1 für den Kontext In-Body-Networks dargestellt. Im Beispiel aus dieser Tabelle werden spezifischen Nachrichten eine höhere Priorität gegeben. Diese können unendlich oft oder mehrfach während eines  $t_{deg}$  Zeitfensters übertragen werden. Nachrichten von niedriger Priorität jedoch nur wenige Male oder nur einmal. Durch einen solchen Mechanismus kann keine Gerechtigkeit garantiert werden, jedoch kann so verhindert werden, dass Beteiligte im Netzwerk das Medium mit „unwichtigen“ Nachrichten flutet.

Dabei ist wichtig zu betonen, dass dieser Mechanismus vor allem dann von Bedeutung ist, wenn angenommen wird, dass das Medium in einem DNA-Tile-basierten Nanonetzwerk überflutet werden kann. Eine Stärke dieser Technologie ist ihre inhärente Parallelität: bei korrekt definierten Tilesets können mehrere Self-Assemblies simultan im selben Medium ablaufen, ohne Fehler zu verursachen. Dennoch könnte es Systeme geben, in denen ein Mechanismus zur Gerechtigkeit im Medium wichtig ist: beispielweise bei einer zentralen Steuerung, die durch eine Vielzahl von Nachrichten überlastet werden könnte. Daher kann ein Gerechtigkeitsmechanismus im Medium in bestimmten Anwendungsfällen durchaus sinnvoll sein.

Da einige Aspekte der Datenflusskontrolle mit Acknowledgements und Prioritätslevels abgedeckt sind, kann mit der nächsten Anforderung weiter gemacht werden.

### 4.10. Nachrichtencodierung in Nanonetzwerken

Die nächste betrachtete Anforderung lässt sich schwer auf einzelne Anforderungen aus Kommunikationsprotokollen abbilden. Dabei geht es um die Darstellung von Nachrichten auf der Nanoebene. So wie in anderen Systemen müssen bei der Kommunikation in Nanonetzwerken Nachrichten zwischen Geräten verschickt werden. Diese werden durch Self-Assembly gebildet und durch die Bindung zwischen Liganden und Rezeptoren empfangen. Dabei muss das Nanogerät, das die Nachricht bindet, eine beliebige Nachricht  $x$  von einer anderen Nachricht  $y$  unterscheiden können.

Dafür soll in dieser Sektion eine Möglichkeit geliefert werden, mit welcher eine beliebige Anzahl an Nachrichten in einem System durch Tilesets und deren Self-Assembly gebildet werden können. Dabei liegt der Fokus darauf, die Nachrichtenmoleküle möglichst klein zu halten, ohne dabei zu große Tilesets zu erstellen. Die genaue Gewichtung dieser beiden Faktoren wird in Kapitel 5 diskutiert.

Zur Codierung von Nachrichten auf Nanonetzwerkebene werden zwei Möglichkeiten vorgestellt: die Kodierung in den Kleberbezeichnern und die Kodierung in den Tilebezeichnern. Wie beim Thema der Adressierung beschrieben wurde, können offene DNA-Stränge zur Informationskodierung verwendet werden. Somit ist auch die Informationskodierung in Kleberbezeichnern möglich. Wenn die Information nicht in den Kleberbezeichnern co-

dient wird, kann sie auch in den Tilebezeichnern codiert werden.

Dabei spielt die Struktur der gebildeten Tiles die entscheidende Rolle für die Codierung, nicht die offenen DNA-Enden. Beispiele für beide Codierungsarten sind in Abbildung 4.8 und Abbildung 4.9 gegeben. Beide Beispiele zeigen im Beispiel b) eine Self-Assembly für die Hexadezimalzahl  $ab1$ . In Abbildung 4.8 sind in a) alle Tiles zu sehen, die zur Codierung von dreistelligen Hexadezimalzahlen in den Kleberbezeichnern benötigt werden. Insgesamt 100 Tiles sind notwendig und die finale Self-Assembly besteht aus zehn Tiles. Es ist möglich zwei Tiles im Tileset und der Self-Assembly zu sparen, indem die Liganden mit den Bezeichnern „Z“ und „K“ entfernt werden. Dann müssten alle Tiles mit den Bezeichnern „Y“ und „J“ die Liganden bilden. Somit würde das Tileset 98 Tiles umfassen und die Self-Assembly acht Tiles. Aus Gründen der Übersichtlichkeit wurde dies im Beispiel aus Abbildung 4.8 nicht getan. Das Prinzip wird jedoch im Beispiel aus Abbildung 4.9 angewendet. Auch hier sind in a) alle Tiles dargestellt, die zur Codierung einer dreistelligen Hexadezimalzahl benötigt werden. In b) ist die beispielhafte Self-Assembly zu sehen. Dieser Ansatz benötigt 48 Tiles im Tileset und vier Tiles in der Self-Assembly. Wird angenommen, dass auch hier die Liganden in einzelnen Tiles gebildet werden, verändert sich das leicht. Dabei wäre das Tileset 49 Tiles und die Assembly fünf Tiles groß.

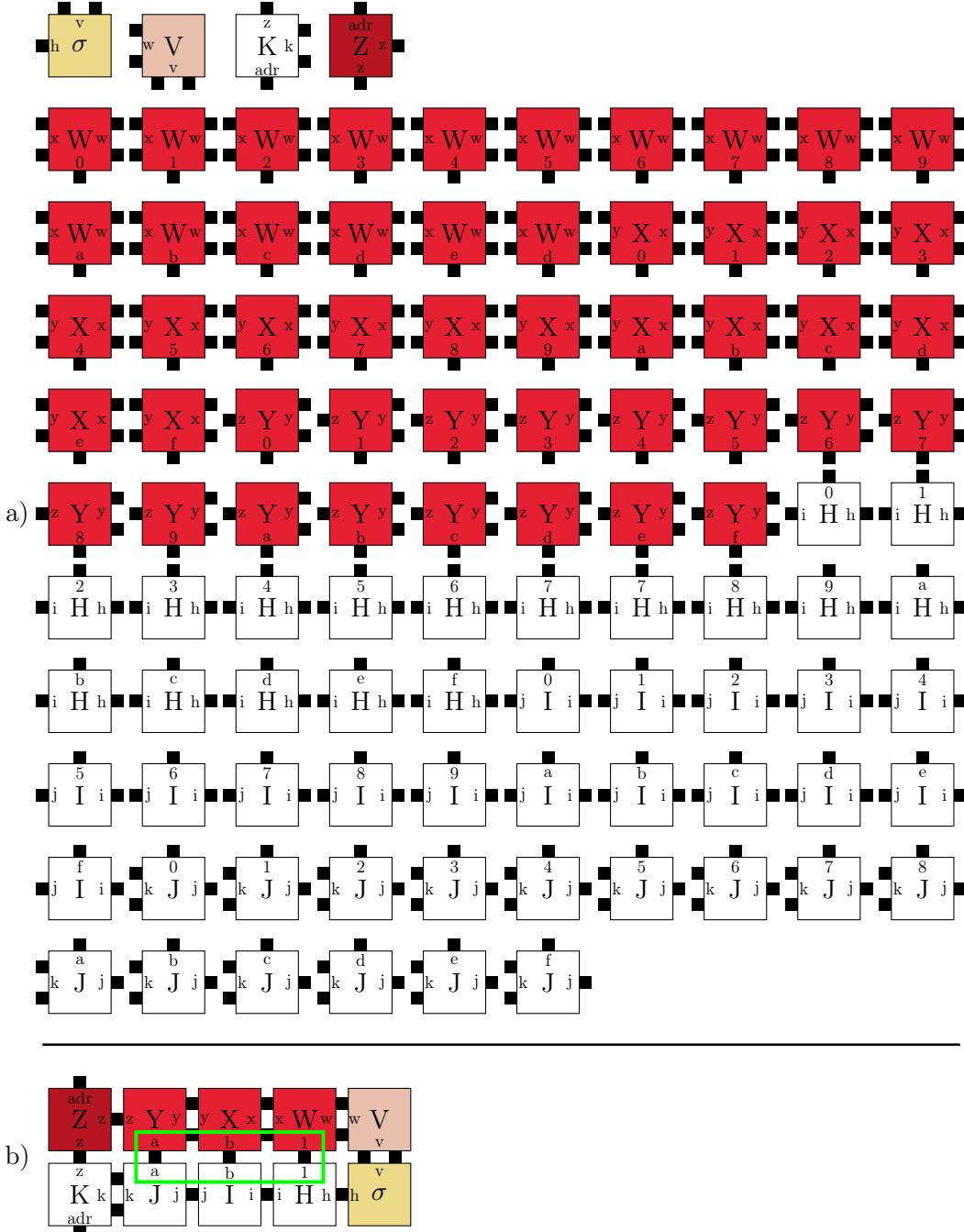
Wird bei beiden Ansätzen davon ausgegangen, dass sie den Beispielen aus den Abbildungen 4.8 und 4.9 folgen, dann lässt sich für eine Zahl im Basis- $x$ -System und der Anzahl der Ziffern  $y$  Folgendes sagen: Wird die Zahl in den Kleberbezeichnern codiert, so hat das Tileset  $2xy + 4$  Tiles und die Self-Assembly  $2y + 4$  Tiles. Wird die Zahl in einem einzelnen Tile codiert, dann hat das Tileset  $yx + 1$  Tiles und die Self-Assembly  $y + 1$  Tiles. In den Beispielen aus Abbildungen 4.8 und 4.9 liegt eine Hexadezimalzahl mit drei Ziffernstellen vor. Somit lässt sich beispielsweise für Kleberbezeichner  $2 \cdot 16 \cdot 3 + 4 = 100$  berechnen. Für sowohl die Größe der Self-Assembly als auch für die Größe des benötigten Tilesets kann erkannt werden, dass die Codierung in den Bezeichnern des Tiles effizienter ist, da jede Ziffer eines beliebigen Zahlensystems durch ein statt zwei Tiles dargestellt werden kann.

Jedoch muss dabei betrachtet werden, dass die Tilebezeichner auf der mathematischen Ebene verwendet werden, damit Nutzer\*innen des Systems eine bessere Übersicht über die verschiedenen Tiles haben. Um darin die Codierung durchzuführen, muss ein neuer Ansatz gefunden werden, der diese Bezeichner auch im praktischen Anwendungsfall ermöglicht. Ein Vorschlag dafür ist die interne Struktur des Tiles zu verwenden, um Bezeichner zu realisieren. Die DNA-Basenpaare im Inneren des Tiles können so analysiert und spezifisch für einen Bezeichner verwendet werden.

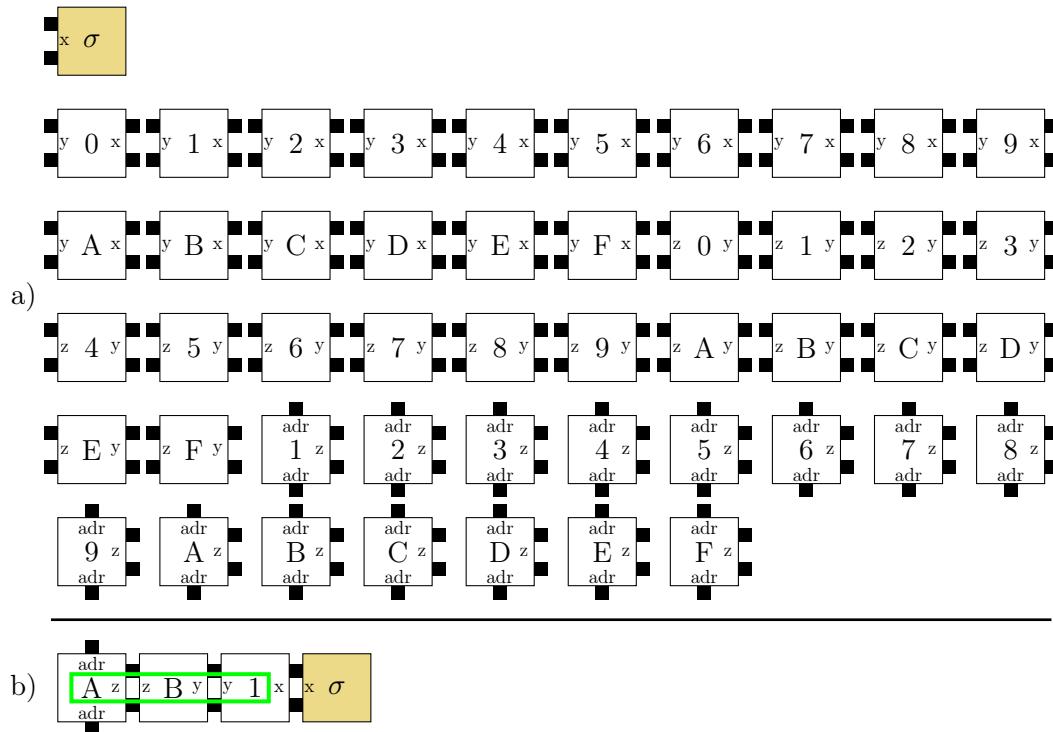
Wenn in einem Nanonetzwerk eine beliebige Anzahl unterschiedlicher Nachrichten vorhanden ist, können diese in ein entsprechendes Zahlensystem übertragen werden. Als Beispiel dient dabei wieder das In-Body-Network. Gibt es im Netzwerk beispielsweise 20 unterschiedliche Kontrollnachrichten und 200 verschiedene Krankheitsbilder, die durch Nachrichten übermittelt werden sollen, so ergibt das insgesamt 220 Nachrichten. Mit

#### 4. Konzeption

---



**Abbildung 4.8.:** Beispielhafte Codierung der dreistelligen Hexadezimalzahl **ab1** (grün), wobei die Codierung in den Kleberbezeichnern stattfindet. In a) ist dafür das Tileset für alle notwendigen Tiles zur Codierung einer dreistelligen Hexadezimalzahl in den Kleberbezeichnern abgebildet. In b) ist die Self-Assembly für die Hexadezimalzahl **ab1** abgebildet.



**Abbildung 4.9.:** Beispielhafte Codierung der dreistelligen Hexadezimalzahl **ab1** (grün), wobei die Codierung in den Tilebezeichnern stattfindet. In a) ist dafür das Tileset für alle notwendigen Tiles zur Codierung einer dreistelligen Hexadezimalzahl in den Tilebezeichnern abgebildet. In b) ist die Self-Assembly für die Hexadezimalzahl **ab1** abgebildet.

einer zweistelligen Hexadezimalzahl könnten 256 Nachrichten codiert werden, wofür 33 Tiles für das Tileset und drei Tiles für die Self-Assembly benötigt werden.

Sollte das Netzwerk jedoch 300 verschiedene Krankheitsbilder unterstützen, wäre eine zweistellige Hexadezimalzahl unzulänglich. Obwohl eine dreistellige Hexadezimalzahl die Kapazität hat, ist es ineffizient, eine Zahl zu wählen, die 4096 Nachrichten kodieren kann. Dabei werden 49 Tiles im Tileset und vier Tiles in der Self-Assembly benötigt. Da nur 320 Nachrichten kodiert werden müssen, sollte ein kleineres Zahlensystem gewählt werden. Ein effizienterer Ansatz wäre die Verwendung einer dreistelligen Oktalzahl. Hierbei würden 25 Tiles für das Tileset und vier Tiles für die Self-Assembly ausreichen, um 320 Nachrichten zu kodieren.

Die Entscheidung für das optimale Zahlensystem hängt von der Gewichtung zwischen der Größe der Self-Assembly und der Größe des Tilesets ab. Wird die Tilebezeichner-Codierung verwendet, sollte das Zahlensystem so gewählt werden, dass die Ergebnisse der Formeln  $xy+1$  und  $y+1$  (für  $y$ -stellige Zahl der Basis  $x$ ) möglichst gering sind. Werden diesen Formeln spezifische Gewichtungen zugewiesen, kann das Auswahlverfahren durch den folgenden Pseudocode repräsentiert werden:

```

1 # m = Anzahl der Nachrichten die codiert werden muss
2 # w1 = Gewichtung Formel 1 (xy+1)
3 # w2 = Gewichtung Formel 2 (y+1)
4 def get_best_base(m, w1, w2):
5     # initialisieren
6     best_base = 2
7     best_score = inf
8     best_f1 = 0
9     best_f2 = 0
10
11    for b in range(2, m+1):
12        # bestimme Anzahl der Ziffern von m zur Basis b
13        y = convert_m_to_base(m, b)
14        f1 = b * y + 1
15        f2 = y + 1
16        score = w1 * f1 * w2 * f2
17        # wenn der aktuelle Score kleiner ist,
18        # liegt effizientere Basis vor
19        if score < best_score:
20            best_score = score
21            best_base = base
22            best_f1 = f1
23            best_f2 = f2
24    return best_base, best_f1, best_f2

```

**Programmcode 4.1:** Pseudocode für eine Funktion zur Definition der besten Basis abhängig von der Anzahl der zu codierenden Nachrichten und den Gewichtungen von Tileset und Assembly. Der gesamte Code ist im Anhang A verlinkt.

Die Gewichtung der Formeln kann beliebig gewählt werden. Ist die Gewichtung beispielsweise  $w_1 = 1$  und  $w_2 = 10$ , so bedeutet dies, dass ein Tile in der Self-Assembly das gleiche

Gewicht hat wie zehn Tiles im Tileset.

In dieser Sektion wurde für Erklärungszwecke vereinfacht angenommen, dass die Codierung von DNA-Tiles auf verschiedene Zahlensysteme trivial ist. Tatsächlich erfolgt die Codierung auf Nanoebene stets in der DNA-Sprache von *ATCG*. Dafür wird angenommen, dass DNA analog auf Binärzahlen übertragen werden kann. So kann jedes *AT* oder *TA* Basenpaar beispielsweise binär als 0 und jedes Basenpaar *CG* oder *GC* binär als 1 codiert werden. Auch ein Zahlensystem der Basis vier ist denkbar, das die Reihenfolge der Basenpaare berücksichtigt.

Die Umwandlung von Binärzahlen in andere Zahlensysteme, wie Hexadezimalzahlen, kann ohne Overhead erfolgen. Beispielsweise lässt sich eine vierstellige Binärzahl direkt in eine Hexadezimalzahl übersetzen. Komplikationen ergeben sich jedoch, wenn man Binärzahlen in ein Dezimalsystem umwandeln möchte. Während eine vierstellige Binärzahl 16 Zahlen darstellen kann, ist eine dreistellige Binärzahl auf acht Zahlen beschränkt. Dies kann zu Overhead in der Codierung führen.

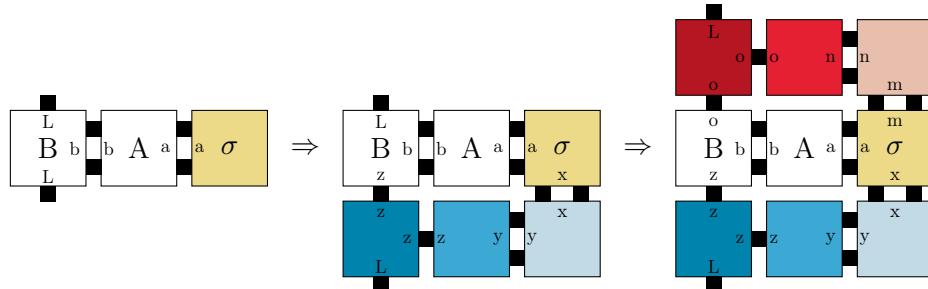
Da in dieser Arbeit das mathematische Tile-Modell betrachtet wird und die Codierung in den mathematischen Bezeichnern stattfindet, wird dies in der Arbeit nicht weiter betrachtet. Sollte der Mechanismus in einem Nanonetzwerk implementiert werden, muss darauf geachtet werden.

Nach Auswahl des effizientesten Zahlensystems können Nachrichten mittels Preprocessing in das entsprechende System umgewandelt werden. Dies ermöglicht die Erstellung von Nachrichtenmolekülen mit minimaler Anzahl an Tiles und kleinstmöglichen Assemblies für das Netzwerk. Weiterhin ermöglicht das Preprocessing eine Optimierung der benötigten Nachrichtenzahl. Einige Nachrichten könnten eventuell durch den Einsatz von Flags optimiert werden, die in der Sektion 4.12 näher erläutert werden.

## 4.11. Self-Assemblies in ihrer Höhe anpassen

Bevor mit weiteren Anforderungen fortgefahrene wird, muss zunächst die Höhe von Self-Assemblies thematisiert werden. In dieser Arbeit werden verschiedene Self-Assemblies dargestellt, die unterschiedliche Höhen aufweisen. Eine Verringerung der Höhe einer Self-Assembly ist nicht immer möglich, da hierbei Informationen verloren gehen könnten. Das bislang verwendete Beispiel für den Äquivalenzvergleich aus Abbildung 2.12 kann beispielsweise nicht in eine Self-Assembly der Höhe zwei oder eins umgewandelt werden, ohne grundlegende Änderungen an der Struktur und Funktionsweise der Self-Assembly vorzunehmen.

Es ist jedoch trivial, die Höhe einer Self-Assembly zu erhöhen. Dies wird beispielhaft in Abbildung 4.10 an einer Self-Assembly der Höhe eins gezeigt. Dabei müssen lediglich die Seed-Assembly und die Liganden angepasst werden. Der im Beispiel gezeigte Ligand wird zu einem Tile umfunktioniert, der die neu hinzugefügten Liganden bindet. Daher



**Abbildung 4.10.:** Beispiel für das Anpassen der Höhe einer Assembly. Links ist dabei eine Self-Assembly der Höhe eins dargestellt. Diese kann durch Anpassung der Seed-Assembly und des Liganden zu einer Assembly der Höhe zwei (mittig) oder der Höhe drei (rechts) erweitert werden. Dabei verändert sich die Information der Self-Assembly selbst nicht. Nur die inneren Kleber beziehungsweise die Liganden verändern sich.

kann einer Self-Assembly der Höhe eins einfach ein Rahmen hinzugefügt werden, um ihre Höhe auf zwei oder drei zu erhöhen, ohne die eigentliche Information der Self-Assembly zu verändern.

Dafür muss nur entweder der südliche oder nördliche Kleber von Seed-Assembly und Liganden angepasst werden. Dieser Mechanismus lässt sich für beliebige Höhen wiederholen. Dabei wird dann nicht die Seed-Assembly angepasst, sondern die Tiles nördlich und südlich der Seed-Assembly.

Mit dem beschriebenen Ansatz ist es möglich, für alle Self-Assemblies in einem System eine einheitliche Größe festzulegen. Dabei wird jedoch nicht betrachtet, ob der hinzugefügte Rahmen Informationen speichern kann, um die Effizienz der erweiterten Self-Assembly zu steigern. Dies muss individuell für jeden Fall betrachtet und neu implementiert werden. Mit diesem Verständnis über die Höhe von Self-Assemblies können nun die weiteren Anforderungen besprochen werden.

## 4.12. Flags in Nanonetzwerken

Flags werden in herkömmlichen Kommunikationsprotokollen verwendet, um bestimmte Eigenschaften oder Zustände einer Nachricht im Header zu kennzeichnen. Typische Flags sind beispielsweise Synchronisationsflags, welche in TCP verwendet werden, um einen Three-Way-Handshake zu signalisieren. Auch Acknowledgements werden in Flags verwendet, um zu signalisieren, dass das letzte Paket beim Empfänger angekommen ist. Die Finish-Flag zeigt an, dass es sich um die letzte Nachricht der Übertragung handelt, während die Reset-Flag zur abrupten Beendung einer Übertragung genutzt wird.

Es können je nach Anwendungsfall verschiedene Flags verwendet werden. Die hier vorgestellten Flags lassen sich jedoch nicht sinnvoll auf Nanonetzwerkebene abstrahieren.

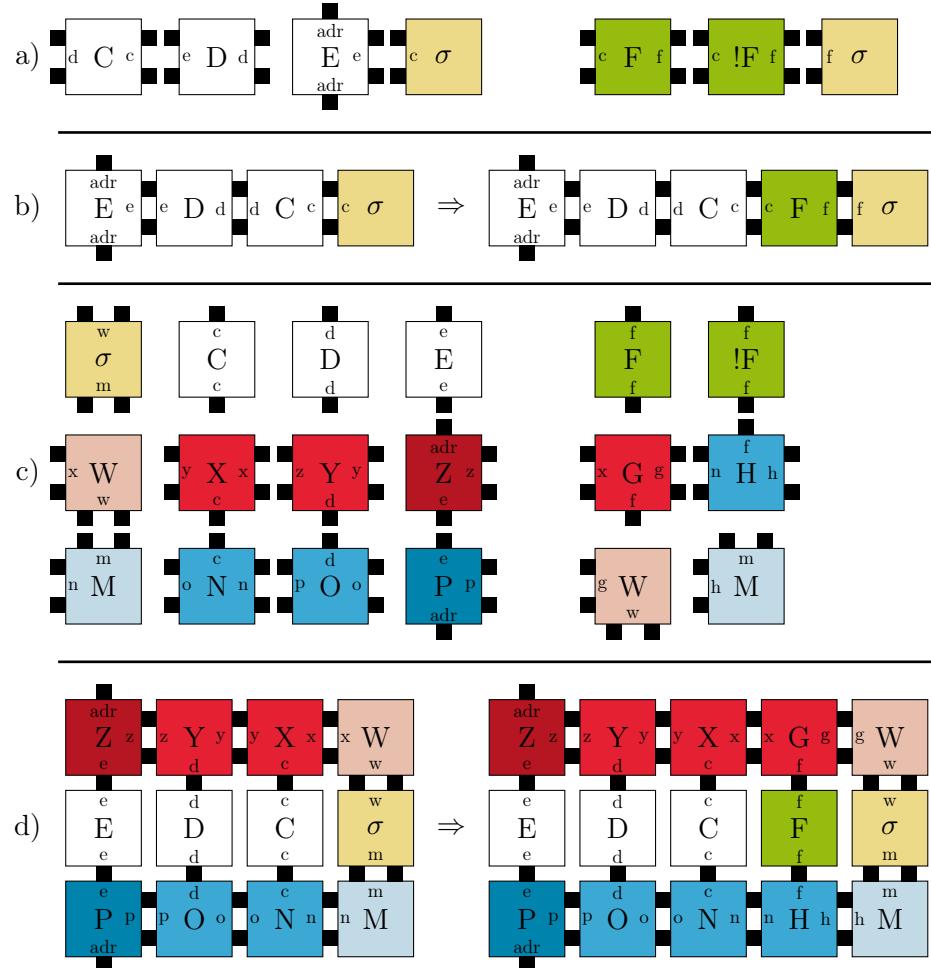
Acknowledgements wurden im letzten Absatz beschrieben und können in einem offenen System als eigene Nachricht verschickt werden. In einem auf DNA und Self-Assembly basierten Nanonetzwerk, in welchem über Nachrichtenmolekülen zwischen zwei Entitäten kommuniziert wird und mehrere Nachrichten in einer Übertragung verschickt werden, könnten sich Acknowledgement-Flags anbieten. Davon wird in dieser Arbeit jedoch nicht ausgegangen.

Synchronisation ist in Nanonetzwerken ein weiteres Problem. Dadurch, dass bei der Self-Assembly der Vorgang der Nachrichtenübertragung aus der Hand des Nutzers gegeben wird und sich alles auf dem sehr kleinen Skalenbereich der Nanoebene bewegt, lässt sich Zeit schwer synchronisieren. Es bietet sich allgemein an, asynchron zu kommunizieren. Ein Three-Way-Handshake oder andere Mechanismen, die eine relative Synchronisation zueinander ermöglichen, könnten eine Möglichkeit auf Nanoebene bieten. Dabei wird nicht die Zeit der Geräte synchronisiert, sondern ein relativer Zähler oder ein anderer Mechanismus verwendet, damit die kommunizierenden Geräte eine gemeinsam festgelegte Möglichkeit haben, um festzustellen, ob eine Nachricht verloren gegangen ist. Jedoch wurde in diesem Kapitel beschrieben, dass in Nanonetzwerken die Nachrichtenmenge minimalisiert werden soll und der Dialog von zwei Geräten über mehrere Nachrichten ungewöhnlich für Systeme auf dieser Skala ist. Somit wird für diese Arbeit asynchrone Kommunikation angenommen, da viele der Mechanismen außerhalb der Tile-basierten Self-Assembly durchgeführt werden müssten.

Einige Anwendungsmöglichkeiten für Flags existieren dennoch. Bei Übertragungen, die sich über mehr als eine Nachricht ziehen, kann eine Finish- oder eine Reset-Flag sinnvoll sein. Auch im Fall, dass immer nur einzelne Nachrichten zwischen zwei Entitäten des Netzwerks verschickt werden, kann es Anwendungsmöglichkeiten für Flags geben. Ein Beispiel dafür kann ein In-Body-Network sein, das  $x$  verschiedene Diagnosenachrichten verwendet. Es wird angenommen, dass diese  $x$  Nachrichten wie in Sektion 4.10 codiert werden. Soll in dem System für jede Diagnose hinzugefügt werden, ob zuvor Medikamente ausgeschüttet wurden oder noch nicht, so kann dies über zwei Wege gemacht werden. Eine Möglichkeit ist es, jede Nachricht auf zwei unterschiedliche Nachrichten aufzuteilen. Somit gibt es statt  $x$  verschiedenen Nachrichten  $2x$  verschiedene Nachrichten, die wiederum codiert werden müssen. Das gleiche Ergebnis könnte jedoch auch durch eine „Medizin ausgeschüttet“-Flag erzielt werden, ohne dabei die Menge der Nachrichten zu verdoppeln. Für welches  $x$  oder für welchen anderen Anwendungsfall dies sinnvoll ist, muss je nach System betrachtet werden.

Die benötigten Tiles hängen dabei von der Höhe der Self-Assembly ab. In Abbildung 4.11 werden die Unterschiede in den Tilesets und Self-Assemblies zwischen einem Molekül ohne Flags und solchen mit Flags gezeigt. In diesem Beispiel für Moleküle der Höhe eins und drei.

Abbildung 4.11 a) zeigt das notwendige Tileset für ein Molekül ohne Flags (links) und die zusätzlichen bzw. veränderten Tiles für die Flag-Integration (rechts). Bei Hinzufügung von Flags werden ein F- und ein !F-Tile benötigt, um die unterschiedlichen Flag-Werte zu



**Abbildung 4.11.:** Darstellung der Erweiterung von Self-Assembly Molekülen durch Flagtiles. In a) und c) sind die jeweiligen Tilesets links dargestellt. Rechts sind jeweils die Tiles dargestellt, die entweder überarbeitet (bei gleichem Bezeichner) oder neu hinzugefügt werden müssen, um Flags zu implementieren. In b) und d) sind jeweils die Umwandlungen der originalen Self-Assembly zur Self-Assembly mit einem Flag Tile abgebildet.

repräsentieren. Bei einem Molekül der Höhe eins muss zudem nur das Seed-Tile geändert werden, um den korrekten Kleberbezeichner für die Flag-Tiles zu haben. Abbildung 4.11 b) zeigt die Transformation eines Moleküls der Höhe eins, bei dem die Flag auf eins gesetzt ist.

