2012•2013
FACULTEIT WETENSCHAPPEN
master in de informatica: databases

Masterproef

Spontaan formerende structuren uit DNA tegels

Promotor :
Prof. dr. Marc GYSSENS

Niki Vandesbosch

Masterproef voorgedragen tot het bekomen van de graad van master in de informatica , afstudeerrichting databases

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

universiteit hasselt
KNOWLEDGE IN ACTION

Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt
Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek

universiteit hasselt | Maastricht University

2012•2013
FACULTEIT WETENSCHAPPEN
master in de informatica: databases

Masterproef

Spontaan formerende structuren uit DNA tegels

Promotor :
Prof. dr. Marc GYSSENS

Niki Vandesbosch

Masterproef voorgedragen tot het bekomen van de graad van master in de informatica , afstudeerrichting databases

universiteit hasselt | Maastricht University

Woord vooraf

Tijdens het tot stand brengen van mijn masterproef heb ik veel hulp en steun gekregen van verschillende mensen. Zo wil ik in de eerste plaats mijn promotor prof. dr. Marc Gyssens bedanken voor mij de kans te geven om een proefschrift rond DNA self assembly te laten schrijven. Verder wil ik mijn begeleider Robert Brijder bedanken voor de tijd die hij in onze bijeenkomsten heeft gestoken, het meermaals nalezen van mijn werk en de vele goede opmerkingen en ideeën die hij mij heeft gegeven.
Graag wil ik mijn vrienden en mijn medestudenten Bas Ketsman en Dimitri Surinx bedanken voor de discussies, hulp en steun die ze mij hebben geboden. Tot slot wil ik ook nog mijn ouders en familie bedanken voor hun steun en vertrouwen en dat zij mij de mogelijk gaven om aan de Universiteit Hasselt te mogen studeren en dit alles mogelijk te maken.

Abstract

Het doel van deze masterproef is om de lezer kennis te laten maken met DNA computing. Eerst wordt er nagegaan welke expressiviteit je kan bereiken met behulp van DNA. Vervolgens introduceren we DNA tiles om op die manier het gedrag van DNA op een meer abstracte manier te kunnen bestuderen. We zullen twee theoretische modellen beschouwen, namelijk aTam en kTam. Binnen in deze modellen kunnen er echter fouten optreden. We zullen deze types van errors bestuderen en we sluiten dit proefschrift af met enkele technieken, namelijk self-healing en proofreading, om zo enkele types van errors te verhelpen, of om de kans dat ze optreden te verlagen.

Inhoudsopgave

Inleiding

DNA computing is een onderzoeksgebied binnen de informatica en de biologie waarbij men gebruik maakt van DNA voor het uitvoeren van algoritmen. DNA is een molecule dat fungeert als de belangrijkste drager van erfelijke informatie in alle bekende organismen. Door zijn robuustheid en kleine afmetingen is DNA uiterst geschikt voor het bewaren van informatie, wat in principe ook zijn natuurlijke functie is.

Deze masterthesis handelt over DNA self assembly, een tak binnen DNA computing. Meer specifiek zullen we bespreken hoe je DNA tiles kan gebruiken om structuren te construeren of algoritmes te simuleren.

In het eerste hoofdstuk bespreek ik hoe je met behulp van enkele DNA structuren reguliere talen, context vrije talen en tot slot recursief opsombare talen kan simuleren.

Het tweede hoofdstuk behandeld de materie vanuit een meer abstract oogpunt. We stappen af van de concrete DNA structuren, zoals in Hoofdstuk 1, en maken vanaf nu gebruik van DNA tiles. We omschrijven twee theoretische modellen voor self assembly met behulp van DNA tiles, namelijk aTam en kTam. ATam is een deterministisch model dat interessant is vanuit een theoretisch oogpunt maar niet realistisch is naar de realiteit toe. Om ook een theoretisch model te hebben dat grotendeels wel overeenkomt met de realiteit introduceren we kTam. Op het einde van dit hoofdstuk bespreken we verschillende types van errors die kunnen optreden binnenin deze modellen.

Het derde hoofdstuk introduceren we self-healing tile sets. Dit zijn tile sets waarin geen backward growth meer kan optreden. We introduceren in totaal drie transformatie die respectievelijk een L-BCA tile sets, transformable tile sets en polyomino tile sets kunnen omvormen naar equivalente self-healing tile set.

In het vierde en laatste hoofdstuk introduceren we proofreading tile sets. Dit zijn tile sets waarin er meerdere fouten na elkaar moeten optreden, voordat er weer normale groei kan plaatsvinden. We bespreken eerst een eenvoudige proofreading tile set die growth error tegengaat. Vervolgens introduceren we een snaked proofreading tile set die beide growth errors en facet roughening errors tegengaat.

Hoofdstuk 1

Expressiviteit van DNA

In dit hoofdstuk wordt de expressiviteit van DNA modellen bestudeerd. Hiervoor introduceren we een formeel begrip, het DNA *complex*, samen met enkele operaties over dit begrip. In een theoretisch model van de wereld zullen we dan gebruik maken van DNA *complexen* om zo DNA self assembly toe te passen. Zo zal er worden geïllustreerd hoe je met behulp van DNA self assembly een willekeurige reguliere taal kan simuleren. Vervolgens wordt er op een analoge wijze aangetoond hoe je dit ook kan doen voor context-vrije talen. We sluiten dit hoofdstuk af door aan te tonen dat een willekeurige *Turing machine* kan simuleren met DNA self assembly. Hiervoor maken we gebruik van het gebrip BCA.

1.1 DNA Complex

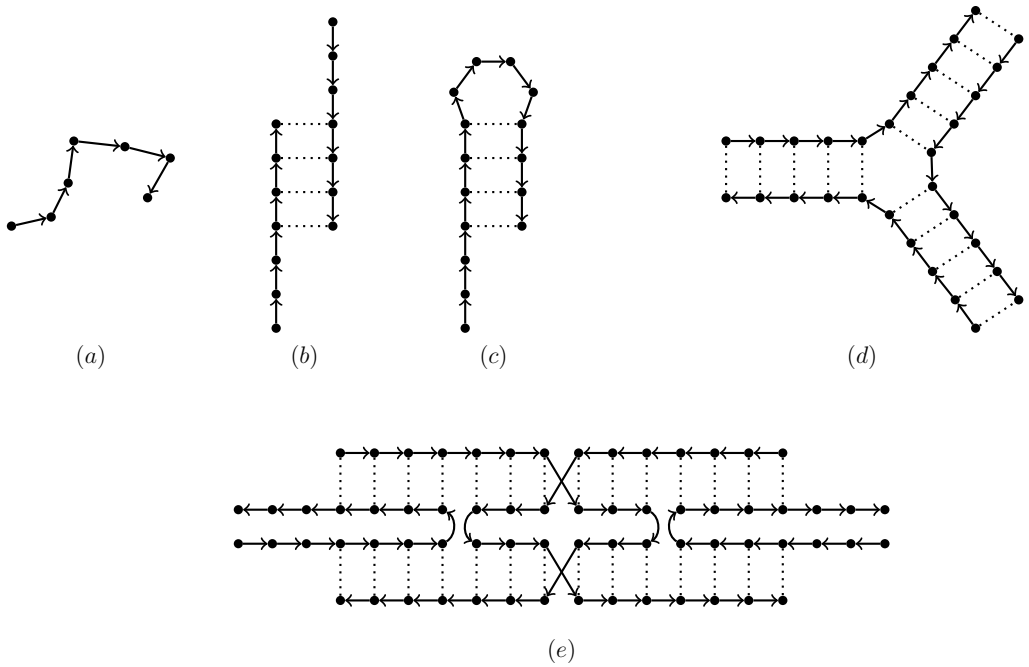
Voordat we uitleggen wat een DNA *complex* is, leggen we eerst kort uit hoe een DNA molecule is opgebouwd. Een *streng* is een reeks van nucleotiden die met elkaar verbonden zijn. Iedere streng heeft een uniek begin en een einde. Een DNA molecule is opgebouwd uit twee strengen die verbonden zijn met elkaar door zogenaamde *baseparen*. Deze baseparen verbinden dan twee nucleotiden, 1 uit iedere streng, met de nucleobasen adenine (A), thymine (T), guanine (G) en cytosine (C). Een belangrijke eigenschap van deze nucleotiden is dat adenine enkel een binding met thymine kan aangaan, en dat guanine enkel een binding met cytosine kan aangaan.

Een DNA *complex* is in essentie een theoretische voorstelling van een DNA molecule, zie Definitie 1.1 [6]. Meer concreet maken we gebruik van een graafstructuur, waarbij de nucleotiden worden gerepresenteerd door de knopen van de graaf, en de baseparen worden gerepresenteerd door de bogen. De baseparen verbinden echter enkel nucleotiden van twee verschillende strengen. Om de nucleotiden van éénzelfde streng met elkaar te verbinden maken we ook gebruik van bogen, maar deze bogen zullen gelabeld worden als backbone edge's. Omdat de strengen in een DNA molecule een richting hebben, maken we gebruik van gerichte bogen. In Figuur 1.1 vind je enkele voorbeelden terug. Deze voorbeelden zijn niet willekeurig uitgekozen: ze zullen als bouwstenen worden gebruikt tijdens de simulatie van o.a. reguliere talen.

Definitie 1.1: *Een* DNA ***complex*** *is een geconnecteerde gelabelde gerichte graaf* $G = (V, E, l_v, l_e)$ *waarbij:*

- V *een eindige verzameling knopen is,*
- $E \subseteq V \times V$ *een eindige verzameling bogen is,*
- $l_v : V \to L_v$*, met* $L_v = \{\text{A,C,G,T}\}$*, is een labelings functie over de knopen,*
- $l_e : E \to L_e$*, met* $L_e = \{backbone, basepair\}$*, is een labelings functie over de bogen,*
- *Iedere knoop heeft maximaal 1 uitgaande boog met een hetzelfde label,*
- *Iedere knoop heeft maximaal 1 binnenkomende boog met een hetzelfde label,*
- *Voor iedere basepair edge* $x \to y$ *bestaat er een basepair edge* $y \to x$*, met* $x, y \in V$*,*
- *Alle basepair edge's zijn Watson-Crick complementair.*

We introduceren nu enkele operaties die kunnen worden toegepast op een DNA *complex*.



Figuur 1.1: Dit zijn 5 verschillende DNA *complexen. Figuur (a) noemen we een* ***strand****, Figuur (b) een* ***Duplex****, Figuur (c) een* ***Hairpin****, Figuur (d) een* ***3-armed branched junction*** *en Figuur (e) een* ***Double-crossover****. De labels op de knopen zijn weggelaten.*

Ligatie

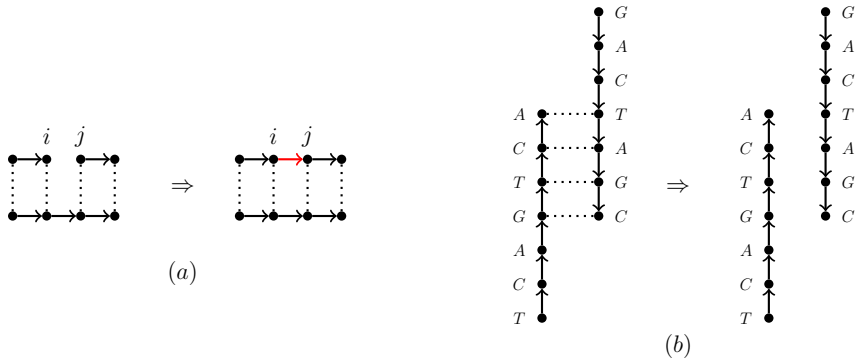
Ligatie, $C' = ligate(C)$ genoteerd, is een operatie die backbone edge's gaat toevoegen voor ieder paar knopen (i, j) in een DNA *complex* dat voldoet aan volgende condities:

- Er bestaat nog geen backbone edge tussen de twee knopen i en j,
- Er bestaat een backbone edge tussen de knopen i' en j', waarbij i' en j' respectievelijk incident zijn aan i en j door middel van basepair edge's;

In Figuur 1.2 (a) wordt de werking van ligatie geïllustreerd.

Denaturatie

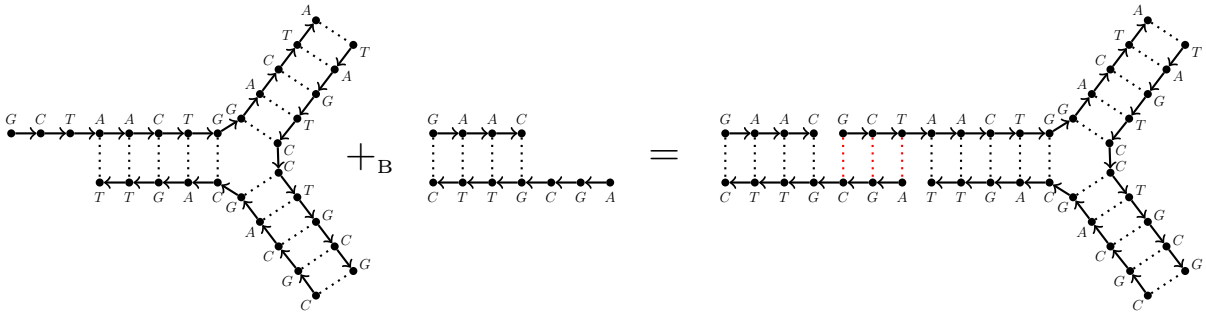
Denaturatie, $\{C_i\} = denature(C)$ genoteerd, gaat in tegenstelling tot ligatie bogen wegnemen uit het DNA *complex*. Denaturatie verwijdert namelijk alle basepair edge's uit het DNA *complex*, wat een verzameling van strand's opleverd. Deze operatie wordt geïllustreerd in Figuur 1.2 (b).



Figuur 1.2: In Figuur (a) wordt ligatie geïllustreerd, in Figuur (b) wordt denaturatie geïllustreerd.

Hybridizatie

Hybridizatie van twee DNA *complexen*, $C_1 +_{\text{B}} C_2 = C_3$ genoteerd, zal deze twee complexen samenvoegen tot één complex indien beide complexen één of meerdere sticky ends hebben die volledig of gedeeltelijk *matchen*. Bij matchen wordt bedoeld dat de labels van knopen van het ene sticky end Watson-Crick complementair moet zijn aan de labels van knopen van het andere sticky end. Hybridizatie zal dan voor alle sticky ends die (gedeeltelijk) matchen basepair edge's toevoegen tussen de knopen van de sticky ends de twee complexen. Zie Figuur 1.3 voor een voorbeeld.
Wanneer ligatie wordt toegepast op het resultaat van hybridizatie, zal dit enkel resulteren in het toevoegen van basepair edge's indien de sticky ends volledig matchen. Indien ze maar gedeeltelijk matchen zal ligatie geen veranderingen toebrengen.



Figuur 1.3: Dit is de hybridizatie-operatie toegepast op twee DNA *complexen om zo een nieuw* DNA *complex te construeren.*

1.1.1 Theoretisch model van de 'wereld'

We hebben zonet het theoretische model van een DNA molecule besproken. Nu gaan we bespreken welke eigenschappen en condities er allemaal gelden in het theoretische model waarin we DNA self assembly willen toepassen.

1. **Constante temperatuur:** gedurende de volledige simulatie zal de temperatuur hetzelfde blijven. Dit heeft als gevolg dat er steeds hetzelfde minimaal aantal basepair edge's nodig is om een stabiele verbinding tussen twee DNA *complexen* te verkrijgen.

2. **Perfect Watson-Crick complementair:** er zal enkel hybridizatie optreden tussen twee DNA *complexen* indien de sticky ends volledig matchen.

3. **Geen intramoleculaire gebeurtenissen:** een DNA *complex* mag niet met zichzelf een operatie ondergaan.

Merk op dat het toegestaan is dat er aan de start van de simulatie reeds een initiëel DNA *complex* aanwezig is. Bovendien mogen meerdere sticky ends van een DNA *complex* verbonden worden met andere sticky ends binnen éénzelfde tijdseenheid, op voorwaarde dat deze sticky ends volledig matchen.

1.1.2 Taal van DNA *complexen*

Om een taal die uitdrukbaar is met behulp van DNA *complexen* te beschrijven hebben we een alfabet en een lijst van regels over dat alfabet nodig. Het alfabet bestaat uit een eindige verzameling van DNA *complexen*. De lijst van regels bepaald welke DNA *complexen* hybridizatie met elkaar mogen aangaan en welke niet. In Definitie 1.2 vindt je de formele definitie van de taal van DNA *complexen*.

Definitie 1.2: *Gegeven een verzameling van* DNA *complexen A, dan construeren we een taal van* DNA *complexen, $\mathcal{L}_A$ genoteerd, die voldoet aan volgende eigenschappen:*

- *A is een eindige verzameling van* DNA *complexen en stelt het alfabet van onze taal voor,*
- *Zij* $\hat{\mathcal{L}}_A$ *de transitieve sluiting over alle toegestane hybridizaties, dan geldt:*
  - $A \subset \hat{\mathcal{L}}_A$,
  - $C_1 \in \hat{\mathcal{L}}_A \wedge C_2 \in \hat{\mathcal{L}}_A \implies C_3 \in \hat{\mathcal{L}}_A$
  - *Geen andere* DNA *complexen behoren tot* $\hat{\mathcal{L}}_A$

We zouden de labels op de knopen van de DNA *complexen* kunnen gebruiken als alfabet van onze taal. Dit geeft ons echter een beperking van vier verschillende symbolen. Om dit probleem te overkomen, en dus willekeurige string te gebruiken in onze taal, introduceren we een *codebook* $\mathcal{C}$. Dit codebook zal een string uit een willekeurig alfabet σ vertalen naar een unieke string over het alfabet $\{A, T, C, G\}$. We noteren dit als $\mathcal{L}_{A,\mathcal{C}}$.