Abbildung 4.11 c) und d) stellen das gleiche für ein Molekül der Höhe drei dar. Für diese Höhe müssen zusätzlich zwei weitere Tiles hinzugefügt werden und die Tiles M und W modifiziert werden. Im Worst-Case können bis zu sieben Tiles betroffen sein, obwohl in diesem Beispiel nur sechs gezeigt werden. Das liegt daran, dass das in c) definierte  $\sigma$ -Tile nicht angepasst werden muss, da es keinen westlichen Kleber besitzt.

Der gleiche Aufbau und Mechanismus lässt sich für die Darstellung von Prioritätsleveln in Self-Assemblies anwenden. Dies wird in der folgenden Sektion vorgestellt.

## 4.13. Prioritätslevel auf Basis von Flags

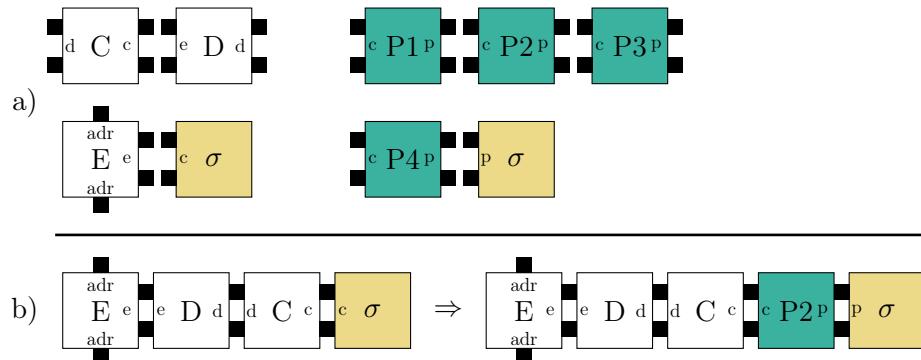
Beim Thema Datenflusskontrolle und Gerechtigkeit wurde über eine Priorität einer Nachricht gesprochen. Ein mögliches Anwendungsbeispiel dafür wurde in Tabelle 4.1 gegeben. Mit dem gleichen Ansatz wie für Flags kann auch eine Priorität in einer Self-Assembly codiert werden. Statt der Verwendung von zwei Flag-Tiles für gesetzte und nicht gesetzte Flags, werden hier Tiles für die verschiedenen Prioritäten benötigt. Dies ist in Abbildung 4.12 beispielhaft am gleichen Molekül wie aus Abbildung 4.11 b) dargestellt. Dabei muss für jede Priorität ein Tile existieren. In der Self-Assembly in Abbildung 4.12 b) wird beispielsweise das P2-Tile verwendet. Die Menge an benötigten Tiles hängt dabei immer von der Menge an Prioritätsleveln ab. In diesem Beispiel werden für die vier Prioritätslevel aus Tabelle 4.1 entsprechende Tiles erstellt.

Für sowohl die Priorität und die Flags muss ein Mechanismus existieren, der immer nur die richtigen Tiles ausschüttet. Wenn alle Flag-Tiles oder Priorität-Tiles ohne Filterung ausgeschüttet werden, dann ist es zufällig, welches Tile sich im Zuge der Self-Assembly bindet. So muss sichergestellt werden, dass bei gesetzter Flag nur das F-Tile ausgeschüttet wird und nicht das !F-Tile. Das Gleiche gilt für das richtige Prioritätstile.

## 4.14. Einordnung der vorgestellten Mechanismen im ISO/OSI-Modell

Da so alle Mechanismen in einem gesammelten Konzept vorgestellt wurden, kann zurück auf das ISO/OSI-Modell gekommen werden, um die Mechanismen in den verschiedenen Schichten einzurordnen. Dabei werden zuerst die anwendungsorientierten und danach die transportorientierten Schichten betrachtet.

Wie in Kapitel 2 beschrieben wurde, könnte es vorteilhaft sein, höherlevelige Anfor-



**Abbildung 4.12.:** Beispielhafte Darstellung durch Implementierung von Prioritätstiles. In a) ist links das originale Tileset dargestellt und rechts die Erweiterung, die für Prioritätstiles notwendig sind. In b) ist die Umwandlung der Self-Assemblies dargestellt, wenn Prioritätstiles hinzugefügt werden sollen. Dabei ist anzumerken, dass verschiedene Prioritätstiles an derselben Stelle in der Self-Assembly binden können. Aus dem Tileset in a) darf nur eines der vier Prioritätstiles freigelassen werden, wenn die Nachricht ein festgelegtes Prioritätslevel haben soll.

derungen aus der Anwendungsschicht auf der Mikroebene statt auf der Nanoebene zu behandeln. Einige Anforderungen, wie die Ressourcenallokation oder die Zugangskontrolle, gestalten sich bei einer Implementierung mit Tiles und Self-Assembly als herausfordernd. Auch die Anforderungen der Darstellungsschicht wurden nicht vertieft. Aus der Kommunikationsschicht wurden zwar Mechanismen wie Routing und Adressierung aus der Kommunikationsschicht berücksichtigt, doch eine konkrete Implementierung dieser anwendungsorientierten Schichten blieb aus.

Aus den transportorientierten Schichten wurde mit der Datenflusskontrolle ein Mechanismus aus der Transport- und Vermittlungsschicht vorgestellt und auf die Nanoebene übersetzt. Auch das Framing und die Fehlerbehandlung aus der Vermittlungs- und Sicherungsschicht wurde näher betrachtet und durch Prüfsummen und Snaked-Proofreading vorgestellt.

Die Codierung von Nachrichten auf die Nanoebene passt nicht nahtlos in das traditionelle ISO/OSI-Modell. Zwar zieht dieser Mechanismus Parallelen zur Modulation aus den transportorientierten Schichten, kann jedoch ebenso als eine Form von Anwendungsschnittstelle interpretiert werden. Dabei dient er als Brücke, um Nachrichten aus Systemen höherer Ebenen in die betrachtete Nanoebene zu übersetzen.

In der folgenden Sektion wird anhand eines Beispiels gezeigt, wie die vorgestellten Mechanismen in Kombination angewendet werden können.

## 4.15. Ein zusammenfassendes Beispiel für alle Anforderungen

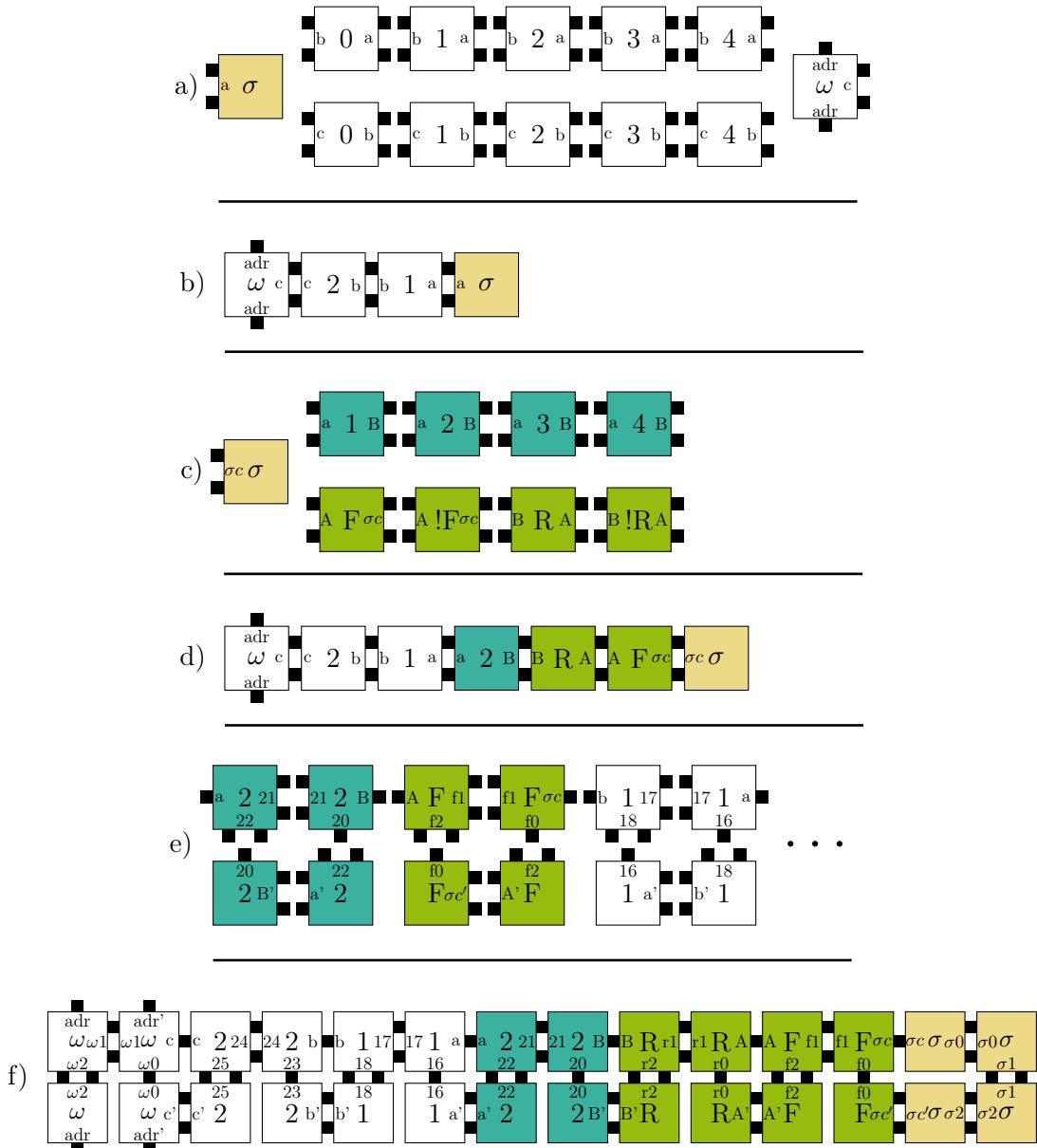
Abschließend werden alle bis hierhin vorgestellten Anforderungen und deren Lösungen in einem Beispiel präsentiert. Dabei liegt der Schwerpunkt auf der Reihenfolge der Implementierung der verschiedenen Ansätze. Als Beispiel dient dabei das Molekül aus Abbildung 4.13. Für das Beispiel wurden die folgenden Parameter gewählt:

- Nachrichtenzahl: 25
- Tileset Gewichtung: 1
- Assembly Gewichtung: 10
- Temperatur: 2
- Prioritätslevel: 4
- Flag 1: F
- Flag 2: R

Die konkrete Bedeutung jedes einzelnen Parameters wird im folgenden Kapitel erläutert, wenn das Skript genauer beschrieben wird. Durch die Gewichtungen für Tileset und Assembly ergibt sich die Basis fünf mit zwei Stellen als beste Basis zur Codierung von 25 verschiedenen Nachrichten. Die Adressierung geschieht, wie beschrieben, in den Liganden des letzten Tiles der Self-Assembly. In Abbildung 4.13 a) ist das so definierte Tileset abgebildet. Mit der Seed-Assembly  $\sigma$  und dem Liganden  $\omega$ . In b) ist das durch Self-Assembly gebildete Molekül für eine Nachricht abgebildet, das mit der Zahl 21 codiert wird.

Es wurde für dieses Beispiel darauf verzichtet, das Molekül mit einer Prüfsumme zu erstellen, da das Tileset dann statt zwölf Tiles 31 Tiles beinhalten würde. Auch die Assembly wäre um ein Tile größer, was in späteren Schritten weiter die Übersichtlichkeit beeinflussen würde. Das Vorgehen bei Prüfsummen ist in Abbildung 4.4 dargestellt.

In diesem Beispiel werden in Schritt c) sowohl zwei Flags als auch vier Prioritätslevel hinzugefügt. Dafür muss das Seedtile angepasst werden und die entsprechenden Tiles erstellt werden. Die Flagtiles sind in der Abbildung 4.13 grün markiert. Die Prioritätstiles haben die Farbe Türkis. Dabei werden die vier Flagtiles F, !F, R und !R erstellt sowie die vier Prioritätstiles 1, 2, 3 und 4. In d) ist die entsprechend neue Self-Assembly des Moleküls dargestellt. Die Flag- und Prioritätstiles binden sich zwischen Seed-Assembly und restlichem Molekül. Somit ist das gesamte Molekül vollständig erstellt. Erst hier kann Proofreading auf das Tileset angewendet werden. Das neue Tileset ist in c) angedeutet. Das gesamte Tileset umfasst  $20 \times 4 = 80$  Tiles, da die insgesamt 20 Tiles aus a) und c) im Snaked-Proofreading durch jeweils vier neue Tiles ersetzt werden. Doch diese Menge an Tiles würde die Abbildung sprengen. In f) ist das finale Molekül abgebildet. Das Tileset mit zwölf Tiles und die Self-Assembly mit vier Tiles, welches in a) und b) dargestellt wurde, wird so durch Flag- und Prioritätstiles in c) und d) auf 20 Tiles im Tileset und



**Abbildung 4.13.:** Darstellung von Tilesets und Assemblies nach dem Durchführen einzelner Mechanismen, die in diesem Kapitel vorgestellt werden. Dabei wurde auf die Prüfsummenerstellung verzichtet. In a) und b) wird die Nachrichtencodirung mit der Assembly für 25 Nachrichten dargestellt. Durch die gewählte Gewichtung ergibt sich die Basis fünf und Zifferanzahl zwei. In c) wird das Tilesset für die zwei Flags F und R und die vier Prioritätslevel erweitert und angepasst. In d) ist die daraus resultierende Self-Assembly dargestellt. In e) wird das Tilesset angedeutet, nachdem Snaked-Proofreading auf das Tilesset von a) und c) angewendet wurde. In f) ist die finale Self-Assembly abgebildet.

sieben Tiles in der Self-Assembly erweitert. Danach wird die Menge an Tiles im Tileset durch Snaked-Proofreading auf 80 erweitert. Die Assembly beinhaltet zum Schluss 28 Tiles.

Mechanismen, die in diesem Beispiel nicht betrachtet wurden, sind: Framing, Routing, Adressierung und Fehlererkennung durch Prüfsummen. Die Fehlererkennung wurde aus Übersichtsgründen nicht in diesem Beispiel aus Abbildung 4.13 vorgestellt. Framing, Routing und Adressierung wurden in diesem Kapitel als in der Implementierung der Self-Assembly inhärent vorgestellt. Somit ist das Konzept vorgestellt und in einem großen Beispiel zusammengefügt.

In diesem Kapitel wurden einige Lösungen für Anforderungen aus herkömmlichen Kommunikationsprotokollen vorgestellt. Es wurden inhärente Anforderungen, wie die Adressierung, das Framing und das Routing vorgestellt. Auch nicht vorhandene Anforderungen wurden betrachtet und auf die Nanoebene übersetzt. Dazu zählt die Datenflusskontrolle, die Nachrichtencodierung, die Fehlererkennung und Fehlerkorrektur. Im folgenden Kapitel werden diese Ansätze konstruiert, um sie im Weiteren analysieren und evaluieren zu können.





## 5. Konstruktion des Skripts

Dieses Kapitel behandelt die Erstellung der Tilesets, die zur Simulation des zuvor erörterten Konzepts mit der Simulationsumgebung NetTAS benötigt werden. Im Zentrum dieses Kapitels steht ein Python-Skript, das im Rahmen dieser Arbeit entwickelt wurde. Das Skript, seine Funktionen und Eigenschaften werden im Folgenden detailliert beschrieben. Interessierte können das Skript über den QR-Code am Seitenrand oder über die Links im Anhang A auf Github einsehen. Zunächst wird die Simulationsumgebung NetTAS erläutert. Im Folgenden wird dargelegt, wie das Skript anhand verschiedener Parameter Tilesets erstellt. Für die generierten Tilesets wird zudem eine Erweiterung vorgestellt, die Prüfsummen auf Tilesets implementiert. Es wird auch erörtert, welche Voraussetzungen für externe Tilesets bestehen, damit sie durch das Skript korrekt modifiziert werden können. Hierzu wird ein Farbcde eingeführt. Zudem wird das grafische Interface des Skripts beleuchtet. Das Kapitel endet mit einer Beschreibung des Gesamtablaufs des Skripts, einschließlich der Umsetzung von Flags, Prioritätstiles und Snaked-Proofreading auf beliebigen Tilesets.

### 5.1. NetTAS

Um die Konstruktion von den im vorherigen Kapitel vorgestellten Anforderungen durchführen zu können, muss zunächst auf die Simulationsumgebung NetTAS und den Aufbau von Tiles und Tilesets in diesem Kontext eingegangen werden.

Die in Kapitel 2 vorgestellten Tile-Assembly-Modelle können in der Simulationsumgebung *NetTAS* verwendet werden, um Tilesets zu simulieren. Dabei handelt es sich um die Tile-Assembly-Modelle aTAM, kTAM, 2HAM und kTHAM. Das Tool wurde in TypeScript programmiert und kann über eine Webanwendung über den Browser verwendet werden [21]. Dieses Tool wird nicht nur wegen der besseren Verfügbarkeit und Dokumentation für die Arbeit verwendet, sondern auch, weil es die einzige Simulationsumgebung ist, die das kTHAM beinhaltet.

In NetTAS können Tilesets erstellt werden, indem Label, Kleberbezeichner und Kleberstärken in einem grafischen Interface eingegeben werden. Diese Tilesets können in JSON-Dateien gespeichert und geladen werden. Die JSON-Dateien haben eine spezifische Form, die in der Konstruktion des Skripts einen entscheidenden Faktor spielt.

Dafür wird zunächst die Form eines Tilesets in JSON vorgestellt und analysiert:

```

1  {
2      "_tiles": [
3          {
4              "label": "A",
5              "glues": [
6                  {
7                      "label": "a",
8                      "strength": 1
9                  },
10                 {
11                     "label": "b",
12                     "strength": 2
13                 },
14                 {
15                     "label": "c",
16                     "strength": 2
17                 },
18                 {
19                     "label": "",
20                     "strength": 0
21                 }
22             ],
23             "color": "white"
24         },
25         {
26             "label": "B",
27             .
28             .
29             .
30         }
31     ]
32 }
```

**Programmcode 5.1:** Darstellung eines beispielhaften Tilesets in JSON-Form. Tiles werden in `_tiles` gelistet. Hier ist ein weißes Tile mit dem Bezeichner A und den folgenden Klebern dargestellt: nördlicher Kleber a der Stärke eins, östlicher Kleber b mit Stärke zwei, südlicher Kleber c mit Stärke zwei und keinem Kleber oder Label auf der westlichen Seite des Tiles. Auch wird ein Tile B angedeutet, um zu zeigen, wie weitere Tiles aufgelistet werden.

Jedes Tile des Tilesets wird in den `_tiles` gespeichert. Ein Tile enthält den Bezeichner *label* und die zugehörigen Kleber *glues*. Letztere bestehen aus vier Tupeln, wobei jedes Tupel den Kleberbezeichner *label* und die Kleberstärke *strength* für eine bestimmte Seite des Tiles repräsentiert. Dabei ist die Reihenfolge wie folgt festgelegt:

1. Tupel: nördlicher Kleber (hier Bezeichner a, mit Stärke eins)
2. Tupel: östlicher Kleber (hier Bezeichner b, mit Stärke zwei)
3. Tupel: südlicher Kleber (hier Bezeichner c, mit Stärke zwei)

4. Tupel: westlicher Kleber (hier Bezeichner "", mit Stärke null)

Die letzte Information ist die Farbe des Tiles. Diese wird in der Variable *color* gespeichert. Durch Aneinanderreihung solcher Tiles kann so ein Tileset mit allen benötigten Informationen gespeichert oder geladen werden.

## 5.2. Die Generierung von Tilesets im Skript

Diese vorgestellte Struktur kann in einem Python-Skript nachgebaut werden. So können Tilesets verändert, beziehungsweise generiert werden. Bei der Generierung eines Tilesets wird im Skript auf die Nachrichtencodierung aus dem letzten Kapitel geachtet. Für eine Anzahl an Nachrichten in einem fiktiven System mit Gewichtungen für Tileset- und Assemblygröße, wird ein Tileset automatisch generiert. Diese Codierung hält sich dabei an den Ansatz, bei welchem die Information in die Tilebezeichner codiert werden kann. So besitzt jedes generierte Tileset bei der Self-Assembly eine Molekülhöhe von eins. Der Vorgang für die Generierung des Tileset kann durch folgenden Pseudocode dargestellt werden:

```

1 # mc = message count
2 # tw = tileset weight
3 # aw = assembly weight
4 def generate_data(mc, tw, aw):
5     base = find_best_base(mc, tw, aw)
6     digits = get_digits_for(mc, base)
7     tiles = []
8     tiles.append(generate_seedtile())
9     for each digit in digits:
10         for each number in base:
11             tiles.append(generate_tile(digit, number))
12     tile.append(generate_ligand())
13     return {"_tiles": tiles}

```

**Programmcode 5.2:** Pseudocode für die Generierung eines Tilesets für gegebene Anzahl zu kodierenden Nachrichten und den Gewichtungen von Tileset- und Assemblygröße. Es wird die optimale Basis für die Anzahl der Nachrichten nach den gegebenen Gewichtungen berechnet. Anschließend werden aus der Anzahl und der Basis die erforderlichen Ziffern ermittelt. Mit beiden Parametern kann das Tileset generiert werden.

Zur Generierung des optimalen Tilesets, wird zuerst die optimale Basis mit der Funktion `find_best_base` bestimmt. Nachdem diese Basis festgelegt wurde, lässt sich ermitteln, wie viele Ziffern codiert werden müssen. Sobald die optimale Basis und die Anzahl der Ziffern durch die Variablen `base` und `digits` festgelegt sind, kann das Tileset generiert werden. Zunächst wird die Seed-Assembly generiert. Danach kann für jede Ziffernstelle und jede Zahl in der Basis ein Tile erstellt werden. Dabei wird die Zahl im Tilebezeichner notiert, die Kleberbezeichner legen die Stelle in der Ziffernfolge fest. Zuletzt wird der

Ligand erstellt, der sich an die letzte Ziffernstelle binden kann.

### 5.3. Die Generierung von Prüfsummen in Nanonetzwerken

Eine Erweiterung zur Generierung von Tilesets, ist die Generierung von Tilesets mit integrierter Prüfsumme. Dabei wird die gleiche Idee verwendet, jedoch wird sie rekursiv durchgeführt. Folgender Pseudocode stellt die Idee zur Generierung von Tilesets mit Prüfsummen dar:

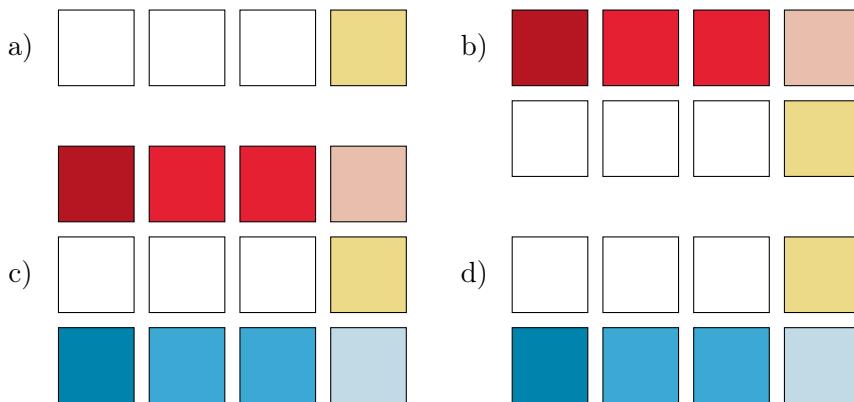
```

1  # mc = message count
2  # tw = tiles set weight
3  # aw = assembly weight
4  def generate_data_with_checksum(mc, tw, aw):
5      base = find_best_base(mc, tw, aw)
6      digits = get_digits_for(mc, base)
7      tiles = []
8      tiles.append(generate_seedtile())
9      for each digit:
10         tiles.append(generate_data_recursive(digit, base))
11      for each num in message_count:
12         tiles.append(generate_checksumligand())
13
14  def generate_data_recursive(digit, base):
15      temp_tiles = []
16      if digit == 1:
17          for each num in base
18              temp_tiles.append(generate_tile())
19      else:
20          temp_tiles.extend(generate_data_recursive(digit -1))

```

**Programmcode 5.3:** Pseudocode für die Generierung eines Tilesets mit Prüfsumme. Die Rekursion wird dabei dazu genutzt, um exponentiell mehr Tiles für jede weitere Ziffernstelle zu erstellen. Auch muss statt einem einzelnen letzten Tile, wie in `generate_tile`, für alle codierten Nachrichten ein Prüfsummentile erstellt werden.

Durch die Konstruktion mit Prüfsumme ist das Tileset in den meisten Fällen sehr groß. Deshalb wurde im Skript und in dieser Arbeit darauf verzichtet, die Prüfsummengenerierung für andere Tilesets und Moleküle zu implementieren. Diese wären jedoch analog, wobei weitere Informationen gesammelt werden müssen. Dabei muss jedes Tileset auf Positionen aller Tiles in der Self-Assembly analysiert werden. Dann kann festgestellt werden, wie viele Tiles an welchen Stellen des Moleküls vorkommen können. Daraus kann durch das gleiche Prinzip wie im Code 5.3 das Tileset mit einer Prüfsumme versehen werden.



**Abbildung 5.1.:** Darstellung der korrekten Farbkodierung von Tilesets: a) für Moleküle der Höhe eins, b) sowie d) für Moleküle der Höhe zwei und c) für Moleküle der Höhe drei. Die Tiles müssen in diesem Fall so codiert werden, damit im Skript aus einzelnen Tiles positionsbezogene Informationen gewonnen werden können. Die Farbkodierung leitet sich immer aus der Position der gelb codierten Seed-Assembly ab.

## 5.4. Farbcode als Anforderung an Tilesets

Nachdem nun die Generierung von Tilesets durch das Skript detailliert erläutert wurde, sollen im Weiteren die Anforderungen für die korrekte Anwendung der Mechanismen im Skript beschrieben werden.

Sollen die Mechanismen auf ein bereits in NetTAS entworfenes Tileset angewendet werden, so gilt es, Informationen über die Positionen der Tiles zu sammeln. Da Tiles intrinsisch keine positionellen Informationen speichern und ihre Position erst während der Self-Assembly festgelegt wird, benötigt das Skript zusätzliche Informationen. Diese können über Farbcodierung erreicht werden. In NetTAS haben die Farben der Tiles primär ästhetische Funktionen, sie erleichtern den Nutzenden die Übersicht und Strukturierung. Genau diese Eigenschaft kann genutzt werden, um zusätzliche Informationen über die Tiles zu speichern.

In Abbildung 5.1 sind beispielhaft korrekte Farbkodierungen für Assemblies dargestellt. Die genaue Bedeutung der Farben für die Tiles wird im Weiteren erläutert. Dabei sollte beachtet werden, dass die Eingabe eines Tilesets für ein Molekül mit einer Höhe von mehr als drei Tiles nicht empfohlen wird, da höhere Self-Assemblies bisher nicht getestet wurden.

Insgesamt werden acht verschiedene Farben zur Codierung benötigt, die alle auf dem Farbschema der Universität Lübeck basieren [7]. Während rot, gelb und blau als Farbkombination ausgewählt wurden, standen ursprünglich die drei Grundfarben rot, grün und blau zur Debatte. Die Kombination von Grün- und Blautönen zeigte, besonders bei helleren Tönen, eine zu große Ähnlichkeit und wurde daher ausgeschlossen. Die Kombi-

nation von Rot und Grün stellt insbesondere für Personen mit einer Rot-Grün-Schwäche ein Problem dar. Zwar können Menschen ohne diese Sehschwäche die Farben klar differenzieren, jedoch birgt diese Kombination eine zweite Herausforderung: die Assoziation von rot und grün mit den Begriffen *inkorrekt* und *korrekt*. Aus beiden Gründen wurde die Rot-Grün-Kombination ausgeschlossen. Somit bleibt als letzte Kombination noch Rot und Blau. Diese sind gut unterscheidbar und eine Rot-Blau-Schwäche ist ungewöhnlicher als eine Rot-Grün-Schwäche [6]. Da es jedoch drei zu betrachtende Teile eines Moleküls gibt, ist eine weitere Farbe notwendig. Diese Farbe muss gut dazu passen, aber dennoch klar von den anderen beiden Farben unterscheidbar sein. Dabei wurde gelb gewählt, da die Farbe beiden Anforderungen entspricht. Als „Farbe“ für alle nicht betrachteten Tiles wird weiß verwendet.

In NetTAS können alle Farben über den HTML-Farbcodes oder den CSS-Farbcodes definiert werden. Die verwendeten Farben sind in Tabelle 5.1 dargestellt. Die Seed-Assembly wird gelb dargestellt. Wenn eine nördliche Grenze über der Seed-Assembly existiert, wird diese in verschiedenen Rottönen kodiert. Dabei repräsentiert der hellste Rotton das Tile direkt über der Seed-Assembly, während der dunkelste Rotton den Liganden der nördlichen Grenze markiert. Alle dazwischen liegenden Tiles werden in einem mittleren, kräftigen Rot dargestellt. Analog zur roten Kodierung wird jede Grenze südlich der Seed-Assembly blau gehalten. Tiles, die westlich der Seed-Assembly liegen, sind in Weiß codiert.

Dabei ist anzumerken, dass die Farben durch CSS nicht vollständig dem Farbschema der Universität Lübeck entsprechen. Sie wurden so gewählt, dass sie möglichst nahe der jeweiligen HTML-Farbe sind. Diese sind wiederum dem Schema entsprechend. Bei der Farbkodierung sollte beachtet werden, dass konsequent entweder der CSS- oder der HTML-Farbcodes genutzt werden sollte. Einzig bei den weißen Tiles ist eine Mischung möglich, da sie in beiden Farbcodes identisch sind. Zum Beispiel kann im HTML-Farbcodes die Bezeichnung `white` verwendet werden. Eine andere Überschneidung existiert jedoch nicht und führt zu Errors im Skript. Es muss eine Codierung exklusiv verwendet werden und die dazugehörige Checkbox im Grafikinterface ausgewählt werden, damit das Skript korrekt ausgeführt werden kann.

## 5.5. Das grafische Interface des Skripts

In dieser Sektion wird das grafische Interface des Skripts `tile-generator.pyw` vorgestellt. Dieses Skript wurde speziell für Windows-Betriebssysteme konzipiert, wobei das Suffix `w` in der Dateiendung zur Ausführung durch `pythonw.exe` führt. Dadurch kann das Skript ohne die Windows-Konsole ausgeführt werden. Möchte man das Skript auf anderen Betriebssystemen nutzen, muss lediglich das `w` aus der Dateiendung entfernt werden, sodass die Datei als `tile-generator.py` vorliegt und in allen pythonfähigen Systemen lauffähig ist.

HTML	CSS	Position in Assembly
#ecda88	khaki	Seed-Assembly
#e8bfad	salmon	nördlich von Seed
#e42034	red	nördliche Grenze
#b51621	crimson	nördlicher Ligand
#c2d9e6	skyblue	südlich von Seed
#3ca9d5	deepskyblue	südliche Grenze
#0083ad	royalblue	südlicher Ligand
#0000	white	Rest

**Tabelle 5.1.:** Tabellarische Darstellung der Farbkodierung in HTML oder CSS wobei rechts davon die entsprechende Farbe abgebildet ist. Jede Farbe betrifft Tiles mit spezifischen Positionen in der Assembly.

Um sicherzustellen, dass das Skript korrekt ausgeführt wird, müssen einige Aspekte beachtet werden. Dafür wird in Abbildung 5.2 eine Übersicht über die grafische Nutzeroberfläche des Skripts gegeben, die zwei Screenshots des Interfaces zeigt. Dabei ist der linke Screenshot das grafische Interface in der Grundform, während der rechte Screenshot die GUI mit allen ausgeklappten und ausgewählten Optionen darstellt.

Von Oben nach Unten sind die Einstellungsoptionen des Skripts wie folgt: Die Auswahl einer Eingabedatei fällt auf den Fall, dass ein in NetTAS erstelltes Tileset verwendet wird. Die entsprechende Datei muss über einen Datenexplorer ausgewählt werden. Wird die Checkbox *Generiere Tileset* darunter ausgewählt, so öffnen sich drei Eingabefelder und eine Checkbox. Button und Eingabefeld der Eingabedatei werden dabei deaktiviert.

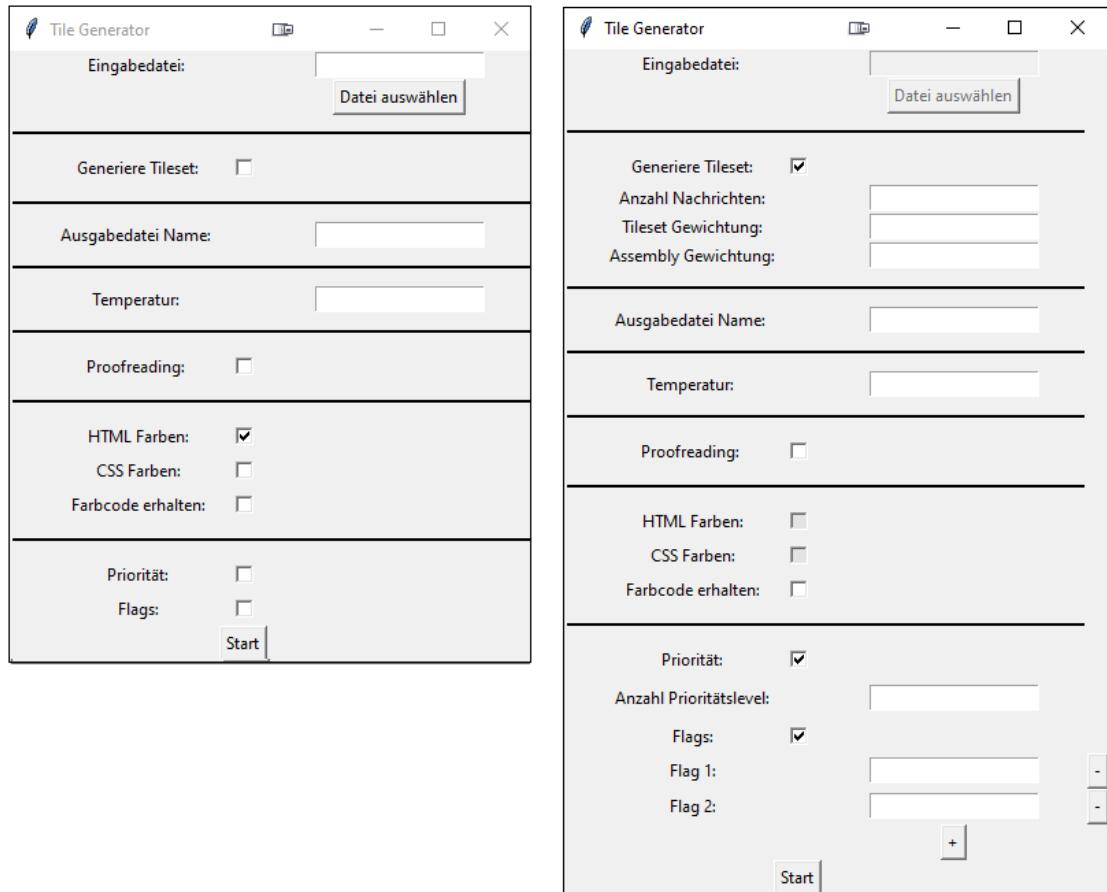
In den neuen Eingabefeldern können die folgenden drei Parameter festgelegt werden: die Anzahl der Nachrichten, die durch das Tileset darstellbar sein sollen, die Gewichtung des Tileset und die Gewichtung der Self-Assembly. Alle drei Eingaben müssen ganze Zahlen beinhalten und sollte eine andere Eingabe getätigt werden, so wird diese mit einer Fehlernachricht abgefangen. Die Gewichtung von Tileset und Assembly ist relativ. Das bedeutet, dass eine Eingabe von eins in der Tileset Gewichtung und einer zehn in der Gewichtung der Assembly äquivalent zu einer Eingabe von zehn in der Tileset Gewichtung zu einer 100 in der Gewichtung der Assembly ist. Durch Aktivieren der Checkbox mit der Beschriftung *Prüfsumme erstellen* wird die Prüfsummenbildung für das generierte Tileset aktiviert.

Unter dem Abschnitt zur Generierung von Tilesets findet sich das Eingabefeld der Ausgabedatei. Darin muss der Name der Datei angegeben werden. Ein `.json` am Ende des Namens ist nicht notwendig. Namen mit anderen Dateiendungen oder ohne Dateiendung werden erkannt und durch ein `.json` erweitert. Die Ausgabedatei wird immer an der gleichen Stelle im Verzeichnis wie das Skript abgespeichert.

Unter dem Eingabefeld für den Namen der Ausgabedatei ist ein weiteres Eingabefeld zu finden. Dieses dient zur Angabe der Temperatur. Auch hier kann erneut nur eine ganze

## 5. Konstruktion des Skripts

---



**Abbildung 5.2.:** Darstellung des grafischen Interfaces des im Zuge dieser Arbeit entstandenen Skripts. Links ist dabei das Fenster ohne weitere Auswahl dargestellt. Rechts sind alle zusätzlichen Eingaben und Optionen gezeigt, die beim Auswählen aller Checkboxes hinzugefügt werden.

Zahl angegeben werden, andere Eingaben führen zu einer Fehlermeldung. Darunter findet sich eine Checkbox mit der Beschriftung *Proofreading*. Wird diese Checkbox ausgewählt, so wird auf das finale Tileset *Snaked-Proofreading* angewendet.

Unter dieser Checkbox finden sich drei weitere Checkboxen. Die oberen beiden sind gegenseitig ausschließend, sodass immer nur eine aktiv sein kann. Mit ihnen kann anhand der Beschriftung angegeben werden, ob *HTML Farben* oder *CSS-Farben* als Farbkodierung verwendet werden. Im Fall einer Eingabedatei muss die Checkbox dem gewählten Farbschema aus dem Eingabetileset entsprechen. Ansonsten erscheint eine Fehlermeldung und das Skript kann nicht ausgeführt werden. Wird ein Tileset generiert, so sind diese beiden Checkboxen ausgegraut und nicht auswählbar, da sie keine Relevanz besitzen. Das generierte Tileset hat immer die Höhe eins und die einzige Farbkodierung ist ein Seed-Tile, das in HTML-Farbcode nach Tabelle 5.1 dargestellt wird. Die restlichen Tiles sind weiß codiert. Die dritte Checkbox, bezeichnet als *Farbcode erhalten*, steuert die Farbkodierung des Ergebnistilesets. Ist diese Option aktiviert, bleibt die Farbkodierung im generierten Tileset bestehen. Andernfalls wird die Farbkodierung entfernt und nur die Tiles, die den Frame des Moleküls darstellen, werden in der Farbe Ozeangrün hervorgehoben, um eine bessere Übersicht zu gewährleisten. Es sei angemerkt, dass Ozeangrün die Primärfarbe des Farbschemas der Universität Lübeck ist.

Unter den drei Checkboxen für die Farbkodierung, finden sich zwei weitere Checkboxen. Die erste Checkbox für die *Priorität* kann ausgewählt werden, wenn für das gegebene Tileset Prioritätstiles benötigt werden. Der genaue Ablauf wird in den folgenden Absätzen genauer betrachtet. Ist die Checkbox ausgewählt, öffnet sich ein Eingabefeld, in welchen eine ganze Zahl angegeben werden muss, um so anzugeben, wie viele Prioritäts-level es gibt. Die andere Checkbox mit dem Bezeichner *Flags* öffnet beim Auswählen einen Button, der mit „+“ bezeichnet wird. Durch diesen Button können beliebig viele Flags erzeugt werden, indem die dabei erscheinenden Eingabefelder mit Namen für die Tiles ausgefüllt werden. Mit dem rechts erscheinenden Button, gekennzeichnet durch ein „-“, kann eine falsch erstellte Flag wieder gelöscht werden. Ganz unten befindet sich der *Start*-Button, der das Skript mit den oben angegeben Optionen ausführt. Danach öffnet sich der Datenexplorer an der Stelle, an der die Ausgabedatei gespeichert wird.

## 5.6. Anforderungen für das korrekte Ausführen des Skripts

Mit allen erforderlichen Optionen und Eingabe, können im Weiteren die Anforderungen für in NetTAS erstellte Tilesets festgelegt werden, die erfüllt werden müssen, damit das Skript funktioniert. Danach sollen auch die Anforderungen angegeben werden, die eingehalten werden sollten, um mögliche Fehler zu vermeiden. Zuletzt sollen die Anforderungen vorgestellt werden, die erfüllt werden können, um ein korrektes Skriptverhalten zu sichern.

### 5.6.1. Notwendige Anforderungen an Tilesets

In dieser Sektion werden die Anforderungen präsentiert, die für eine korrekte Funktionsweise des Skripts berücksichtigt werden müssen. Ein Nichtbefolgen dieser Anforderungen führt fast immer zu Fehlern. Einige dieser Fehler werden durch das Skript mit einer Fehlermeldung identifiziert, während andere zu fehlerhaften Tilesets führen können. Die erste Anforderung, das Einhalten des Farbcodes, wurde bereits bei der Beschreibung des grafischen Interfaces gegeben.

Die zweite Anforderung betrifft die Ausrichtung des Moleküls im Tileset. Das Molekül muss so konstruiert werden, dass es mit der Seed-Assembly rechts beginnt und nach links wächst. Beginnt die Seed-Assembly hingegen rechts, oben oder unten und das Molekül wächst dann in eine andere Richtung (links, unten oder oben), kann dies im Skript zu schwerwiegenden Fehlern führen.

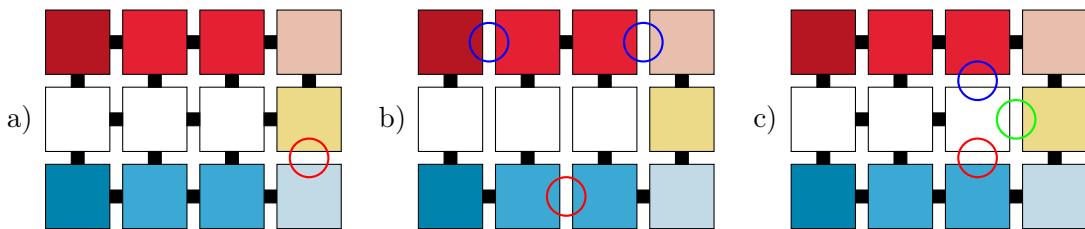
Ein weiterer essentieller Aspekt ist die klare Definition einer Wachstumsfront an einer Molekülseite. Zur Verdeutlichung dieser Anforderung sind in Abbildung 5.3 Negativbeispiele gezeigt. Es ist wichtig zu betonen, dass diese spezifische Anforderung primär im Kontext des Proofreadings relevant ist. Bei Konstruktionen ohne Proofreading könnten solche Strukturen problemlos sein. Bei Anwendung des Proofreadings ist es jedoch von zentraler Bedeutung, dass das verwendete Tileset konform mit den Wachstumsvorgaben ist. Im Proofreading-Verfahren beginnt das Wachstum stets in einem von vier festgelegten Tiles. Der Farocode ermöglicht eine differenzierte Wachstumsdynamik in verschiedenen Molekülteilen, aber es muss stets eine konsistente Wachstumsfront beibehalten werden.

In Abbildung 5.3 a) ist ein Beispiel für nicht-konformes Wachstum im Kontext des Proofreadings dargestellt. Bei Molekülen mit einer Höhe von mehr als einem Tile wird davon ausgegangen, dass das Wachstum vom Seed-Tile ausgeht. Für das in der Abbildung gezeigte Molekül wäre an der mit einem roten Kreis hervorgehobenen Position ein Kleber erforderlich.

In Abbildung 5.3 b) wird ein weiteres Problem bei der Wachstumsfront dargestellt. Für das Proofreading der zentralen Tiles bei Molekülen der Höhe drei ist es von Bedeutung, aus welcher Richtung das Wachstum erfolgt. Alle Tiles starten ihr Wachstum bei einem der vier Proofreadingtiles. Um dies zu gewährleisten, muss eine der beiden Seiten so gebildet werden, dass sie ohne Verbindung eines zentralen Tiles auskommt. In dem dargestellten Beispiel erfordert dies das Hinzufügen von Klebern an den Stellen, die durch die blauen oder den roten Kreis markiert sind.

Das letzte Negativbeispiel ist in Abbildung 5.3 c) zu sehen und ist mit dem Beispiel aus b) verbunden. In Molekülen der Höhe drei müssen die zentralen Tiles entweder von der Seed-Assembly, von der südlichen Grenze oder von der nördlichen Grenze startend wachsen. Ein Wachstum wie in c) wird durch das Skript nicht unterstützt, weshalb hier ein Kleber an einer der farbig markierten Stellen notwendig wäre.

Die vierte Anforderung besagt, dass das ausgewählte Tileset für die gewählte Temperatur



**Abbildung 5.3.:** Drei Beispiele von Tile-Assemblies, die im Skript zu Fehlern führen. Fehlen die Kleber zwischen Seedtiles und ihren nördlichen und südlichen Nachbarn, kann im Skript beim Proofreading keine korrekte Wachstumsrichtung gefunden werden. Dies ist in a) dargestellt. Auch muss entweder die nördliche Grenze oder die südliche Grenze des Moleküls durchgängig verbunden sein, da im Skript eine der beiden Seiten als Wachstumsfront definiert wird und das nicht mitten im Molekül wechseln kann. Das ist in b) zu sehen. Auch müssen alle zentralen Tiles (hier weiß codiert) von Norden, Süden oder Osten wachsen. Ein Wachstum von Westen bei zentralen Tiles ist nicht vorgesehen.

korrekt konstruiert sein muss. Das Skript selbst nimmt keine Korrekturen an fehlerhaften Verbindungen oder Tiles vor.

Die fünfte und abschließende Anforderung bezieht sich auf die Konsistenz von Kleberbezeichnern und Kleberstärken. Ein Tile darf auf keiner Seite einen Kleberbezeichner aufweisen, wenn dieser keine zugehörige Kleberstärke hat. Umgekehrt sollte keine Seite eines Tiles eine Kleberstärke  $> 0$  besitzen, ohne durch einen entsprechenden Kleberbezeichner gekennzeichnet zu sein. Im Skript werden beide Werte zur Herleitung des jeweils anderen genutzt.

### 5.6.2. Empfehlenswerte Anforderungen an Tilesets

Im Folgenden werden empfohlene Anforderungen vorgestellt, die, obwohl nicht zwingend erforderlich, dazu beitragen, potenzielle Fehlerquellen zu minimieren. Es sollte betont werden, dass ein Nichteinhalten dieser Anforderungen nicht notwendigerweise zu Problemen führt.

Ein zentrales Element dieser Empfehlungen ist die Notwendigkeit eines eindeutigen Bezeichners für jedes Tile. Dies ist insbesondere relevant, da beim Erstellen von Flag-, Prioritäts- und Snaked-Proofreading Tiles zusätzliche Kleberbezeichner benötigt werden. Diese werden durch Konkatenation des Tilebezeichners mit Zeichen aus einer nachfolgend dargestellten Liste generiert. Da so einige Kleberbezeichner automatisch entstehen, muss darauf geachtet werden, dass erstellte Kleberbezeichner nicht gleich heißen. Im Skript wird für die erste Stelle der drei inneren Kleber eine kleingeschriebene Version des Tilebezeichners verwendet und danach mit drei Zeichen aus folgender Liste konkateniert:

$$[0, 1, \dots, 8, 9, a, b, \dots, y, z, aa, ab, \dots, zy, zz]$$

Um zu verdeutlichen, wie die Kleberbezeichner generiert werden, wird das folgende Beispiel betrachtet:

**gegeben:**

- vier Tiles mit dem Bezeichner  $X$ ,
- ein Tile mit dem Bezeichner  $Y$ ,
- zwei Tiles ohne Bezeichner.

**innere Bezeichnertripel:**

- $[x0, x1, x2], [x3, x4, x5], [x6, x7, x8], [x9, xa, xb]$ ,
- $[y0, y1, y2]$ ,
- $[0, 1, 2], [3, 4, 5]$ .

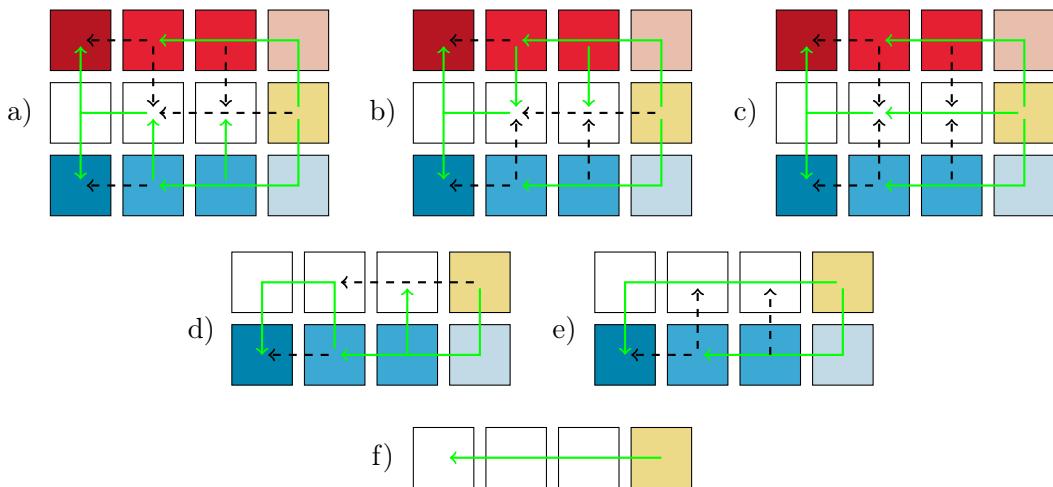
Dementsprechend muss beachtet werden, dass bei leeren Tilebezeichnern die generierten Kleberbezeichner einstellige Zahlen oder Buchstaben sind. Durch die Regeln der Self-Assembly kann es so beim automatischen Generieren von Kleberbezeichnern zu Wachstumsfehlern kommen. Wenn jedoch bei der Konstruktion des Tilesets in NetTAS darauf geachtet wird, muss dies keine Wachstumsfehler verursachen.

Damit verbunden sind allgemeine Regeln zur Kleberbezeichnung. Selbst wenn alle Tiles einen Bezeichner erhalten, sollte darauf geachtet werden, dass die genutzten Kleberbezeichner entweder einstellig und kleingeschrieben sind oder den folgenden Regeln entsprechen:

1. Bei Kleberbezeichnern mit einer Zeichenlänge von zwei oder drei darf das letzte Zeichen weder eine Zahl noch ein klein geschriebener Buchstabe sein.
2. Ein  $<$  am Ende von Bezeichnern der Länge zwei kann zu Fehlern führen.
3. Kombinationen von  $\{1, 2\}$  am Anfang, beliebig vielen Zeichen in der Mitte und  $\{F, P\}$  am Ende können von Flag- oder Prioritätstiles belegt sein.
4. Die Bezeichner  $\sigma t, \sigma c$  und  $\sigma b$  können belegt sein.
5. Großgeschriebene Buchstaben wie  $\{A, B, \dots, Y, Z, AA, AB, \dots, ZY, ZZ\}$  werden vom Skript generiert und sollten daher nicht manuell vergeben werden.

Während die Verwendung dieser Bezeichner nicht zwangsläufig Probleme verursacht, wird empfohlen, sie zu vermeiden, um potentiellen Fehlern vorzubeugen.

Insgesamt können durch die Zeichenmengen bis zu 237 gleich benannte Tiles unterschieden werden. Der Vollständigkeit halber muss trotzdem angemerkt werden, dass nicht mehr als 238 Mal der gleiche Tilebezeichner in einem Tileset verwendet werden sollte. Auch sind die Farbcodes für Flag- und Prioritätstiles im Skript zugelassen und führen nicht zu einem Error, wenn sie verwendet werden. Doch sind diese Farben nicht dafür gedacht, importiert zu werden. Sie werden im Skript als *white* gewertet und müssen so



**Abbildung 5.4.:** Darstellung von verschiedenen Molekülformen, die empfohlen werden, um die korrekte Ausführung des Skripts zu gewährleisten. Die grünen Pfeile stellen dabei die minimal benötigten Kleber dar, die schwarz gestrichelten Pfeile sind nicht zwingend notwendig, könnten aber benötigt werden, wenn es die Logik der Assembly verlangt. a), b) und c) sind dabei verschiedene Möglichkeiten für Moleküle der Höhe drei. d) und e) sind die zwei möglichen Wachstumsrichtungen für Moleküle der Höhe zwei und f) stellt die eine triviale Wachstumsrichtung für Moleküle der Höhe eins dar.

platziert werden, damit es zu keinen Fehlern kommt. Es ist empfehlenswert diese Farben im gegebenen Tileset nicht zu verwenden.

### 5.6.3. Vorschläge für Tilesets

Zuletzt werden noch einige Leitideen gegeben, denen gefolgt werden kann. Die sicherste Variante ist es, ein Tileset zu generieren. Das funktioniert bei sinnvollen und korrekten Eingaben im grafischen Interface immer fehlerfrei. Wenn jedoch ein Tileset in NetTAS erstellt wird, dann können die folgenden zwei Regeln eingehalten werden, um Fehlern vorzubeugen: Die Temperatur sollte zwischen zwei und drei gewählt werden und Tilesets für Moleküle sollten den Formen aus den Beispielen in dieser Arbeit folgen. Diese wurden alle getestet und funktionieren fehlerfrei. In Abbildung 5.4 sind die Molekülformen für die Höhen drei (in a,b und c), zwei ( in d und e) und eins (in f) gegeben. All diese Strukturen wurden ausreichend getestet und funktionieren im Skript. Durch die Pfeile ist das Wachstum im Proofreading verbildlicht. Die grünen Pfeile repräsentieren die minimalen Kleberverbindungen zwischen den Molekülen. Die schwarz gestrichelten Pfeile hingegen symbolisieren zusätzliche, optionale Verbindungen. Während sie nicht zwingend für die Bildung eines Moleküls dieser Form benötigt werden, können sie dennoch notwendig sein, um eine entsprechende Berechnung in der Self-Assembly zu ermöglichen.

## 5.7. Ablauf des Skripts

Es wurden die Voraussetzungen vorgestellt, um möglichst fehlerfrei den Start-Button im grafischen Interface des Skripts drücken zu können. Damit kann im Folgenden genauer darauf eingegangen werden, was im Skript passiert, nachdem der Start-Button gedrückt wird.

Wenn das Skript mit einer beliebigen Eingabe ausgeführt wird, erfolgt zunächst eine Prüfung der Eingabe. Dabei wird ermittelt, welche Checkboxen aktiviert wurden. Anschließend werden die zugehörigen Eingabefelder kontrolliert. So wird beispielsweise überprüft, ob die Eingabefelder von Temperatur, Nachrichtenzahl, Tileset/Assembly Gewichtung und Priorität Zahlen sind. Für Flags, Eingabedatei und Ausgabedatei wird überprüft, ob sie nicht leer sind. Nur wenn alle Checks positiv sind, wird die Eingabe in die Main-Funktion übergeben, um dort das finale Tileset zu erstellen. Die Main-Funktion gibt das Tileset im `_tiles` Dictionary zurück, wodurch dieses ohne weitere Änderungen in die Ausgabedatei geschrieben werden kann. Es erscheint eine Bestätigungsnachricht, dass das Skript erfolgreich ausgeführt wurde und der Datenexplorer öffnet sich am Speicherort der Ausgabedatei.

Für eine vollständig korrekte Ausführung muss noch betrachtet werden, was in der Main-Funktion passiert. Dies kann durch folgenden Pseudocode beschrieben werden:

```

1 def main(data, input)
2     new_tiles = []
3     gatherinfo(data)
4     if flags in input:
5         data.add(flags)
6     if priorities in input:
7         data.add(priorities)
8     sort(data, colorcode aus input)
9     if proofreading in input:
10        new_tiles = snakedproofreading(data)
11    else:
12        new_tiles = data
13    if not color_kept in input:
14        change_colors(new_tiles)
15    return {"_tiles": new_tiles}

```

**Programmcode 5.4:** Pseudocode der Main-Funktion des Skripts. Es wird je nach Input eine Menge von Tiles erstellt, die für die Ausgabe vorbereitet werden. Dabei werden Informationen gesammelt, die im Code genutzt werden, um ein korrekt funktionierendes Tileset zu garantieren. Wurden in der GUI bestimmte Checkboxen aktiviert, so werden dementsprechend Flag-, Prioritäts- und Proofreadingtiles generiert.

Der Hauptfunktion muss das gegebene oder generierte Tileset übergeben werden, sowie die angegebene Systemtemperatur und die Booleans für die Checkboxen im grafischen Interface. Daraufhin werden Informationen aus dem gegebenen Tileset entnommen. Dabei wird geprüft, ob das Tileset nördlich und südlich der Seed-Assembly entsprechende

Tiles besitzt. Des Weiteren erfolgt die Überprüfung, ob mögliche südliche oder nördliche Grenzen innere Kleber besitzen, über die das Wachstum im Proofreading für die inneren Tiles gestartet werden kann. Zusätzlich wird der linke Kleber der Seed-Assembly vermerkt, um sicherzustellen, dass er bei der Erstellung der Flag- und Prioritätstiles erhalten bleibt. Wenn helleblaue oder hellrote Tiles existieren, werden auch ihre linken Kleber gespeichert.

In dieser Arbeit wird immer wieder von der Generierung oder Erstellung von Tiles gesprochen. Dies geschieht immer in einer Liste und durch eine allgemeine Definition einer `generate_tile`-Funktion. Diese Funktion gibt ein einzelnes Tile zurück, das in korrekter Darstellung nach dem Programmcode 5.1 angegeben wird. Im Folgenden ist ein Ausschnitt aus dem Skript aus Anhang A dargestellt, der jedes Mal mit passenden Parametern verwendet wird, um ein Tile zu erstellen.



```

1 # t = tile
2 # l = label
3 # s = strength
4 # c = color
5 def generate_tile(tl, l1, s1, l2, s2, l3, s3, l4, s4, c):
6     return {
7         "label": tl,
8         "glues": [
9             {"label": l1, "strength": s1},
10            {"label": l2, "strength": s2},
11            {"label": l3, "strength": s3},
12            {"label": l4, "strength": s4},
13        ],
14         "color": c
15     }

```

**Programmcode 5.5:** Die zentrale Funktion des Skriptes, mit welcher ein einzelnes Tile generiert werden kann. In `label` wird der Tilebezeichner gespeichert, im `glues`-Array die nördlichen, östlichen, südlichen und westlichen Kleber mit Bezeichner und Stärke. Dabei liegen sie auch in dieser Reihenfolge vor. Zuletzt kann noch die Füllfarbe des Tiles angegeben werden.

## 5.8. Erstellung von Flag- und Prioritätstiles

Wenn alle Informationen gesammelt sind, dann werden Flags und Prioritätstiles erstellt. Dies passiert nur, wenn die jeweilige Checkbox ausgewählt wurde und die Eingabefelder nicht leer sind. Für alle erstellten und benannten Flags werden Tiles erstellt.

Die Menge an benötigten Tiles ist dabei abhängig von der Höhe der Self-Assembly. Das heißt, dass aus den gesammelten Informationen entnommen wird, ob ein Tile der Farbe *salmon* oder *skyblue* existiert. Diese müssen bei Existenz von Flag- oder Prioritätstiles am westlichen Kleber angepasst werden. Sie erhalten einen neuen Kleberbezeichner  $\sigma_t$ ,  $\sigma_c$

und  $\sigma b$  für `top`, `center` und `bottom` Kleber. Somit wird in der Self-Assembly verhindert, dass sich das Molekül bindet, ohne die Flag- und Prioritätstiles hinzuzufügen.

Auch muss beim Erstellen der Flagtiles beachtet werden, dass je nachdem, wie viele Flagtiles oder Prioritätstiles vorhanden sind, die westlichen Kleber des generierten Tiles verändert werden können. Dies ist in Abbildung 5.5 dargestellt. In a) wird nur ein Flagtile ohne weitere Flag- oder Prioritätstiles erstellt. Dadurch ist der westliche Kleber des Flagtiles so gewählt, dass sich das restliche Molekül hier binden kann. Dies ist analog für ein einzelnes Prioritätstile ohne Flag Tiles. In b) wird ein Flag- und ein Prioritätstile hinzugefügt. Dadurch braucht es einen zusätzlichen eindeutigen inneren Kleber zwischen Flag- und Prioritätstile. Der Vorgang ist wiederum analog für zwei Flagtiles. In c) ist der letzte Fall dargestellt, in welchem zwei Flagtiles und ein Prioritätstile hinzugefügt werden. Dabei werden mehrere innere Kleber benötigt. Es werden immer drei Kleberbezeichner aus der Menge  $\{A, B, C, \dots, AA, AB, AC, \dots, ZX, ZY, ZZ\}$  entnommen. Somit sind bis zu 234 Flag- oder Prioritätstiles nebeneinander möglich.

Wie angedeutet ist die Erstellung von Prioritätstiles analog zu den Flagtiles. Dabei muss nur darauf geachtet werden, ob und wie viele Flagtiles existieren, um so die korrekten östlichen Kleber erstellen zu können. Die westlichen Kleber sind immer die Kleber, die das restliche Molekül binden.