1.2 Reguliere Talen

We nemen aan dat de lezer gekend is met de basisbegrippen van formele talen, zie Sipser[8]. Er worden verschillende klassen van formele talen bekeken, gebaseerd op hun uitdrukkingskracht. De klasse van reguliere talen omvat alle talen die je kan beschrijven met behulp van een deterministische eindige automaat. Kenmerkend aan deze talen is dat ze een relatief eenvoudige structuur hebben, waardoor een computer bijvoorbeeld snel en eenvoudig kan nagaan of een bepaald woord tot een bepaalde reguliere taal behoort.

Er zijn verschillende mogelijkheden om een reguliere taal te beschrijven. Een deterministische eindige automaat of een reguliere expressie zijn hier voorbeelden van. Maar om de vertaling van reguliere talen naar *de taal van* DNA *complexen* makkelijker, en vooral intuïtiever, te maken, kiezen we ervoor om reguliere talen voor de stellen door een rechts-reguliere grammatica, zie Definitie 1.3 en Voorbeeld 1.2.1.

Definitie 1.3: *Een* ***rechts-reguliere grammatica*** *is een viertuppel* (N, Σ, P, S)*, waarbij:*

- N *een eindige verzameling van* ***niet-terminale symbolen****;*
- Σ *een eindige verzameling van* ***terminale symbolen****, disjunct met N;*
- P *een eindige verzameling van* ***productieregels****, elk van een van de volgende vormen:*
  - $A \to a$*, met* $A \in N$ *en* $a \in \Sigma$*;*
  - $A \to aB$*, met* $A, B \in N$ *en* $a \in \Sigma$*;*
  - $A \to \epsilon$*, met* $A \in N$ *en* ϵ *stelt het lege symbool voor.*
- $S \in N$ *het* ***startsymbool****.*

Voorbeeld 1.2.1: Veronderstel dat we over de reguliere taal L beschikken die alle woorden bevat die beginnen bij een willekeurig aantal a's, gevolgd door precies één b en eindigt op een willekeurig aantal c's. Dan kunnen we de reguliere grammatica $G = (N, \Sigma, P, S)$, die de reguliere taal L uitdrukt, beschrijven als volgt:

- $N = \{S, A, B, C\}$
- $\Sigma = \{a, b, c\}$
- $P = \{$

  $S \to aA,$
  $S \to bB,$
  $A \to aA,$
  $A \to bB,$
  $B \to cC,$
  $B \to \epsilon,$
  $C \to cC,$
  $C \to \epsilon\ \}$

♣

Zoals gebruikelijk gebruiken we een reguliere grammatica om een reguliere taal te beschrijven door ieder woord over die taal te genereren op de volgende manier:

1. Schrijf het startsymbool neer
2. Ga nu op zoek naar niet-terminaal symbool dat je eerder had neergeschreven. Ga vervolgens op zoek naar een productieregel waarvan de linkerkant overeenkomt met het zonet gevonden symbool. Vervang nu dat symbool door de volledige rechterkant van die productieregel.
3. Herhaal stap 2 zolang er nog niet-terminale symbolen overblijven.

We noemen bovenstaande sequentie van substituties van productieregels, om zo een woord te genereren, een afleiding over een reguliere grammatica, zie Voorbeeld 1.2.2.

Voorbeeld 1.2.2: We hernemen de reguliere grammatica β uit Voorbeeld 1.2.1, die de reguliere taal α uitdrukt. De afleiding die het woord $w = aabcc$ hoort is de volgende:

$$S \Rightarrow aA \Rightarrow aaA \Rightarrow aabB \Rightarrow aabcC \Rightarrow aabcc$$

De afleiding van het woord $v = b$ ziet er als volgt uit:

$$S \Rightarrow bB \Rightarrow b\epsilon = b$$

Aan deze voorbeelden kan je duidelijk zien dat de nieuwe symbolen steeds rechts worden toegevoegd aan het tussenresultaat. Dit volgt rechtstreeks uit het feit dat de niet-terminale symbolen ook steeds uiterst rechts worden toegevoegd. ♣

Eigenschap 1.2.1

Zij $G = (N, \Sigma, P, S)$ een rechts-reguliere grammatica. Dan kan er tijdens iedere tussenstap in een afleiding over G maximaal 1 niet-terminaal symbool aanwezig zijn. Bovendien zal dit niet-terminale symbool, als het aanwezig is, steeds uiterst rechts voorkomen.

Bewijs: Om te bewijzen dat Eigenschap 1.2.1 correct is, veronderstellen we dat we over een rechts-reguliere grammatica $G = (N, \Sigma, P, S)$ beschikken. We gaan nu via inductie aantonen dat er tijdens iedere tussenstap in een afleiding maximaal 1 niet-terminaal symbool aanwezig kan zijn.

Basisstap: Een afleiding begint altijd door het startsymbool neer te schrijven. Hierdoor zal er na één stap altijd precies één niet-terminaal symbool aanwezig zijn.

Inductiestap: We veronderstellen nu dat deze stelling nog steeds geldt na n stappen. Dus na n keer een substitutie te hebben toegepast is er nog steeds maximaal één niet-terminaal symbool aanwezig.

Inductiehypothese: Uit onze inductiestap weten we dat het maximaal aantal niet-terminale symbolen, na n stappen, nog steeds 1 bedraagt. We moeten nu controleren of het mogelijk is dat de substitutie van het rechterdeel een productieregel ervoor kan zorgen dat het maximaal aantal niet-terminale symbolen groter wordt dan 1. Uit Definitie 1.3 kunnen we afleiden dat iedere productieregel altijd één van onderstaande vormen zal aannemen:

- $A \to a$, met $A \in N$ en $a \in \Sigma$;
- $A \to aB$, met $\{A, B\} \in N$ en $a \in \Sigma$;
- $A \to \epsilon$, met $A \in N$ en ϵ stelt het lege symbool voor.

Het is eenvoudig in te zien dat geen enkele van deze productieregels meer dan één niet-terminaal symbool aan de rechterzijde heeft staan. Wanneer we een niet-terminaal symbool substitueren door de rechterkant van een productieregel, zal er maximaal één nieuw niet-terminaal symbool worden toegevoegd aan het tussenresultaat. Maar omdat we het symbool dat we hebben gesubstitueerd ook moeten verwijderen, zal het totaal aantal niet-terminale symbolen dus niet kunnen toenemen.

We kunnen dus concluderen dat het maximaal aantal niet-terminale symbolen na $n+1$ stappen nog steeds 1 bedraagt. ∎

***Stelling 1.2.1* [11]**

Voor alle reguliere talen $\mathcal{L}$ bestaat er een positieve integer T, een codebook C en een verzameling A van duplexen, zodat $\mathcal{L} = \mathcal{L}_{T,A,C}$.

Bewijs: Voor het bewijs van Stelling 1.2.1 veronderstellen we dat we beschikken over een rechts-reguliere grammatica $G = (N, \Sigma, P, S)$ die $\mathcal{L}$ beschrijft. We construeren nu voor ieder symbool $i \in N \cup \Sigma$ een unieke string s_i over het alfabet $\{A, C, T, G\}$. We noteren hun Watson-Crick complementaire string als s'_i. Nu zijn we klaar om voor iedere productieregel $p \in P$ een DNA *complex* te construeren, hiervoor houden we rekening met de vorm van p:

- p is van de vorm $A \to bB$, met $b \in \Sigma$ en $B \in N$. In dit geval construeren we een DNA *complex* met twee sticky ends. Het linker sticky end wordt gelabeld met s'_A, het rechter sticky end wordt gelabeld met s_B en de interne structuur van het DNA *complex* wordt gelabeld met s_b en s'_b. Zie Figuur 1.4 (a).

- p is van de vorm $A \to b$, met $b \in \Sigma \cup \{\epsilon\}$. In dit geval construeren we een DNA *complex* met enkel een sticky end aan de linkerkant. Dit sticky end wordt gelabeld met s'_A en de interne structuur van het DNA *complex* wordt gelabeld met s_b en s'_b. Zie Figuur 1.4 (b).

- We produceren ook één speciaal DNA *complex* dat links geen sticky end heeft, maar rechts wel. Dit sticky end labelen we met s_S, waarbij S het startsymbool voorstelt. We noemen dit DNA *complex* het *startcomplex*, zie Figuur 1.4 (c). ∎

Voor iedere woord $w \in \mathcal{L}$ bestaat er een afleiding D_w over G. Omdat G een rechts-reguliere grammatica is, weten we dat D_w volgende vorm heeft:

$$S \to_1 \underbrace{aA}_{\text{lengte 2}} \to_2 \ldots \to_i \underbrace{ab\ldots cD}_{\text{lengte i+1}} \to_{i+1} \ldots \to_{n-1} \underbrace{ab\ldots cd}_{\text{lengte n}}$$

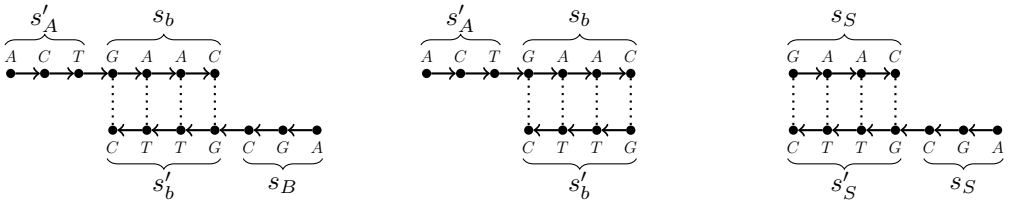
We gaan nu proberen aan te tonen met behulp van inductie, dat het mogelijk is om deze afleiding D_w te simuleren met het DNA *complex* dat we zonet hebben geconstrueerd.

Basisstap: Initieel starten we de constructie met het startcomplex, zie Figuur 1.4 (c). Omdat het startcomplex slechts 1 sticky end heeft, met als label s'_S, kan er tijdens hybridizatie slechts één DNA *complex*, met aan de linkerzijde een sticky end met als label s_S, worden toegevoegd aan het geheel. Er kan dus enkel een DNA *complex* worden toegevoegd dat een productieregel representeert waarvan het linkerlid het startsymbool S is. In een afleiding zal het startsymbool steeds als eerste worden gesubstitueerd omdat initieel enkel het startsymbool aanwezig is. We kunnen dus concluderen dat de eerste hybridizatiestap correct verloopt.

Inductiestap: We veronderstellen nu dat de gevormde molecule na n hybridizatiestappen correct is, en dat deze dus hetzelfde tussenresultaat representeert als een afleiding na n substituties.

Inductiehypothese: Na n substituties zullen we een molecule hebben dat uiterst rechts nog 1 sticky end heeft met als label s'_X. We moeten nu aantonen dat er een DNA *complex* is dat aan zijn linkerkant een sticky end heeft met als label s_X, zodat er ook nu hybridizatie kan optreden. Het label s'_x geeft aan dat er in de afleiding een niet-terminaal X staat dat tijdens de $n+1$de substitutiestap moet worden gesubstitueerd. Opdat deze substitutie zou kunnen plaatsvinden, moet er ook een productie p aanwezig zijn in β die aan zijn linkerlid het niet-terminale symbool X heeft staan. En omdat we voor iedere productieregel ook een overeenkomstig DNA *complex* werd gegenereerd, is het dus mogelijk het huidige molecule uit te breiden en dus de $n+1$de hybridizatiestap toe te passen. Omdat we willen dat het molecule hetzelfde woord oplevert als D_w, kiezen we het DNA *complex* dat de productieregel p representeert. Hierbij is p de productieregel die werd gebruikt tijdens de $n+1$de substitutiestap van D_w.

We weten nu dat het mogelijk is om voor iedere afleiding d een molecule m_d te genereren. Wanneer we eerst ligatie en vervolgens denaturatie toepassen op deze molecule, dan zal deze de vorm van een duplex hebben zonder sticky ends. Het duplex bestaat uit twee lange strands die ieder een bepaald woord w omschrijven. Deze strands bevatten sequenties van niet-terminale en terminale labels, die elkaar steeds afwisselen. De niet-terminale labels zijn afkomstig van de sticky ends, de terminale van de interne delen van de gebruikte DNA *complexen*. Om de niet-terminale labels te verwijderen kunnen we gebruik maken van een codebook $\mathcal{C}$. Voor ieder terminaal symbool i, wordt er een regel $\mathcal{C}_i = S_i$ toegevoegd aan $\mathcal{C}$. Hieruit volgt dat $\mathcal{C}(denaturatie(ligatie(m_d))) = w$. En omdat dit geldt voor iedere afleiding over G, kunnen we concluderen dat $\mathcal{L} = \mathcal{L}_{T,A,C}$.



(a) $A \to bB$ (b) $A \to b$ (c) S

Figuur 1.4: Dit zijn de DNA *complexen, gebruikt in Bewijs 1.2.2, van de bijhorende productieregels die onder de afbeelding worden weergegeven. De accolades duiden aan welk deel van het* DNA *complex gebruikt wordt om een bepaald (niet)-terminaal symbool te encoderen.*

***Stelling 1.2.2* [11]**

Voor alle positieve integers T, codebooks C en verzamelingen van duplexen A, beschrijft $\mathcal{L}_{T,A,C}$ een reguliere taal.

Bewijs: Voor het bewijs van Stelling 1.2.2 gaan we een rechts-reguliere grammatica G construeren die dezelfde taal genereert als $\mathcal{L}_{T,A,C}$. Voor ieder sticky end creëeren we een niet-terminaal symbool. Hierbij wordt hetzelfde niet-terminale symbool toegekend aan sticky ends met hetzelfde of een Watson-Crick complementair label. We voegen ook het niet-terminale symbool S toe, dit is het startsymbool. Voor de labels van de interne gedeeltes van alle duplex $d \in A$ creëren we een terminaal symbool. Ook hier zorgen we ervoor twee labels dit hetzelfde zijn vertaald worden naar hetzelfde terminaal symbool. Voor het construeren van de productieregels beschouwen we alle sticky ends. Voor ieder duplex d onderscheiden we volgende situaties:

- Het duplex d kan hybridizatie aangaan met een ander duplex i, dus $d +_{\text{B}} i \neq \emptyset$. Afhankelijk van de vorm van i onderscheiden we 2 situaties:
  - i heeft enkel een sticky end aan de linkerzijde. In deze situatie zal er na toevoegen van i aan het geheel geen uitbreiding meer mogelijk zijn naar rechts toe. Het woord dat deze molecule beschrijft kan dus niet meer groeien naar rechts toe, hierdoor dan de productieregel geen niet-terminaal symbool nodig hebben in de rechterzijde. We voegen dus $R_d \rightarrow m_i$ toe, hierbij is R_d het niet-terminale symbool dat bij het rechtse sticky end van d hoort en m_i het terminale symbool dat bij het interne gedeelde van i hoort.
  - i heeft een sticky end aan beide zijdes. Na toevoegen van i is er mogelijk nog uitbreiding mogelijk, hierdoor zullen we wel een niet-terminaal symbool aan de rechterzijde van de productieregel plaatsen, deze zal er als volgt uitzien: $R_d \rightarrow m_i R_i$. Hierbij zijn R_d en R_i de niet-terminale symbolen die respectievelijk bij het rechtse sticky end van d en i horen en is m_i het terminale symbool dat bij het interne gedeelde van i hoort.
- Het duplex d kan geen hybridizatie aangaan. Omdat er geen verdere uitbreiding mogelijk is moeten we ervoor zorgen dat onze grammatica ook kan stoppen. Dit lossen we op door volgende productieregel toe te voegen: $R_d \rightarrow \epsilon$, hierbij is R_d het niet-terminale symbool van het rechtse sticky end van d.
- Het duplex d heeft links wel een sticky end, maar er is geen ander duplex i waarvoor geldt dat $i +_{\text{B}} d \neq \emptyset$ OF d heeft links geen sticky end. Dit betekend dus dat dit duplex altijd aan het begin van een groter geheel zal voorkomen. Hierdoor moeten we volgende productieregel toevoegen: $S \rightarrow m_d R_d$. Hierbij is R_d het niet-terminale symbool van het rechtse sticky end van d en m_d het terminale symbool dat bij het interne gedeelde van d hoort. ■

1.3 Context vrije Talen

We hebben in de vorige paragraaf besproken wat een reguliere grammatica is. Er bestaan echter nog andere methodes voor het beschrijven van talen, die veel krachtiger talen dan de reguliere talen kunnen uitdrukken. De context-vrije taal is hier een voorbeeld van. We zullen gebruik maken van een context-vrije grammatica om deze taal te beschrijven, zie Definitie 1.4.

Definitie 1.4: *Een **context-vrije grammatica** is een viertuppel (V, Σ, P, S), hierbij is:*

- *V een eindige verzameling van **variabelen**;*
- *Σ een eindige verzameling van **terminale** symbolen, disjunct met V;*
- *P een eindige verzameling van **productieregels**. Het linkerlid bevat precies 1 variabele, het rechterlid een string van variabelen en terminale symbolen;*
- *$S \in V$ de **startvariabele**.*

We gebruiken een context-vrije grammatica om een context-vrije taal te beschrijven door ieder woord over die taal te genereren op de volgende manier:

1. Schrijf de startvariabele neer
2. Ga nu op zoek een variabele die je eerder had neergeschreven. Ga vervolgens op zoek naar een productieregel waarvan het linkerlid overeenkomt met de zonet gevonden variabele. Vervang nu die variabele door het rechterlid van die productieregel.
3. Herhaal stap 2 zolang er variabelen overblijven.