Sind Flag- und Prioritätstiles erstellt, wird zunächst das Tileset nach den Farben der Tiles sortiert. Dies wird zum einen aus Gründen der Übersicht vorgenommen, zum anderen, um sicherzustellen, dass die Seed-Assembly an erster Stelle steht. Denn so wird die Seed-Assembly in NetTAS festgelegt. Auch wird diese Sortierung bei der Anwendung von Snaked Proofreading verwendet. Wenn an dieser Stelle jedoch kein Proofreading vorgenommen werden soll, wird die Eingabemenge in die Liste `new_tiles` übernommen. In dieser Liste wird dann noch die Farbkodierung entfernt, wenn nicht die entsprechende Checkbox ausgewählt wurde, durch die der Farbcde erhalten bleibt. Die Liste kann wie folgt zurückgegeben werden:

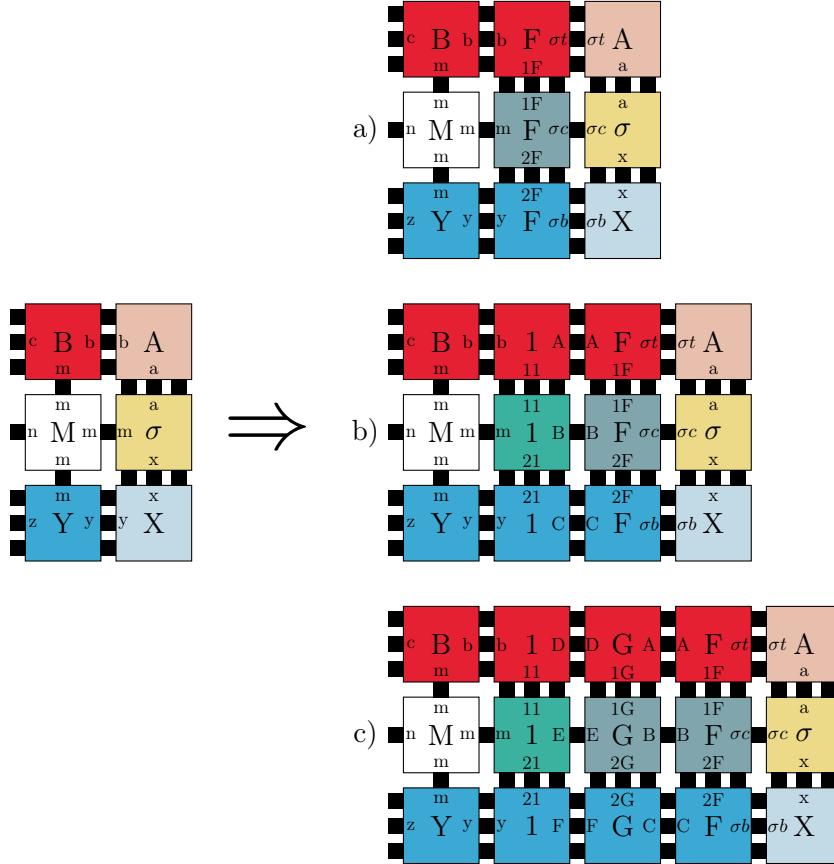
```
1   return {"_tiles": new_tiles}
```

**Programmcode 5.6:** Die Rückgabe der Main-Funktion, die so erstellt wird, dass sie ohne Probleme in eine JSON-Ausgabedatei geschrieben werden kann.

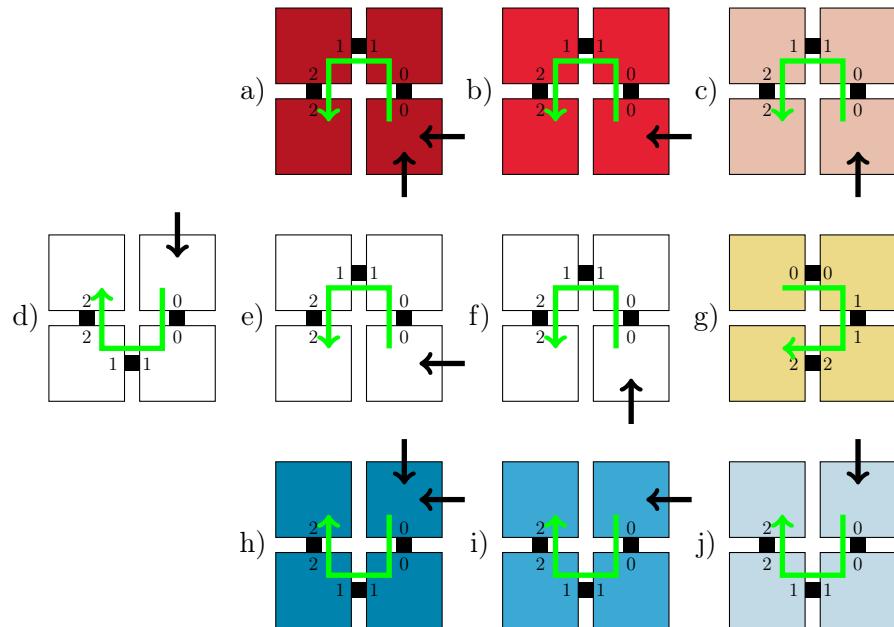
## 5.9. Proofreading im Skript

Wird jedoch Snaked-Proofreading auf das Tileset angewendet, dann wird die `new_tiles`-Liste neu erstellt. Jedes Tile aus der sortierten Liste von Tiles in `data[_tiles]` wird durch vier neue Tiles ersetzt. Dabei folgen die Tiles den Regeln des Snaked-Proofreadings aus dem Kapitel 2, dargestellt in Abbildung 2.10. Diese vier Tiles müssen zusammen wieder die gleiche Information und Position des originalen Tiles beinhalten.

Die so erstellten Tiles müssen dabei von einem der vier Tiles aus mit ihrem Wachstum



**Abbildung 5.5.:** Darstellung des inneren Kleberverhaltens für Flag- und Prioritätstiles. Sowohl a), b) und c) stammen original aus dem linken Molekül. In a) wird ein einzelnes Flag Tile im Molekül hinzugefügt. Dabei muss nur der Kleber an den Tiles  $\sigma$ , A und X geändert werden. Die Flag Tiles können dann östlich dieser Bezeichner und westlich die alten Bezeichner erhalten, an welchen B, M und Y gebunden werden. In b) ist das innere Kleberverhalten dargestellt. Die Kleberbezeichner A, B, C werden verwendet, um die Flag- und Prioritätstiles zu verbinden. Die restlichen Kleberbezeichner sind analog zu a) gesetzt. In c) wird gezeigt wie die inneren Kleber sich im Weiteren verhalten. Das Flagtile mit dem Bezeichner G hat so komplett eigene Kleberbezeichner. Die restlichen Bezeichner sind wieder analog zu b) und a). Auch ist anzumerken, dass keine Unterscheidung in der Konstruktionsweise von zwei Flagtiles zu einem Flagtile und einem Prioritätstile gemacht wird.



**Abbildung 5.6.:** Darstellung der verschiedenen Wachstumsrichtungen von Tiles mit Snaked-Proofreading. Die Richtung ist durch die grünen Pfeile dargestellt, während durch die schwarzen Pfeile der Kleber angegeben wird, über welchen die Tiles beim Start gebunden werden. In g) ist die Seed-Assembly abgebildet, die nicht durch irgendwelche Kleberverbindungen gestartet wird. a), b) und c) sind die Tiles der nördlichen Grenze. h), i) und j) sind die Tiles der südlichen Grenze. Die drei Möglichkeiten in d), e) und f) für die weißen zentralen Tiles hängen von der Wachstumsfront während der Self-Assembly ab. d) wächst von Norden, e) von Osten und f) von Süden.

starten. Je nach Position im Molekül kann dieses Wachstum mit einem anderen Tile starten. In Abbildung 5.6 ist dies für alle farbcodierten Tiles dargestellt.

Die Seed-Assembly wird als einziges Tile aus „Nichts“ gebildet und ist dementsprechend an keine Regel gebunden, wenn es um die Wachstumsrichtung geht. Im Skript wurde diese Wachstumsrichtung gewählt, da in NetTAS, dass erste angegebene Tile als neues Seedtile dient. Da dies das linksobere Tile im Proofreading ist, kann somit die in Abbildung 5.6 g) dargestellte Richtung verwendet werden.

Das hellrote Tile nördlich des Seedtiles muss das Wachstum mit dem Tile rechts unten starten. Der Grund dafür ist, dass sobald die beiden linken Tiles gebunden sind, die roten Tiles daneben gebunden werden können. Das soll aber erst passieren, wenn alle hellroten Tiles gebunden sind. Somit müssen zuerst die beiden rechten und danach die beiden linken Tiles gebunden werden. Die Kleber zwischen Seedtiles und den nördlichen Tiles müssen so angepasst werden, dass die rechten Tiles sich verbinden können. Gleichzeitig muss die Kleberstärke zwischen den linken Tiles so gewählt werden, dass sie sich erst dann verbinden, wenn alle drei anderen hellroten Tiles bereits gebunden sind. Die roten Tiles

bilden die nördliche Grenze des Moleküls. Für sie gilt dieselbe Wachstumsrichtung. Nur der Kleber, über welchen das erste Tile gebunden wird, ist in diesem Fall östlich, statt südlich. Die dunkelroten Tiles repräsentieren die nördlichen Liganden. Es wird davon ausgegangen, dass sie sich erst dann bilden, wenn das gesamte Molekül korrekt verbunden ist. Dementsprechend wächst dieses zwar genau wie das hellrote und rote Tile, jedoch benötigt das erste Tile sowohl den östlichen als auch den südlichen Kleber dafür. Wenn ein Kleber auf einer Seite fehlt, muss der jeweils andere Kleber ausreichend stark sein, um die korrekte Bindung sicherzustellen.

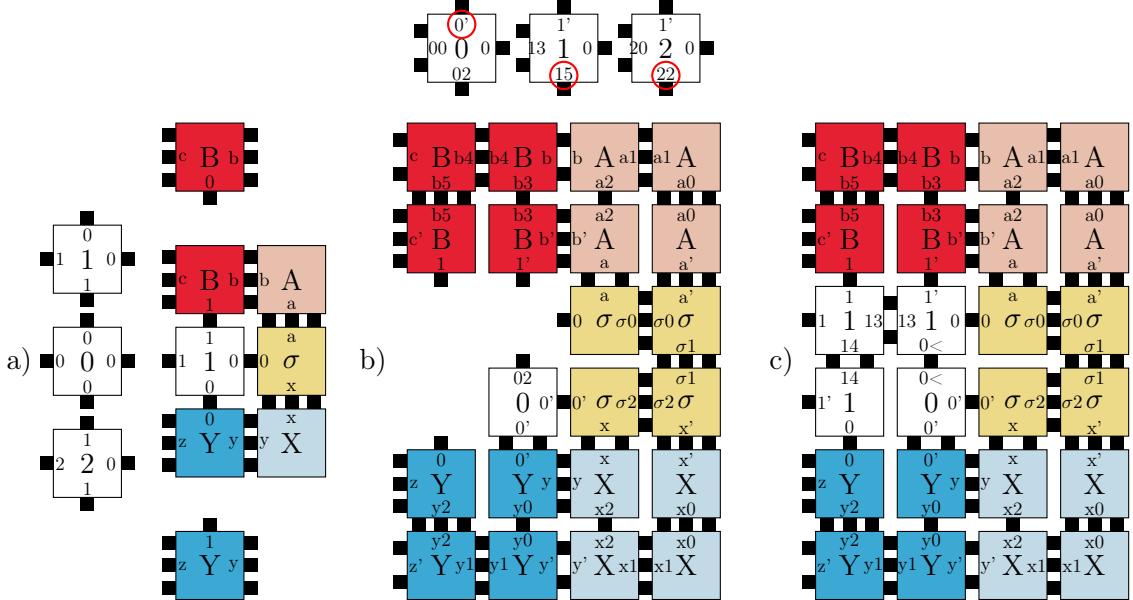
Für die hellblauen, blauen und dunkelblauen Tiles ist dieses Verhalten analog zu den roten, nur dass das Wachstum mit den oberen Tiles starten muss, da die Tiles nach Süden wachsen müssen. Diese Tiles werden immer genau gleich gebildet. Jedoch hat nicht jedes durch Self-Assembly gebildete Molekül die gleiche Höhe. Dementsprechend müssen die inneren Tiles (weiß) immer abhängig von den Informationen über das Tileset gebildet werden. Auch kann es bei Molekülen der Höhe drei zu dem Problem kommen, dass sowohl der nördliche als auch der südliche Kleber benötigt wird, um ein Tile zu binden. Da beim Snaked-Proofreading jedoch zwei Tiles gebunden werden müssen, bevor südliche und nördliche Kleber verbunden werden, könnte Information verloren gehen.

Das beschriebene Problem ist in Abbildung 5.7 aufgezeichnet. Dabei ist in a) eine Self-Assembly abgebildet, die bei Snaked-Proofreading ein Problem verursacht. Die inneren und weißen Tiles sind abhängig vom südlichen und nördlichen Tile. Je nach nördlichem Kleber des Tiles Y und südlichem Kleber des Tiles B bindet sich ein anderes zentrales Tile.

In b) kann das Problem erkannt werden, das entsteht, wenn auf solchen Tilesets Snaked-Proofreading angewendet wird. Das Wachstum der vier zentralen Tiles startet unten rechts. Damit sich dieses Tile bindet, muss der Kleber verstärkt werden. Dadurch kann sich jedoch ein falsches Tile binden, da noch keine Information des nördlichen Klebers verwendet wird. In b) bindet sich das Tile 0. Damit gibt es im Tileset kein Tile mehr, das sich oben rechts binden kann. Die einzigen potenziellen Kandidaten sind über dem Molekül dargestellt. Die roten Markierungen weisen dabei darauf hin, weshalb auch diese Tiles nicht gebunden werden können.

Um das Problem zu lösen, muss wie in c) dargestellt, die Information des südlichen Klebers weitergegeben werden. Durch die Konkatenation mit dem Zeichen „<“ wird sichergestellt, dass keine falsche Bindung entstehen kann. Die Information des südlichen Klebers wird dabei im inneren Kleber des Proofreadings mitgegeben. Somit kann sich das „falsche“ Tile unten rechts bilden, ohne ein Problem zu verursachen. Das korrekte Tile bindet sich immer oben rechts.

Wie in Abbildung 5.6 dargestellt wird, gibt es je nach Existenz der jeweiligen Tiles drei mögliche Wachstumsrichtungen. Gibt es blaue Tiles, die alle nördliche Kleber besitzen, so wachsen die inneren Tiles vom südlichen Kleber des rechten unteren Tiles aus. Das ist in Abbildung 5.6 f) dargestellt. Gibt es keine blauen Tiles im Tileset oder haben nicht alle blauen Tiles nördliche Kleber, so wird überprüft, ob es rote Tiles gibt. Falls



**Abbildung 5.7.:** Darstellung eines Problems von Snaked-Proofreading bei Molekülen der Höhe drei. In a) ist ein solches Molekül dargestellt. Abhängig vom südlichen Kleber von Tile B und dem nördlichen Kleber von Tile Y existieren die vier unterschiedlichen Tiles: 0, 1, 1 und 2. Das Tile mit dem Bezeichner 1 gibt es dabei zweimal, da einmal die 1 von oben und einmal von unten geliefert wird. In b) kann das Problem bei dieser Self-Assembly erkannt werden. Da das zentrale Tile an der unteren rechten Seite das Wachstum starten muss, ist dort der Kleber stärker. So kann sich das Tile 0 binden und bei Temperatur drei halten. Jedoch müsste sich zur korrekten Bindung das Tile 1 binden. Es kann sich jedoch kein weiteres Tile binden, wie durch die rote Markierung darüber dargestellt wurde. Die Lösung für das Problem wird in c) dargestellt. Der Kleberbezeichner im Süden der Tiles (0 bei Tiles 1 und 0 und 1 bei Tiles 1 und 2) muss nach oben weitergegeben werden. Dafür wird der Bezeichner mit < konkateniert. Damit bindet sich rechtsoben das richtige Tile. Auch wenn unten 0 gebunden wird, kann durch die Weitergabe der 0 an den nördlichen Kleber, das korrekte Tile darüber gebunden werden.

alle diese Tiles einen südlichen Kleber besitzen, beginnt das Wachstum der inneren Tiles vom nördlichen Kleber aus, wobei mit dem rechtsoberen Tile gestartet wird. Das wird in Abbildung 5.6 d) gezeigt. Gilt keiner der beiden genannten Fälle, so müssen die inneren Tiles von der Seed-Assembly aus gebildet werden. Dabei wachsen die vier Tiles vom linken unteren Tile und dessen östlichem Kleber aus. Dies ist in Abbildung 5.6 e) zu sehen.

Damit sind alle Regeln für das Bilden der Tiles ausgelegt. So können die vier Tiles durch die entsprechenden Abfragen generiert werden. Für jedes Tile werden diese in der `new_tiles`-Liste gesammelt. Sind alle Schritte und Funktionen aus Code 5.4 durchlaufen, so wird der Inhalt in eine JSON-Datei geschrieben, die in NetTAS geladen und simuliert werden kann.

In diesem Kapitel wurde die Konstruktion des Python-Skripts vorgestellt, mit welchem einige der Mechanismen aus dem vorherigen Kapitel in Tilesets implementiert werden. Auch wurde näher beschrieben, wie diese Mechanismen sinnvoll in Tilesets eingebaut werden können. Mit dem Skript und der Vorstellung dieser Mechanismen können im Weiteren Simulationen durchgeführt und die Ergebnisse evaluiert werden.



# 6. Simulationen

In diesem Kapitel steht nach der Darstellung von Konzept und Struktur der Anforderungen die Evaluation der Simulationsergebnisse an. Einige dieser Ergebnisse werden mithilfe von Boxplots präsentiert, da diese eine umfassende Darstellung der Messwerte ermöglichen: Sie zeigen das Minimum und Maximum sowie den Median und die Quartile. Dadurch wird die gesamte Messreihe kompakt dargestellt. Die vorgestellte Simulationsumgebung NetTAS dient zur Generierung der Messergebnisse. Zunächst wird die Gewichtung bei der Generierung von Tilesets untersucht. Danach werden für acht durch das Skript generierte und in NetTAS erstellte Tilesets Simulation und Evaluation von Acknowledgements, Prioritätseveln, Flags, Prüfsummen und Snaked-Proofreading durchgeführt.

## 6.1. Simulation und Evaluation der Gewichtungen zur Generierung von Tilesets

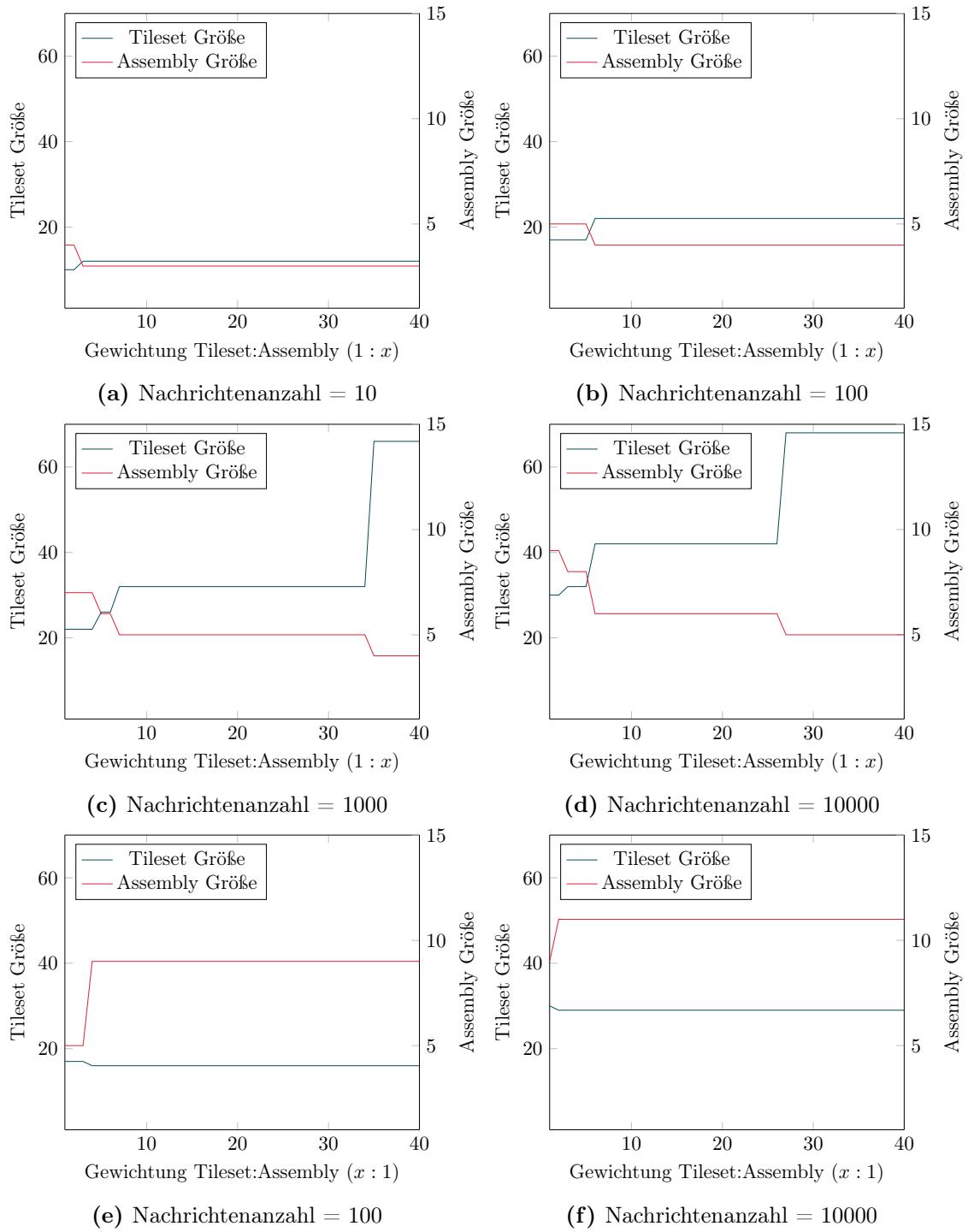
Zur Evaluation und Analyse der im vorherigen Kapitel vorgestellten und implementierten Anforderungen werden Tilesets zur Simulation und Evaluation benötigt. Vier Tilesets wurden in NetTAS erstellt. Diese Tilesets werden später in diesem Kapitel noch genauer beschrieben. Vier weitere Tilesets wurden mit dem Skript generiert. Bei der Beschreibung des Vorgangs im vorherigen Kapitel wurde jedoch eine sinnvolle Gewichtung zwischen Tileset- und Assemblygröße offengelassen.

So muss zunächst die Generierung von Tilesets analysiert werden. Da die Tilegenerierung von drei Faktoren abhängt, müssen diese zuerst betrachtet und analysiert werden. Die Anzahl an Nachrichten in einem System ist zwar eine anwendungsabhängige Eingabe, jedoch muss für unterschiedlich große Nachrichtensätze gute Gewichtungen von Tileset und Assembly gefunden werden. Dabei ist erneut anzumerken, dass eine „gute Gewichtung“ anwendungsabhängig sein kann.

Eine gute Gewichtung wird hier durch die Simulationsumgebung NetTAS gefunden, indem das Tileset und die Self-Assembly in **kTAM** simuliert wird. In **kTAM** wird durch die Bond Breaking Cost und Binding Cost definiert, wie sich ein zufällig an einer Stelle der Self-Assembly gebundenes Tile bindet oder löst. Dies ist in der Simulation abhängig von der Größe der Self-Assembly, da so mehr mögliche Wachstumsfronten entstehen. Auch die Größe des Tilesets beeinflusst die Simulation, da häufiger „falsche“ Bindungen getestet werden.

## 6. Simulationen

---



**Abbildung 6.1.:** Graphen zur Relation von Gewichtungen und Anzahl von zu codierenden Nachrichten. Die Anzahl der Nachrichten ist unter den Graphen abgebildet. Die Graphen (a) bis (d) stellen die Tileset:Assembly Gewichtung mit wachsender Tileset-Gewichtung dar, (e) und (f) mit wachsender Assembly-Gewichtung. In den Graphen ist die Tilesetgröße an der linken y-Achse und Assemblygröße an der rechten y-Achse dargestellt.

Dieser Faktor ist eine der größten Schwächen von kTAM, da in einer realitätsnäheren Betrachtung nicht immer nur ein zufälliges Tile an einer zufälligen Stelle versucht eine Bindung aufzubauen. In der Realität läuft dies parallel an allen Stellen mit mehreren Tiles ab. Doch für die Betrachtung und Analyse der Gewichtungen von Tileset und Assembly hilft es dabei eine Argumentationsbasis aufzubauen. Braucht kTAM bedeutend länger für die Bildung einer Self-Assembly, dann kann davon ausgegangen werden, dass die Gewichtung falsch gewählt wurde, weil entweder die Assembly oder das Tileset in Relation zum jeweils anderen zu groß gewählt wurde.

Mit dieser Information können Tilesets mit unterschiedlichen Gewichtungen erstellt werden. Dafür werden verschiedene Nachrichtenzahlen verwendet und durch unterschiedliche Gewichtungen auf die Größe von Tileset und Assembly getestet. Dies ist in Abbildung 6.1 abgebildet. In Abbildung 6.1 wird die Gewichtung zwischen Tileset und Assembly dargestellt. Für das Tileset-Gewicht wird ein Spektrum bis zu einer Gewichtung von 100:1 im Vergleich zur Assembly untersucht. Umgekehrt wird für das Assembly-Gewicht eine Betrachtung bis zu einer Gewichtung von 1:100 vorgenommen. Gewichtungen darüber werden nicht betrachtet. Die Temperatur des Systems wird auf zwei festgelegt.

Aus der Abbildung 6.1 lässt sich entnehmen, welche Gewichtungen für eine gegebene Nachrichtenmenge angemessen sind. Bei einer kleinen Nachrichtenmenge machen Verhältnisse wie 1 : 4 und 1 : 40 keinen Unterschied, daher genügt die Simulation eines Verhältnisses. Aus den Abbildungen zeigt sich, dass für Nachrichtenzahlen unter 1000 relativ ähnliche Verhältnisse getestet werden können. Bei höheren Nachrichtenzahlen sollte ein größeres Verhältnis berücksichtigt werden. Gemäß den Gewichtungen aus Abbildung 6.1 e) und f) erhält das Tileset ein größeres Gewicht als die Assembly. Sowohl bei kleinen als auch großen Nachrichtenmengen bleibt die Größe des Tilesets und der Assembly für bereits geringe Verhältnisse konstant. Alle sechs Graphen bilden das Verhältnis zwar nur bis zur Gewichtung von 40:1, beziehungsweise 1:40 ab, jedoch wurden alle Verhältnisse bis zu der Gewichtung 1:100 und 100:1 getestet. In keinem der Fälle gab es zwischen 40 und 100 weitere Veränderungen in Tileset- und Assemblygröße. So werden nun folgende Simulationen durchgeführt, um eine sinnvolle Gewichtung von Tileset- und Assemblygröße für den Rest der Evaluation zu finden. Dafür werden die Nachrichtenmengen und Gewichtungen wie folgt gewählt:

**Nachrichtenmenge** 10: 1:1, 1:3, 4:1

**Nachrichtenmenge** 100: 1:1, 1:5, 1:6, 4:1

**Nachrichtenmenge** 1.000: 1:1, 1:5, 1:7, 1:35, 2:1

**Nachrichtenmenge** 10.000: 1:3, 1:6, 1:27, 1:1, 2:1

Für die Messungen in Tabelle 6.1 wurden 20 Messungen in kTAM durchgeführt. Die generierten Daten wurden mit Temperatur zwei erstellt und die dafür gewählten Parameter in kTAM sind wie folgt gewählt:

**Sleep Time:** 10

Verhältnis	Nachrichtenmenge			
	10	100	1.000	10.000
4:1	122	1.425		
2:1		735	3.153	
1:1	<u>106</u>	335	1249	
1:3	109			1.316
1:5		353	1.038	
1:6		<u>250</u>		1.214
1:7			<u>531</u>	
1:27				<u>1.021</u>
1:35			721	

**Tabelle 6.1.:** Darstellung des Medians der Messergebnisse in kTAM für verschiedene Verhältnisse (Tileset:Assembly), die aus Abbildung 6.1 entnommen wurden. Zur Berechnung der dargestellten Werte wurden 20 Simulationsdurchläufe in kTAM durchgeführt. Ange-sichts signifikanter Ausreißer wurde der Median für alle 20 Durchläufe bestimmt und dann die 15 % der Messungen mit dem größten Abstand zum Median ausgeschlossen. Nachdem so die drei größten Ausreißer entfernt wurden, wurde der Median der verbleibenden 17 Messungen genommen. Diese Ergebnisse sind hier abgebildet. Für jede Nachrichtenmenge wurde die niedrigste Messung unterstrichen.

**Forward Rate:** 10

**Binding Cost:** 16

**Bond Breaking Cost:** 11

Die Werte für *Binding Cost* und *Bond Breaking Cost* sind passend zur Temperatur zwei gewählt. Um größere Moleküle in realistischer Zeit simulieren zu können, müssen die Kosten für die Bindung von Tiles in einem System der Temperatur zwei unter 20 gewählt werden. Jede niedriger der Wert, desto wahrscheinlicher bindet und hält sich jedoch eine Verbindung mit Stärke eins. Dementsprechend darf der Wert nicht zu niedrig sein, da sonst fehlerhafte Verbindungen entstehen können. Bei Kosten zum Brechen von Verbindungen von über elf wird gewährleistet, dass Verbindungen der Stärke eins schnell wieder getrennt werden. Gleichzeitig brechen Verbindungen der Stärke zwei nur selten, wodurch auch größere Moleküle stabil gebunden werden können. Auch folgt aus niedrigerer *Binding Cost* und höherer *Bond Breaking Cost* eine geringe Zahl von Ausreißern im kTAM. Somit kann ein leicht angepasster Median auf die Messungen angewandt werden. Bei 20 Messungen kann ein Cutoff von 15 % für den Median gewählt werden. So wird zunächst der Median der gesamten Messreihe berechnet. Für den Cutoff werden die drei Messwerte aus der Reihe entfernt, die den größten Abstand zum Median haben. Aus den restlichen 17 Messwerten wird erneut der Median berechnet. Aus den Messungen in Tabelle 6.1 kann gelesen werden, dass sich das Verhältnis in Relation zur Nachrichtenmenge verhält. Bei einer großen Nachrichtenmenge (hier: 10000) sollte ein Verhältnis gewählt

Name	H-1-mini	H-1-klein	H-1-norm	H-1-groß
Tilset Größe	10	22	32	68
Assembly Höhe	1	1	1	1
Assembly Länge	4	4	5	5
Nachrichtenanzahl	10	100	1000	10000
Tileset:Assembly Gewichtung	1:1	1:6	1:7	1:27

**Tabelle 6.2.:** Darstellung der in diesem Kapitel zur Simulation verwendeten Tilesets mit einigen Parametern. Die Tilesets in dieser Tabelle sind alle durch das vorgestellte Skript generiert worden und haben dementsprechend in der Assembly eine Höhe von eins. Die nicht generierten Tilesets sind in Tabelle 6.3 dargestellt.