We noemen bovenstaande sequentie van substituties van productieregels, om zo een woord te genereren, een afleiding over een context-vrije grammatica. Het is mogelijk om iedere context-vrije grammatica voor te stellen in een speciale vorm, namelijk *Chomsky normal form*, zie Definitie 1.5 en Stelling 1.3.1. Het voordeel van deze vorm is dat de productieregels nu een vaste vorm hebben, hierdoor wordt de vertaling naar *de taal van* DNA *complexen* makkelijker.

Definitie 1.5: *Een context-vrije grammatica G is in Chomsky normal formwanneer iedere productieregel in één van onderstaande vormen is:*

- $A \to BC$
- $A \to a$

Hierbij zijn A, B en C variabelen en a een terminaal symbool uit G, maar B en C mogen niet de startvariabele zijn. Bovendien laten we de regel $S \to \epsilon$ toe, waarbij S de startvariabele is.

***Stelling 1.3.1* [8]**

Voor iedere context-vrije grammatica bestaat er een equivalente context-vrije grammatica in Chomsky normal form.

***Stelling 1.3.2* [11]**

Voor alle context-vrije talen $\mathcal{L}$ bestaat er een positieve integer T, een codebook C en een verzameling A van duplexen, hairpins en 3-armed junctions, zodat $\mathcal{L} = \mathcal{L}_{T,A,C}$.

Bewijs: Voor het bewijs van Stelling 1.2.1 veronderstellen we dat we beschikken over context-vrije grammatica G' die de context-vrije taal $\mathcal{L}$ uitdrukt. Wegens Stelling 1.3.1, weten we dat er een equivalente context-vrije grammatica bestaat in *Chomsky normal form*, we noemen deze $G = (V, \Sigma, P, S)$.

We maken gebruik van een codebook $\mathcal{C}$ om de variabelen en terminale symbolen uit G om te zetten naar unieke strings over het alfabet $\{A, C, T, G\}$. We gebruiken de notatie $\mathcal{C}_i$ om de vertaling, door het codebook $\mathcal{C}$, van een variabele of terminaal symbool i voor te stellen. Voor hun Watson-Crick complement gebruiken we $\mathcal{C}_{i'}$ $\mathcal{C}_0$ stelt een speciale 0-string voor, deze zullen we gebruiken op plaatsen waar we geen labels nodig hebben.

We voeren ook een notatie in voor de labels van de sticky ends van de DNA *complexen*. Stel dat b een 3-armed junction is, dat stelt L_b de string voor die op de labels staan van het linkse sticky end van b. Analoog stellen B_b en O_b de labels van respectievelijk het bovenste en onderste sticky end voor. Het label van het interne gedeelte tussen de het linkse sticky end en het bovenste sticky end noteren we door mlb_j. Analoog stellen mbo_j en mol_j de labels tussen respectievelijk het bovenste en onderste, en het onderste en linkse sticky end voor. Voor een hairpin h zullen we L_h gebruiken voor het label van haar enige sticky end en m_h voor het interne gedeelte.

We zullen nu voor iedere productieregel $p \in P$, afhankelijk van zijn vorm, een hairpin of 3-armed junction construeren. De vormen die p kan aannemen zijn:

- p is van de vorm $A \to BC$, met A, B en C variabelen en $A \neq S$. In deze situatie construeren we een 3-armed junction j, waarbij $L_j = \mathcal{C}_A$, $B_j = \mathcal{C}_{B'}$ en $_j = \mathcal{C}_{C'}$. Omdat we voor de interne gedeeltes geen invulling nodig hebben stellen we deze gelijk aan de 0-string, dus $mlb_j = mbo_j = mol_j = \mathcal{C}_0$, zie Figuur 1.5 (a).

- p is van de vorm $A \to a$, met A een variabele en a en terminaal symbool. In deze situatie construeren we een hairpin h, met $L_H = \mathcal{C}_A$ en $m_h = \mathcal{C}_a$, zie Figuur 1.5 (b).■

Voor de startvariabele S construeren we een duplex d. Hierbij is $m_d = \mathcal{C}_0$ en $R_d = \mathcal{C}_{S'}$, zie Figuur 1.5 (c).

Voor iedere woord $w \in \mathcal{L}$ bestaat er een afleiding D_w over G. We gaan nu proberen aan te tonen met behulp van inductie, dat het mogelijk is om deze afleiding D_w te simuleren met de DNA *complex* die we zonet hebben geconstrueerd.

Basisstap: Initieel starten we de constructie door het duplex d, dat we construeerde voor de startvariabele s, toe te voegen. Omdat $R_d = \mathcal{C}_{S'}$ zal er enkel hybridizatie kunnen optreden met een 3-armed junction j, waarbij $mbo_j = \mathcal{C}_S$. Dit komt overeen met de substitutie van de startvariabele S door een productieregel p in de afleiding. Hierbij is het linkerlid van p gelijk aan S. We kunnen dus concluderen dat de eerste hybridizatiestap correct verloopt.

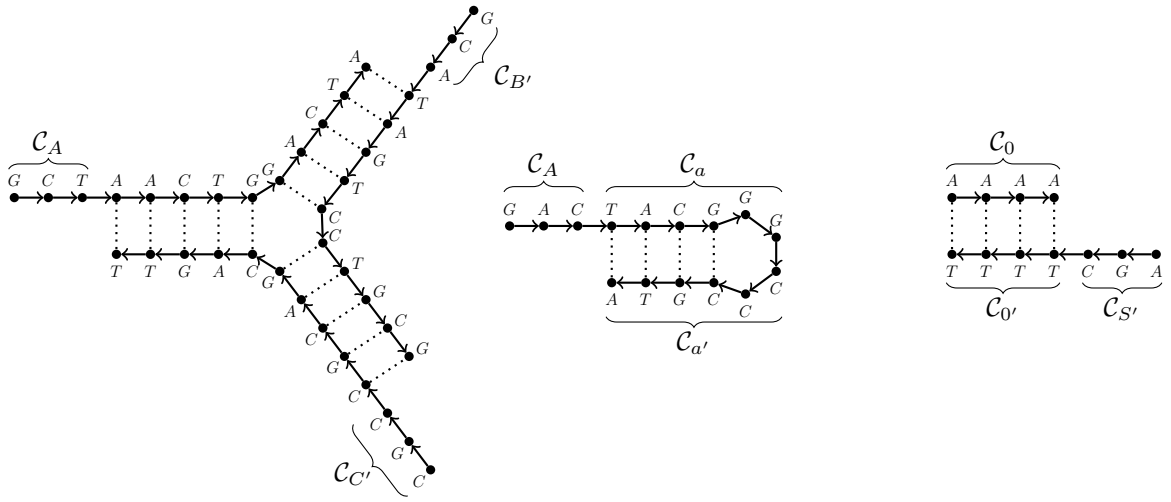
Inductiestap: We veronderstellen nu dat de gevormde molecule na n hybridizatiestappen correct is, en dat deze dus hetzelfde tussenresultaat representeert als een afleiding na n substituties.

Inductiehypothese: Na n substituties zal de afleiding bestaan uit een afwisseling van variabelen en terminale symbolen. Voor iedere variabele v in die afwisseling zal er een sticky end s_v van een 3-armed junction bestaan dat nog geen hybridizatie is aangegaan met een ander sticky end. Stel dat tijdens de $n + 1$de substitutiestap, de variabele v_i zal worden gesubstitueerd door een productieregel p. Dan zal er hybridizatie moeten plaatsvinden tussen het sticky end van een 3-armed junction, dat de v_i vertegenwoordig in onze molecule, en een nieuwe 3-armed junction dat de productieregel p vertegenwoordigd. Op deze manier zal onze molecule, na $n + 1$ hibridizatiestappen, nog steeds hetzelfde tussenresultaat representeren als de afleiding na $n + 1$ substitutiestappen.

We weten nu dat het mogelijk is om een molecule m_d te genereren voor iedere afleiding d, die een woord w_d produceert. Wanneer we eerst ligatie en vervolgens denaturatie toepassen op deze molecule, dan zullen we 1 lange strand verkrijgen. Deze strand zal opgebouwd zijn door de labels van de variabelen, terminale symbolen en de 0-string. We kunnen gebruik maken van het codebook $\mathcal{C}$ om enkel de labels van de terminale symbolen eruit te filteren. De volgorde waarin deze terminale symbolen voorkomen in de strand is dezelfde volgorde als in het woord w_d. Hieruit volgt dat $\mathcal{C}(denaturatie(ligatie(m_d))) = w_d$. En omdat dit geldt voor iedere afleiding over G, kunnen we concluderen dat $\mathcal{L} = \mathcal{L}_{T,A,C}$.

Stelling 1.3.3 **[11]**

Voor alle positieve integers T, codebooks C en verzamelingen van duplexen, hairpins en 3-armed junctions A, $\mathcal{L}_{T,A,C}$ is equivalent aan een reguliere taal.



(a) $A \to BC$ (b) $A \to a$ (c) S

Figuur 1.5: Dit zijn de DNA *complexen, gebruikt in Bewijs 1.3.1, van de bijhorende productieregels die onder de afbeelding worden weergegeven. De accolades duiden aan welk deel van het* DNA *complex gebruikt wordt om een bepaald (niet)-terminaal symbool in te encoderen.*

Bewijs: Voor het bewijs van Stelling 1.3.3 gaan we een context-vrije grammatica G construeren die dezelfde taal genereert als $\mathcal{L}_{T,A,C}$. Voor ieder sticky end creëeren we een variabele. Hierbij wordt dezelfde variabele toegekend aan sticky ends met hetzelfde of een Watson-Crick complementair label. We voegen ook de variabele S toe, dit is het startsymbool. Voor de labels van de interne gedeeltes van alle hairpins $h \in A$ creëren we een terminaal symbool. Ook hier zorgen we ervoor twee labels dit hetzelfde zijn vertaald worden naar hetzelfde terminaal symbool.

We voeren een notatie in voor de labels van de sticky ends van de DNA *complexen*. Stel dat b een 3-armed junction is, dat stelt L_b de string voor die op de labels staan van het linkse sticky end van b. Analoog stellen B_b en O_b de labels van respectievelijk het bovenste en onderste sticky end voor. Het label van het interne gedeelte tussen de het linkse sticky end en het bovenste sticky end noteren we door mlb_j. Analoog stellen mbo_j en mol_j de labels tussen respectievelijk het bovenste en onderste, en het onderste en linkse sticky end voor. Voor een hairpin h zullen we L_h gebruiken voor het label van haar enige sticky end en m_h voor het interne gedeelte.

Voor het construeren van de productieregels beschouwen we alle DNA *complexen* $d \in A$. Afhankelijk van welk type DNA *complex* d is doen we het volgende:

- d is een 3-armed junction. In dit geval voegen we de productieregel $\mathcal{C}_{L_d} \to \mathcal{C}_{B_d}\mathcal{C}_{O_d}$ toe.
- d is een hairpin. In dit geval voegen we de productieregel $\mathcal{C}_{L_d} \to \mathcal{C}_{m_d}$ toe.

- d is een duplex. Hier onderscheiden we 3 gevallen:
  - Indien d twee sticky ends heeft, doen we niets.
  - Indien d enkel een rechter sticky end heeft, dan voegen we de productieregel $S \to \mathcal{C}_{R_d}$ toe.
  - Indien d enkel een linker sticky end heeft, dan voegen we de productieregel $\mathcal{C}_{L_d} \to \mathcal{C}_{m_d}$ toe. In dit laatste geval behandelen we d alsof het een hairpin was.■

1.4 Blocked Cellular Automata

In deze paragraaf wordt een model voor berekenbaarheid besproken, namelijk een Blocked Cellular Automata (BCA) [7]. Eerst zullen we aantonen dat een BCA even expressief is als een *Turing machine*, zie ook Winfree [11]. Dit doen we door een constructie te geven die een *Turing machine* omzet naar een equivalente BCA. Later in deze paragraaf zullen we dan aantonen hoe je een BCA kan simuleren met behulp van DNA *complexen*.

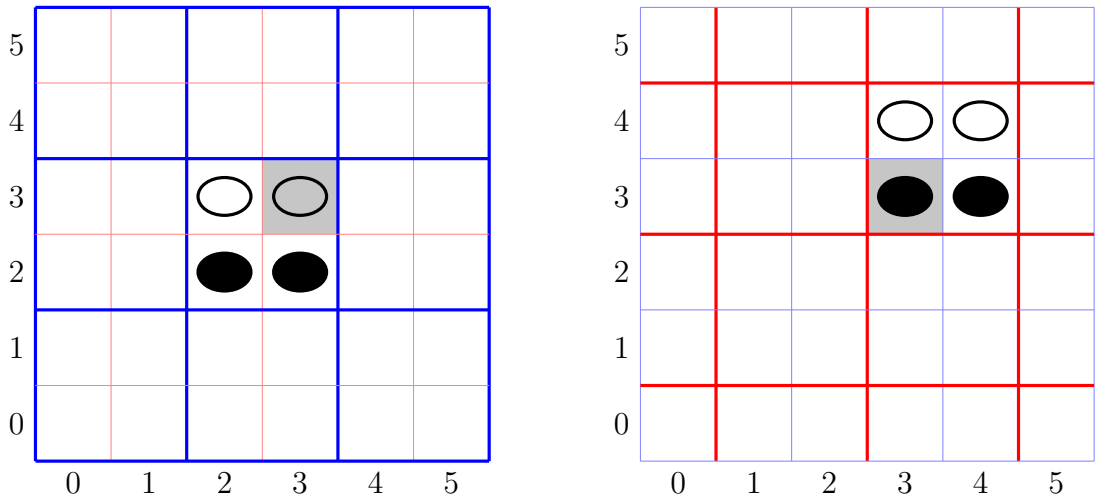
Ik ga er vanuit dat de lezer al bekend is met het begrip *Turing machine*, zie ook Sipser [8]. Om notatie en terminologie vast te leggen, herhalen we hier de definitie van een *Turing machine*, samen met enkele proposities die ik later zal gebruiken.

Definitie 1.6: *Een (deterministische) Turing machine is een 7-tuppel,* $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$*, waarbij*

- Q *een eindige verzameling van* ***toestanden*** *is,*
- Σ *een eindig* ***input-alfabet*** *is,*
- Γ *een eindig* ***tape-alfabet*** *is, waarbij* $\sqcup \in \Gamma$ *en* $\Sigma \subseteq \Gamma \setminus \{\sqcup\}$*,*
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ *een* ***transitie functie*** *is,*
- $q_0 \in Q$ *de begin-toestand is,*
- $q_{accept} \in Q$ *de accept-toestand is, en*
- $q_{reject} \in Q$ *de reject-toestand is, waarbij* $q_{accept} \neq q_{reject}$*.*

Lemma 1.4.1 **[8]**

Iedere niet-deterministische Turing machine heeft een equivalente deterministische Turing machine.



Figuur 1.6: De blauwe randen stellen de eerste indeling voor en de rode randen de tweede indeling. In beide figuren stellen de vol gekleurde ovalen een invoercel voor en de niet gekleurde ovalen een uitvoercel. Het grijs-gekleurde vakje stelt in beide figuren dezelfde cel c voor. Je kan zien dat c in het linker figuur een uitvoercel is en in het rechter figuur een invoercel.

***Lemma 1.4.2* [8]**

Iedere multi-tape Turing machine heeft een equivalente single-tape Turing machine.

Een ander model voor berekenbaarheid is de *Cellular Automata*. Een CA maakt gebruik van een n-dimensionaal raster, met $n \geq 1$, dat is opgebouwd uit cellen die een bepaalde vorm aannemen. Meestal is deze vorm een vierkant, maar dit mag bijvoorbeeld ook een zeshoek zijn. Bovendien zal een cell zich steeds in één van zijn k *states* bevinden, met $k \geq 2$. Een van deze states stelt de zogenaamde *halting state* voor. Wij gaan ons nu concentreren op een specifiek geval van een CA, namelijk de BCA.

Een BCA slaat zijn informatie op in een oneindig 2-dimensionaal **raster**. Dit raster is opgebouwd uit *vierkante* cellen. Hierbij heeft iedere cel een waarde afkomstig uit het *raster-alfabet*. Dit alfabet wordt voorgesteld door een eindige verzameling van symbolen.

De BCA maakt gebruik van de *Margolus omgeving* om zijn raster op te delen in groepjes van grootte 2×2. Deze omgeving bestaat uit twee indelingen, die ervoor zorgen dat wanneer twee cellen zich samen in een groepje bevinden in de eerste indeling, ze niet samen zullen voorkomen in een groepje in de andere indeling. In Figuur 1.6 worden de twee mogelijke indelingen weergegeven.

Daarnaast heeft iedere BCA zijn eigen *rule-table*. Deze tabel bestaat uit een eindig aantal regels. Iedere regel neemt als invoer een paar cellen en zal op basis van de inhoud

van deze cellen een waarde toekennen aan twee andere cellen. Merk op dat deze nieuwe waarde eventueel dezelfde waarde mag zijn als de waarde die oorspronkelijke in de cel stond. De twee invoer cellen zijn altijd de twee onderste cellen uit een groepje gegenereerde door de Margolus omgeving, de twee bovenste cellen zijn dan de uitvoer cellen, zie Figuur 1.6. We definiëren een BCA als volgt:

Definitie 1.7: *Een Blocked Cellular Automata is een 2-tuppel,* (Γ, δ)*, waarbij*

- Γ *een eindig raster-alfabet is,*
- $\delta : \Gamma \times \Gamma \longrightarrow \Gamma \times \Gamma$ *de transitie functie is.*

Een BCA begint zijn berekeningen op een tijdstip i. Tijdens dit tijdstip staat de *begin-configuratie* op het raster. Daarnaast zal het raster ook opgedeeld zijn in groepjes volgens de Margolus omgeving. We spreken af dat de eerste Margolus indeling wordt gebruikt op een even tijdstip, en dus de tweede indeling tijdens een oneven tijdstip. Hierdoor zal er dus altijd gealterneerd worden tussen de twee indelingen. Om de berekening van stap i naar stap $i + 1$ te laten overgaan moet het raster worden vertaalt naar een nieuw raster. Hiervoor maken we gebruik van δ.