Name	H-2-klein	H-2-norm	H-3-klein	H-3-norm
Tilset Größe	6	22	9	26
Assembly Höhe	2	2	3	3
Assembly Länge	3	5	3	5

**Tabelle 6.3.:** Darstellung der in diesem Kapitel zur Simulation verwendeten Tilesets mit einigen Parametern. Die Tilesets in dieser Tabelle wurden alle in NetTAS erstellt. Die generierten Tilesets sind in Tabelle 6.2 dargestellt.

werden, das die Größe der Assembly stärker gewichtet. Bei kleinen Nachrichtenmengen kann dieses Verhältnis kleiner gewählt werden. Jedoch sollte trotzdem die Assembly ein klar größeres Gewicht haben. Nur bei kleinen Nachrichtenmengen kann die Gewichtung gleichmäßig gewählt werden. Dem Tileset ein höheres Gewicht zuzuordnen, ist für keine der gegebenen Nachrichtenmengen ratsam. Ein solches Vorgehen würde in der Simulation zu einer erhöhten Anzahl an Schritten führen.

## 6.2. Die Tilesets

Mit diesen ersten Simulationsergebnissen können im Folgenden Datensätze generiert werden, um die Mechanismen aus Kapitel 4 simulieren und evaluieren zu können. Um alle Anforderungen vergleichen zu können, werden die acht Tilesets erstellt, die in Abbildung 6.2 dargestellt werden. Ihre Parameter und Eigenschaften sind in Tabelle 6.2 und Tabelle 6.3 abgebildet.

Die Tilesets wurden aus folgenden Gründen ausgewählt: Im Sinne des im vorherigen Kapitel vorgestellten Skriptes werden nur Moleküle erstellt und getestet, die durch das Skript entweder generiert oder erweitert werden können. Somit werden Tilesets für As-

## 6. Simulationen

---

Größe	1-mini	1-klein	1-norm	1-groß	2-klein	2-norm	3-klein	3-norm
1	10	22	32	68	6	22	9	26
2	24	120	220	1012	3	25	5	26
3	32	200	1200	11616	1	39	3	29
4	16	100	2000	21296	1	39	2	36
5	-	-	1000	10648	1	36	1	68
6	-	-	-	-	1	36	1	128
7	-	-	-	-	1	27	1	200
8	-	-	-	-	1	27	2	288
9	-	-	-	-	-	27	1	400
10	-	-	-	-	-	27	-	672
11	-	-	-	-	-	-	-	960
12	-	-	-	-	-	-	-	1088
13	-	-	-	-	-	-	-	512
14	-	-	-	-	-	-	-	512
15	-	-	-	-	-	-	-	192
16	-	-	-	-	-	-	-	-

**Tabelle 6.4.:** 2HAM Ergebnisse für alle acht erstellten Tilesets. Das führende „H-“ in den Namen der Tilesets wurde in dieser Tabelle aus Platzgründen weggelassen. Aus diesen Ergebnissen lässt sich ablesen, ob eine abgeschlossene Assembly erreicht wird und dass das Molekül nicht unendlich durch einen Fehler wächst. Auch kann anhand des Wertes der größten Assembly (beispielsweise 1000 für Assemblies der Größe fünf für H-1-norm) festgestellt werden, ob die richtige Anzahl an möglichen Molekülen gebildet werden.

semblies der Höhe ein, zwei und drei erstellt. Für die Höhe zwei und drei wird jeweils ein Tileset mit minimalen Tiles erstellt und ein Tileset mit einigen Variationsmöglichkeiten in der finalen Assembly. Für die Moleküle der Höhe eins werden vier verschiedene Größen in Hinsicht auf die Nachrichtenzahl generiert und für weitere Simulationen verwendet.

Die Eigenschaften der Tilesets können in den Simulationsergebnissen von 2HAM dargestellt werden. Diese Ergebnisse sind in Tabelle 6.4 dargestellt. Aus der Tabelle lassen sich die maximalen Assemblygrößen für alle Tilesets ablesen. Auch kann bei der jeweils maximalen Größe die Anzahl an verschiedenen Assemblies erkannt werden, die durch das Tileset gebildet werden können.

So kann abgelesen werden, dass H-2-klein nur ein eindeutiges finales Molekül mit sechs Tiles in der Assembly bilden kann, während H-2-norm 27 Assemblies der Größe zehn bilden kann. Da die Tabelle hier für noch eher kleine Tilesets bereits sehr groß wird, werden die Ergebnisse für 2HAM für keine weiteren Tilesets dargestellt. Die 2HAM-Simulationen für die Tilesets H-1-groß und H-3-norm erwiesen sich außerdem bereits in ihrer Grundform als besonders zeitaufwendig.

Dementsprechend werden im Folgenden alle getesteten Implementierungen nur auf den

Tilesets **H-1-mini**, **H-2-klein** und **H-3-klein** in 2HAM getestet, um evaluieren zu können, ob die Tilesets und Assemblies korrekt und fehlerfrei erweitert werden. Die Ergebnisse werden wie beschrieben nicht tabellarisch dargestellt. Wenn in den jeweiligen Absätzen nichts Weiteres beschrieben wird, kann davon ausgegangen werden, dass die 2HAM-Simulationen wie beschrieben durchgeführt wurden. Nur wenn unerwartete Ergebnisse oder Probleme in der 2HAM Simulation entstanden sind, werden diese Simulationen erwähnt.

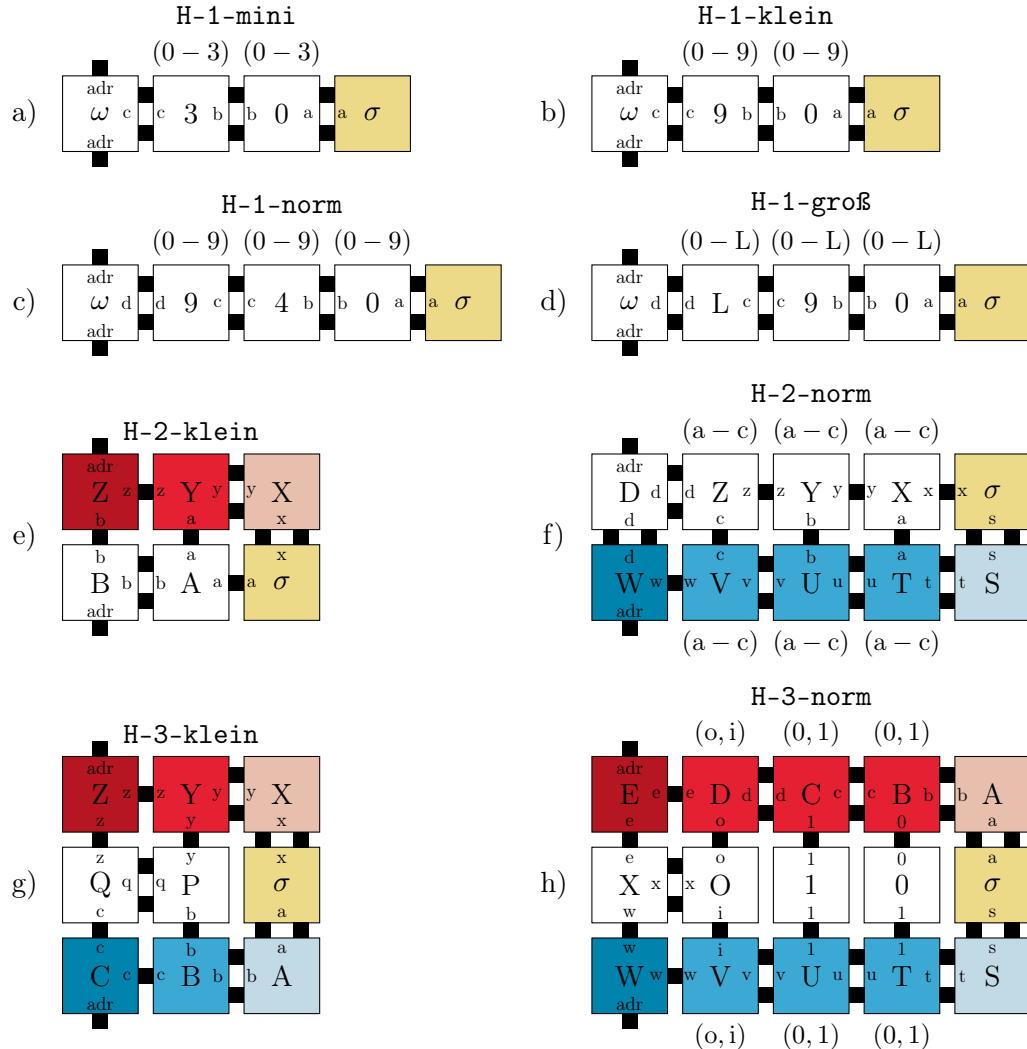
Bei der 2HAM-Simulation von **H-3-norm** trat ein Problem mit dem Tileset auf. Für eine Assemblygröße von 15 wurden mehr Assemblies gefunden, als durch die Konstruktion des Tilesets erwartet wurde. Das Tileset sollte 64 verschiedene Self-Assemblies der Größe 15 darstellen. Doch in Tabelle 6.4 zeigt sich, dass 192 solcher Self-Assemblies in 2HAM gefunden wurden. Dabei handelt es sich um einen Facet-Error, der in der Sektion 6.8.1 bei der Evaluation von Snaked-Proofreading näher betrachtet wird.

Des Weiteren ist vor der Evaluation der Mechanismen anzumerken, dass das Assembly-Modell kTHAM und die Simulation dieses Modells in NetTAS nur durchgeführt wurde. Das Modell ist zwar für dieses Kapitel interessant, allerdings stand zum Zeitpunkt der Arbeit nicht die nötige Rechenleistung zur Verfügung, um kTHAM für alle betrachteten Tilesets durchzuführen zu können. Während die Tilesets **H-1-mini**, **H-2-klein** und **H-3-klein** in kTHAM simuliert wurden, erwiesen sich schon die Simulationen der unmodifizierten Tilesets als sehr rechenaufwendig. Bei allen anderen getesteten Tilesets dauerte die Berechnung einzelner Iterationen mehrere Minuten bis Stunden. Bei der Durchführung dieser Simulationen kam es zum Einfrieren der Anwendung. Es war möglich, einige kleinere Simulationen abzuschließen, jedoch sind einzelne Iterationen nicht unbedingt repräsentativ. Für eine aussagekräftige Datenbasis wären 20 bis 30 Messungen besser. Da die Durchführung so vieler Messungen für einzelne Tilesets aus Zeitgründen unpraktikabel erschien, wurde entschieden, keine Simulationen in kTHAM durchzuführen. Obwohl kTAM in Bezug auf Realitätsnähe kTHAM unterlegen ist, zeigt es sich bei der Simulationszeit großer Assemblies und Tilesets als überlegen, weshalb es als zentrales Simulationsmodell gewählt wurde.

Alle erstellten Tilesets werden mit einer Temperatur von zwei definiert. Eine Variation der Temperaturen in den Simulationen ist aus folgendem Grund nicht sinnvoll: Die Temperatur spielt in allen Assembly-Modellen eine zentrale Rolle, und in NetTAS ist dies nicht anders. Wenn die Simulation mit einer anderen als der vorgesehenen Temperatur durchgeführt wird, resultiert dies in fehlerhaften Bindungen der Tiles.

Zwei Self-Assemblies können trotz unterschiedlicher Temperatur identisch strukturiert sein, wobei nur die Kleber entsprechend angepasst werden müssen. In kTAM wird die Temperatur jedoch nicht direkt angegeben, sondern durch *Binding Cost* und *Bond Breaking Cost* repräsentiert. Das führt dazu, dass für unterschiedliche Tilesets unterschiedliche Parameter in kTAM angegeben werden müssen, wenn sich die Temperaturen unterscheiden.

Dadurch entsteht eine potenzielle Fehlerquelle, die durch eine einheitliche Temperatur über alle Tilesets verhindert wird. Dementsprechend wird in der Simulation und Analyse



**Abbildung 6.2.:** Grafische Darstellung der Assemblies, die für die Simulationen ausgewählt wurden. In a), b), c) und d) sind die Assemblies der Tilesets abgebildet, die durch das Skript generiert wurden. Über den Assemblies sind die Namen sowie die möglichen Wertebereiche der darunterliegenden Tiles angegeben. Am Beispiel von a) gibt es im Tileset jeweils vier Tiles mit den Bezeichnern 0,1,2 und 3. Damit können 16 Werte abgebildet werden, unter denen sich die zehn benötigten Nachrichten befinden. Neben den vier generierten Tilesets wurden zwei Beispiele für die Molekülhöhe zwei erstellt: ein Minimalbeispiel in e) und ein Beispiel mit Variation in f). Das Gleiche für Moleküle der Höhe drei in g) und h). In h) umfasst das Tileset zusätzlich zu den unten und oben angegebenen Variationen noch entsprechende Tiles für die Mitte. Diese wurden aus Platzgründen nicht angegeben, umfassen aber fünf weitere Tiles für alle Fälle der inneren Kleber.

der Ergebnisse immer eine Temperatur von zwei verwendet, um alle Messungen miteinander vergleichen zu können und mögliche Fehlerquellen auszuschließen.

### 6.3. Evaluation der Adressierung in DNA-Tile-basierten Tile-Assemblies

Die Simulation von Adressierung ist in NetTAS nicht möglich. Dies liegt daran, dass die Adressierung in dieser Implementierung im Kleberbezeichner definiert ist und die konkreten Basenpaare im offenen DNA-Strang des Tiles nicht im mathematischen Modell berücksichtigt werden. Eine Möglichkeit zur Evaluierung von mehreren Adressen besteht darin, mehrere Liganden zu erstellen. Dabei muss beachtet werden, dass dies eher die Varianz im Tileset simuliert als die eigentliche Adressierung. Für eine umfassendere Simulation der Adressen wäre ein Modell interessant, das den Einfluss von offenen DNA-Strängen unterschiedlicher Länge und Codierung testet.

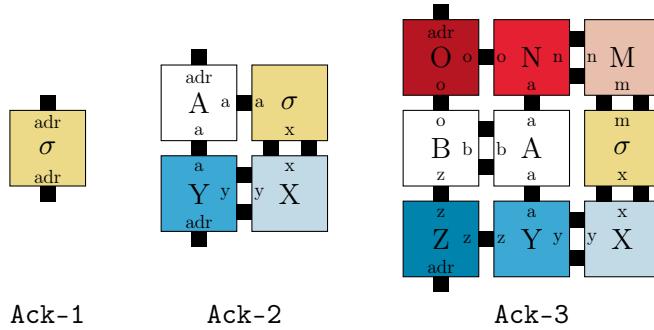
Zur Evaluation von Adressierung kann hier also festgestellt werden, dass die Implementierung trivial ist, da sie in der Bildung von Liganden inhärent ist. Auch kann eine große Menge von verschiedenen Adressen dazu führen, dass viele Liganden für das gleiche Tileset benötigt werden. Dadurch wären Mechanismen notwendig, die die richtigen Liganden abhängig von der Empfängeradresse im Medium freilassen. Dieser Vorgang kann jedoch nicht in NetTAS simuliert werden. Deshalb werden im Folgenden die Mechanismen betrachtet, welche durch Simulationsergebnisse in NetTAS evaluiert werden können.

### 6.4. Simulation und Evaluation von Acknowledgements

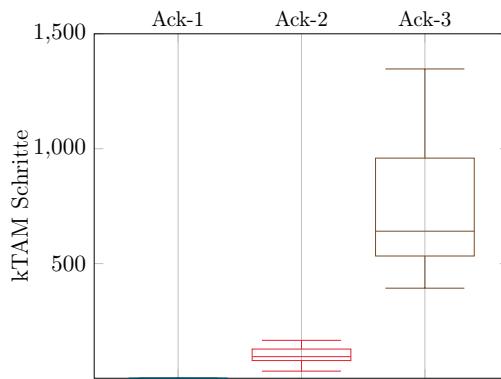
In dieser Sektion wird ein Mechanismus der Datenflusskontrolle evaluiert. Dafür wird zunächst betrachtet, wie aufwendig verschiedene Acknowledgements in kTAM sind.

Simuliert werden drei unterschiedliche Möglichkeiten für Acknowledgements. Diese sind in Abbildung 6.3 dargestellt. Im Folgenden werden sie auch mit den in der Abbildung gegebenen Namen angesprochen. Bei der Simulation der **Ack-1** Assembly kann zu allen Simulationsmodellen in NetTAS gesagt werden, dass die Simulationen trivial sind. Da das Tileset und die Assembly nur aus der Seed-Assembly besteht, kann auch keine Self-Assembly im herkömmlichen Sinne simuliert werden. Das Problem mit diesem Tile ist auch, so wie bei allen Assemblies der Höhe eins, dass sich das Tile als Ligand an Rezeptoren binden kann, ohne den Prozess der Self-Assembly durchlaufen zu haben. Wenn das kein Problem im System darstellt, so ist **Ack-1** die effizienteste Möglichkeit zur Implementierung von Acknowledgements.

In dieser Sektion werden jedoch noch zwei weitere Möglichkeiten simuliert und analysiert. Beide Tilesets benötigen für die korrekte Bindung der Tiles die Temperatur zwei. Obwohl die **Ack-2** Self-Assembly mit vier Tiles komplexer ist als **Ack-1**, bleibt ihr Aufbau dennoch



**Abbildung 6.3.:** Darstellung der drei simulierten und analysierten Acknowledgement Self-Assemblies mit den Bezeichnern unter den Self-Assemblies, mit welchen sie in dieser Arbeit angesprochen werden.



**Abbildung 6.4.:** Grafische Darstellung der Simulationsergebnisse für Acknowledgement Assemblies. Da Ack-1 nur aus einem Tile besteht, ist der Boxplot nur angedeutet dargestellt. Alle Durchläufe werden theoretisch bei null Schritten beendet, da keine Verbindung benötigt wird. Es kann jedoch ein klarer Unterschied zwischen Ack-2 und Ack-3 erkannt werden.

relativ einfach. Ein besonderes Merkmal dieses Moleküls ist, dass durch eine fehlerhafte Bindung des A- und Y-Tiles (siehe Abbildung 6.3) eine Bindung an den Rezeptoren eines Empfängergeräts möglich ist, ohne dass die Seed-Assembly gebunden werden muss.

Dementsprechend gibt es mit Ack-3 einen Vorschlag für eine Self-Assembly, bei welcher sich die Liganden nur durch zwei gleichzeitig fehlerhafte Bindungen an Rezeptoren binden können. Sonst muss das gesamte Molekül gebildet werden, damit es sich binden kann. Eine andere Möglichkeit wäre Snaked-Proofreading auf den Acknowledgements anzuwenden, um dieses Problem zu erschweren. Dazu wird jedoch in späteren Sektionen mehr beschrieben.

Die Simulation der Acknowledgement Assemblies zeigt, dass Ack-3 im Median fast siebenmal so viele Schritte in kTAM benötigt wie Ack-2. Dies ist in Abbildung 6.4 dargestellt.

Auch Minima, Quartile und Maxima sind eindeutig kleiner in der **Ack-2** Simulation. Die Gründe für **Ack-3** statt **Ack-2** wurden beschrieben. Hat die mögliche Fehlbildung in **Ack-2** eine geringere Priorität, bietet es sich wegen der Komplexität an, dieses Molekül oder sogar **Ack-1** zu verwenden. Ist die korrekte Bindung jedoch von hoher Relevanz, so kann zu **Ack-3** gesagt werden, dass ein Median von 641 Schritten in kTAM eine immer noch sehr schnelle Simulationszeit darstellt. Eine weitere Möglichkeit ist das **Ack-1**, **Ack-2** oder **Ack-3** Molekül mit Snaked-Proofreading zu erweitern, um fehlerhafte Bindungen weiter zu erschweren. Das wird in der Sektion 6.8.3 später in diesem Kapitel aufgegriffen und betrachtet.

## 6.5. Simulation und Evaluation von Prioritätsleveln

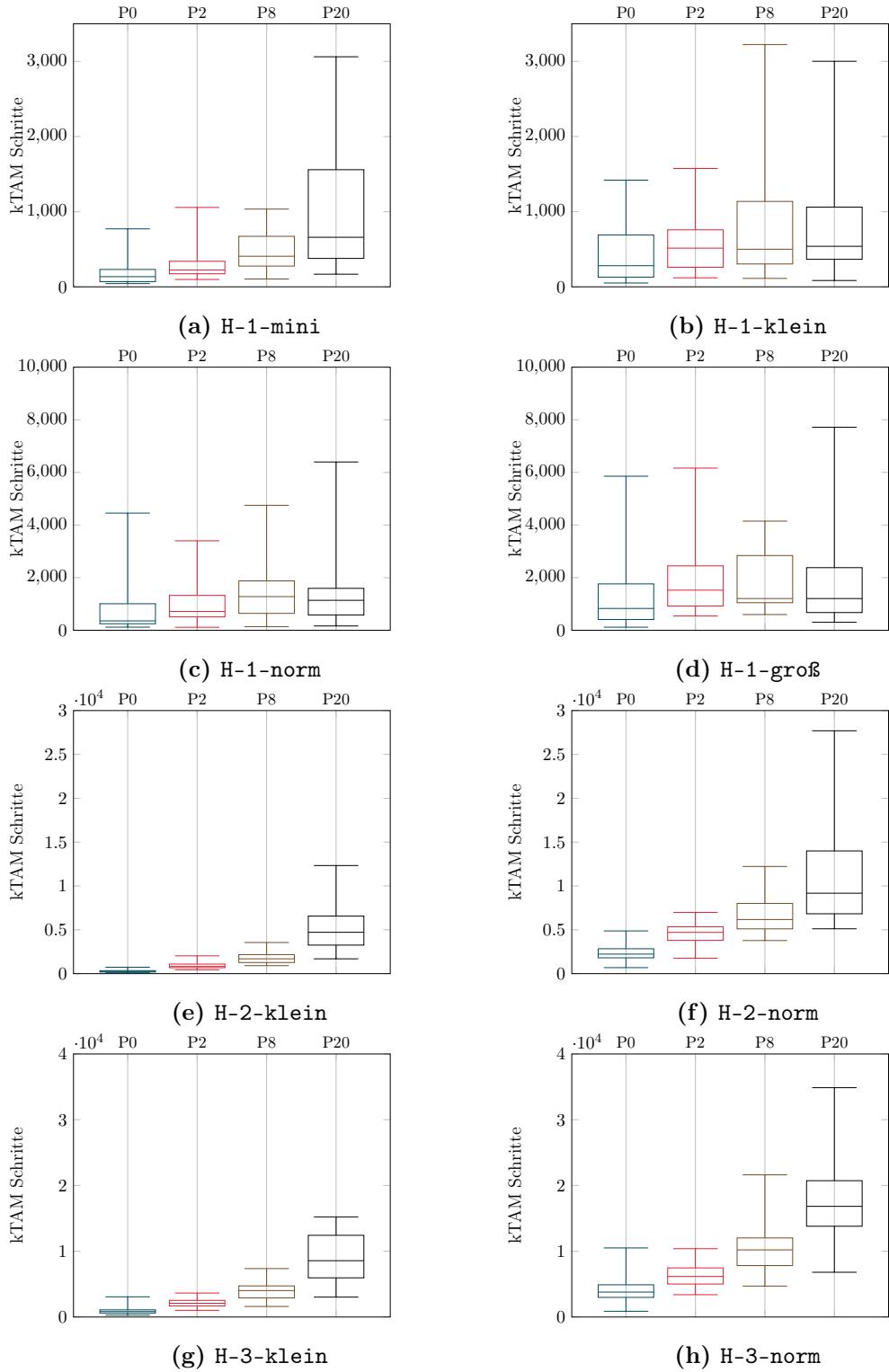
Zu Datenflusskontrolle und speziell zur Gerechtigkeit im System gehören die Prioritätslevel. Auch diese können mit der Laufzeit der Self-Assembly in verschiedenen Tilesets simuliert werden. Die Ergebnisse sind in Abbildung 6.5 angegeben. Jedes in Tabelle 6.2 und Tabelle 6.3 dargestellte Tileset wird dabei mit drei unterschiedlichen Prioritätsleveln erweitert und verglichen. Sie wurde wie folgt gewählt: ein Minimalbeispiel mit zwei Leveln, ein größeres aber noch realistisches Beispiel mit acht Prioritätsleveln und ein Maximalbeispiel mit 20 Leveln. Ein einzelnes Prioritätslevel als Minimalbeispiel widerspricht dem Sinn des Mechanismus, weshalb zwei Level das Minimalbeispiel darstellen. Mehr Level als 20 Prioritätslevel werden für ein Maximalbeispiel nicht angenommen. Ein höherer Wert kann implementiert werden, aber 20 Prioritätslevel werden in den meisten Anwendungsfällen nicht benötigt und können so als Maximalbeispiel verwendet werden.

Die Simulationsergebnisse aus Abbildung 6.5 zeigen, dass Prioritätslevel in kleineren Assemblies geringen Mehraufwand verursachen. In (a),(b),(c) und (d) lässt sich aus dem Median ablesen, dass sich die Schritte in kTAM zwischen unterschiedlichen Prioritätsleveln nur gering verändern. Die Ausreißer und Quartile zeigen, dass das Tileset größer ist, jedoch bleibt bei den Prioritätstiles die Assembly konstant in ihrer Größe. Bei Assemblies mit einer Höhe von zwei, wie in (e) und (f), oder einer Höhe von drei, wie in (g) und (h), präsentiert sich die Situation anders. Die Assembly wächst je nach Höhe an, was einen größeren Einfluss auf die Laufzeit der Self-Assembly hat als die Größe des Tilesets. Die Simulationsergebnisse zeigen auch, dass für Assemblies der Höhe eins das Prioritätslevel nur einen geringen Einfluss auf die Laufzeit hat. Assemblies und Molekülen der Höhe zwei und drei können niedrige Prioritätslevel noch mit geringem Mehraufwand implementiert werden, bei höheren Leveln wird jedoch das Tileset weit aufgebläht, was zu größeren Ausreißern in allen Messungen führt.

Es kann jedoch in sowohl (c) als auch (d) erkannt werden, dass die Simulationen mit 20 Prioritätsleveln in Quartilen und im Median schneller als mit acht Prioritätsleveln konstruiert werden. Eine mögliche Erklärung dafür könnte sein, dass das originale Tileset bereits groß ist. Die Tilesets sind mit den Größen 32 für H-1-norm und 68 für H-1-groß relativ groß. Durch acht Prioritätstiles wird diese Menge auf 40 und 76 erhöht, mit 20

## 6. Simulationen

---



**Abbildung 6.5.:** Darstellung der Simulationsergebnisse von Assemblies mit unterschiedlichen Prioritätsleveln. Jedes Tileset wurde dabei mit keinen, zwei, acht und 20 Prioritätsleveln simuliert.

Prioritätstiles auf 52 und 88. Aus allen Simulationen kann gefolgert werden, dass die Assemblygröße ein bedeutenderer Faktor für die Laufzeit ist als die Tilesetgröße. Ein größeres Tileset führt in kTAM zu mehr Ausreißern und größerer Differenz zwischen den Simulationen. Damit kann erklärt werden, warum in Abbildung 6.5 (c) und (d) die Quartile und der Median für 20 Prioritätslevel jeweils niedriger ist als für acht Prioritätslevel, während die Maxima deutlich größer sind.

Allgemein lässt sich zum Mechanismus der Prioritätstiles sagen, dass eine geringe Anzahl an Prioritätslevel einen geringen Einfluss auf Assembly und Tileset haben. Je größer die Self-Assemblies in ihrer Höhe sind und je mehr Prioritätslevel benötigt werden, desto mehr schlägt sich dieser Mechanismus auf die Laufzeit aus. Für  $p$  Prioritätslevel und Assemblyhöhe  $h$  wird hier das Tileset um  $ph$  und die Assembly um  $h$  Tiles erweitert. Dadurch werden vor allem die Ausreißer in den Simulationen größer, da die Assemblygröße konstant bleibt, die Tilesetgröße jedoch abhängig vom Prioritätslevel wächst.

Damit sind beide Mechanismen der Datenflusskontrolle evaluiert und vorgestellt. Sowohl Acknowledgements als auch Prioritätslevel lassen sich durch DNA-basierte Self-Assembly realisieren. Beide Mechanismen zeigen in den Simulationen, dass sie ohne großen Mehraufwand implementiert werden können. Es kann so mit der Evaluation der Flags weitergemacht werden.

## 6.6. Simulation und Evaluation von Flags

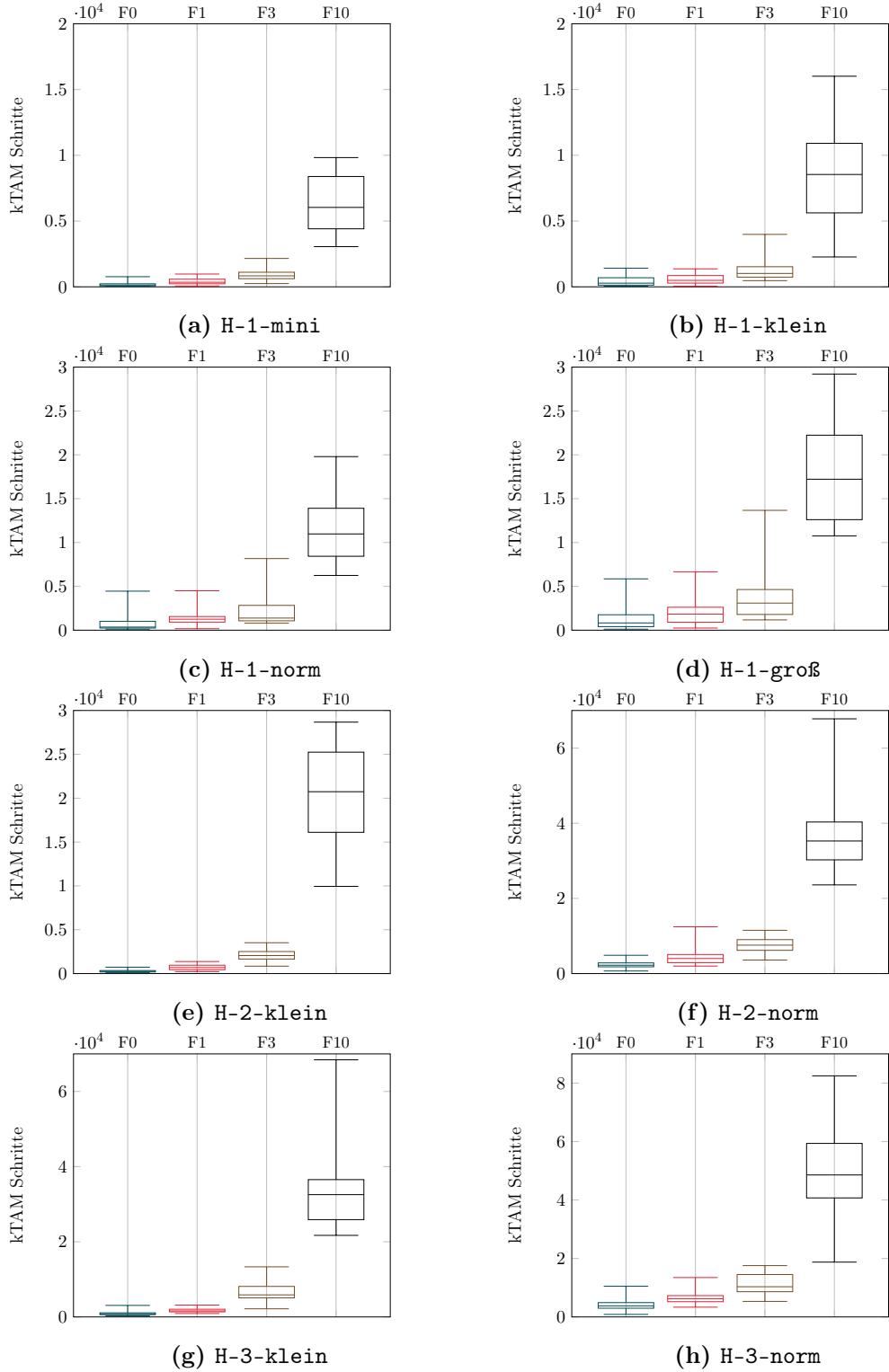
So wie die Prioritätslevel wird in dieser Sektion der Mechanismus der Flags simuliert. Die Ergebnisse dafür sind in Abbildung 6.6 zu sehen. Genau wie bei den Prioritätsleveln wurden für jedes Tileset die Minimalgröße von Flags mit einer Flag, eine realistische Flagmenge mit drei Flags und eine Maximalgröße mit zehn Flags simuliert. Mehr als zehn Flags können auch hier implementiert werden. Jedoch gibt es kaum Anwendungsfälle, in denen zehn oder mehr Flags benötigt werden.