Voor iedere groepje g uit de Margolus omgeving spreken we δ aan. Hierbij zal de inhoud van de twee onderste cellen van g de invoer van δ voorstellen. Vervolgens zal er aan de twee bovenste cellen van g de uitvoer van δ toegewezen worden. We noemen een bepaalde invulling van het raster een **configuratie** van de BCA.

We zeggen dat een configuratie C_1 de configuratie C_2 **oplevert** indien we C_1 op een legale manier kunnen omzetten naar C_2. Formeel komt dit er op neer we het raster gespecifiëerd door C_1 moeten kunnen vertalen met behulp van δ om zo het raster gespecifiëerd door C_2 te bekomen.

Stelling 1.4.1

Zij $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ *een willekeurige Turing machine, dan bestaat er een equivalente* BCA *die* M *kan simuleren in polynomiale tijd.*

Bewijs: Om Stelling 1.4.1 aan te tonen is het, wegens Lemma 1.4.1 en 1.4.2, voldoende om een transformatie te vinden van een deterministische single-tape Turing machine naar een BCA.

Zij $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ een willekeurige *deterministische single-tape Turing machine*. Het idee van dit bewijs is om een BCA te construeren die initiëel start met de omgevormde start-configuratie van M, met een bepaalde invoer op zijn tape. Vervolgens zal deze stap voor stap deze configuratie aanpassen, precies op dezelfde manier als hoe de configuratie van M zou veranderen als we M zouden uitvoeren. We construeren dus een equivalente BCA $B = (\Gamma', \delta')$ als volgt:

- $\Gamma' = \Sigma' \cup \Omega \cup \{\sqcup\}$, hierbij is:
  - $\Sigma' = \Gamma$;
  - $\Omega = \{s_1, \ldots, s_i\, \dot{s_1}, \ldots, \dot{s_i}\}$, dus voor iedere state in M voegen we twee speciale symbolen toe aan het state-alfabet van B. De symbolen met een punt zullen we later gebruiken om een beweging naar links te simuleren, de symbolen zonder punt voor een beweging naar rechts.
- δ' bouwen we op als volgt:
  1. Voor alle mogelijke koppels (x, y), met $x, y \in \Gamma$, voegen we de regel $(x, y) \to (x, y)$ toe aan δ'. Hierdoor blijft de tape ongewijzigd indien er geen state-symbool in het koppel voorkomt. In een *Turing machine* kan een symbool op de tape namelijk enkel worden overschreven indien het tape-hoofd zich langs een symbool bevindt.
  2. Voor iedere regel $(q_i, x) \to (q_j, y, R)$ uit δ, met $q_i, q_j \in Q$, $x, y \in \Gamma$ en $s_i, s_j \in \Omega$, voegen we $(s_i, x) \to (y, s_j)$ toe aan δ'.
  3. Voor iedere regel $(q_i, x) \to (q_j, y, L)$ uit δ, met $q_i, q_j \in Q$, $x, y \in \Gamma$ en $s_i, \dot{s_j} \in \Omega$, voegen we $(s_i, x) \to (\dot{s_j}, y)$ aan δ' toe. Merk op dat we nu een bolletje boven s_j hebben geplaatst. Dit dient om aan te geven dat de verplaatsing naar links nog moet gebeuren.
  4. Voor alle mogelijke koppels $(x, \dot{s})$, met $x \in \Gamma$ en $\dot{s} \in \Omega$ voegen we $(x, \dot{s}) \to (s, x)$ toe aan δ'. De symbolen met een bolletje worden dus gebruikt om een verplaatsing naar links op de tape uit te voeren.
  5. Voor alle mogelijke koppels (x, s), met $x \in \Gamma$, $s \in \Omega$ en waarbij (x, s) *niet* voorkomt in het linkerdeel van de regels geproduceerd door één van bovenstaande regels, voegen we $(x, s) \to (x, s)$ aan δ' toe.
  6. Voor alle mogelijke koppels (s, x), met $x \in \Gamma$, $s \in \Omega$ en waarbij (s, x) *niet* voorkomt in het linkerdeel van de regels geproduceerd door bovenstaande regels, voegen we $(s, x) \to (s, x)$ aan δ' toe.

  De laatste twee regels zijn nodig zodat δ' altijd een resultaat kan generen op eender welk gegenereerd groepje door λ. Merk ook op dat iedere regel die we toevoegen aan δ', waarbij er een state-symbool in het linkerdeel van de regel staat, er ook altijd precies 1 state-symbool in het rechterdeel van diezelfde regel staat. Hierdoor blijf het aantal state-symbolen op de tape altijd hetzelfde.

Een *Turing machine* kan 2 soorten acties uitvoeren in 1 stap. Ofwel wijzigt hij de waarde van het symbool onder het leeshoofd en verplaatst hij vervolgens zijn leeshoofd 1 plaats naar *rechts* op de tape, of hij wijzigt de waarde van het symbool onder het leeshoofd en verplaats hij vervolgens zijn leeshoofd 1 plaats naar *links* op de tape. We gaan nu aantonen dat we beide operaties kunnen uitvoeren in lineaire tijd.

Veronderstel dat onderstaande configuratie, met $v, w, x, y \in \Gamma'$ en $s_i \in \Omega$, op de tape van B staat, hierbij geven de vette lijnen aan welke margolus groepering er van toepassing is.

1	⊔	⊔	⊔	⊔	⊔	⊔	⊔
0	⊔	v	w	s_i	x	y	⊔
	0	1	2	3	4	5	6

Zij $(q_i, x) \to (q_j, z, R) \in \delta$. Hierdoor is $(s_i, x) \to (z, s_j) \in \delta'$, zie stap 2 uit de constructie van δ'. Wegens stap 1 uit de constructie van δ' weten we ook dat de groepjes zonder s-symbool ongewijzigd blijven. Hierdoor wordt de configuratie na 1 iteratie het volgende:

2	⊔	⊔	⊔	⊔	⊔	⊔	⊔
1	⊔	v	w	z	s_j	y	⊔
0	⊔	v	w	s_i	x	y	⊔
	0	1	2	3	4	5	6

Hieruit kunnen we dus besluiten dat een beweging naar rechts kan gesimuleerd worden in 1 iteratie. Voor de simulatie van $(q_j, y) \to (q_k, a, L) \in \delta$, een beweging naar links, weten we dat $(s_j, y) \to (\dot{s_k}, a) \in \delta'$, zie stap 3 uit de constructie van δ'. Als we vertrekken vanuit bovenstaande configuratie verkrijgen we het volgende na 1 iteratie:

3	⊔	⊔	⊔	⊔	⊔	⊔	⊔
2	⊔	v	w	z	$\dot{s_k}$	a	⊔
1	⊔	v	w	z	s_j	y	⊔
0	⊔	v	w	s_i	x	y	⊔
	0	1	2	3	4	5	6

Wegens stap 4 uit de constructie van δ' is ook $(z, \dot{s_k}) \to (s_k, z) \in \delta'$, dit levert dan het gewenste resultaat op:

4	⊔	⊔	⊔	⊔	⊔	⊔	⊔
3	⊔	v	w	s_k	z	a	⊔
2	⊔	v	w	z	$\dot{s_k}$	a	⊔
1	⊔	v	w	z	s_j	y	⊔
0	⊔	v	w	s_i	x	y	⊔
	0	1	2	3	4	5	6

■

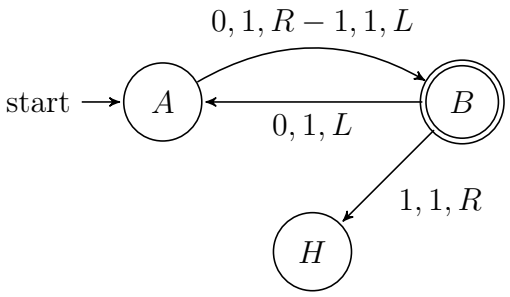
We hebben dus in twee iteraties de beweging naar links kunnen simuleren.

Zowel bij een beweging naar links als een beweging naar recht, is het noodzakelijk dat het state-symbool het meest linkse symbool in een groepje is. We stellen vast dat na de executie van een beweging naar rechts, dit nog steeds het geval is. We kunnen in dit geval onmiddellijk een nieuwe beweging naar links of rechts starten.

Bij de beweging naar links is dit echter niet het geval. Hierdoor zijn we genoodzaakt om 1 iteratie lang niets te doen, iedere keer als we een beweging naar links hebben uitgevoerd. Tijdens de volgende iteratie zullen de groepjes zodanig gewijzigd zijn, zodat het state-symbool weer het meest linkse symbool in een groepje is. Hierdoor is het weer mogelijk om een beweging naar links of rechts te simuleren.

Samengevat hebben we dus 1 iteratie nodig om een beweging naar rechts te simuleren en 3 iteraties voor een beweging naar links zodat we onmiddellijk een volgende beweging kunnen uitvoeren.

Veronderstel nu dat de executie van M gebeurd in $\mathcal{O}(n)$ tijd, dan zal B M kunnen simuleren in $\mathcal{O}(3n) = \mathcal{O}(n)$ tijd, dus polynomiale tijd.



Figuur 1.7: Dit is state graph van de deterministische Turing machine M uit voorbeeld 1.4.1.

Voorbeeld 1.4.1: In dit voorbeeld illustreren we hoe je een Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ kan omzetten naar een equivalente BCA $B = (\Sigma', \Gamma', \lambda, \delta', \alpha, \beta)$. We definiëren M als volgt:

- $Q = \{A, B, H\}$
- $\Sigma = \{0, 1\}$
- $\Gamma = \{0, 1, \sqcup\}$
- $\delta = \{$

  $(A, 0) \to (B, 1, R),$
  $(A, 1) \to (B, 1, L),$
  $(B, 0) \to (A, 1, L),$
  $(B, 1) \to (H, 1, R)$ $\}$
- $q_0 = A$
- $q_{accept} = H$
- $q_{reject} = \emptyset$

In Figuur 1.7 vind je de state graph van M terug. We gaan nu B construeren op basis van M, hiervoor gebruik ik de constructie uit Bewijs 1.4.1.

- $\Gamma = \Gamma' \cup \{A, B, \dot{A}, \dot{B}\} \cup \sqcup$.
- δ', zie Tabel 1.2. Zoals je kan zien is de transitie functie van de BCA veel groter dan die van de Turing machine. Dit komt omdat een Turing machine werkt met states, terwijl een BCA dit niet doet. Hierdoor moeten we de states van de Turing machine gaan vervangen door symbolen, waardoor het tape-alfabet groeit en bijgevolg ook de transitie functie. In voorbeeld 1.4.2 wordt de werking van deze BCA geïllustreerd. ♣

$(0,0) \to (0,0)$	$(0,1) \to (0,1)$	$(0,\sqcup) \to (0,\sqcup)$
$(1,0) \to (1,0)$	$(1,1) \to (1,1)$	$(1,\sqcup) \to (1,\sqcup)$
$(\sqcup,0) \to (\sqcup,0)$	$(\sqcup,1) \to (\sqcup,1)$	$(\sqcup,\sqcup) \to (\sqcup,\sqcup)$
$(A,0) \to (1,B)$	$(B,1) \to (1,H)$	$(B,0) \to (\dot{A},1)$
$(A,1) \to (\dot{B},1)$	$(0,\dot{A}) \to (A,0)$	$(1,\dot{A}) \to (A,1)$
$(\sqcup,\dot{A}) \to (A,\sqcup)$	$(0,\dot{B}) \to (B,0)$	$(1,\dot{B}) \to (B,1)$
$(\sqcup,\dot{B}) \to (B,\sqcup)$	$(0,A) \to (0,A)$	$(0,B) \to (0,B)$
$(0,H) \to (0,H)$	$(1,A) \to (1,A)$	$(1,B) \to (1,B)$
$(1,H) \to (1,H)$	$(\sqcup,A) \to (\sqcup,A)$	$(\sqcup,B) \to (\sqcup,B)$
$(\sqcup,H) \to (\sqcup,H)$		

Tabel 1.2: Dit is de invulling van δ' uit Voorbeeld 1.4.1.

Voorbeeld 1.4.2: In voorbeeld 1.4.1 hebben we een BCA B geconstrueerd die een *busy beaver Turing machine* met drie states, zonder de accept-state mee te tellen, kan simuleren. In dit voorbeeld gaan we illustreren hoe B, met als invoer alleen maar nullen op de tape, stap voor stap de berekeningen van de *busy beaver Turing machine* zal simuleren om zo uiteindelijk hetzelfde resultaat als de *Turing machine* te bekomen. Figuur 1.8 toont de tape-inhoud van B na de simulatie. Initiëel was enkel de onderste rij ingevuld. Na in iteratie werd de tweede rij ingevuld, dan de derde, ... De margolus indeling op de afbeelding is die gebruikt werd tijdens de laatste iteratie, waar rij 12 werd ingevuld.

12	0	0	1	1	H	1	1	0	0
11	0	0	1	B	1	1	1	0	0
10	0	0	A	0	1	1	1	0	0
9	0	0	A	0	1	1	1	0	0
8	0	0	0	$\dot{A}$	1	1	1	0	0
7	0	0	0	B	0	1	1	0	0
6	0	0	0	B	0	1	1	0	0
5	0	0	0	0	$\dot{B}$	1	1	0	0
4	0	0	0	0	A	1	1	0	0
3	0	0	0	0	A	1	1	0	0
2	0	0	0	0	1	$\dot{A}$	1	0	0
1	0	0	0	0	1	B	0	0	0
0	0	0	0	0	A	0	0	0	0
	0	1	2	3	4	5	6	7	8

Figuur 1.8: Deze figuur toont de tape-inhoud van de BCA *uit Voorbeeld 1.4.2. De margolus indeling die is afgebeeld, is de indeling die werd gebruikt tijdens de laatste iteratie.*

1.5 Recursief Opsombare Talen

We hebben in de vorige paragraaf besproken wat een BCA is. We toonde aan dat er voor iedere BCA een equivalente Turing machine bestaat, hierdoor zijn ze dus in staat om recursief opsombare talen uit te drukken. In deze paragraaf zullen aantonen hoe je een willekeurige BCA kan simuleren met behulp van DNA. Bijgevolg tonen we dus aan dat het mogelijk is om met DNA een recursief opsombare taal uit te drukken.

***Stelling 1.5.1* [11]**

Voor alle recursief opsombare talen $\mathcal{L}$ bestaat er een positieve integer T, een codebook C en een verzameling A van duplexen en double-crossovers, zodat $\mathcal{L} = \mathcal{L}_{T,A,C}$.

Bewijs: Voor het bewijs van Stelling 1.5.1 veronderstellen we dat we beschikken over een *Turing machine* T die de *recursief opsombare taal* $\mathcal{L}$ uitdrukt. Wegens Stelling 1.4.1, weten we dat er een BCA $B = (\Gamma, \delta)$ equivalent met T bestaat.

Wanneer we de constructie van B uit het bewijs van Stelling 1.4.1 bekijken, merken we op dat cellen met een waarde verschillend van $\sqcup$ niet meer kunnen worden gewijzigd. Hierdoor is het dus niet nodig dat DNA *complexen* terug van elkaar kunnen loskoppelen tijdens de simulatie van B met DNA *complexen*.

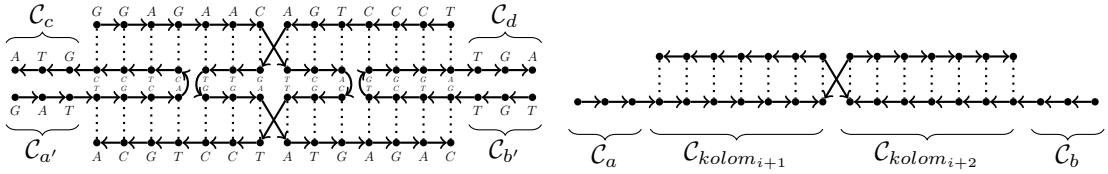
We maken gebruik van een codebook $\mathcal{C}$ om de symbolen uit Γ om te zetten naar unieke strings over het alfabet $\{A, C, T, G\}$. We gebruiken de notatie $\mathcal{C}_i$ om de vertaling, door het codebook $\mathcal{C}$, van symbool i voor te stellen. Hun Watson-Crick complement noteren we als $\mathcal{C}_{i'}$. $\mathcal{C}_0$ stelt een speciale 0-string voor, deze zullen we gebruiken op plaatsen waar we geen labels nodig hebben.

We voeren een notatie in voor de sticky ends van de double-crossovers. Met LB_d, RB_d, LO_d en RO_d duiden we respectievelijk de sticky ends links boven, rechts boven, links onder en rechts onder van een double-crossover d aan. Voor een duplex d noteren we het linker en rechter sticky end respectievelijk door L_d en R_D. Het onderste linker, bovenste linker, onderste rechter en bovenste rechter label van het interne gedeelte van d noteren we respectievelijk door LO_d, LB_d, RO_d en RB_d. Op basis van B construeren we nu de volgende DNA *complexen*:

- Voor iedere iedere regel $r_i \in \delta$ construeren we een double-crossovers d_i en zijn invers dr_i. Veronderstel dat r_i van de vorm $(a, b) \to (c, d)$ is. Dan stellen we $LO_d = \mathcal{C}_{a'}$, $RO_d = \mathcal{C}_{b'}$, $LB_d = \mathcal{C}_c$ en $RB_d = \mathcal{C}_d$, zie Figuur 1.9 (a). Het double-crossover dr_i in identiek aan d_i, behalve dat de richting van de pijlen van alle bogen in het complex omgekeerd is. Deze complexen zullen we gebruiken om de vertaling van een groepje uit de Margolus indeling te simuleren.

- $\forall (i, j) \in \Gamma^2$ construeren we een duplex d. Hierbij is $L_d = \mathcal{C}_i$, $R_d = \mathcal{C}_j$, $LO_d = \mathcal{C}_{kolom_i}$ en $RO_d = \mathcal{C}_{kolom_j}$, zie Figuur 1.9 (b). De interne gedeeltes van d gebruiken we dus om aan te geven in welke kolommen, van het raster van de BCA, de gegevens die de sticky ends van d representeren voorkomen. Deze DNA *complexen* zullen we gebruiken om de initiële DNA *complexen*, waarin we de invoer in encoderen, aan elkaar te koppelen.

De uitvoering van B kunnen we voorstellen door een sequentie van configuraties. We gaan nu proberen aan te tonen met behulp van inductie, dat het mogelijk is om deze configuratie te simuleren met de DNA *complex* die we zonet hebben geconstrueerd.



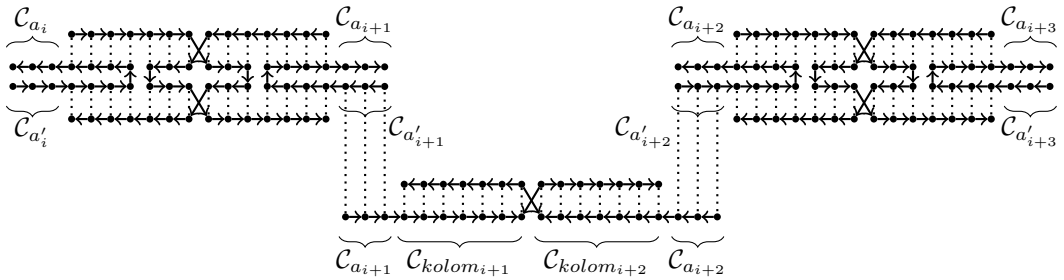
(a) $(a, b) \to (c, d)$ (b) (a, b)

Figuur 1.9: Figuur (a) illustreert hoe we een regel uit δ vertalen naar een double-crossover. Figuur (b) illustreert hoe we een duplex construeren dat wordt gebruikt om de initiële double-crossovers aan elkaar te koppelen.

Basisstap: De start-configuratie van een BCA, die geconstrueerd is volgens de constructie uit Bewijs 1.4.1, heeft zijn invoer op precies één rij in het raster staan. De overige cellen zijn leeg. Voor het initiële molecuul dat we gaan construeren is het voldoende om er enkel de invoer van B in te verwerken. Veronderstel dat $w = a_1, \ldots, a_k$ een invoerstring van lengte k is. Dan construeren we het initieel molecuul als volgt:

- Voor a_1 construeren we een double-crossover DNA *complex* d_{a_1}, hierbij is $LO_{d_{a_1}} = \mathcal{C}_0$, $RO_{d_{a_1}} = \mathcal{C}_0$, $LB_{d_{a_1}} = \mathcal{C}_{a'}$ en $RB_{d_{a_1}} = \mathcal{C}_a$.
- Voor alle paren (a_i, a_{i+1}), met $i \in [2, \ldots, k-1]$ een even getal, construeren we een double-crossover DNA *complex* d_i. Hierbij is $LO_{d_i} = \mathcal{C}_{a'_i}$, $RO_{d_i} = \mathcal{C}_{a'_{i+1}}$, $LB_{d_i} = \mathcal{C}_{a_i}$ en $RB_{d_i} = \mathcal{C}_{a_{i+1}}$.
- Voor alle paren (a_i, a_{i+1}), met $i \in [1, \ldots, k-1]$ een oneven getal, construeren we een duplex DNA *complex* d_i. Hierbij is $L_{d_i} = \mathcal{C}_{a_i}$, $R_{d_i} = \mathcal{C}_{a_{i+1}}$ $LO_{d_i} = \mathcal{C}_{kolom_i}$ en $RO_{d_i} = \mathcal{C}_{kolom_{i+1}}$.
- Als k een even getal is, dan construeren we een double-crossover d, waarbij $LO_d = \mathcal{C}_{a'_k}$, $RO_d = \mathcal{C}_0$, $LB_d = \mathcal{C}_{a_k}$ en $RB_d = \mathcal{C}_0$. Als k een oneven getal is hoeven we niets te construeren. In dit geval werd k namelijk al toegevoegd aan de rechterzijde van het double-crossover d_{k-1}.

De double-crossovers die we zonet hebben geconstrueerd koppelen we nu aan elkaar met behulp van duplexen, dit doen we door de onderste sticky ends van twee double-crossovers te verbinden met het duplex, zie Figuur 1.10. Merk op dat het noodzakelijk is dat de volgorde van de double-crossovers, die symbolen uit w representeren, overeen komt met de volgorde waarin de symbolen in w voorkomen. Op deze manier hebben we dan een molecuul geconstrueerd waarin de invoer van B is verwerkt. Dit initieel molecuul is dus equivalent aan configuratie C_1. Merk op dat alle double-crossovers via deze constructie langs elkaar zullen worden geplaatst op eenzelfde niveau. We zeggen dat alle double-crossovers die tijdens deze stap werden toegevoegd aan het molecuul zich op niveau 1 bevinden. Merk ook op dat er aan de bovenkant van het molecuul gaten zijn gevormd waar precies éé double-crossover inpast, zie Figuur 1.10 en Figuur 1.11. Deze gaten bevinden zich allemaal op niveau 1 en zullen worden gebruikt om het molecuul uit te breiden.



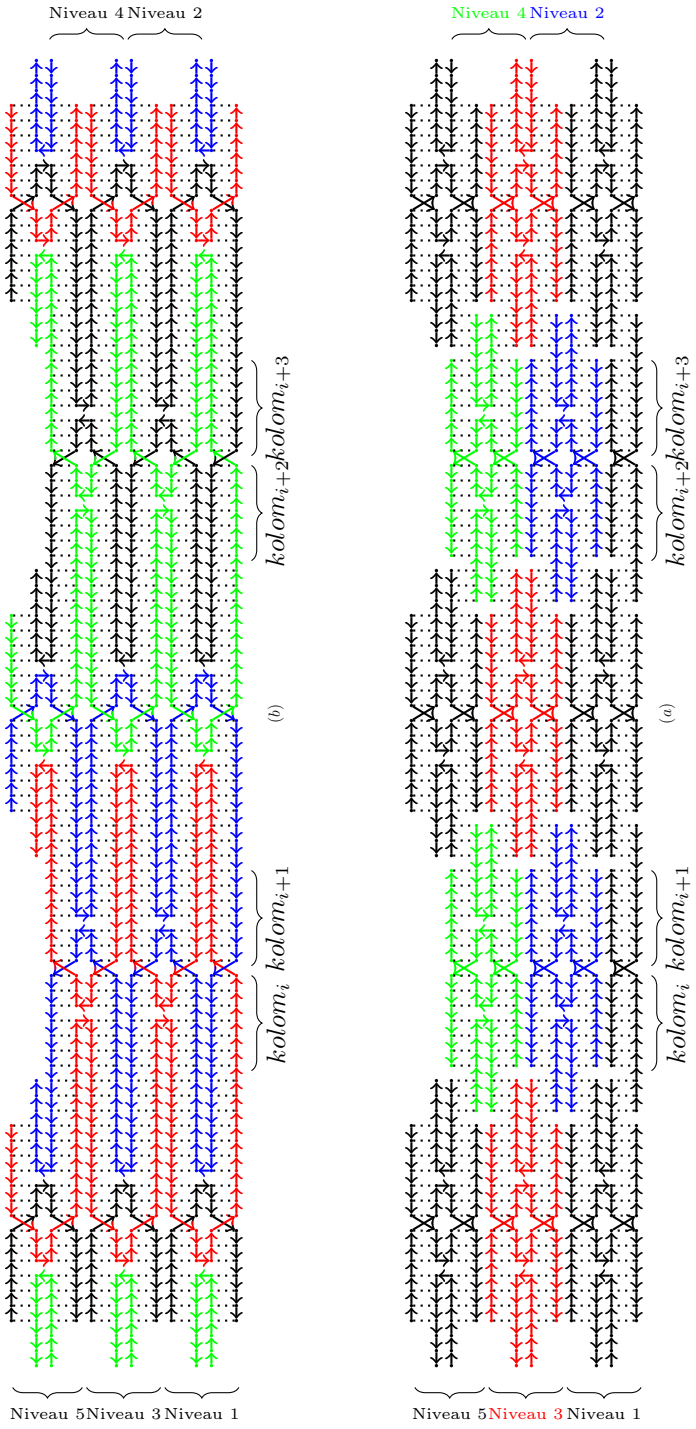
Figuur 1.10: Deze figuur illustreert hoe je twee double-crossovers aan elkaar kan koppelen met behulp van een duplex. Deze twee double-crossovers representeren de invoerstring $a_i a_{i+1} a_{i+2} a_{i+3}$.

Inductiestap We veronderstellen nu dat de gevormde molecule na n hybridizatiestappen correct is, en dat deze dus hetzelfde tussenresultaat representeert als configuratie C_n. De laatste double-crossovers die werden toegevoegd aan dit molecuul bevinden zich op niveau n, hierdoor bevinden de gaten, waaraan andere double-crossovers zich kunnen gaan hechten tijdens volgende iteraties, zich ook op niveau n. Merk op dat we onder één hybridizatiestap verstaan dat er steeds zoveel mogelijk DNA *complexen* zich zullen hechten aan het gevormde molecule.

Inductiehypothese: Na n hybridizatiestappen hebben we een molecule dat de configuratie C_n representeert. We moeten nu aantonen dat we deze molecule kunnen uitbreiden naar een representatie voor configuratie C_{n+1}. We weten dat de gaten zich op niveau n bevinden, dus wanneer we al deze gaten opvullen met double-crossovers zullen deze zich allemaal op niveau $n+1$ bevinden. Bovendien zullen de nieuw gecreëerde gaten zich ook op niveau $n+1$ bevinden, zie Figuur 1.11 (a). We moeten nu dus enkel nog aantonen dat we de correcte double-crossovers toevoegen aan het molecuul.

Een double-crossover representeert steeds 4 symbolen, twee bovenaan en twee onderaan. Net zoals bij de Margolus omgeving, kunnen we de twee onderste symbolen als de invoer symbolen beschouwen en de twee bovenste als uitvoer symbolen. Wanneer we een nieuw double cross-over toevoegen aan het molecuul, dan zal deze zich steeds hechten aan één uitvoer symbool van een double-crossover d_1 en aan een ander uitvoersymbool van een double-crossover d_2. Dit gedrag komt overeen met het alterneren tussen de twee Marlogus omgevingen. Hierdoor kunnen we de gaten, die zich op niveau n bevinden, opvullen met double-crossovers die de uitvoer symbolen zullen opleveren voor de gaten op niveau $n+1$. In Figuur 1.11 (a) kan je dit alternerende gedrag tussen de kolommen duidelijk zien. We kunnen dus concluderen we na 1 hybridizatiestap het molecuul hebben uitgebreid naar een representatie van configuratie C_{n+1}.

We weten nu dat het mogelijk is om een molecule m_{b_w} te genereren die de uitvoering van een BCA b op invoer w simuleert. Wanneer we eerst ligatie en vervolgens denaturatie toepassen op deze molecule, dan zullen we 1 lange strand verkrijgen voor iedere kolom uit het raster van b. Deze strands zijn opgebouwd met labels van symbolen uit δ en een



Figuur 1.11: Figuur (a) illustreert hoe de double-crossovers niveau per niveau worden toegevoegd aan het molecuul. De gaten die gecreëerd worden na het toevoegen van een niveau zijn ook goed zichtbaar. Figuur (b) illustreert hetzelfde molecuul na uitvoering van ligatie en denaturatie. Je kan duidelijk zien hoe dat alle waarden uit in kolom omvat zijn door 1 strand. Deze strand kunnen we makkelijk identificeren dankzij het label van de duplexen uit de initiële stap.

label dat aangeeft tot welke kolom deze strand behoord. We kunnen gebruik maken vaan een codebook $\mathcal{C}$ om de labels van de strands te vertalen naar symbolen over Γ om op die manier de uitvoer van de BCA te kunnen aflezen. Omdat we een dergelijk molecuul kunnen genereren voor iedere BCA en op iedere mogelijk invoer w, kunnen we concluderen dat $\mathcal{L} = \mathcal{L}_{T,A,C}$.

***Stelling 1.5.2* [11]**

Voor alle positieve integers T, codebooks C en verzamelingen van duplexen en double-crossovers A, $\mathcal{L}_{T,A,C}$ is equivalent aan een recursief opsombare taal.

Bewijs: Voor het bewijs van Stelling 1.5.2 gaan we een BCA $B = (\Gamma, \delta)$ construeren die dezelfde taal genereert als $\mathcal{L}_{T,A,C}$. Voor ieder label van een sticky end creëeren we een symbool s en voegen dit toe aan Γ. Hierbij wordt dezelfde symbool toegekend aan sticky ends met hetzelfde of een Watson-Crick complementair label. De labels van de interne gedeeltes van alle double-crossovers en duplexen laten we achterwegen.

We voeren een notatie in voor de sticky ends van de double-crossovers. Met LB_d, RB_d, LO_d en RO_d duiden we respectievelijk de sticky ends links boven, rechts boven, links onder en rechts onder van een double-crossover d aan. Voor ieder double-crossover d voegen we een regel $r = (LO_d, RO_d) \to (LB_d, RB_d)$ toe aan de transitie functie δ. ∎

1.6 Slotopmerking

We hebben in dit hoofdstuk besproken en bewezen hoe je met behulp van enkele DNA structuren reguliere talen, context vrije talen en tot slot recursief opsombare talen kan simuleren. De focus lag hier voornamelijk op het aantonen van deze equivalentie. Om af te sluiten wil ik nog even opmerken dat deze methodes echter niet optimaal zijn om het lidmaatschap van een bepaald woord w na te gaan in een reguliere of context vrije taal. Dit komt omdat we bij de methodes, die we hebben besproken in dit hoofdstuk, geen mechanisme hebben voorzien dat kan controleren welke hybridisatie er zal plaatsvinden indien er meerdere correcte mogelijkheden mogelijk zijn.

Een methode om het lidmaatschap na te gaan, zoals voorgesteld door Adleman [2], is om eerst alle mogelijke woorden genereren. Dit doen we door zeer grote hoeveelheden van alle DNA *complex* in één grote pot samen te gooien. Deze pot laten we dan een tijdje staan zodat de DNA *complex* de tijd krijgen om zich aan elkaar te hechten. Wanneer dit proces klaar is, moet je één voor één de gevormde moleculen in de pot controleren om te zien of ze overeen komen met w. Je kan natuurlijk nooit met 100% zekerheid garanderen dat alle woorden die kunnen gevormd worden, ook effectief zullen gevormd worden in de pot. Maar het idee is dat, als je voldoende DNA *complexen* neemt, dat de kans dat w er tussen zal zitten zeer groot is.

Dit is echter geen efficiënte manier van werken. In de volgende hoofdstukken zullen we een andere, meer abstracte, aanpak bespreken. Daar zullen we DNA tiles beschouwen om bijvoorbeeld patronen te gaan construeren met DNA. Deze patronen kan je visueel, mits het gebruik van een microscoop, wel gemakkelijk herkennen.

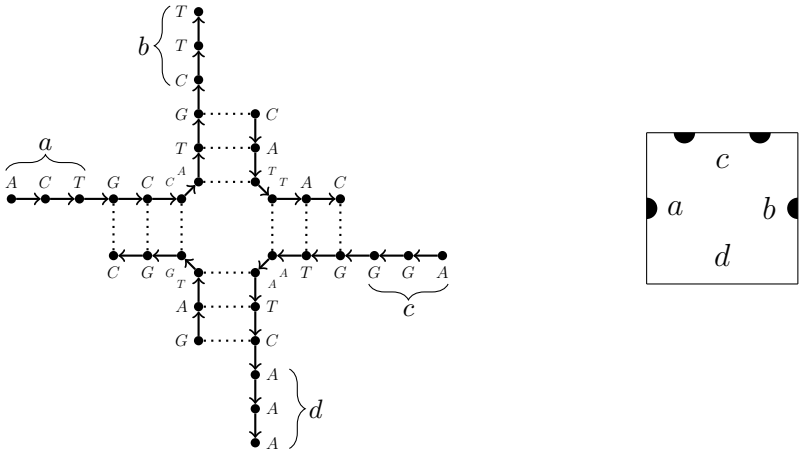
Hoofdstuk 2

DNA Tile Assembly

In Hoofdstuk 1 hebben we een methode gezien om een *Turing machine* te simuleren met DNA door gebruik te maken van een BCA. Er bestaan echter nog andere modellen, die ook Turing Universal zijn. Het Abstract Tile Assembly Model (aTam) is hier een voorbeeld van. In dit hoofdstuk zullen we eerst de basis concepten van een tile system uitleggen, vervolgens zullen we twee specifieke modellen beschouwen, namelijk aTam en kTam.

2.1 Tile System

In tegenstelling tot voordien, zullen we bij een DNA Tile Assembly spreken over *tiles* in plaats van DNA *complexen*. Een tile type, zie Definitie 2.1, is een vierkant waarbij aan iedere zijde een bond type wordt toegekend. Figuur 2.1 illustreert hoe je een DNA tile kan voorstellen door een DNA *complex*.