Im Gegensatz zu den Simulationen der Prioritätslevel ist bei Flags der Unterschied zwischen diesen drei Flagmengen bedeutend größer. Während für eine oder drei Flags die Simulationen zwar merkbar mehr Schritte benötigen, sind die Unterschiede der Laufzeit relativ gering. Die Simulationen mit zehn Flags zeigen, dass viele Flags zu stark erhöhten Simulationsschritten führen. Das liegt daran, dass Flags im Gegensatz zu Prioritätsleveln nicht nur das Tileset, sondern auch noch zusätzlich die Größe der Assembly beeinflussen. Für  $f$  Flags und Molekülhöhe  $h$  wird das Tileset mit  $f(h + 1)$  Tiles und die Assemblygröße mit  $fh$  Tiles erweitert. Somit wächst nicht nur die Tilesetgröße, sondern auch die Assemblygröße abhängig von der Menge an Flags. Wenn nur eine Flag verwendet wird, ist die Assemblygröße analog zu der Implementierung mit zwei Prioritätsleveln. Jedoch, mit einer steigenden Anzahl von Flags wächst die Größe der Assembly linear, während sie bei einem Anstieg der Prioritätslevel konstant bleibt.

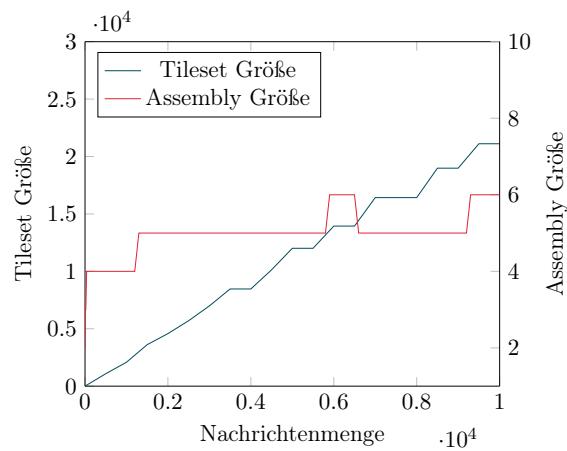
Die Simulationsergebnisse in Abbildung 6.6 zeigen, dass die Implementierung von einer

## 6. Simulationen

---



**Abbildung 6.6.:** Darstellung der Simulationsergebnisse von Assemblies mit unterschiedlichen Anzahlen von Flags in Boxplot Form. Jedes Tileset wurde dabei mit keiner, einer, drei und zehn Flags simuliert.



**Abbildung 6.7.:** Grafische Darstellung des Wachstums von Tilesetgröße und Assemblygröße im Verhältnis zur Nachrichtenmenge bei Prüfsummen. Dabei ist zu erkennen, dass die Assemblygröße schwankt, allgemein jedoch klein und fast konstant bleibt, während die Tilesetgröße linear steigt, mit einigen kleineren Plateaus.

bis drei Flags zu einer erkennbaren Zunahme der Schritte in der Self-Assembly führt. Dennoch liegt die resultierende Erhöhung innerhalb eines akzeptablen und realisierbaren Rahmens. Bei einer größeren Menge von Flagtiles wird die Laufzeit in kTAM jedoch bedeutend größer. Dabei sollte beim kTAM Simulationsmodell beachtet werden, dass eine große Assembly die Schrittzahl erheblich beeinflusst. Die Simulation von so großen Tilesets war in 2HAM mit den für diese Arbeit vorhandene Rechenleistung nicht möglich, da die Simulationen zu lange gedauert hätten.

## 6.7. Simulation und Evaluation von Prüfsummen

In den folgenden zwei Sektionen wird die letzte betrachtete Anforderung analysiert: die Fehlerbehandlung. Begonnen wird mit der Implementierung von Prüfsummen und somit mit der Fehlererkennung. Bei Prüfsummen in Self-Assemblies ist zur Simulation Folgendes zu sagen: Die Self-Assembly wächst durch Prüfsummen nicht bedeutend an, da nur ein weiteres Tile am Ende des Moleküls hinzukommt. Die Größe des Tilesets vergrößert sich abhängig von der Anzahl der Nachrichten wiederum deutlich.

Dies ist in Abbildung 6.7 verdeutlicht. In dem Graphen ist klar zu erkennen, dass das Tileset linear zur Nachrichtenmenge wächst. Die Assemblygröße steigt in den ersten 1.000 Nachrichten zwar an, jedoch bleibt die Größe danach zwischen fünf und sechs relativ konstant. Die Assemblygröße wächst jedoch nicht durch die implementierte Prüfsumme, sondern wegen der wachsenden Nachrichtenmenge. Prüfsummen vergrößern Assemblies der Höhe eins konstant um ein einzelnes Tile. Es ist bei dem Graphen anzumerken, dass eine konstante Gewichtung gewählt wurde, da die Assemblygröße und Tilesetgröße

Tileset	Assemblygröße	Tilesetgröße
H-1-mini	4	10
H-1-mini + chksm	3	21
H-1-klein	4	22
H-1-klein + chksm	4	211
H-1-norm	5	32
H-1-norm + chksm	4	2.081
H-1-groß	5	68
H-1-groß + chksm	6	21.111

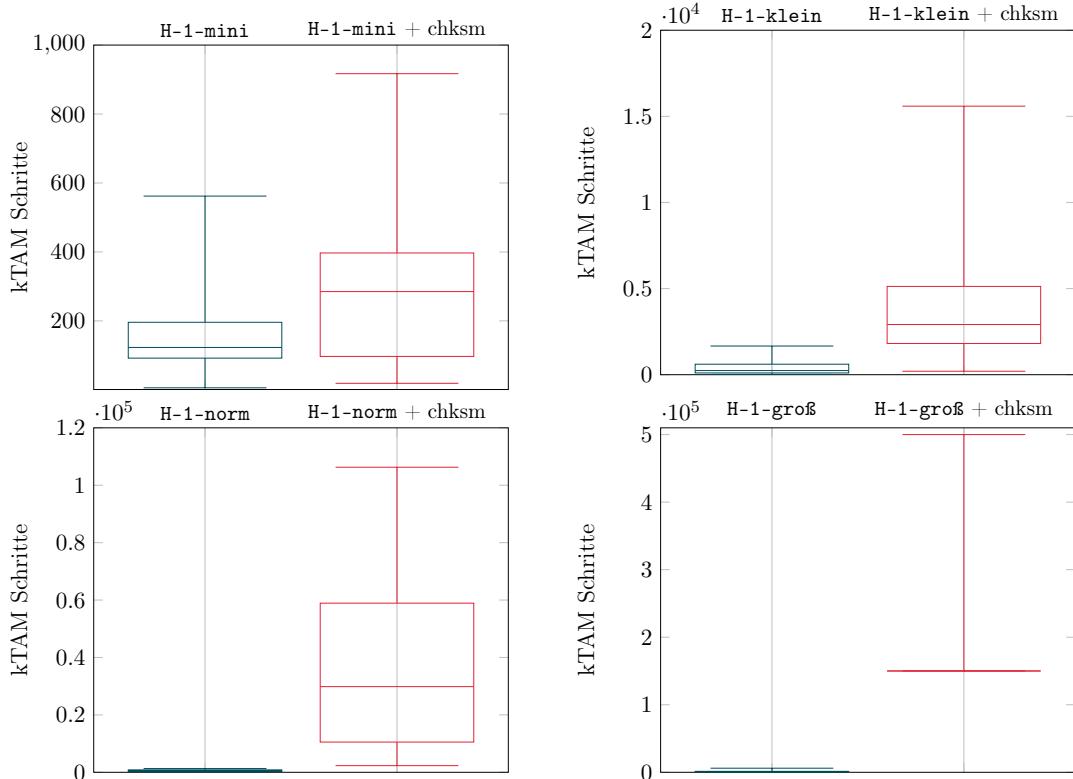
**Tabelle 6.5.:** Darstellung von Tileset- und Assemblygrößen für die vier generierten Tilesets aus Tabelle 6.2 und ihre Prüfsummenäquivalenten.

ungleich anwächst. Bei einer Nachrichtenmenge von 10.000 ist das Tileset mit einer Größe von 21.111 im Vergleich zur Assembly mit Größe sechs derart größer, dass ein Verhältnis von mindestens 1:3519 benötigt wird, um eine Veränderung herbeizuführen.

Mit diesen Voraussetzungen kann die Simulation und Analyse von Prüfsummen in Self-Assemblies durchgeführt werden. Dazu ist jedoch anzumerken, dass in einem realistischeren Modell nicht alle Tiles zur Self-Assembly verwendet werden. Nur Tiles, welche die korrekte Nachricht bilden, sollten auf diese Weise verwendet werden. Dabei wäre die Simulation in NetTAS jedoch trivial. Eine Self-Assembly der Höhe eins mit fünf Tiles bildet sich schnell und wäre identisch zu allen anderen Self-Assemblies dieser Größe. Deshalb wird in dieser Evaluation die Laufzeit der Self-Assembly mit allen Tiles verwendet, um einen einheitlichen Vergleich zwischen verschiedenen Tilesets zu haben. Somit kann ein Vergleich zwischen Prüfsummen-Assemblies für 100 Nachrichten mit Assemblies für 1000 Nachrichten verglichen werden. Diese haben gleich große Assemblies und wären somit in der Laufzeit im deterministischen Fall identisch. Die Simulationsergebnisse sind in Abbildung 6.8 dargestellt.

Alle bisherigen Simulationen nutzen die acht Tilesets aus Tabelle 6.2 und Tabelle 6.3. Bei den Prüfsummen werden jedoch nur die vier Tilesets aus Tabelle 6.2 herangezogen, da die Prüfsummenerstellung in Self-Assemblies im Skript lediglich für die generierten Tilesets der Höhe eins implementiert wurde. Obwohl das Prinzip für höhere Moleküle analog ist, und die Information jedes Tiles von dem Seed-Tile zum letzten Tile in jeder Reihe weitergegeben werden muss, genügt es für den Konzeptionsbeweis, den Mechanismus bei Molekülen der Höhe eins zu betrachten.

Aus den Simulationsergebnissen in Abbildung 6.8 lässt sich Folgendes ablesen. Für den kleinsten Datensatz ist die Laufzeit für Prüfsummen in *kTAM* noch relativ gering. Der Median des Tilesets ohne Prüfsumme liegt bei 123 Schritten in kTAM, während der Median mit Prüfsumme bei 285 Schritten liegt. Das untere Quartil ohne und mit Prüfsumme zeigt ebenfalls nur geringe Unterschiede mit 92 bzw. 97 Schritten. Ein etwas größerer Unterschied ist im oberen Quartil sichtbar, wo es ohne Prüfsumme 196 Schritte und



**Abbildung 6.8.:** Boxplots zur Darstellung der Messergebnisse von den Tilesets im Vergleich zu den Prüfsummenäquivalenten. Die zugehörigen Namen sind links und rechts über dem Graphen abgebildet und sind der Tabelle 6.5 entnommen. Bis auf H-1-groß + chksm wurde für jedes Tileset 30 Messungen durchgeführt, aus welchen in diesen Graphen Minimum, unteres Quartil, Median, oberes Quartil und Maximum abgebildet werden. Nur bei H-1-groß + chksm wurde auf die 30 Messungen verzichtet, da die Simulationen auf einem so großen Tileset zu lange gedauert hätten. Die dargestellten Werte sind aus insgesamt vier Messungen, die im Fließtext näher erläutert werden.

mit Prüfsumme 397 Schritte sind, allerdings bleibt dieser Unterschied im Gesamtkontext noch relativ unbedeutend.

Jedoch muss für das kleinste Tileset folgendes betrachtet werden. Die Assembly- und Tilesetgröße wird aus der Gewichtung gewonnen. Dadurch, dass die Gewichtung im **H-1-mini** Tileset 1:1 ist, wird die Assembly im Prüfsummenfall kleiner, da das Tileset sonst stark vergrößert wird. Die Assemblygröße mit Prüfsumme für zehn Nachrichten ist also drei. Das Tileset besteht dabei aus einem Seed-Tile, zehn verschiedenen Tiles für die verschiedenen Nachrichten (die sich neben dem Seed-Tile befinden) und den Prüfsummen-Tiles. Dabei sind die Prüfsummen-Tiles im Wesentlichen Kopien der mittleren zehn Tiles, mit dem Unterschied, dass sie sich nicht am Seed-Tile, sondern an den mittleren Tiles binden. Dieses Konzept kann auch auf Tilesets mit mehr Nachrichten ausgeweitet werden. Jedoch würde das den Sinn von Prüfsummen missinterpretieren, da so immer nur ein Tile mit identischem Prüfsummentile daneben erstellt werden würde. Die Größe des Tilesets für  $x$  zu kodierenden Nachrichten wäre so immer  $2x + 1$ . Um Prüfsummen besser testen zu können, wurde für die anderen Tilesets ausgeschlossen, eine Assemblygröße von drei zu verwenden, auch wenn dies bei der gewählten Gewichtung für besser erachtet wird. Somit erklären sich die Assembly- und Tilesetgrößen in Abbildung 6.7 und Tabelle 6.5. Auch wenn durch die zuvor beschriebene Methode kleinere Assemblies und Tilesets möglich wären, widerspricht dies dennoch dem Prinzip der Prüfsummen.

In den Boxplots der anderen drei Tilesets zeigt sich, wie lange die Simulation braucht, um die gleiche Assembly mit einer Prüfsumme zu erstellen. Der Boxplot des größten Tilesets sieht wie in der Abbildung dargestellt aus, da nur vier Messungen durchgeführt wurden, die alle abgebrochen wurden. Das liegt daran, dass selbst nach 150.000 Schritten in **kTAM** mit den gleichen Parametern (Binding Cost = 16 und Bond Breaking Cost = 11) die Assembly nur zwischen den Größen eins und zwei hin und her wechselt. Auch bei niedrigerer Binding Cost und vergleichsweise hoher Bond Breaking Cost zeigt die Simulation, dass das erste Tile neben dem Seed-Tile häufig bricht, bevor ein weiteres Tile sich binden kann. Obwohl die finale Assembly eine Größe von sechs aufweist, erreicht sie bis zum 150.000. Schritt nie eine Größe von mehr als zwei. Aufgrund der langen Dauer dieser Simulationen wurden sie in drei weiteren Durchläufen getestet. In keinem dieser Durchläufe zeigte sich eine Verbesserung gegenüber dem ersten Lauf.

Zuletzt wurde ein Durchlauf mit Parametern gewählt, die bei einem Molekül der Höhe eins mit Temperatur zwei funktioniert, da es nur eine Bindungsmöglichkeit von rechts nach links gibt. Die gewählten Parameter Binding Cost = 11 und Bond Breaking Cost = 21 führen dazu, dass sich Verbindungen der Stärke zwei nicht mehr lösen können. Solche Parameter für ein Tileset der Stärke zwei entfernen einen der zentralen Aspekte des **kTAM**. Um das Problem der Simulation der Prüfsumme im größten Tileset zu lösen, können die Parameter trotzdem angepasst werden. Dieser Simulationsdurchlauf wurde nach 500.000 Schritten abgebrochen, da zwischen dem 40.000. Schritt und dem 500.000. Schritt die Assembly immer auf Größe fünf geblieben ist. Dies liegt daran, dass in **kTAM** zwölf offene Wachstumsmöglichkeiten bei einer Assembly der Höhe eins und Länge fünf existieren. Bei einem Tileset der Größe 21.111 führt dies dazu, dass statistisch 253.332 verschiedene

zufällige Tileplatzierungen in kTAM möglich sind. Weitere Simulationsdurchläufe wurden aus zeittechnischen Gründen nicht durchgeführt.

Abschließend zu den Prüfsummen in DNA-Tile-basierten Assemblies lässt sich Folgendes festhalten: Bei kleinen Assemblies, die eine geringe Anzahl an variablen Tiles enthalten, bieten Prüfsummen eine Möglichkeit, die korrekte Bildung nach der Seed-Assembly zu bestätigen. Allerdings wird hierbei nicht überprüft, ob das korrekte Tile am Seed-Tile gebunden wurde. Auch muss angemerkt werden, dass bei größeren Assemblies mit variablen Tiles die Anzahl der notwendigen Prüfsummentiles rapide ansteigt. Falls solch eine umfangreiche Tilemenge in einem konkreten Anwendungsfall vertretbar ist, könnte der Mechanismus dennoch implementiert werden. Dies begründet sich darin, dass die in Abbildung 6.8 dargestellten Simulationen nicht die Laufzeit eines einzelnen Prüfsummenmoleküls, sondern das gesamte Tileset berücksichtigen. Insgesamt erscheint dieser Fehlerbehandlungsmechanismus aufgrund der hohen Anzahl an Tiles jedoch nur für geringe Nachrichtenmengen praktikabel.

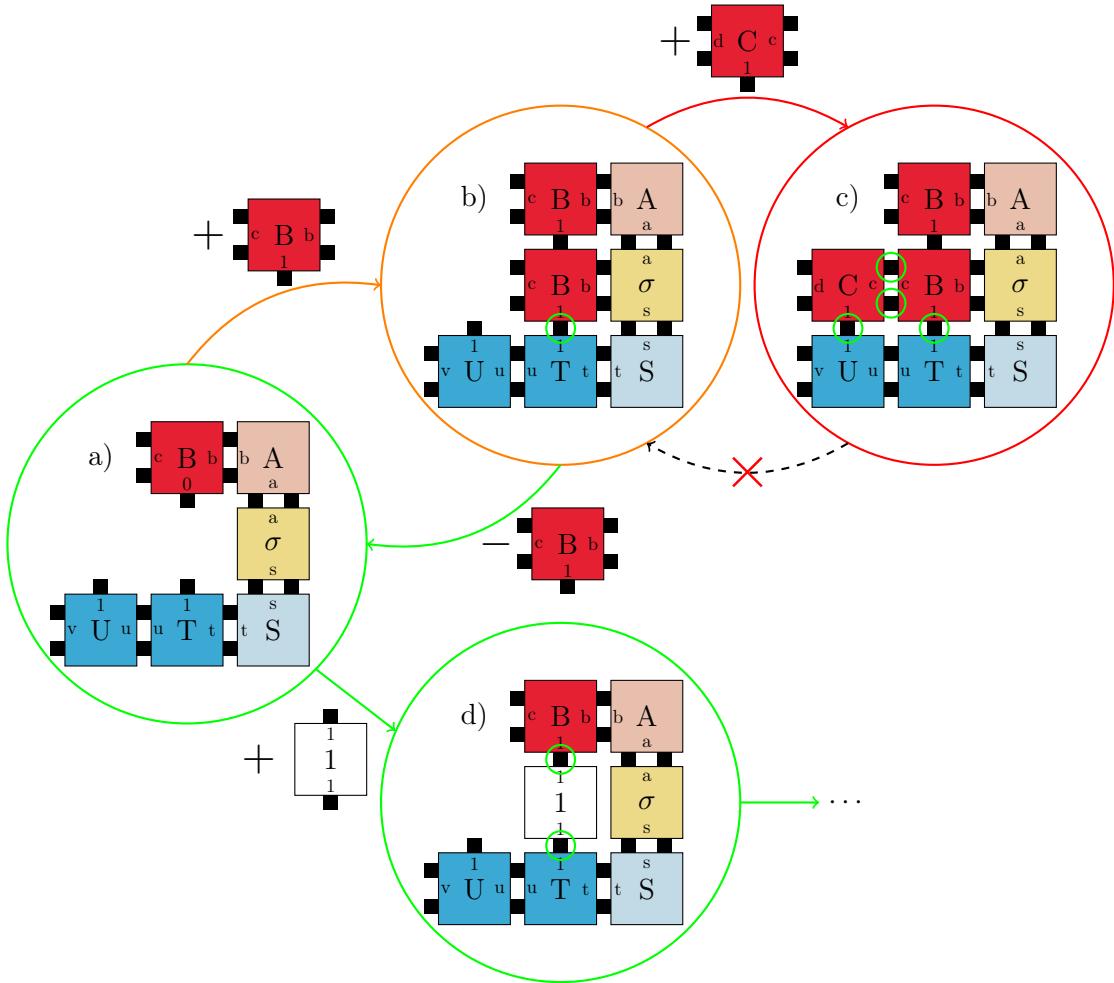
## 6.8. Simulation und Evaluation von Snaked-Proofreading

Der andere vorgestellte Mechanismus zur Fehlerbehandlung ist das Snaked-Proofreading, die Fehlerkorrektur. Im Vergleich zu den vorherigen Betrachtungen, die sich gegenseitig nicht beeinflussen und dementsprechend getrennt voneinander betrachtet werden können, muss beim Snaked-Proofreading betrachtet werden, wie sich die Simulationszeit im Zusammenhang zu allen vorherigen Mechanismen verhält. Deshalb wird das Snaked-Proofreading als Letztes in diesem Kapitel analysiert.

### 6.8.1. Snaked-Proofreading für das H-3-norm Tileset

An dieser Stelle ist anzumerken, dass in den vorherigen Simulationen einige Fehlkonstruktionen entstanden sind. In den Simulationen des Tilesets H-3-norm entstand jeweils einmal eine fehlerhafte Konstruktion, sowohl in der Simulation mit drei Flags als auch in der mit zehn Flags. Eine Fehlerquote von zwei aus 120 Läufen ist hoch genug, um über Fehlerkorrektur für diese Assemblies nachzudenken. Der Fehler ist in Abbildung 6.9 dargestellt und ist unabhängig zu den Flagtiles. In a) ist die Assembly, wie vorhergesehen konstruiert. Bindet sich das Tile mit dem Bezeichner „1“, so gibt es in dem Durchlauf kein Problem. Dies ist durch die Farbe Grün im Übergang und in d) dargestellt.

Bindet sich das Tile mit dem Bezeichner „B“, wie im Übergang zwischen a) und b) zu sehen ist, so kommt es dadurch zu einem gefährlichen Zustand. Das Molekül ist in b) jedoch noch nicht fehlkonstruiert, da das System auf eine Temperatur von zwei festgelegt ist. In b) wird grün markiert, dass das Tile nur mit einer Stärke von eins mit dem restlichen Molekül gebunden ist. Mit den richtigen Parametern in kTAM kann dieses Molekül durch die Bond Breaking Cost wieder entfernt werden. Dies ist durch den grünen



**Abbildung 6.9.:** Grafische Darstellung des Errors im H-3-norm Tilesets. Die farblichen Kreise deuten dabei Zustände an, zwischen welchen in kTAM durch Aufbau und Lösung von Bindungen gewechselt werden kann. Bindet sich, wie abgebildet, das Tile mit dem Bezeichner „B1“ an das Molekül in a), so bildet sich das Molekül wie in d) korrekt und das Tile kann durch Kleberstärke zwei nur schwer wieder gebrochen werden. Das ist durch die grüne Farbe des Zustandes gekennzeichnet. Bindet sich jedoch das Tile mit dem Bezeichner „B“ an das Molekül in a), so wird ein gefährlicher Zustand in b) erreicht. Noch ist das Molekül nicht falsch gebildet, dementsprechend ist der Zustand mit der Farbe Orange dargestellt. Das Tile hat nur die Kleberstärke eins, dargestellt durch den kleinen grünen Kreis. Es sollte im richtigen System gelöst werden. Bindet sich das Tile vor dem Lösen des B-Tiles mit dem Tile, das den Bezeichner „C“ trägt, entsteht ein gravierender Fehler in der Konstruktion. Wie in c) durch die kleinen grünen Kreise dargestellt, sind die beiden Tiles mit Kleberstärke zwei verbunden, sowie jeweils mit einem Kleber am restlichen Molekül. Somit ist die Verbindung stark genug, um in einem System der Temperatur zwei nicht nach kurzer Zeit wieder gelöst zu werden. Es ist zwar möglich, dass das Molekül so wieder aufbricht, jedoch ist dies laufzeittechnisch nicht wünschenswert.

Tileset	Assemblygröße	Tilesetgröße
H-3-norm (mit SP)	15 (60)	26 (104)
H-3-norm-1F (mit SP)	18 (72)	30 (120)
H-3-norm-3F (mit SP)	24 (96)	38 (152)
H-3-norm-10F (mit SP)	45 (180)	66 (264)
H-3-norm-2P (mit SP)	18 (72)	32 (128)
H-3-norm-8P (mit SP)	18 (72)	50 (200)
H-3-norm-20P (mit SP)	18 (72)	86 (344)

**Tabelle 6.6.:** Tileset- und Assemblygrößen für das H-3-norm Tileset und Varianten. In Klammern ist die jeweilige Größe mit Snaked-Proofreading (SP) abgebildet, die immer das Vierfache sind. Die Größen sind für sowohl alle drei Flag- (F) als auch Prioritätslevelfälle (P) mit den entsprechenden Quantitäten angegeben.

Übergang von b) zu a) angedeutet.

Wird jedoch das Tile mit dem Bezeichner „C“ gebunden, bevor sich das Tile lösen kann, so kommt es zu einem fehlerhaften Zustand in der Konstruktion. In c) ist zu erkennen, dass für Temperatur zwei sowohl die beiden Tiles untereinander als auch die zwei Tiles zusammen mit dem restlichen Molekül eine ausreichende Kleberstärke besitzen, um nur schwer wieder gebrochen zu werden. Da in kTAM die Kleberstärke jedes einzelnen Tiles betrachtet wird, hat jedes Tile in dem Modell sogar eine Kleberstärke von drei. Dadurch brechen diese Tiles noch unwahrscheinlicher. Mit genügend Zeit und Schritten in kTAM kann das zwar trotzdem passieren, jedoch ist das dem Zufall überlassen.

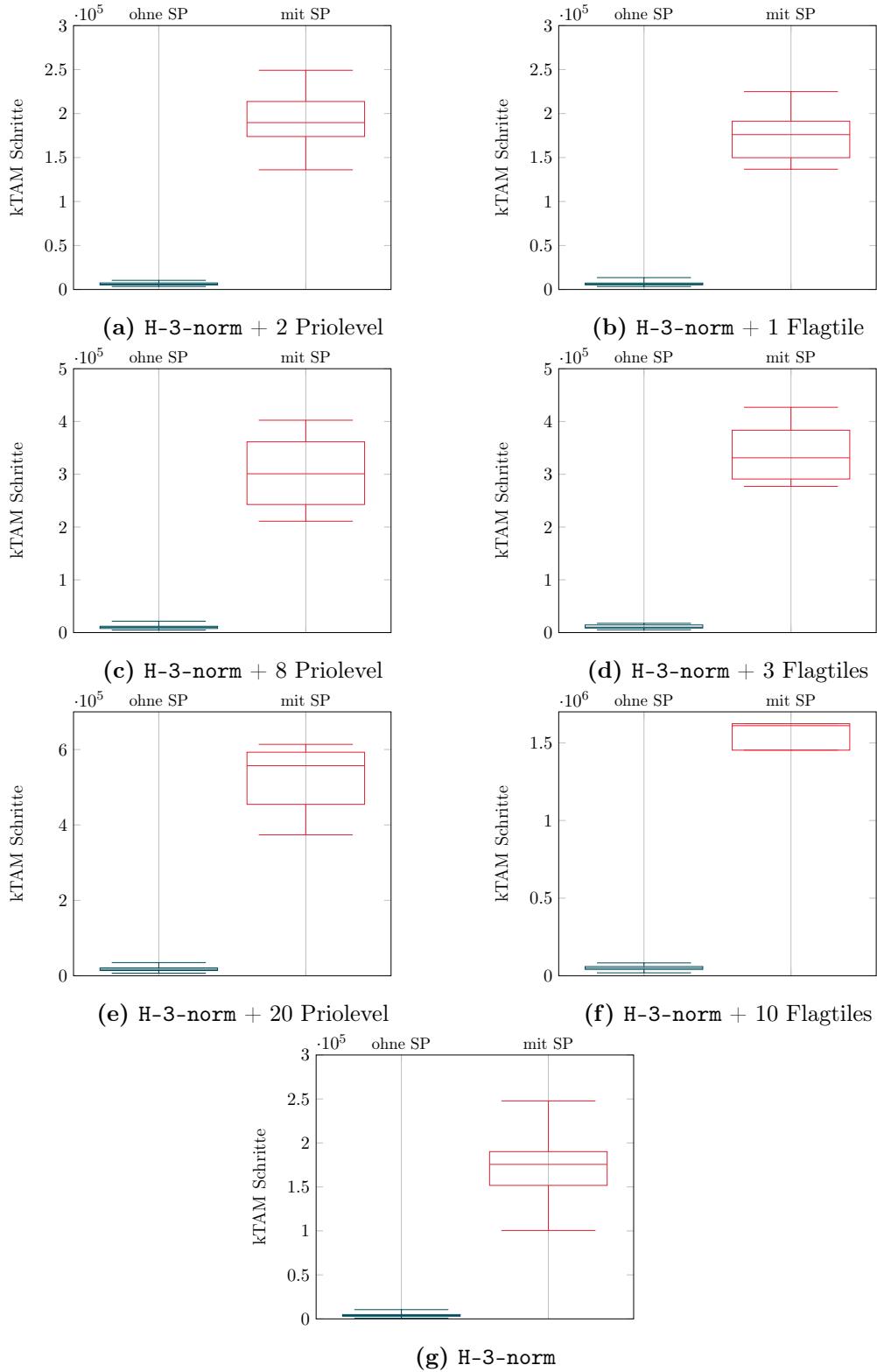
Zur Lösung des Problems stehen verschiedene Ansätze jenseits des Snaked-Proofreadings zur Verfügung. Eine alternative Implementierung der Assembly könnte in Erwägung gezogen werden. Jedoch ist zu beachten, dass bei bestimmten Implementierungen, etwa beim Äquivalenzvergleich, ähnliche Fehlerstrukturen auftreten können. Dieser Fehler wurde im Kapitel 2 vorgestellt. Facet-Errors können durch Snaked-Proofreading erschwert werden, da im hier implementierten Snaked-Proofreading vier Tiles falsch gebunden werden müssen, um den gleichen Fehler zu erhalten. Somit soll im Folgenden zunächst das Tileset H-3-norm simuliert und betrachtet werden.