Figuur 2.1: Links een DNA *complex met* 4 *sticky ends, opgebouwd uit* 4 DNA*-strengen. Rechts de* DNA *tile die dit* DNA *complex representeert.*

Definitie 2.1: *Zij |Sigma een eindige verzameling bond types. Een* ***tile type*** *is dan een viertuppel* $t = (\sigma_N, \sigma_E, \sigma_S, \sigma_W)$ *over* Σ*, met* $\sigma_N, \sigma_E, \sigma_S, \sigma_W \in \Sigma$*, die respectievelijk de bond types van de boven, rechter, onder en linker zijde van een vierkant voorstellen.*

Merk op dat het toegestaan is om aan verschillende zijden hetzelfde bond type toe te kennen. Gegeven een tile type t, dan gebruiken we de functies $bond_N(t)$, $bond_E(t)$, $bond_S(t)$ en $bond_W(t)$ om respectievelijk de bond types van de boven, rechter, onder en linker zijde van t op te vragen. We veronderstellen dat er van iedere tile type een onuitputtelijke voorraad aanwezig is. Bovendien is het niet toegestaan om tile types te roteren of te plooien, een geroteerd tile type wordt namelijk beschouwd als een ander tile type.

Definitie 2.2: *Een* ***tile*** *is een tuppel* $t = (tt, p)$*, met* tt *een tile type en* $p \in \mathbb{Z}^2$*.*

Een tile wordt gebruikt om de positie van een tile type in een tweedimensionaal raster aan te geven. We maken gebruiken van de **directions** $\mathcal{D} = \{N, E, S, W\}$ om een relatieve positie ten opzichte van een andere positie in het raster aan te geven. Formeel gezien zijn dit functies van de vorm: $(\mathbb{Z}, \mathbb{Z}) \to (\mathbb{Z}, \mathbb{Z})$. Dus $N(i, j) = (i, j + 1)$, $E(i, j) = (i + 1, j)$, $S(i, j) = (i, j - 1)$ en $W(i, j) = (i - 1, j)$. Daarnaast noteren we de inverse directions als volgt: $N^{-1} = S$, $E^{-1} = W$, $S^{-1} = N$ en $W^{-1} = E$. Zij t een *tile*, dan vragen we de positie op met $pos(t)$ en het tile type met $tile(t)$.

Definitie 2.3: *Zij* Σ *een verzameling bond types, dan is* $g : \Sigma \times \Sigma \to \mathbb{Z}$ *een* ***bond strength function****.*

Om aan te duiden hoe sterk twee aangrenzende tile types elkaar aantrekken maken we gebruik van een bond strength function, zie Definitie 2.3. Het resultaat van g noemen we de bond strength. We noemen g **diagonaal** indien $\forall(\sigma, \sigma') \in \Sigma^2 : g(\sigma, \sigma') = 0$ als $\sigma \neq \sigma'$. We zeggen dat twee aangrenzende tile types elkaar aantrekken indien $g(\sigma, \sigma') > 0$, hierbij zijn σ en σ' de bond types van de aangrenzende zijden.

Definitie 2.4: *Een* ***configuratie*** *is een verzameling van tiles. Hierbij is het is niet toegestaan dat twee tiles, binnen eenzelfde configuratie, dezelfde positie hebben.*

Zij A een configuratie, dan gebruiken we $A(i, j)$ om de tile op positie (i, j) aan te duiden. De totale bond strength van een tile t binnen een configuratie A noteren we door

$$\Gamma^A_{\mathcal{D}}(t) = \sum_{d \in \mathcal{D}} g(bond_{d^{-1}}(d(t)), bond_d(t))$$

We mogen $\mathcal{D}$ ook vervangen door $\mathcal{L} \subseteq \mathcal{D}$; in dat geval nemen we enkel de directions van $\mathcal{L}$ in beschouwing.

Definitie 2.5: *Een **tile system** $\mathcal{T}$ is een quadrupel (T, t_s, g, τ), hierbij is g een bond strength function van Σ, T een verzameling tile types over Sigma, $t_s \in T$ een speciale seed-tile en $\tau \in \mathbb{N}$ de temperatuur.*

Definitie 2.6: *Gegeven een tile system $\mathcal{T} = (T, t_s, g, \tau)$, dan definiëren we het begrip **self assembly** als de relatie $A \rightarrow_{\mathcal{T}} B$ tussen twee configuraties A en B. Hierbij zijn A en B identiek aan elkaar met de uitzondering dat B één tile $t \in T$ extra bevat. We noteren $A \rightarrow_{\mathcal{T}} B$ om aan te geven dat A B enkel kan opleveren indien $\Gamma^A_{\mathcal{D}}(t) \geq \tau$.*

Dit wil zeggen dat we t enkel toevoegen aan A indien zijn totale bond strength ten minste τ is en dat er nog geen tile aanwezig was op de positie van t. Op deze manier levert de configuratie A ons B op. Indien de totale bond strength strikt kleiner is dan τ, kan A B niet opleveren (door middel van self assembly). De transitieve sluiting van $\rightarrow_{\mathcal{T}}$ noteren we als $\rightarrow^*_{\mathcal{T}}$.

De definitie van een configuratie houdt er geen rekening mee of twee aangrenzende tiles al dan niet met elkaar verbonden kunnen zijn, alle willekeurige plaatsingen van tiles in een raster zijn namelijk een geldige configuratie. Wij zijn echter geïnteresseerd in een deelverzameling van configuraties, namelijk de configuraties die we kunnen bekomen door enkel self assembly toe te passen vertrekkend vanuit een seed-tile.

Definitie 2.7: *Gegeven een tile system $\mathcal{T} = (T, t_s, g, \tau)$ en de relatie $\rightarrow^*_{\mathcal{T}}$, dan definiëren we $Prod(\mathcal{T}) = \{A \mid \{t_s\} \rightarrow^*_{\mathcal{T}} A\}$. Een configuratie $c \in Prod(\mathcal{T})$ noemen we een **assembly**. $Term(\mathcal{T}) = \{A \in Prod(\mathcal{T}) \text{ en } \nexists B \mid A \rightarrow_{\mathcal{T}} B\}$ bevat alle **terminale assemblies**, dit zijn assemblies die je niet meer kan uitbreiden.*
*We zeggen dat $\mathcal{T}$ A **uniek produceert** indien $Term(\mathcal{T}) = \{A\}$.*

Definitie 2.8: *Een **assembly sequence** $\vec{A}$ over $\mathcal{T} = (T, t_s, g, \tau)$ is een sequentie van tuppels van de vorm (A_n, t_n), hierbij is $A_0 = \{t_0\}$, $t_0 = t_s$ en $A_{n-1} \rightarrow_{\mathcal{T}} A_n = A_{n-1} \cup \{t_n\}$. Als $\vec{A}$ eindig is, noteren we de laatste assembly van $\vec{A}$ als A.*

2.1.1 Unieke productie garanderen

Wanneer je een tile system ontwerpt dat een bepaalde assembly moet genereren, dan is het interessant als je kan aantonen dat het ontworpen tile system deze assembly uniek kan produceren. Het is namelijk mogelijk dat het tile system wel het gewenste assembly kan genereren, maar daarnaast ook nog vele andere assemblies genereert. Om dit aan te kunnen tonen, introduceren we eerst enkele begrippen.

Definitie 2.9: *Gegeven een assembly sequence $\vec{A}$, dan definiëren we volgende verzamelingen van directions voor $\forall i, j \in \mathbb{Z}$, met $t = A(i, j)$:*

- $inputsides^{\vec{A}}(t) = \{D \in \mathcal{D} : t = t_n \text{ en } \Gamma^{A_n}_{\{D\}}(t) > 0\}$
- $propsides^{\vec{A}}(t) = \{D \in \mathcal{D} : D^{-1} \in inputsides^{\vec{A}}(A(D(pos(t))))\}$
- $termsides^{\vec{A}}(t) = \mathcal{D} - inputsides^{\vec{A}}(t) - propsides^{\vec{A}}(t)$

Intuïtief zijn de inputsides de zijden waarmee een tile zich initieel aan andere tiles in een assembly vasthecht. De propsides zijn de zijden waar er in de toekomst andere tiles zullen aan hechten. De overige zijden, als die er zijn, noemen we termsides. Dit zijn de zijden waaraan er nooit een hechting zal plaatsvinden.

Definitie 2.10: *Een eindige assembly sequence $\vec{A}$ van een tile system $\mathcal{T} = (T, t_s, g, \tau)$ noemen we* ***lokaal deterministisch*** *als $\forall i, j \in \mathbb{Z}$, met $t = A(i, j)$ geldt dat:*

1. $\Gamma^{A}_{inputsides^{\vec{A}}(t)}(t) \leq \tau$; *en*
2. $\forall t' \mid type(t') \in T, pos(t') = pos(t) \text{ en } type(t') \neq type(t) \implies \Gamma^{A}_{\mathcal{D}-propsides^{\vec{A}}(t)}(t') < \tau$

Omdat we alle tiles uit een assembly sequence halen, weten we dat voor alle tiles, buiten de seed-tile, de totale bond strength van de inputsides minstens τ zal bedragen. Intuïtief zegt de eerste eigenschap dus dat de totale bond strength van een tile, op het moment dat hij wordt toegevoegd aan een assembly, precies τ moet bedragen. Omdat de seed-tile niet aan deze eigenschap voldoet, deze heeft namelijk geen andere tiles om zich aan te hechten, laten we ook $<$ toe. De tweede eigenschap zegt dat als we een tile verwijderen uit een assembly samen met alle tiles die verbonden zijn via zijn propsides, dat er dan precies één tile type is dat we op een legale manier kunnen toevoegen op diezelfde positie.

In het bewijs van Lemma 2.1.1 werd er in de originele paper [10] impliciet aangenomen dat Stelling 2.1.1 geldt, zonder dit ook effectief aan te tonen. Voor de volledigheid heb ik deze Stelling 2.1.1 zelf opgesteld en bewezen.

Stelling 2.1.1

Gegeven een lokaal deterministische assembly sequence $\vec{A}$ over een tile system $\mathcal{T} = (T, t_s, g, \tau)$, dan geldt voor iedere tile t en voor iedere $D \in \mathcal{D}$ het volgende: $\exists t' : D^{-1}(pos(t')) = pos(t) \implies \Gamma^{\vec{A}}_{D}(t) \geq 0$.

Bewijs: Gegeven een lokaal deterministische assembly sequence $\vec{A}$ over een tile system $\mathcal{T} = (T, t_s, g, \tau)$. Veronderstel dat $t = t_n$ de *eerste* tile is die werd toegevoegd aan $\vec{A}$, waarvoor aan de eisen van Stelling 2.1.1 niet werd voldaan. Dit wil zeggen dat er minstens één direction D bestaat waarvoor $\Gamma^{A^n}_{\{D\}}(t) < 0$. Omdat we t konden toevoegen aan $\vec{A}^n$, weten we dat

$$\Gamma^{A^n}_{\mathcal{D}}(t) \geq \tau \tag{2.1}$$

We hoeven echter geen rekening te houden met de propsides van t, deze directions hadden zich namelijk op het moment van de toevoeging van t nog niet gehecht aan een andere tile. Hierdoor is hun bijdrage in Vergelijking (2.1) steeds 0, we kunnen deze dus herschrijven naar:

$$\Gamma^{A^n}_{inputsides^{\vec{A}}(t) \,\cup\, termsides^{\vec{A}}(t)} \geq \tau \tag{2.2}$$

Omdat de bond strength van een inputside per definitie strikt positief moet zijn en omdat de bond strength van de propsides 0 bedraagt (in $\vec{A}^n$), kunnen we concluderen dat $D \in termsides^{\vec{A}^n}(t)$ moet zijn. Bovendien weten we ook dat de bond strength van eender welke termsides maximaal 0 kan bedragen, anders zou het namelijk een inputside zijn. Dit wil zeggen dat termsides de totale bond strength alleen maar kunnen laten afnemen vanwege de aanname dat er minstens één direction D bestaat waarvoor $\Gamma^{A^n}_{\{D\}}(t) < 0$. Als we dit combineren met Vergelijking (2.2) kunnen we het volgende concluderen:

$$\Gamma^{A^n}_{inputsides^{\vec{A}}(t)} > \Gamma^{A^n}_{inputsides^{\vec{A}}(t) \,\cup\, termsides^{\vec{A}}(t)} \geq \tau \tag{2.3}$$

Omdat $\vec{A}$ lokaal deterministische is, weten we dat

$$\Gamma^{A^n}_{inputsides^{\vec{A}^n(t)}} \leq \tau \tag{2.4}$$

Vergelijking (2.3) en (2.4) leveren echter een contradictie op, hierdoor is hetgeen wat we oorspronkelijk aannamen, namelijk dat er minstens één direction D bestaat waarvoor geldt dat $\Gamma^{A^n}_{D}(t) < 0$, niet waar.

■

***Lemma 2.1.1* [10]**

Gegeven een lokaal deterministische assembly sequence $\vec{A}$ over een tile system $\mathcal{T} = (T, t_s, g, \tau)$, met $g : \Sigma \times \Sigma \to \mathbb{N}$ een bond strength function, dan geldt voor iedere assembly sequence $\vec{A}'$ over $\mathcal{T}$ en voor iedere tile $t' = t'_n \in \vec{A}'$, met $t = A(pos(t'))$, volgende eigenschappen:

1. $inputsides^{\vec{A}'}(t') = inputsides^{\vec{A}}(t)$,
2. $t' = t$.

Bewijs: Veronderstel dat $t' = t'_n$ de *eerste* tile is die werd toegevoegd aan $\vec{A}'$ waarvoor minstens één van bovenstaande eigenschappen niet geldt. Beschouw $D \in inputsides^{\vec{A}'}(t')$. We weten dan dat $t_D = A'(D(pos(t')))$ op een eerder moment dan t' werd toegevoegd aan $\vec{A}'$, bijgevolg is $D^{-1} \notin inputsides^{\vec{A}'}(t_D) = inputsides^{\vec{A}}(t_D)$. Deze laatste gelijkheid geldt omdat t' de eerste tile is waarvoor de eigenschappen van Stelling 2.1.1 niet gelden. Hieruit

volgt dat $D \notin propsides^{\vec{A}}(t)$, want moest D wel tot de $propsides^{\vec{A}}(t)$ behoren, dan zou $D^{-1} \in inputsides^{\vec{A}(t_D)}$ moeten zijn. Omdat dit voor geldt voor alle $D \in inputsides^{\vec{A'}}(t')$ kunnen we afleiden dat

$$inputsides^{\vec{A'}}(t') \cap propsides^{\vec{A}}(t) = \emptyset \tag{2.5}$$

Omdat t' de eerste tile is waarvoor de eigenschappen van Stelling 2.1.1 niet gelden, weten we dat alle tiles in $\vec{A'_n}$, behalve t', ook voorkomen in $\vec{A}$. Hierdoor geldt dat

$$\forall D \in \mathcal{D} \mid \Gamma_D^{\vec{A'_n}}(t') \leq \Gamma_D^{\vec{A}}(t') \tag{2.6}$$

Dit komt omdat g enkel positieve gehele getallen als resultaat geeft, zie Stelling 2.1.1, hierdoor kunnen de extra tiles in $\vec{A}$ de totale bond strength dus enkel maar verhogen. Wegens Vergelijking 2.5 geldt dat

$$\Gamma^{\vec{A}}_{inputsides^{\vec{A'}}(t')}(t') \leq \Gamma^{\vec{A}}_{\mathcal{D}-propsides^{\vec{A}}(t)}(t')$$

Hieruit volgt, in combinatie met Vergelijking 2.6, dat:

$$\Gamma^{\vec{A'_n}}_{inputsides^{\vec{A'}}(t')}(t') \leq \Gamma^{\vec{A}}_{\mathcal{D}-propsides^{\vec{A}}(t)}(t')$$

Wegens eigenschap 2 van Definitie 2.10 weten we dat indien $type(t') \neq type(t)$, dat dan $\Gamma^{\vec{A}}_{\mathcal{D}-propsides^{\vec{A}}(t)}(t') < \tau$. Bijgevolg is $\Gamma^{\vec{A'_n}}_{inputsides^{\vec{A'}}(t')}(t') < \tau$. Dit betekend dat t' enkel aan $\vec{A'}$ kon worden toegevoegd indien $type(t') = type(t)$. Omdat $pos(t') = pos(t)$ kunnen we concluderen dat $t' = t$. Hierdoor voldoen t' aan de tweede eigenschap van de stelling. $\langle\star\rangle$

Veronderstel nu dat $\Gamma^{A}_{inputsides^{\vec{A'}}(t')}(t') > \tau$, t' voldoet dus niet aan de eerste eigenschap van de stelling. Wegens eigenschap 1 van Definitie 2.10 weten we dat $\Gamma^{A}_{inputsides^{\vec{A}}(t')}(t') \leq \tau$ en wegens Definitie 2.6 weten we dat $\Gamma^{A}_{inputsides^{\vec{A}}(t')}(t') \geq \tau$. Bijgevolg is $\Gamma^{A}_{inputsides^{\vec{A}}(t')}(t') = \tau$. Dit kan echter enkel voorkomen indien $\exists D \mid D \in inputsides^{\vec{A'}}(t') - inputsides^{\vec{A}}(t')$. Dit wil zeggen dat $D \notin inputsides^{\vec{A}}(t')$, hieruit volgt dat t_D na t' werd toegevoegd in $\vec{A}$. Omdat t_D verbonden is aan t', is $D^{-1} \in inputsides^{\vec{A}}(t_D)$ en bijgevolg is $D \in propsides^{\vec{A}}(t)$. Maar dit is onmogelijk wegen Vergelijking 2.5. Hierdoor is $inputsides^{\vec{A'}}(t') = inputsides^{\vec{A}}(t)$. Onze aanname was dus fout en t' voldoet dus ook aan eigenschap 1 van de stelling. $\langle\star\star\rangle$

We kunnen dus concluderen wegens $\langle\star\rangle$ en $\langle\star\star\rangle$ dat er geen tile t' kan bestaan indien $\vec{A}$ lokaal deterministische is.

■

***Stelling 2.1.2* [10]**

Als er een terminale lokaal deterministische assembly sequence $\vec{A}$ bestaat over een tile system $\mathcal{T}$, dan zal $\mathcal{T}$ A uniek produceren.

Bewijs: Deze stelling is een rechtstreeks gevolg van Stelling 2.1.1. Deze stelling impliceert namelijk dat als $\vec{A}$ een lokaal deterministische assembly sequence is over $\mathcal{T}$, dat dan voor alle assembly sequence $\vec{A'}$ over $\mathcal{T}$ geldt dat $A' \subseteq A$. Iedere tile uit $\vec{A'}$ komt namelijk ook voor in $\vec{A}$. Hierdoor zal $\mathcal{T}$ dus steeds A opleveren. ∎

2.2 Abstract Tile Assembly Model

In de vorige paragraaf hebben we besproken wat een tile system is. In deze paragraaf zullen we een concrete invulling van een dergelijk tile system bespreken, namelijk aTam.

Definitie 2.11: *Het **Abstract Tile Assembly Model (aTam)** is een tile system $\mathcal{T} = (T, t_s, g, 2)$, waarbij is $g : \Sigma \times \Sigma \to \{0, 1, 2\}$ een diagonale bond strength function is.*

De diagonale eigenschap van g zorgt ervoor dat tiles die *mismatchen* geen invloed hebben op de totale bond strength van een tegel, ze zullen de constructie van een assembly dus noch helpen, noch hinderen. Het resultaat van g is altijd $0, 1$ of 2, we spreken respectievelijk over een **null bond**, **weak bond** en een **strong bond**. Dit betekent dat een tile zich aan een assembly kan hechten indien er een match is bij één zijde met een strong bond of twee zijden met weak bonds.

We kunnen de tile types binnen aTam onderverdelen in de volgende categorieën:

- **seed-tiles:** de initiële assembly is opgebouwd met uitsluitend seed-tiles. Tijdens de simulatie mogen er geen nieuwe seed-tiles meer worden toegevoegd aan de assembly.
- **boundairy-tiles:** dit type tiles heeft een onuitputbare voorraad en wordt gebruikt om te controleren in welke richting de te construeren structuur kan uitbreiden. Deze tiles hebben typisch één zijde waarmee geen enkele tile een binding kan aangaan. Het is dankzij deze tiles dat men de typische L-structuur kan verwezenlijken.
- **regular-tiles:** deze tiles zullen onuitputbaar aanwezig zijn tijdens de constructie van een assembly, maar hebben verder geen speciale eigenschap.

Binnen aTam, kunnen twee tiles zich enkel aan elkaar hechten indien hun bond types **perfect Watson-Crick complementair** zijn. En wanneer twee tiles zich aan elkaar hechten, dan is dit een **permanent** gebeuren, de tiles zullen dus niet meer van elkaar los kunnen komen zonder tussenkomst van externe factoren. In Voorbeeld 2.2.1 illustreren we hoe een assembly wordt opgebouwd binnen aTam.

Voorbeeld 2.2.1: Voordat we onze assembly kunnen opbouwen, moeten we eerst het tile system $\mathcal{T}_{BinairyCounter} = (T, t_s, g, 2)$ beschrijven. We zullen in totaal gebruik maken van 4 bond types, dus $\Sigma = \{0, 1, W, N\}$. We stellen T op als volgt:

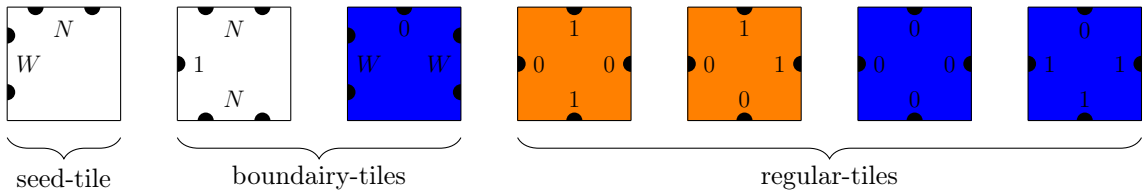
$$T = \{ \quad t_s = (N,null,null,W), \quad (N,null,N,1), \quad (0,W,null,W), \\ (1,0,1,0), \quad (1,1,0,0), \quad (N,null,null,W), \\ (0,0,0,0), \quad (0,1,1,1) \quad \}$$

In Figuur 2.2 vind je de visuele weergave van alle tile types uit T. We construeren g als volgt:

$$W \times W \to 2 \quad N \times N \to 2 \\ 0 \times 0 \to 1 \quad 1 \times 1 \to 1$$

Voor alle andere combinaties van $\Sigma \times \Sigma$ is het resultaat van g 0. Nu zijn we klaar om onze assembly op te bouwen, hierbij beginnen we bij t_s. De seed-tile heeft twee strong bonds, hierdoor kunnen we dus tijdens de eerste iteratie twee boundairy-tiles aan de assembly toevoegen. Tijdens de volgende iteratie kunnen we weer twee boundairy-tiles toevoegen, dit maal aan de boundairy-tiles die we tijdens de vorige iteratie hadden toegevoegd. Dit maal kunnen we echter ook een tile toevoegen op positie $(1, 1)$, dit omdat deze tile twee weak bonds heeft die matchen. Als we op deze manier tiles blijven toevoegen, krijgen we na eindige tijd de assembly in Figuur 2.3. Merk op dat de constructie van de assembly hier niet stopt, deze zal namelijk oneindig lang blijven groeien.

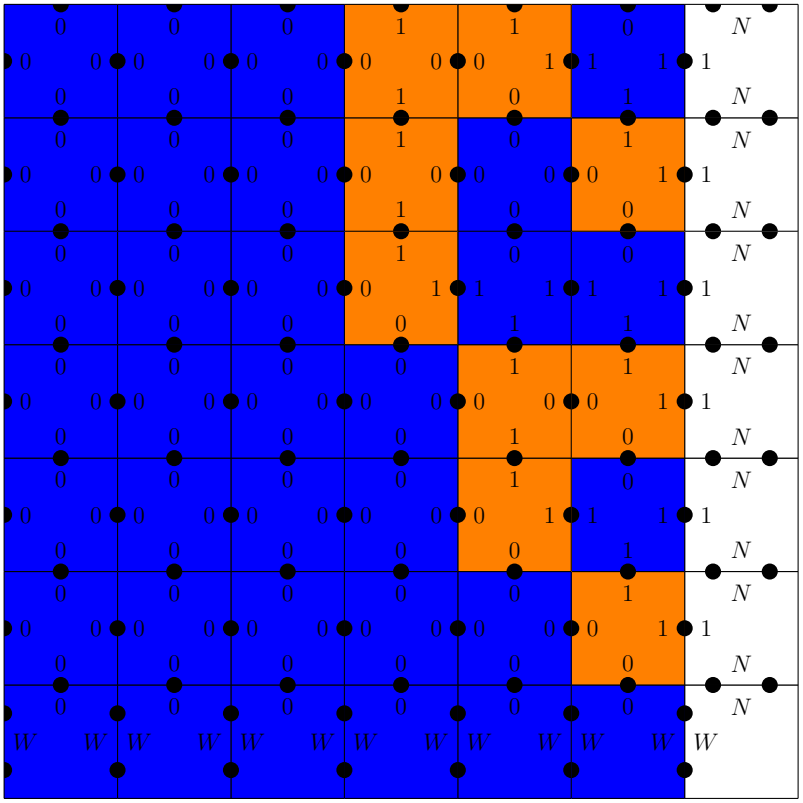
♣



Figuur 2.2: Dit is de visuele weergave van een verzameling tiles die wordt gebruikt om een patroon te genereren dat het binair tellen representeert, zie Figuur 2.3. Het aantal (halve) cirkels op een zijde representeert de bond strength van diezelfde zijde.

2.3 Kinetic Tile Assembly Model

In de vorige paragraaf hebben we aTam besproken. Een probleem met dat model is dat het niet erg realistisch is. In de realiteit kan het voorkomen dat twee tiles zich aan elkaar gaan hechten, ondanks dat hun labels niet 100% Watson-Crick complementair zijn. Deze twee tiles zullen elkaar echter niet zo sterk aantrekken als wanneer ze wel perfect Watson-Crick complementair waren. Er bestaat dus een kans dat tiles terug loskomen van een assembly,



Figuur 2.3: Voor de constructie van deze assembly werden de tiles uit Figuur 2.2 gebruikt. Initiëel was enkel de seed-tile aanwezig, maar door op een iteratieve manier tiles toe te voegen kan men na eindige tijd deze assembly bekomen. Wanneer je blauwe tiles als een 0 *beschouwd en de oranje tiles als een* 1 *kan je zien dat iedere rij een binair getal representeert. Meer concreet stelt de* n*-de rij het binaire getal* n *voor.*

iets wat bij aTam niet toegestaan is. De kans dat een bepaald tile type zich zal hechten aan een assembly staat in verband met de aanwezige concentratie tiles die van dat tile type zijn. Met deze verschillen in het achterhoofd definiëren we kTam als volgt:

Definitie 2.12: *Het **Kinetic Tile Assembly Model (kTam)** is een hextuppel (T, t_s, g, G_{mc}, G_{se}, k_f), hierbij is T een verzameling tile types, $t_s \in T$ een speciale seed-tile, $g : \Sigma \times \Sigma \to \{0, 1, 2\}$ een diagonale bond strength function, $G_{mc} > 0$ de entropische kost om een tile op een bepaalde positie te plaatsen, $G_{se} > 0$ de kost die nodig is om één sticky end lost te koppelen en k_f de forward rate constante.*

Net zoals bij aTam mogen bij kTam enkel alleenstaande tiles worden toegevoegd aan een assembly, het verwijderen gebeurd ook tile per tile. Wanneer een tile wordt toegevoegd aan een assembly, dan gebeurt dit volgens een associatie ratio r_f, ook wel forward rate genoemd. Analoog gebeurt het verwijderen van een tile volgens een dissociatie ratio $r_{r,b}$, ook wel reverse rate genoemd, hierbij stelt b een bond type voor. Op iedere positie aan de buitenrand van een assembly mag ieder tile type worden toegevoegd aan de assembly, ongeacht of de zijden een match opleveren of niet.

Het forward rate wordt bepaald door de aanwezige hoeveelheid tiles van een bepaald tile type t, dit noteren we als $\lceil t \rceil$. Hierdoor kunnen we het forward rate schrijven als:

$$r_f = k_f \lceil t \rceil = k_f e^{-G_{mc}} \tag{2.7}$$

hierbij is $G_{mc} > 0$ de entropische kost (energie) om een tile om een tile toe te voegen aan een assembly. Voor de eenvoud veronderstellen we in kTam dat de concentratie tiles $\lceil t \rceil = e^{-G_{mc}}$ constant blijft. Hieruit volgt, wegens Vergelijking 2.7, dat het forward rate ook een constante is.

Het reverse rate wordt bepaald door de bond strength b van de zijde waarbij de tile zich probeert vast te hechten aan de assembly. We schrijven het reverse rate als volgt:

$$r_{r,b} = k_f e^{-bG_{se}} \tag{2.8}$$

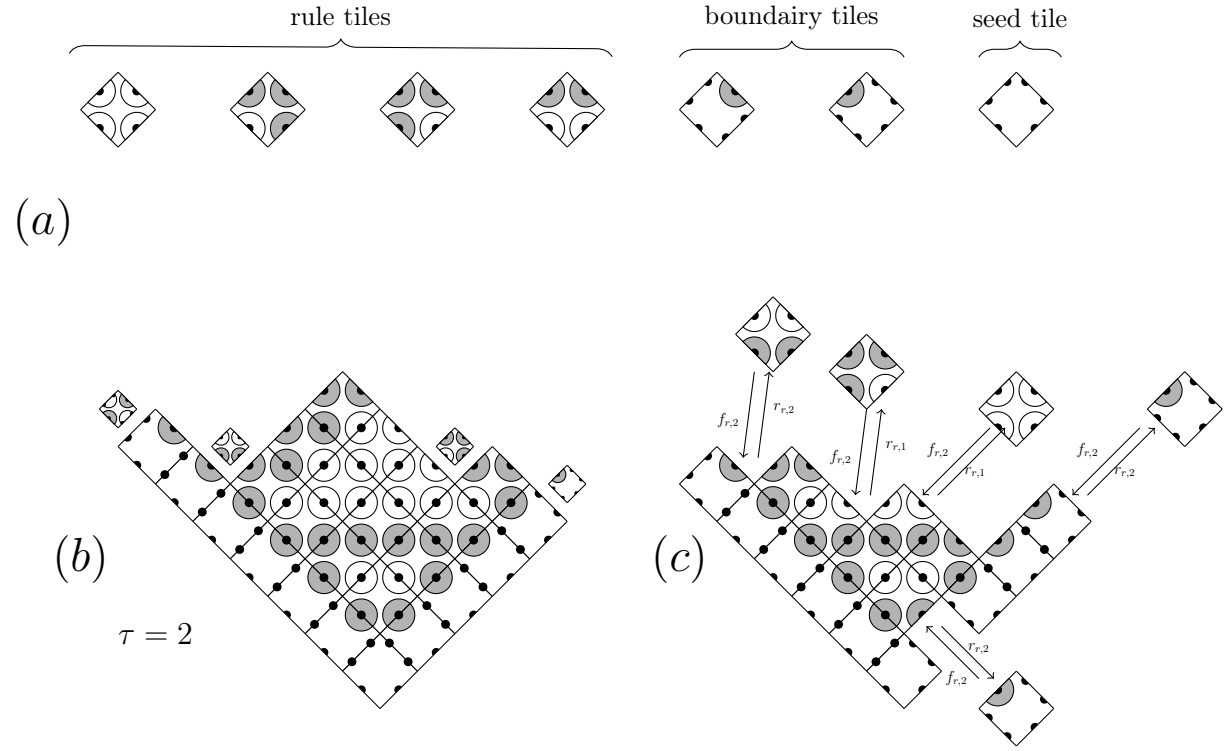
hierbij is $G_{se} > 0$ de kost (energie) die nodig is om één sticky end lost te koppelen waarbij b de bond strength van dat sticky end is.

De manier waarop een assembly wordt geconstrueerd bij aTam staat volledig in functie van de temperatuur τ. kTam heeft, in tegenstelling tot aTam, geen parameter voor de temperatuur. We kunnen deze parameter echter wel benaderen door het ratio tussen de concentratie van tiles en de kracht van de bond strength van die tiles te nemen, of kort $\tau = G_{mc}/G_{se}$. Indien we dit ratio zouden verlagen, wat intuïtief overeenkomt met het verlagen van de temperatuur in aTam, dan zal er sneller een assembly geconstrueerd worden. Het nadeel van deze snelheidswinst is dat de kans op fouten op deze manier groeit.

Veronderstel dat ε het error rate is per tile. Dit wil zeggen dat een tile een kans ε heeft om een foutieve hechting aan een assembly te maken. Afhankelijk van welke waardes je kiest voor G_{mc} en G_{se} kan je het error rate proberen te controleren. Maar het verlagen van het error rate heeft als gevolg dat de assembly veel trager zal groeien. Het is dus interessant

om een goede trade-off te kiezen tussen snelheid en het error rate. Deze trade-off wordt uitgedrukt door het **growth rate** r. In Paragraaf 4.1 zullen we onder specifieke condities een concrete waarde voor r uitrekenen.

Om een idee te krijgen wat nu een goede waarde is voor G_{mc} en G_{se} bespreken we een experiment dat Winfree[12] heeft uitgevoerd, op een tile set die de Sierpinski driehoeken construeert, om zo het gedrag van kTam te bestuderen, zie Figuur 2.4.



Figuur 2.4: Figuur (a) toont de regular-tiles, Figuur (b) de boundairy-tiles en seed-tiles voor de Sierpinski driehoeken, Figuur (c) de assembly geproduceerd met deze tiles volgens aTam met $\tau = 2$ en Figuur (d) de assembly geproduceerd met kTam.

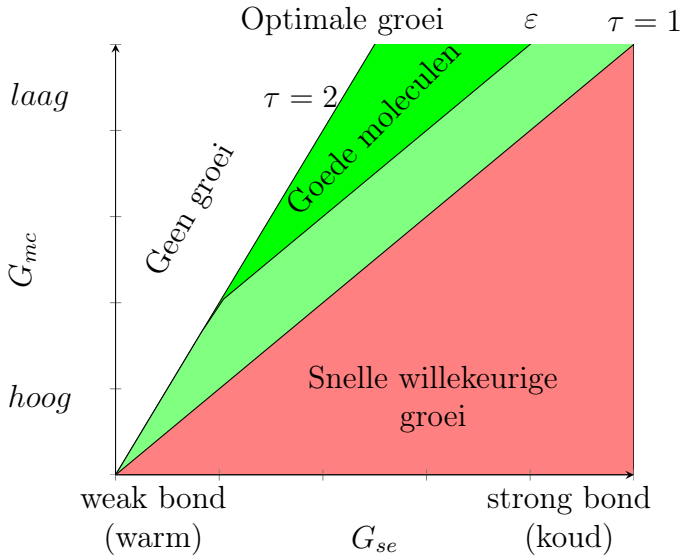
In Figuur 2.5 vind je een overzicht van het aantal fouten dat er optreed voor verschillende waardes van G_{mc} en G_{se}. We kunnen een onderscheid maken tussen 3 situaties:

- $G_{mc} < 2G_{se}$: in deze situatie treed er geen self assembly plaats, er wordt dus geen assembly gevormd. Dit is logisch omdat de tile set is ontworpen om te werken voor $\tau = 2$. Door $G_{mc} < 2G_{se}$ te kiezen moeten de bond strength méér dan 2 bedragen opdat er self assembly kan plaatsvinden.