Die Fehler in den vorherigen Simulationen traten zwar im Fall von drei und zehn Flags auf, doch das Problem liegt nicht bei den Flags. Der Fehler kann bei allen Self-Assemblies des H-3-norm Tileset vorkommen, weshalb im Folgenden alle Fälle betrachtet werden, obwohl die Fehler nur bei Flags vorgekommen sind.

Das Tileset H-3-norm weist ohne weitere Ergänzungen eine beachtliche Größe von 26 Tiles auf. Bei Anwendung des Snaked-Proofreading-Verfahrens führt allein dies zur vierfachen Größe in der Assembly. Kombiniert mit einer Molekülhöhe von drei, erhöht sich die Anzahl der notwendigen Tiles für Mechanismen wie Flags und Prioritätslevel weiter. Das Resultat dieses deutlichen Wachstums ist in Tabelle 6.6 zu sehen. Solch umfangreiche Assemblies resultieren in kTAM in langwierigen Simulationen, da ein umfangreiches

## 6. Simulationen

---



**Abbildung 6.10.:** Darstellung der Simulationsergebnisse von Proofreading für die fehleranfällige Assembly des Tilesets H-3-norm mit verschiedenen Variationen des Tilessets.

Molekül in Kombination mit einem großen Tileset eine Vielzahl an möglichen Verbindungsstellen für zufällige Tests bietet. Bei der ersten Messung von **H-3-norm** mit drei Flags und Snaked-Proofreading wurde die Messung in kTAM nach zwei Millionen Schritten abgebrochen, da die Simulation einige Stunden lief und die Assembly für 500.000 Schritte zwischen 86 und 89 Tiles von finalen 96 Tiles hin- und herschwankte. Dementsprechend wurden alle folgenden Simulationen mit anderen Parametern durchgeführt. Die Binding Cost wurde auf 15 gesetzt. Diese ist in allen anderen Simulationen auf 16 gesetzt. Die Bond Breaking Cost wurde auf zwölf gesetzt. Diese ist in allen anderen Simulationen auf elf.

Die Simulationsergebnisse für Snaked-Proofreading sind in Abbildung 6.10 zu sehen und zeigen, dass Snaked-Proofreading auf einer Assembly wie **H-3-norm** zu großen Problemen in der Laufzeit führt. Auch auf dem unveränderten Tileset ist der Unterschied in kTAM hoch, doch sobald weitere Mechanismen hinzukommen, wird dieser Unterschied noch merkbarer. Das liegt an der Höhe und Länge der Assembly. Dadurch, dass **H-3-norm** eine Höhe von drei besitzt, müssen für die Implementierung von Flags oder Prioritätsleveln mindestens drei Tiles in der Assembly hinzugefügt werden. Mit Snaked-Proofreading bedeutet das mindestens zwölf zusätzliche Tiles für dieselbe Implementierung. Das Extrembeispiel mit zehn Flags verdeutlicht das Problem besonders: Hier wird die Assembly statt um 30 Tiles um ganze 120 Tiles erweitert, was zu besonders hohen Schrittzahlen führt.

Für **H-3-norm** kann festgehalten werden, dass das Snaked-Proofreading mit einem hohen Preis in der Laufzeit einhergeht. In keiner der teilweise mehrere Stunden dauernden Simulationen trat jedoch derselbe Fehler auf wie in den Simulationen ohne Snaked-Proofreading. In bestimmten Anwendungsfällen kann Snaked-Proofreading trotzdem die richtige Entscheidung sein. Dies ist insbesondere dann der Fall, wenn die Laufzeit eine geringe Priorität hat oder die korrekte Bildung der Self-Assembly eine hohe Priorität besitzt.

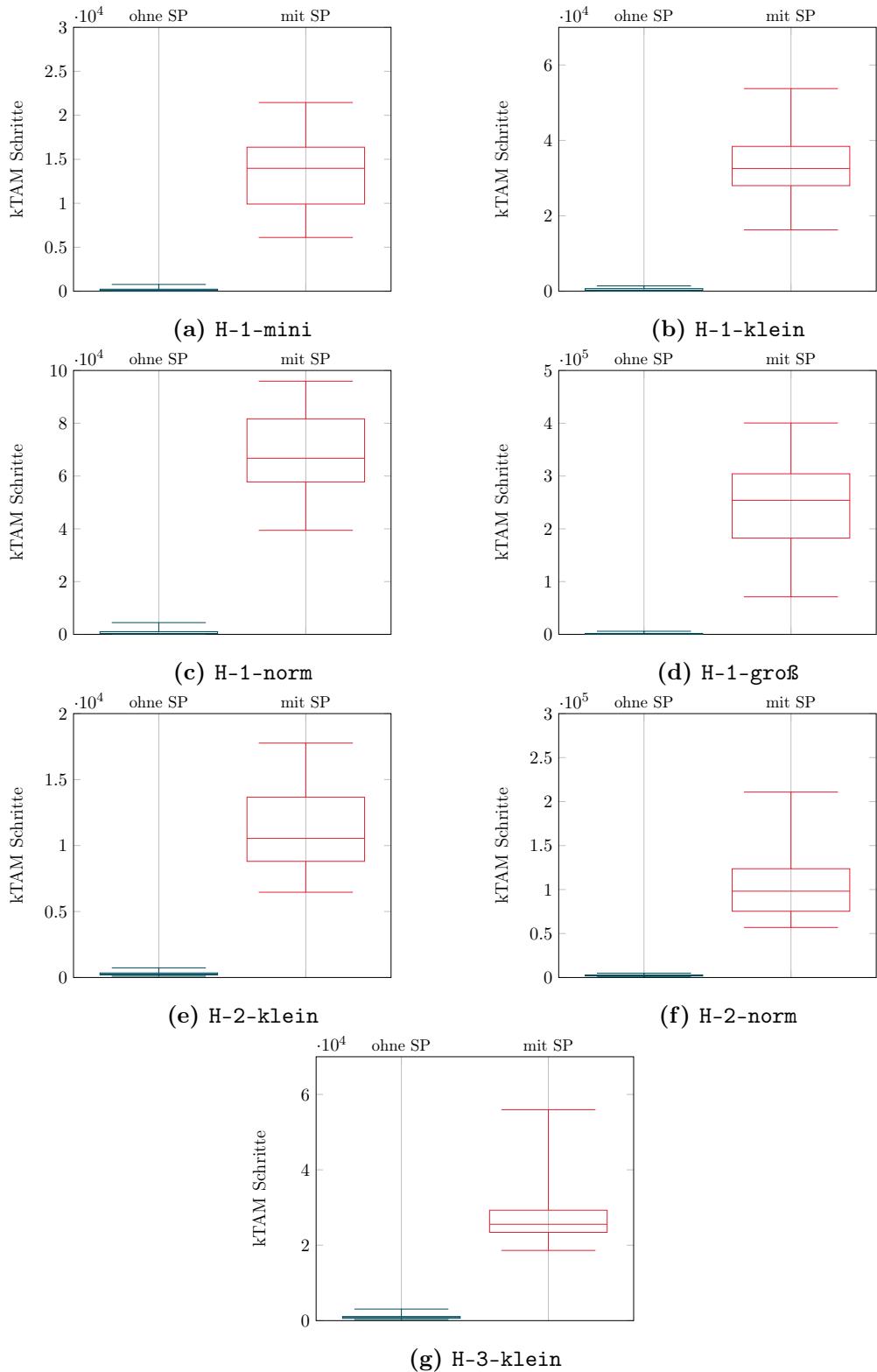
### 6.8.2. Snaked-Proofreading für die erstellten Tilesets

Die Messergebnisse in Abbildung 6.11 weisen ein ähnliches Bild zur vorherigen Sektion auf. Der Unterschied in den Simulationen der Tilesets in kTAM ist merklich größer für Snaked-Proofreading. So groß, dass für die Tilesets wie zum Beispiel in d) in einem verhältnismäßig kleinen Skalenbereich die Messungen ohne Snaked-Proofreading kaum noch zu erkennen sind. Für kleine Tilesets kann jedoch festgestellt werden, dass Snaked-Proofreading einen kleineren Einfluss hat.

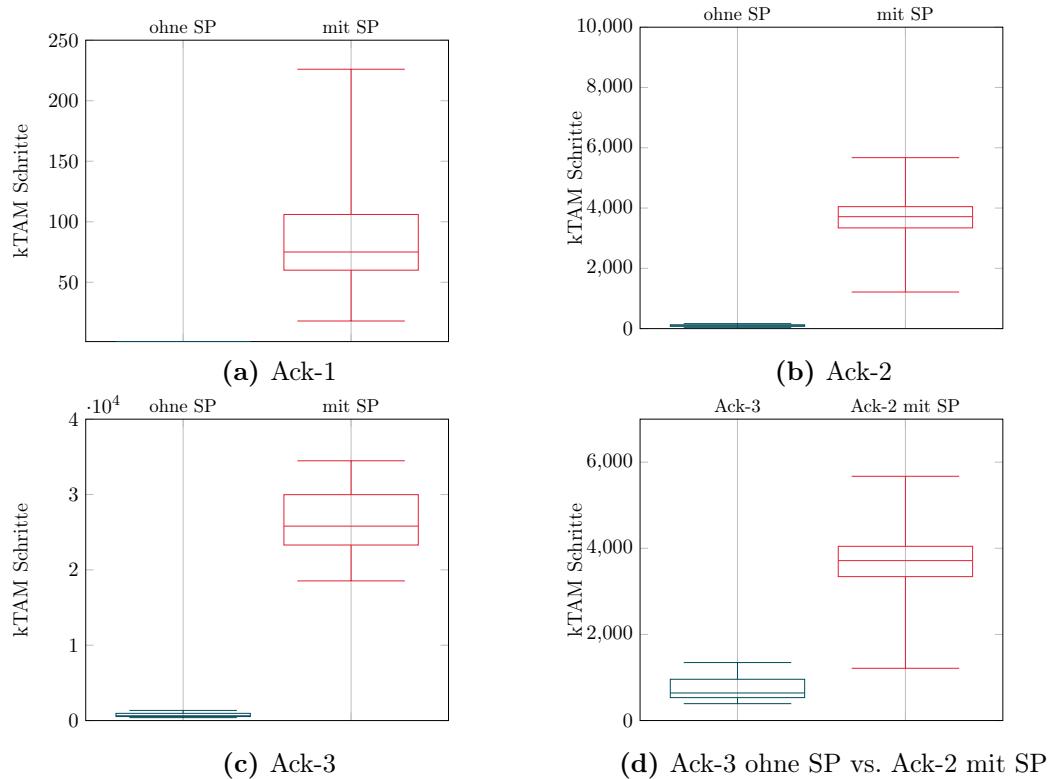
Wenn eine präventive Fehlerbehandlung in einem Kontext benötigt wird, so könnte der Einsatz von Snaked-Proofreading trotzdem gerechtfertigt sein. Dabei zeigen die Messreihen, dass kleinere Assemblies generell weniger Probleme mit diesem Verfahren haben. Es ist zu beachten, dass solche kleineren Assemblies auch insgesamt weniger fehleranfällig sind. In Tilesets für Assemblies der Höhe eins ist es beispielsweise fast unmöglich einen

## 6. Simulationen

---



**Abbildung 6.11.:** Darstellung der Simulationsergebnisse von Proofreading auf den erstellten Assemblies abzüglich des zuvor betrachteten H-3-norm Tilesets.



**Abbildung 6.12.:** Darstellung der Simulationsergebnisse von Proofreading auf den Assemblies für Acknowledgements. Neben den Vergleichen der Tilesets in (a), (b) und (c), wird in (d) das Ack-3 Tileset ohne Snaked-Proofreading mit dem Ack-2 Tileset mit Snaked-Proofreading verglichen.

Facet-Error zu erhalten, da das Molekül nicht über mehrere Kleber innerhalb verbunden ist. Daraus kann gefolgert werden, dass Snaked-Proofreading für die Tilesets H-1-groß und H-1-norm nicht notwendig sein dürfte, da bei so großen Tilesets für Moleküle der Höhe eins der Effekt gering und die Kosten hoch sind. Gleichzeitig sind die hier diskutierten Ergebnisse mit anderen Parametern verbunden als die Simulationen von H-3-norm. Dementsprechend kann angenommen werden, dass der Effekt von Snaked-Proofreading hier nicht so stark ist wie bei dem Größten der acht Tilesets.

### 6.8.3. Snaked-Proofreading für Acknowledgements

In dieser Untersektion soll betrachtet werden, was in Sektion 6.4 angedeutet wurde: der Vergleich von Acknowledgement Self-Assemblies mit und ohne Snaked-Proofreading. Besonders interessant ist der Vergleich des Ack-2 Tilesets mit Snaked-Proofreading gegenüber dem Ack-3 Tileset ohne dieses Verfahren.

In Abbildung 6.12 sind die Messergebnisse dargestellt. Der leere Boxplot in a) zeigt, dass **Ack-1** ohne Snaked-Proofreading konstant ohne einen Schritt in jeglichen Simulationsmodellen durchgeführt werden kann. Mit Snaked-Proofreading gibt es in **Ack-1** vier Tiles, die simuliert werden können, weshalb dafür Ergebnisse dargestellt werden können. Der Unterschied zwischen **Ack-2** und **Ack-3** ist klarer erkennbar. Trotz Snaked-Proofreading bleiben die Tilesets, aufgrund der ursprünglichen Größe, relativ klein. Im Vergleich zu vorherigen Messreihen zeigt nur **Ack-3** eine deutlich höhere Komplexität im Snaked-Proofreading. Doch auch diese führt mit einem Median von 25.000 Schritten in **kTAM** recht schnell die Self-Assembly durch. Das ist vergleichbar mit **H-2-klein** oder **H-3-klein** aus der vorherigen Untersektion.

Der bedeutende Vergleich in dieser Untersektion ist jedoch zwischen dem Tileset **Ack-3** und dem Snaked-Proofreading Tileset von **Ack-2**. Das Problem des **Ack-2** Tilesets ist, dass durch einen einzelnen Wachstums- und Nukleationsfehler die Liganden gebunden werden können, ohne dass sich die gesamte Assembly bildet. Durch Snaked-Proofreading wird dies auf acht wiederholte Wachstumsfehler erweitert, während in **Ack-3** zwei Wachstumsfehler benötigt werden, um den gleichen Fehler von zuvor zu wiederholen. In Abbildung 6.12 d) sind die Simulationsergebnisse beider Tilesets visualisiert. Es wird deutlich, dass **Ack-2** mit Snaked-Proofreading mehr Schritte für die Self-Assembly benötigt als **Ack-3**. Im Durchschnitt beansprucht **Ack-2** mit Snaked-Proofreading viermal so viele Schritte wie **Ack-3**. Jedoch können in der Self-Assembly auch viermal so viele Wachstumsfehler gemacht werden.

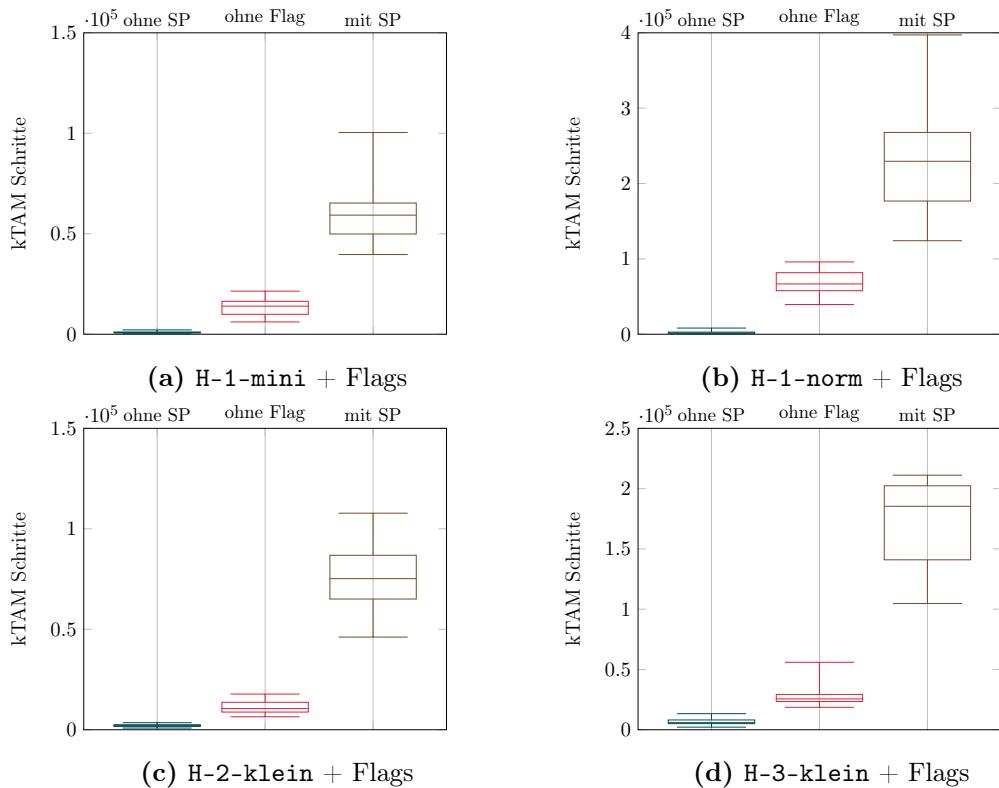
Die Entscheidung für eine bestimmte Acknowledgement-Implementierung hängt von der jeweiligen Anwendung ab: Es muss abgewogen werden zwischen der Schnelligkeit der Self-Assembly und der gewünschten Zuverlässigkeit.

### 6.8.4. Snaked-Proofreading für Flags

Diese Untersektion wird die Simulationsergebnisse von Snaked-Proofreading im Hinblick auf die Implementierung von Flags evaluieren. Dabei wurde das Beispiel mit drei Flags für mehrere Tilesets verwendet und simuliert. Es wurden drei Flags gewählt, da es sich dabei um ein nicht minimales und gleichzeitig realistisches Beispiel handelt.

Die Messergebnisse in Abbildung 6.13 zeigen, dass Snaked-Proofreading auf Flags einen großen Einfluss hat. Die Abbildung ist für jeden Graphen wie folgt aufgeteilt: Links sind die Simulationsergebnisse für Tilesets mit Flags, jedoch ohne Snaked-Proofreading, dargestellt. In der Mitte befinden sich die Messergebnisse für Tilesets mit Snaked-Proofreading, aber ohne Flags. Rechts werden schließlich die Ergebnisse für Tilesets mit sowohl Snaked-Proofreading als auch Flags präsentiert.

Der hohe Aufwand für Snaked-Proofreading mit Flags liegt daran, dass Flags einen großen Einfluss auf die Assemblygröße haben. Im Beispiel des **H-1-mini** Tilesets erhöht sich durch die drei Flags die Assembly von der Größe vier auf die Größe sieben. Mit



**Abbildung 6.13.:** Darstellung der Simulationsergebnisse von Proofreading-Assemblies mit Flags. Dabei werden für jedes Tileset links die Messergebnisse mit Flags, aber ohne Snaked-Proofreading dargestellt. Mittig sind die Messergebnisse ohne Flags aber mit Snaked-Proofreading dargestellt und rechts die Messergebnisse mit beiden Mechanismen.

Snaked-Proofreading wächst die Assembly von 16 Tiles auf 25 Tiles. Es ist dabei klar zu erkennen, dass in a) und b) für H-1-mini und H-1-norm ein großer Sprung zwischen Snaked-Proofreading ohne Flags zu Snaked-Proofreading mit Flags existiert. Dieser Unterschied ist in c) und d) für H-2-klein und H-3-klein noch einmal deutlicher. Das liegt daran, dass bei Molekülhöhe zwei und drei weitere Tiles für Flags benötigt werden.

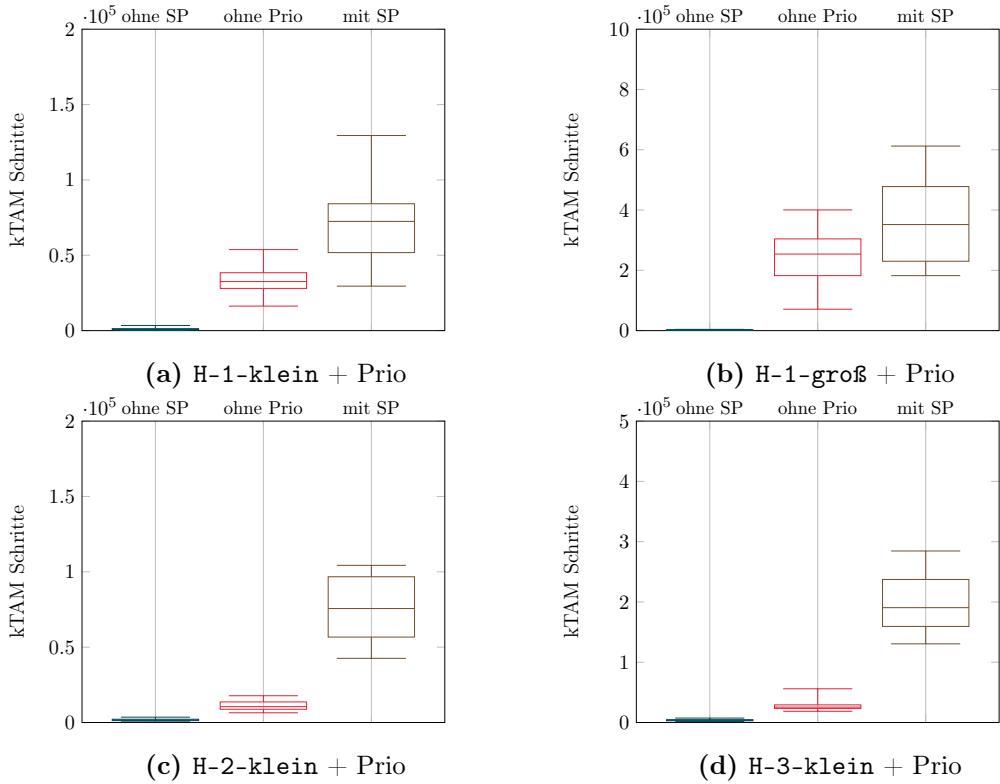
Ein Tileset mit Flags durch Snaked-Proofreading zu erweitern, wirkt durch die Messungen schwierig. Wird beides benötigt, so ist aber für Assemblies mit niedriger Molekülhöhe mit weniger Aufwand verbunden, diese Self-Assembly in kTAM durchzuführen.

### 6.8.5. Snaked-Proofreading für Prioritätslevel

In dieser Untersektion wird der Einfluss von Snaked-Proofreading auf Prioritätslevel in Assemblies betrachtet. Die Graphen in Abbildung 6.14 sind dabei gleich strukturiert wie für die Flag-Simulationen aus der vorherigen Untersektion. In jedem Graphen werden

## 6. Simulationen

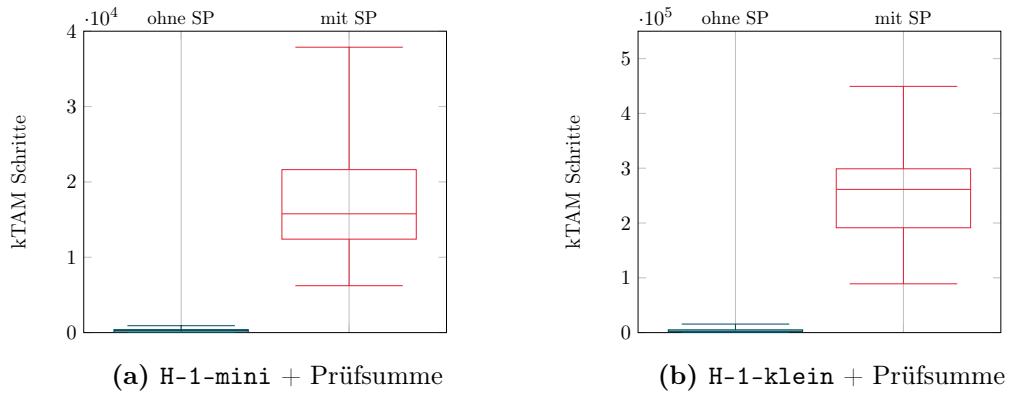
---



**Abbildung 6.14.:** Darstellung der Simulationsergebnisse von Proofreading-Assemblies mit Prioritätsleveln. Dabei werden links für jedes Tileset die Messergebnisse dargestellt, die das Tileset mit Prioritätslevel, aber ohne Snaked-Proofreading, zeigen. Mittig sind die Messergebnisse des Tilesets ohne Prioritätslevel aber mit Snaked-Proofreading dargestellt und rechts die Messergebnisse des Tilesets mit beiden Mechanismen.

links Tilesets mit Prioritätsleveln ohne Snaked-Proofreading dargestellt. Mittig finden sich die Messergebnisse des Tilesets mit Snaked-Proofreading ohne Prioritätslevel und rechts die Messergebnisse des Tilesets mit Snaked-Proofreading und Prioritätsleveln. Für die Prioritätslevel wurde, genauso wie zuvor bei den Flags, ein nicht minimales und gleichzeitig noch realistisches Beispiel mit acht Prioritätsleveln gewählt.

Die Simulationsergebnisse zeigen, dass der Aufwand für Snaked-Proofreading in Kombination mit Prioritätsleveln in kTAM merklich ansteigt. Dennoch ist dieser Anstieg nicht so ausgeprägt wie bei der Verwendung von Flags. Das liegt daran, dass mit acht Prioritätstiles das Tileset zwar stark anwächst, die Assembly jedoch konstant gleich groß bleibt, unabhängig von der Größe des Prioritätslevels. Dies ist bei Flags nicht der Fall. In Abbildung 6.14 a) und b) ist eindeutig zu sehen, dass Prioritätslevel für Moleküle der Höhe eins einen relativ geringen Einfluss haben. Für höhere Moleküle, wie in c) und d) ist der Aufwand eindeutiger, jedoch ist auch dieser nicht so groß, wie zuvor bei den Flags.



**Abbildung 6.15.:** Darstellung der Simulationsergebnisse von Proofreading-Assemblies für Prüfsummen. Dabei ist anzumerken, dass die Messreihen für H-1-norm und H-1-groß nicht abgebildet werden. Das liegt daran, dass die Simulationen der Tilesets mit Prüfsummen nicht abgeschlossen werden konnten, da bei so großen Tilesets die Simulationen teilweise nach über zwei Millionen Schritte noch immer nur ein einzelnes Tile beinhalten. Die Größe der Tilesets und damit das Problem wird in Tabelle 6.5 dargestellt.

Somit ist zu sagen, dass Snaked-Proofreading auf einem Tileset, welches Prioritätslevel implementiert, einen klar höheren Aufwand verursacht. Im Vergleich zu Flags ist dies jedoch bedeutend geringer. Je nach Anwendung ist es somit möglich, die beiden Mechanismen zusammen zu implementieren, ohne dabei die Dauer der Self-Assembly zu stark zu erhöhen.

### 6.8.6. Snaked-Proofreading für Prüfsummen

Abschließend wird in dieser Sektion das Snaked-Proofreading für Prüfsummen evaluiert. Während Prüfsummen primär zur Fehlererkennung dienen, fällt das Proofreading in den Bereich der Fehlerkorrektur. Eine Kombination beider Mechanismen kann also in Betracht gezogen werden.

Die Messungen für Prüfsummen mit Snaked-Proofreading haben sich als schwierig herausgestellt. Wie in Abbildung 6.15 zu erkennen ist, konnte für H-1-mini noch simuliert werden, wie sich das Tileset mit Prüfsumme und Snaked-Proofreading bildet. Auch wenn in Abbildung 6.15 b) Messergebnisse für das Tileset H-1-klein dargestellt werden, wurde bereits diese Simulation zuerst abgebrochen. Der Grund davor war, dass nach über zwei Millionen Schritten in kTAM immer noch nicht die Seed-Assembly abgeschlossen wurde. Diese umfasst zwar vier Tiles, doch die finale Self-Assembly endet erst mit 16 Tiles.

Das liegt daran, dass das H-1-klein Tileset mit Prüfsumme und Snaked-Proofreading 844 Tiles beinhaltet. Deshalb wurden wie zuvor schon bei H-3-norm die Parameter auf Binding Cost = 15 und Bond Breaking Cost = 12 gesetzt, wobei dabei immer noch der in Abbildung 6.15 b) dargestellte Abstand vorhanden ist. Simulationen für H-1-norm oder

H-1-groß wurden abgebrochen, da diese das Problem nur noch weiter verstärken.

Wenn sowohl Prüfsummen als auch Snaked-Proofreading berücksichtigt werden, kann die Implementierung in einem Tileset komplex werden. Dennoch sollte betont werden, dass in speziellen Anwendungsfällen, bei denen die Größe des Tilesets in kTAM-Simulationen nicht im Vordergrund steht, eine kombinierte Implementierung beider Ansätze durchaus machbar ist.

Allgemein lässt sich aus allen Simulationen für Snaked-Proofreading schließen, dass dieser Mechanismus stark erhöhte Laufzeiten für die Bildung der Self-Assembly bedeutet. Jedoch gibt es auf der Ebene der Nanogeräte und spezifisch im Anwendungsgebiet der DNA-Tiles wenige Mechanismen, die Fehlerkorrekturen ermöglichen. Der zuvor betrachtete Ansatz für Prüfsummen ist noch aufwendiger als Snaked-Proofreading und beeinflusst keine Growth- oder Facet-Errors. Auch Acknowledgements behandeln diese Fehler nicht. Diese zwei Errorarten sind in Self-Assemblies jedoch von großer Bedeutung. Es kann daher gerechtfertigt sein, Snaked-Proofreading auf ein Tileset anzuwenden, auch wenn dies mit einem erhöhten Aufwand im Prozess der Self-Assembly verbunden ist.

### 6.9. Evaluation Zusammenfassung

Zusammenfassend lässt sich zur gesamten Evaluation sagen, dass die Größe der Assembly den mit Abstand größten Einfluss auf die Laufzeit einer Self-Assembly hat. Die Größe des Tilesets beeinflusst auch die Laufzeit, jedoch mehr in den Extremen. Das bedeutet, dass der Abstand zwischen den minimal benötigten Schritten und den maximal benötigten Schritten mit der Größe des Tilesets ansteigt.

Die Ergebnisse dieses Kapitels müssen jedoch immer in dem Kontext des zentralen Problems der Tilesetbetrachtung analysiert werden. Für die Evaluation wurde immer das gesamte Tileset gebracht, auch wenn in einem realistischen Anwendungsfall immer nur ausgewählte Tiles aus der Menge für die Self-Assembly verwendet werden. Da die Simulation und der Vergleich verschiedener Tilesets sonst jedoch trivial wäre, wurde die Entscheidung für alle Messungen getroffen. Daraus resultieren höhere Schrittzahlen und extremeren Messungen in allen Messungen in kTAM. Bei so großen Molekülen und Tilesets konnten 2HAM und kTHAM wegen der verfügbaren Rechenleistung nicht durchgeführt werden, weshalb kTAM als weniger optimales Simulationsmodell verwendet wurde.

In diesem Kapitel wurde deutlich, dass die Simulationsergebnisse für die meisten der vorgestellten Mechanismen überzeugende Ergebnisse erzielt haben. Lediglich die beiden Mechanismen zur Fehlerbehandlung, nämlich Prüfsummen und Proofreading, haben sich als besonders anspruchsvoll erwiesen. Besonders die Kombination mehrerer Mechanismen mit Snaked-Proofreading resultierte in deutlich erhöhten Laufzeiten. Dennoch demonstrieren die Ergebnisse dieses Kapitels, dass alle vorgestellten Mechanismen, trotz des gelegentlichen hohen Aufwands, erfolgreich durch Self-Assembly implementiert werden können.

# **7. Zusammenfassung und Aussichten**

Da alle Themenblöcke der Arbeit somit abgearbeitet wurden, kann im Folgenden ein finaler Ausblick in die Zukunft dieses Themengebiets gegeben werden. Auch wird die gesamte Arbeit abschließend zusammengefasst.

## **7.1. Aussicht für die Zukunft**

Das in dieser Arbeit vorgestellte und behandelte Themengebiet steckt trotz jahrelanger Forschung immer noch in den Kinderschuhen. Die DNA-Tile-basierte Self-Assembly wird fast ausschließlich auf dem Papier und in der Theorie diskutiert und analysiert, da praktische Tests und Versuche aufwendig und teuer sind.

Trotzdem lässt sich in den präsentierten theoretischen Modellen das Potenzial dieser Technologie erkennen. DNA ermöglicht einen Bottom-Up Ansatz für Nanostrukturen. Durch Arbeiten wie diese wird klar, dass mit DNA-Tiles nicht nur komplexe Berechnungen möglich sind, sondern auch komplexere Kommunikation. Die Vorstellung, auf diese Weise Nanonetzwerke zu implementieren, wird immer greifbarer. Besonders für den Bereich der Medizin ist dies von großem Interesse. Ein In-Body-Kommunikationssystem, das Krankheiten frühzeitig identifizieren und lokal behandeln kann, rückt durch kontinuierliche und intensive Forschung im Bereich der DNA-Tiles immer näher.

Die Medizin ist nur eines von vielen potenziellen Anwendungsfeldern. Auf Nanoebene gibt es einige vielversprechende Möglichkeiten. Mit weiterer Forschung könnten Nanonetzwerke in verschiedenen Bereichen wertvolle Beiträge leisten.

Wenn auf dieser Arbeit aufbauend weiter geforscht werden soll, dann wäre zum Beispiel eine Möglichkeit, die Simulationsergebnisse durch Simulationen in kTHAM zu erweitern. Dies war aus Gründen der Rechenleistung in dieser Arbeit nicht möglich. Ein Vergleich der Ergebnisse zwischen kTAM und kTHAM wäre jedoch interessant und könnte bessere Einblicke in die Umsetzbarkeit der Mechanismen bieten.

Auch wurden einige Konzepte offengelassen. Der Dialogaufbau, das Routing und Framing wurden beispielsweise in dieser Arbeit als unnötig oder inhärent in DNA-Tile-basierter Self-Assembly beschrieben. Ist ein Mechanismus dieser Art jedoch für ausgewählte Anwendung notwendig, könnten weitere Implementierungen vorgestellt werden, die noch spezifischere Aspekte von Kommunikationsmechanismen in Netzwerken umsetzen.

In dieser Arbeit wurde oft über den medizinischen Kontext gesprochen. Es wurde oft-

mals angenommen, dass so ein System im Blutkreislauf eines Menschen implementiert werden soll. Doch die Simulationen und die Modelle basieren alle auf rein mathematischer Ebene und betrachten nicht den Unterschied zwischen in-vivo und in-vitro. Ein darauf fokussiertes Modell könnte weitere Informationen für die in dieser Arbeit entwickelten Mechanismen bringen.

Auch wurde in den Grundlagen angedeutet, dass einige Mechanismen sinnvoller auf Mikroebene durchgeführt werden könnten. Als Beispiel wurde ein Body-Area-Netzwerk genannt, das die zentrale Steuerung des Systems übernimmt. Für diese Arbeit wurde immer angenommen, dass Mechanismen existieren, die DNA-Tiles bilden und kontrolliert freilassen. Weitere Forschung könnte sich mit solchen Systemen befassen.

### 7.2. Zusammenfassung

Diese Arbeit hat sich tiefgründig mit DNA-Tile-basierten Nanonetzwerken durch Tiles und deren Self-Assembly befasst. Nach der Einleitung wurden dafür die Grundlagen angefangen bei der Desoxyribonukleinsäure, über die Tilebildung, bis hin zu Self-Assembly und den Tile-Assembly Modellen dargelegt. Auch setzen sich die Grundlagen in der Arbeit mit Mechanismen auseinander, die in herkömmlichen Netzwerken durch Protokolle definiert werden. Zusätzlich zu den Grundlagen stellt diese Arbeit drei verwandte Publikationen vor, die genauer vorgestellt wurden. Jedoch gibt es keine Arbeiten, die sich mit dem konkreten Problem befassen, das in dieser Arbeit behandelt wird. Übersetzung von Kommunikationsprotokollen und ihren Mechanismen auf Nanoebene wird in den meisten wissenschaftlichen Publikationen in anderen Nanosystemen durchgeführt.

In dieser Arbeit wurde ein umfassendes Konzept für eine Vielzahl von Mechanismen vorgestellt, darunter Adressierung, Fehlererkennung, Fehlerkorrektur, Framing, Datenflusskontrolle, Nachrichtencodierung und Flags. Zu jedem dieser Themen wurden sowohl die Herausforderungen als auch die Implementierungsansätze beschrieben. Nach Möglichkeit wurden diese Mechanismen in einem Python-Skript umgesetzt, um die in der Simulationsumgebung NetTAS erstellten Tilesets zu erweitern.

Im Rahmen dieser Arbeit wurden die vorgestellten Mechanismen mithilfe der Simulationsumgebung NetTAS untersucht und evaluiert. Die Ergebnisse zeigen, dass bei einigen Mechanismen der Self-Assembly-Prozess durch den jeweiligen Ansatz deutlich verlängert wird. Dennoch können einige dieser Mechanismen in bestimmten Anwendungen essenziell sein. Ein Limitierungsfaktor dieser Untersuchung war, dass aufgrund der zur Verfügung stehenden Rechenkapazitäten das komplexere und realitätsnähere Simulationsmodell kT-HAM nicht eingesetzt werden konnte. Stattdessen basieren alle Simulationsergebnisse ausschließlich auf dem Tile-Assembly-Modell kTAM. Empfehlungen für zukünftige Forschungsansätze im Kontext dieser Limitation und darüber hinaus finden sich im Ausblick dieses Kapitels.



# Abbildungsverzeichnis

1.1.	Nanotechnologien Unterscheidungen . . . . .	2
1.2.	In-Body-Netzwerke . . . . .	3
2.1.	DNA Modell . . . . .	6
2.2.	DNA Origami . . . . .	7
2.3.	Wang Tiles . . . . .	8
2.4.	DX- und TX-Tiles . . . . .	9
2.5.	Holliday Junction . . . . .	10
2.6.	Nanostrukturen Venn-Diagramm . . . . .	12
2.7.	Mathematisches Schema eines DNA-Tiles . . . . .	14
2.8.	Biologisches Schema einer Holliday Junction . . . . .	15
2.9.	Beispiel Assembly . . . . .	16
2.10.	Tile Ersetzung durch Proofreading Beispiel . . . . .	21
2.11.	Nanonetzwerk Ablauf Beispiel . . . . .	23
2.12.	Binärer Äquivalenzvergleich auf Tile Ebene Beispiel . . . . .	26
2.13.	OSI-Modell . . . . .	28
3.1.	Grafische Darstellung eines gatewayorientierten Hop-Counts . . . . .	32
3.2.	CORONA Ankerpunkte Verbildlichung . . . . .	34
3.3.	DEROUS Protokoll Szenarien . . . . .	36
4.1.	Konzeptionelle Darstellung offener DNA-Stränge . . . . .	40
4.2.	Framing Problem Beispiel . . . . .	42
4.3.	Skizzierter Ansatz zur Nachrichtenverkleinerung . . . . .	43
4.4.	Checksum Beispiel . . . . .	45
4.5.	Einweg Kommunikation . . . . .	48
4.6.	Mehrweg Kommunikation . . . . .	49
4.7.	Acknowledgements Tiles und Moleküle . . . . .	50
4.8.	Dreistellige Hexadezimalzahl in den Kleberbezeichnern codiert . . . . .	54
4.9.	Dreistellige Hexadezimalzahl in den Tile Bezeichnern codiert . . . . .	55
4.10.	Beispiel zur Vergrößerung einer Assembly . . . . .	58
4.11.	Flags Beispiel . . . . .	60
4.12.	Priorität Beispiel . . . . .	62
4.13.	Konzept Zusammenfassung Beispiel Molekül . . . . .	64
5.1.	Positive Farbkodierungsbeispiele . . . . .	71
5.2.	GUI Screenshot . . . . .	74

5.3.	Negativbeispiele für die Konstruktion durch das Skript . . . . .	77
5.4.	Empfohlene Molekülformen . . . . .	79
5.5.	Kleberverhalten beim Hinzufügen von Flag- und Prioritätstiles . . . . .	83
5.6.	Wachstumrichtung für Tiles mit Snaked-Proofreading je nach Farbcode . .	84
5.7.	Snaked-Proofreading Problem für Moleküle der Höhe drei. . . . .	86
6.1.	Graphen zur Relation von Gewichtungen zur Nachrichtenmenge . . . . .	90
6.2.	Grafische Darstellung der Assemblies für die Evaluation . . . . .	96
6.3.	Acknowledgement Assemblies . . . . .	98
6.4.	Simulationsergebnisse für Acknowledgement Assemblies . . . . .	98
6.5.	Simulationsergebnisse für Prioritätsassemblies . . . . .	100
6.6.	Simulationsergebnisse für Flagassemblies . . . . .	102
6.7.	Wachstum des Tilesets in Prüfsummen-Assemblies . . . . .	103
6.8.	Simulationsergebnisse für Prüfsummen in Assemblies . . . . .	105
6.9.	Errors im H-3-norm Tileset . . . . .	108
6.10.	Simulationsergebnisse für Proofreading für fehleranfällige Assembly . . .	110
6.11.	Simulationsergebnisse für Proofreading-Assemblies . . . . .	112
6.12.	Simulationsergebnisse für Proofreading-Assemblies von Acknowledgements.	113
6.13.	Simulationsergebnisse für Proofreading-Assemblies mit Flags. . . . .	115
6.14.	Simulationsergebnisse für Proofreading-Assemblies mit Prioritätsleveln. .	116
6.15.	Simulationsergebnisse für Proofreading-Assemblies mit Prüfsummen. . .	117

# Tabellenverzeichnis

2.1. Self-Assembly Errors . . . . .	20
2.2. formale Problemdefinitionen . . . . .	25
4.1. Prioritäten Beispiel . . . . .	51
5.1. Farbschema der Tilesets . . . . .	73
6.1. Simulationsergebnisse für verschiedene Gewichtungsverhältnisse . . . . .	92
6.2. Automatisch generierte Tilesets für Simulation und Analyse . . . . .	93
6.3. In NetTAS erstellte Tilesets für Simulation und Analyse . . . . .	93
6.4. 2HAM Ergebnisse . . . . .	94
6.5. Tileset- und Assemblygrößen für die generierten Tilesets . . . . .	104
6.6. Tileset- und Assemblygrößen für das H-3-norm Tileset . . . . .	109



# Listings

4.1.	Pseudocode für eine Funktion zur Definition der besten Basis. . . . .	56
5.1.	Darstellung des Tilesets in einer JSON-Datei . . . . .	68
5.2.	Pseudocode für die Generierung eines Tilesets . . . . .	69
5.3.	Pseudocode für die Generierung eines Tilesets mit Prüfsumme . . . . .	70
5.4.	Pseudocode der Main-Funktion des Skripts . . . . .	80
5.5.	Codeausschnitt der <code>generate_tile</code> -Funktion . . . . .	81
5.6.	Main-Funktion Rückgabe . . . . .	82



# Literaturverzeichnis

- [1] R. H. Abdel Hady, H. Z. Thabet, N. E. Ebrahem und H. A. Yassa. 2021. Thermal effects on dna degradation in blood and seminal stains: forensic view. *Acad Forensic Pathol*, 11, 1, (März 2021), 7–23. Epub 2021 Mar 19; PMID: 34040682; PMCID: PMC8129487. DOI: 10.1177/19253621211998547.
- [2] Ian Akyildiz, Josep Jornet und Massimiliano Pierobon. 2011. Nanonetworks: a new frontier in communications. *Communications of the ACM*, 54, (November 2011), 84–89. DOI: 10.1145/2018396.2018417.
- [3] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts und Peter Walter. 2015. *Molecular Biology of the Cell*. (6th Auflage). Garland Science.
- [4] Apostolos Almpanis, Christophe Corre und Adam Noel. 2019. Agent based modeling of the rhizobiome with molecular communication and game theory. In *Proceedings of the Sixth Annual ACM International Conference on Nanoscale Computing and Communication (NANOCOM '19)* Article 20. Association for Computing Machinery, Dublin, Ireland, 7 pages. ISBN: 9781450368971. DOI: 10.1145/3345312.3345476. <https://doi.org/10.1145/3345312.3345476>.
- [5] Monique A. Axelos und Marcel H. Van de Voorde, Herausgeber. 2017. *Nanotechnology in Agriculture and Food Science*. First published: 25 March 2017. Wiley-VCH Verlag GmbH & Co. KGaA. ISBN: 9783527339891. DOI: 10.1002/9783527697724.
- [6] Jennifer Birch. 2012. Worldwide prevalence of red-green color deficiency. *Journal of the Optical Society of America. A, Optics, image science, and vision*, 29, (März 2012), 313–20. DOI: 10.1364/JOSAA.29.000313.
- [7] Dr. Stefan Braun. 2018. *Handbuch Corporate Design - Universität Lübeck*. Gestaltung: Dipl.-Designer Uli Schmidts M.Sc. | Büro für Gestaltung. Zugriff: 07.10.23. Stabsstelle Kommunikation, Bereich Marketing, Universität Lübeck. (Juli 2018). [https://www.uni-luebeck.de/fileadmin/uzl\\_kommunikation/corporatedesign/1541077714unihl-ci-handbuch-v2018.pdf](https://www.uni-luebeck.de/fileadmin/uzl_kommunikation/corporatedesign/1541077714unihl-ci-handbuch-v2018.pdf).
- [8] Florian Büther, Florian Lau, Marc Stelzner und Sebastian Ebers. 2017. A formal definition for nanorobots and nanonetworks. In *The 17th International Conference on Next Generation Wired/Wireless Advanced Networks and Systems + The 10th conference on Internet of Things and Smart Spaces (NEW2AN ruSMART 2017)*. Springer, St.Petersburg, Russia, (September 2017). DOI: 10.1007/978-3-319-67380-6\_20. [https://dx.doi.org/10.1007/978-3-319-67380-6\\_20](https://dx.doi.org/10.1007/978-3-319-67380-6_20).

- [9] Florian Büther, Immo Traupe und Sebastian Ebers. 2018. Hop count routing: a routing algorithm for resource constrained, identity-free medical nanonetworks. *NANOCOM '18 Proceedings of the 5th ACM International Conference on Nanoscale Computing and Communication*, (September 2018). DOI: 10.1145/3223795.3223821. <https://doi.org/10.1145/3223795.3223821>.
- [10] Ho-Lin Chen und Ashish Goel. 2005. Error free self-assembly using error prone tiles. In *DNA Computing*. Claudio Ferretti, Giancarlo Mauri und Claudio Zandron, Herausgeber. Springer Berlin Heidelberg, Berlin, Heidelberg, 62–75. ISBN: 978-3-540-31844-6.
- [11] SM Douglas, I Bachelet und GM Church. 2012. A logic-gated nanorobot for targeted transport of molecular payloads. *Science*, 335, 6070, (Februar 2012), 831–834. DOI: 10.1126/science.1214081.
- [12] Falko Dressler und Stefan Fischer. 2015. Connecting in-body nano communication with body area networks: challenges and opportunities of the internet of nano things. *Nano Communication Networks*, 6, 2, 29–38. Pervasive and Ubiquitous Environment Interactions with Nano Things. ISSN: 1878-7789. DOI: <https://doi.org/10.1016/j.nancom.2015.01.006>. <https://www.sciencedirect.com/science/article/pii/S1878778915000071>.
- [13] Brandt F. Eichman, Jeffrey M. Vargason, Blaine H. M. Mooers und P. Shing Ho. 2000. The holliday junction in an inverted repeat dna sequence: sequence effects on the structure of four-way junctions. *Proceedings of the National Academy of Sciences*, 97, 8, 3971–3976. DOI: 10.1073/pnas.97.8.3971. eprint: <https://www.pnas.org/doi/pdf/10.1073/pnas.97.8.3971>. <https://www.pnas.org/doi/abs/10.1073/pnas.97.8.3971>.
- [14] Nariman Farsad, H. Birkan Yilmaz, Andrew Eckford, Chan-Byoung Chae und Weisi Guo. 2016. A comprehensive survey of recent advancements in molecular communication. *IEEE Communications Surveys & Tutorials*, 18, 3, 1887–1919. DOI: 10.1109/COMST.2016.2527741.
- [15] International Organization for Standardization (ISO). 1984. Information processing systems - open systems interconnection - basic reference model. 7498.
- [16] Tsu Ju Fu und Nadrian C. Seeman. 1993. Dna double-crossover molecules. *Biochemistry*, 32, 13, 3211–3220. PMID: 8461289. DOI: 10.1021/bi00064a003. eprint: <https://doi.org/10.1021/bi00064a003>. <https://doi.org/10.1021/bi00064a003>.
- [17] Zhen Gu, Alex A. Aimetti, Qun Wang, Tram T. Dang, Yunlong Zhang, Omid Veiseh, Hao Cheng, Robert S. Langer und Daniel G. Anderson. 2013. Injectable nano-network for glucose-mediated insulin delivery. *ACS Nano*, 7, 5, 4194–4201. PMID: 23638642. DOI: 10.1021/nn400630x. eprint: <https://doi.org/10.1021/nn400630x>. <https://doi.org/10.1021/nn400630x>.
- [18] Robin Holliday. 1974. Molecular aspects of genetic exchange and gene conversion. *Genetics*, 78, 1, (September 1974), 273–287. DOI: 10.1093/genetics/78.1.273.

- [19] Liuyi Jin, Lihua Zuo, Zhipei Yan und Radu Stoleru. 2019. Nanocommunication-based impermeable region mapping for oil reservoir exploration. In *Proceedings of the Sixth Annual ACM International Conference on Nanoscale Computing and Communication* (NANOCOM '19) Article 31. Association for Computing Machinery, Dublin, Ireland, 7 pages. ISBN: 9781450368971. DOI: 10.1145/3345312.3345487. <https://doi.org/10.1145/3345312.3345487>.
- [20] Neville R. Kallenbach, Rong-Ine Ma und Nadrian C. Seeman. 1983. An immobile nucleic acid junction constructed from oligonucleotides. *Nature*, 305, 5937, (Oktober 1983), 829–831. ISSN: 1476-4687. DOI: 10.1038/305829a0. <https://doi.org/10.1038/305829a0>.
- [21] Max Kaussow. 2022. *Modeling and simulation of dna-based nanonetworks*. master thesis. Universität zu Lübeck.
- [22] Yonggang Ke, Shawn M. Douglas, Minghui Liu, Jaswinder Sharma, Anchi Cheng, Albert Leung, Yan Liu, William M. Shih und Hao Yan. 2009. Multilayer dna origami packed on a square lattice. *Journal of the American Chemical Society*, 131, 43, 15903–15908. PMID: 19807088. DOI: 10.1021/ja906381y. eprint: <https://doi.org/10.1021/ja906381y>. <https://doi.org/10.1021/ja906381y>.
- [23] 2000. *Experimental progress in computation by self-assembly of dna tilings*. (November 2000), 123–140. ISBN: 9780821820537. DOI: 10.1090/dimacs/054/11.
- [24] Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system.
- [25] James I. Lathrop, Jack H. Lutz und Scott M. Summers. 2009. Strict self-assembly of discrete sierpinski triangles. Zugriffen am 30. April 2023. (2009). DOI: 10.48550/arXiv.0903.1818. arXiv: 0903.1818 [cs.DM]. <https://arxiv.org/abs/0903.1818>.
- [26] Florian-Lennert Adrian Lau. 2020. *DNA-basierte Nanonetze*. PhD Thesis. Universität zu Lübeck.
- [27] Florian-Lennert Adrian Lau, Florian Büther, Regine Geyer und Stefan Fischer. 2019. Computation of decision problems within messages in dna-tile-based molecular nanonetworks. *Nano Communication Networks*, 21, 100245. ISSN: 1878-7789. DOI: <https://doi.org/10.1016/j.nancom.2019.05.002>. <https://www.sciencedirect.com/science/article/pii/S1878778919300018>.
- [28] Shang Li, Qiao Jiang, Shiyuan Liu und et al. 2018. A dna nanorobot functions as a cancer therapeutic in response to a molecular trigger in vivo. *Nature Biotechnology*, 36, 3, 258–264. Received 16 June 2016; Accepted 09 January 2018; Published 12 February 2018. DOI: 10.1038/nbt.4071.
- [29] Christos Liaskos, Ageliki Tsioliariidou, Andreas Pitsillides, Ian F. Akyildiz, Nikolaos V. Kantartzis, Antonios X. Lalas, Xenofontas Dimitropoulos, Sotiris Ioannidis, Maria Kafesaki und C.M. Soukoulis. 2015. Design and development of software defined metamaterials for nanonetworks. *IEEE Circuits and Systems Magazine*, 15, 4, 12–25. DOI: 10.1109/MCAS.2015.2484098.

- [30] Christos Liaskos und Angeliki Tsoliaridou. 2015. A promise of realizable, ultra-scalable communications at nano-scale:a multi-modal nano-machine architecture. *IEEE Transactions on Computers*, 64, 5, 1282–1295. DOI: 10.1109/TC.2014.2329684.
- [31] Christos Liaskos, Angeliki Tsoliaridou, Sotiris Ioannidis, Nikolaos Kantartzis und Andreas Pitsillides. 2016. A deployable routing system for nanonetworks. In *2016 IEEE International Conference on Communications (ICC)*, 1–6. DOI: 10.1109/ICC.2016.7511151.
- [32] Xiaojun Ma, Masoud Hashempour, Lei Wang und Fabrizio Lombardi. 2010. Manufacturing yield of qca circuits by synthesized dna self-assembled templates. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI (GLSVLSI '10)*. Association for Computing Machinery, Providence, Rhode Island, USA, 275–280. ISBN: 9781450300124. DOI: 10.1145/1785481.1785546. <https://doi.org/10.1145/1785481.1785546>.
- [33] Toshiaki Matsushima, Yoshihiro Mizoguchi und Alexandre Derouet-Jourdan. 2016. Verification of a brick wang tiling algorithm. In *SCSS 2016. 7th International Symposium on Symbolic Computation in Software Science* (EPiC Series in Computing). James H. Davenport und Fadoua Ghourabi, Herausgeber. Band 39. EasyChair, 107–116. DOI: 10.29007/2m5f. <https://easychair.org/publications/paper/4mz>.
- [34] Quentin Mauvisseau, Lynsey R. Harper, Michael Sander, Robert H. Hanner, Hannah Kleyer und Kristy Deiner. 2022. The multiple states of environmental dna and what is known about their persistence in aquatic environments. *Environmental Science & Technology*, 56, 9, 5322–5333. PMID: 35435663. DOI: 10.1021/acs.est.1c07638. eprint: <https://doi.org/10.1021/acs.est.1c07638>. <https://doi.org/10.1021/acs.est.1c07638>.
- [35] M. Meselson und F. W. Stahl. 1958. The replication of dna in escherichia coli. *Proceedings of the National Academy of Sciences*, 44, 7, 671–682. DOI: 10.1073/pnas.44.7.671.
- [36] C. A. Mirkin, R. L. Letsinger und R. C. Mucic. 1996. A dna-based method for rationally assembling nanoparticles into macroscopic materials. *Nature*, 382, 6592, 607–609.
- [37] Thomas Naef, Anne-Laure Besnard, Lisa Lehnens, Eric J. Petit, Jaap van Schaik und Sébastien J. Puechmaille. 2023. How to quantify factors degrading dna in the environment and predict degradation for effective sampling design. *Environmental DNA*, 5, 3, (Mai 2023), 403–416. DOI: 10.1002/edn3.414.
- [38] [n. d.] Osi-modell. [https://www.elektroniktutor.de/internet/net\\_pict/osi.png](https://www.elektroniktutor.de/internet/net_pict/osi.png). [Online; accessed 4-May-2023]. () .

- [39] Matthew J. Patitz. 2014. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13, 2, (Juni 2014), 195–224. Zugegriffen am 30. April 2023. ISSN: 1572-9796. DOI: 10 . 1007 / s11047 - 013 - 9379 - 4. <https://doi.org/10.1007/s11047-013-9379-4>.
- [40] G. S. Rahi, J. L. Adams, J. Yuan, D. J. Devone und K. M. Lodhi. 2021. Whole human blood dna degradation associated with artificial ultraviolet and solar radiations as a function of exposure time. *Forensic Sci Int*, 319, (Februar 2021), 110674. Epub 2020 Dec 24; PMID: 33422800. DOI: 10 . 1016 / j.forsciint.2020.110674.
- [41] young hoon Roh, Roanna Ruiz, Songming Peng, Jong Bum Lee und Dan Luo. 2011. Engineering dna-based functional materials. *Chemical Society reviews*, 40, (August 2011), 5730–44. DOI: 10 . 1039 / c1cs15162b.
- [42] Paul W. K. Rothemund. 2006. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440, 7082, 297–302. ISSN: 1476-4687. DOI: 10 . 1038 / nature04586. <https://doi.org/10.1038/nature04586>.
- [43] Nadrian C. Seeman. 1982. Nucleic acid junctions and lattices. *Journal of Theoretical Biology*, 99, 2, 237–247. ISSN: 0022-5193. DOI: [https://doi.org/10.1016/0022-5193\(82\)90002-9](https://doi.org/10.1016/0022-5193(82)90002-9). <https://www.sciencedirect.com/science/article/pii/0022519382900029>.
- [44] Sunil Singhal, Shuming Nie und May D. Wang. 2010. Nanotechnology applications in surgical oncology. *Annual Review of Medicine*, 61, 359–373. DOI: 10 . 1146 / annurev.med.60.052907.094936.
- [45] AbdelMajid Taibi, Antoine Durant, Valeria Loscri, Anna Maria Vegni und Luigi La Spada. 2019. Controlling light by curvilinear metasurfaces. In *Proceedings of the Sixth Annual ACM International Conference on Nanoscale Computing and Communication (NANOCOM '19)* Article 28. Association for Computing Machinery, Dublin, Ireland, 6 pages. ISBN: 9781450368971. DOI: 10 . 1145 / 3345312 . 3345484. <https://doi.org/10.1145/3345312.3345484>.
- [46] Andrew Tanenbaum. 2011. *Computer Networks*. Pearson Education.
- [47] Ageliki Tsioliaridou, Christos Liaskos, Sotiris Ioannidis und Andreas Pitsillides. 2015. Corona: a coordinate and routing system for nanonetworks. In (September 2015). DOI: 10 . 1145 / 2800795 . 2800809.
- [48] 1990. *Dominoes and the aea case of the decision problem. Computation, Logic, Philosophy: A Collection of Essays*. Springer Netherlands, Dordrecht, 218–245. ISBN: 978-94-009-2356-0. DOI: 10 . 1007 / 978 - 94 - 009 - 2356 - 0 \_ 11. [https://doi.org/10.1007/978-94-009-2356-0\\_11](https://doi.org/10.1007/978-94-009-2356-0_11).
- [49] Sijia Wang, Gereon Hüttmann, Zhenhua Zhang, Alfred Vogel, Reginald Birngruber, Shashi Tangutoori, Tayyaba Hasan und Ramtin Rahmazadeh. 2015. Light-controlled delivery of monoclonal antibodies for targeted photoinactivation of ki-67. *Molecular Pharmaceutics*, 12, 9, 3272–3281. Epub 2015 Aug 13. DOI: 10 . 1021 / acs . molpharmaceut . 5b00260.

- [50] J. D. Watson und F. H. C. Crick. 1953. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171, 4356, 737–738. DOI: 10.1038/171737a0.
- [51] Wikimedia Commons contributors. [n. d.] Dna simple. Zugriff am 28. April 2023. (). [https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/DNA\\_simple.svg/300px-DNA\\_simple.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/e/e7/DNA_simple.svg/300px-DNA_simple.svg.png).
- [52] Wikipedia. 2023. Desoxyribonukleinsäure. [Online; 29.09.2023]. (2023). <https://de.wikipedia.org/wiki/Desoxyribonukleins%C3%A4ure>.
- [53] Wikipedia. 2023. Kopfhaar. [Online; 29.09.2023]. (2023). <https://de.wikipedia.org/wiki/Kopfhaar>.
- [54] Erik Winfree und Renat Bekbolatov. 2004. Proofreading tile sets: error correction for algorithmic self-assembly. In *DNA Computing*. Junghuei Chen und John Reif, Herausgeber. Springer Berlin Heidelberg, Berlin, Heidelberg, 126–144. ISBN: 978-3-540-24628-2.
- [55] Cuiping Yao, Florian Rudnitzki, Gereon Hüttmann, Zhenxi Zhang und Ramtin Rahmanzadeh. 2017. Important factors for cell-membrane permeabilization by gold nanoparticles activated by nanosecond-laser irradiation. *International Journal of Nanomedicine*, 12, 5659–5672. Published online 2017 Aug 7. DOI: 10.2147/IJN.S140620.

## A. Zugriff auf den Code

Der für diese Arbeit entwickelte Code umfasst über 1600 Zeilen und ist aus Gründen der Übersichtlichkeit nicht direkt in dieser Arbeit aufgenommen. Das vollständige Code-Repository ist jedoch öffentlich auf GitHub zugänglich. Ein QR-Code für den direkten Zugriff ist unten abgebildet.

<https://github.com/Falkenheim/Tile-Generator>