- $2G_{se} > G_{mc} > G_{se}$: in deze situatie zal er self assembly plaatsvinden. Dit komt omdat $r_f > r_{r,2}$ voor tiles waarvan we verwachten dat ze zullen hechten aan een assembly, maar voor tiles waarvoor we wensen dat ze niet hechten aan een assembly geldt $r_f < r_{r,1}$. Het is echter nog steeds mogelijk dat er zich enkelen fouten zullen

voordoen. Het error rate per tile noteren we door $\varepsilon = 2Ne^{-G_{se}}$, met $N+1$ het aantal tiles.

- $G_{se} > G_{mc}$: in deze situatie zal er wel *een* assembly gevormd worden, maar alles gebeurd willekeurig en de kans dat het gewenste resultaat geconstrueerd wordt, in dit geval de Sierpinski driehoeken, is nagenoeg onbestaand.



Figuur 2.5: Deze figuur illustreert de groei van een assembly bij verschillende waarden voor G_{mc} en G_{se}.

2.4 Type errors

Om dit hoofdstuk af te sluiten geven we kort nog een overzicht van het type errors dat kan optreden tijdens self assembly. In de volgende hoofdstukken zullen we dan technieken bespreken over hoe we enkele van deze types errors kunnen verminderen of zelfs volledig verhelpen.

- **Growth error:** dit type van error houdt in dat er een tile zich zal hechten aan een assembly, ondanks dat dit geen geldige toevoeging is. Dit kan bijvoorbeeld een tile zijn die zich toevoegt aan een assembly, hoewel zijn totale bond strength over zijn inputsides slechts 1 bedraagt en $\tau = 2$ is.

- **Gross damage:** dit type van error houdt in dat er, door een externe kracht, één of meerdere tiles vanuit de assembly worden verwijderd.

- **Backward growth:** dit type van error houdt in dat er een foutieve assembly wordt geconstrueerd doordat tiles worden toegevoegd aan een assembly tegen hun groeirichting in. Deze errors nemen typisch plaatst na het optreden van gross damage.
- **Nucleation errors:** dit type van error houdt in dat er assemblies ontstaan, zonder de aanwezigheid van een seed-tile.
- **Facet roughening errors:** dit type van error houdt in dat een tile zich met precies één weak bond hecht aan een assembly op een willekeurige positie. Vervolgens moet deze tile aan de basis liggen van een oneindige reeks van andere foutieve tiles die worden toegevoegd aan de assembly.

Hoofdstuk 3

Self-healing

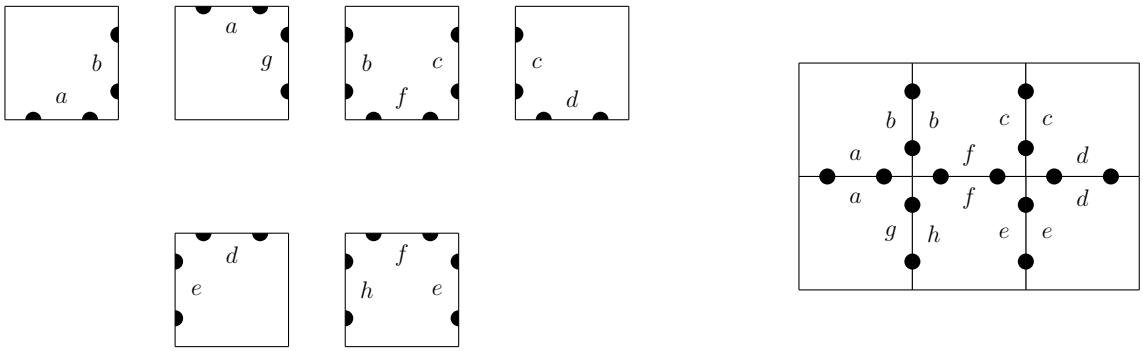
In Hoofdstuk 2 hebben we enkele types van errors besproken die kunnen optreden tijdens self assembly in aTam en kTam. In dit hoofdstuk bespreken we een techniek die een van deze errors, namelijk gross damage, kan wegwerken door iedere tile te vervangen door een $N \times N$ block van tiles. Deze nieuwe set van tiles zullen we self-healing tiles noemen [13, 5].

Definitie 3.1: *We noemen een tile system $\mathcal{T}$* ***self-healing*** *(in aTam) als voor iedere assembly A over $\mathcal{T}$ volgende eigenschap geldt: Indien een willekeurig aantal tiles verwijderd worden uit A zodat de overgebleven tiles allemaal nog verbonden zijn aan de seed-tile, dan wordt herstel van de assembly gegarandeerd zodat uiteindelijk iedere tile die was verwijderd terug toegevoegd wordt aan de assembly zonder dat er fouten zullen optreden.*

Wanneer er gross damage optreed in een assembly, dan kan het voorkomen dat de assembly wordt opgedeeld in meerdere kleinere assemblies. Je zou misschien verwachten dat iedere van deze kleinere assemblies in staat is om zichzelf te herstellen, maar deze aanname is niet erg realistisch omdat sommige deeltjes heel erg klein kunnen zijn (bijvoorbeeld 1 of 2 tiles groot). We weten echter wel dat de seed-tile in staat is om de hele assembly te (her)construeren, op voorwaarde dat tiles in de juiste volgorde worden toegevoegd aan de assembly. Daarom zijn we tevreden als we self-healing kunnen garanderen voor de deel-assembly waarin de seed-tile nog aanwezig is.

Een vraag die je jezelf nu kan stellen is of het wel mogelijk is om een self-healing tile set te ontwikkelen. Het antwoord op deze vraag is gelukkig ja. Een heel eenvoudig voorbeeld is een tile set waarbij iedere bond type uniek wordt toegewezen aan de twee tile types die aan elkaar moeten worden gehecht en waarbij er enkel gebruik wordt gemaakt van strong bonds, zie Figuur 3.1 voor een voorbeeld.

In de rest van dit hoofdstuk zullen we eerst een 3×3 block transformatie voorstellen die kan toegepast worden op een L-BCA tile set, dit is een tile set die een assembly produceert met een L-structuur. Vervolgens zullen we aantonen dat de nieuwe gegenereerde tile set inderdaad self-healing is. Vervolgens stellen we een 5×5 transformatie voor die werkt voor vele, maar niet allemaal, lokaal deterministische tile sets. Tot slot zien we een 7×7 transformatie die werkt indien we toestaan dat er deel-assemblies, die gevormd werden



Figuur 3.1: Links een self-healing tile set en rechts de assembly die deze tile set produceert.

zonder de aanwezigheid van een seed-tile, worden toegevoegd aan de assembly met seed-tile tijdens het genezing proces.

3.1 Self-healing transformatie voor L-BCA tile sets

In deze paragraaf beschrijven we een transformatie, die kan worden toegepast op L-BCA tile sets, om zo een equivalente tile set te produceren die self-healing is. Voordat we deze transformatie bespreken leggen we eerst kort uit wat een L-BCA tile sets is.

Definitie 3.2: *Een* L-BCA *tile set is een tile set waarbij de tiles aan een specifieke vorm voldoen:*

- ***regular-tile**: deze tile heeft twee strong bonds die langs elkaar moeten liggen. De bond types van de strong bonds moeten matchen met de bond types van een boundairy-tile. Zijn overige twee zijden hebben null bonds.*
- ***boundairy-tile**: deze tiles hebben ook twee strong bonds, maar deze moeten echter tegen over elkaar liggen. Verder heeft deze tile nog één weak bond en één null bond.*
- ***seed-tile**: deze tiles hebben 4 weak bonds.*

De verzameling inputsides van iedere tile is uniek, er zijn dus geen twee tiles die dezelfde inputsides hebben. Bovendien liggen de inputsides van de tiles van een L-BCA *tile set vast als volgt:*

- ***regular-tile**: deze tile heeft 2 inputsides, namelijk E en S, met $E, S \in \mathcal{D}$.*
- ***boundairy-tile**: deze tile heeft 1 inputside. Dit is ofwel E, ofwel S, met $E, S \in \mathcal{D}$.*
- ***seed-tile**: deze tile heeft geen inputsides.*

Lemma 3.1.1

Iedere L-BCA *tile set is lokaal deterministisch.*

Bewijs: Dit lemma volgt rechtstreeks uit Definitie 3.2. Omdat een regular-tile enkel weak bonds heeft en omdat hij precies 2 inputsides heeft weten we dat zijn totale bond strength precies 2 bedraagt op het moment dat hij wordt toegevoegd aan een assembly. Een boundairy-tile heeft slechts 1 inputside. Veronderstel dat dit een strong bond is, dan kan zijn totale bond strength maximaal 2 bedragen. En een seed-tile heeft geen inputsides, dus zijn initiële bond strength is steeds 0. De L-BCA tile set voldoet dus al aan punt 1 van Definitie 2.10.

Het bewijs van punt 2 van Definitie 2.10 volgt uit het feit dat ieder paar van inputsides uniek toebehoort aan 1 tile. Dit betekend dat er minstens 1 zijde, uit de inputsides van de correcte tile, zal zijn die een mismatch oplevert. Op deze manier wordt de totale bond strength strikt kleiner dan die van de correcte tile. En omdat die zijn totale bond strength maximaal τ bedrag is de totale bond strength van een andere tile strikt kleiner dan τ. Merk op dat de bond strength van een termsides steeds 0 bedraagt, want moest dit meer zijn dan was deze zijde ofwel een inputside, ofwel een propsides geweest. Een termsides beïnvloed het resultaat dus niet. ∎

Stelling 3.1.1

De klasse van L-BCA *tile sets is Turing Universal.*

Bewijs: Voor het bewijs van Stelling 3.1.1 is het voldoende om een constructie te geven die een BCA, geconstrueerd volgens de constructie uit Bewijs 1.4.1, omzet naar een L-BCA tile set. Gegeven een BCA b, dan construeren we een equivalente L-BCA tile set als volgt:

- We construeren een seed-tile t_s, zie Figuur 3.2. Deze heeft twee strong bonds met L en R als hun bond types. Deze bond types zullen we gebruiken bij de boundairy-tile om zo de L-structuur te kunnen creëren.

- We construeren twee boundairy-tiles t_{b1} en t_{b2}. Eén van deze tiles heeft L als zijn bond type en de andere heeft R als zijn bond type. Deze tiles zorgen ervoor dat we de L-structuur creëren.

- Veronderstel dat $w = i_1, \ldots, i_k$ de invoerstring is waarop b moeten worden uitgevoerd. Dan construeren we volgende tiles op basis van w.

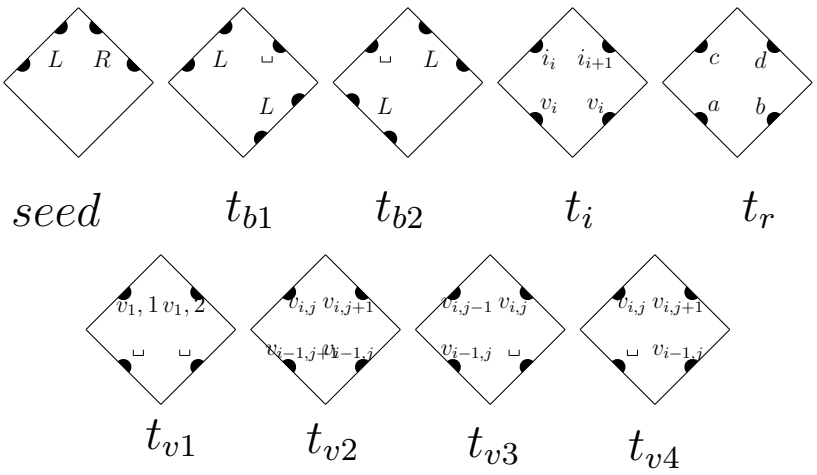
– Op ieder niveau wordt de assembly 1 tile breder. Omdat w uit k symbolen is opgebouwd, moeten we eerst $k/2+2$ niveau's construeren, zodat we op niveau k, k tiles kunnen toevoegen waarin we w coderen. Tile t_{v1} uit Figuur 3.2 illustreert hoe de tile voor het eerste niveau er moet uitzien. Voor niveau $l+2$ construeren we l tiles en @ boundairy-tiles. Tile t_{v3} en t_{v4} beschrijven hoe de twee uiterste tiles, die zich hechten aan de boundairy-tiles, eruit moeten zien. Voor de overige $l-2$ tiles gebruiken we het patroon van tile t_{v2} uit Figuur 3.2. Merk op dat het belangrijk is dat al deze tiles unieke bond types krijgen om te kunnen garanderen dat de inputsides van iedere tile uniek blijven.

– Tile t_i uit Figuur 3.2 geeft het patroon weer van hoe we de invoer w gaan verwerken in tiles. We creëren zo een tile voor ieder paar symbolen (i_j, i_{j+1}) in w, met j een oneven getal.

In Figuur 3.3 vind je een voorbeeld van hoe een assembly er moet uitzien nadat je er een invoer string in hebt verwerkt.

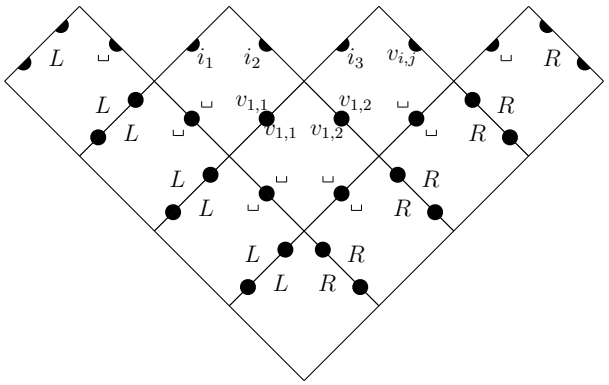
- Tot slot creëren we een tile voor iedere rule $r = (a, b) \to (c, d)$ uit b. Tile t_r uit Figuur 3.2 geeft het patroon weer dat deze tiles moeten aannemen.

Tot slot merken we op dat alle tiles unieke inputside hebben. Hierdoor zal steeds de correcte rule tile worden gebruikt tijdens de constructie van de assembly. Ook zal tile steeds verbonden zijn met propsides van *twee* andere tiles. Dit gedrag komt overeen met het alterneren van de Margolus indelingen tijdens de uitvoering van een BCA. Hieruit volgt dat de assembly equivalent is aan de uitvoer van b op invoer w. ∎



Figuur 3.2: Deze figuur toont de tiles die nodig zijn voor de constructie van een L-BCA *tile set op basis van een* BCA*.*

De moeilijkheid van self-healing ligt in het feit dat een tile set vaak ontworpen is om te groeien in één bepaalde richting, dit betekent dat de inputside en propsides van een



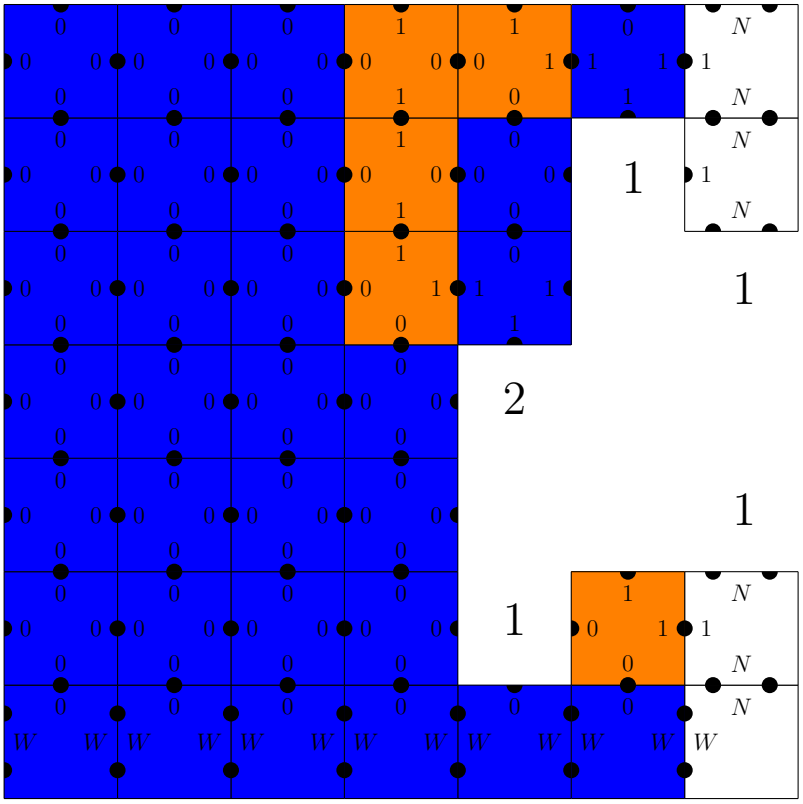
Figuur 3.3: Deze figuur toont een assembly van een L-BCA *tile set waarin de invoer string* $w = i_1 i_2 i_3$ *in is verwerkt.*

bepaalde tile al op voorhand vastliggen. Maar desondanks dat de originele tile set een assembly construeerde op een deterministische manier, is het mogelijk dat diezelfde tile set geen deterministische groei meer kan garanderen vanaf het moment dat de groeirichting veranderd. Dit komt omdat het veranderen van de groeirichting de inputside en propsides van een tile kan doen veranderen, zie Figuur 3.4.

Voorbeeld 3.1.1: We hernemen de tiles uit Figuur 2.2, herinner je dat deze tiles een patroon construeren dat binair tellen representeert en merk op dat dit een L-BCA tile set is. In Figuur 3.4 wordt een assembly over deze tile set weergegeven waar gross damage op heeft plaatsgevonden. De cijfers op de lege plaats duiden aan hoeveel type tile er op die plaats, op een geldige manier, kunnen worden toegevoegd aan de assembly. Op de plaats waar een 2 staat, is het dus mogelijk dat de assembly foutief zal terug groeien. Deze tile set is dus niet self-healing. ♣

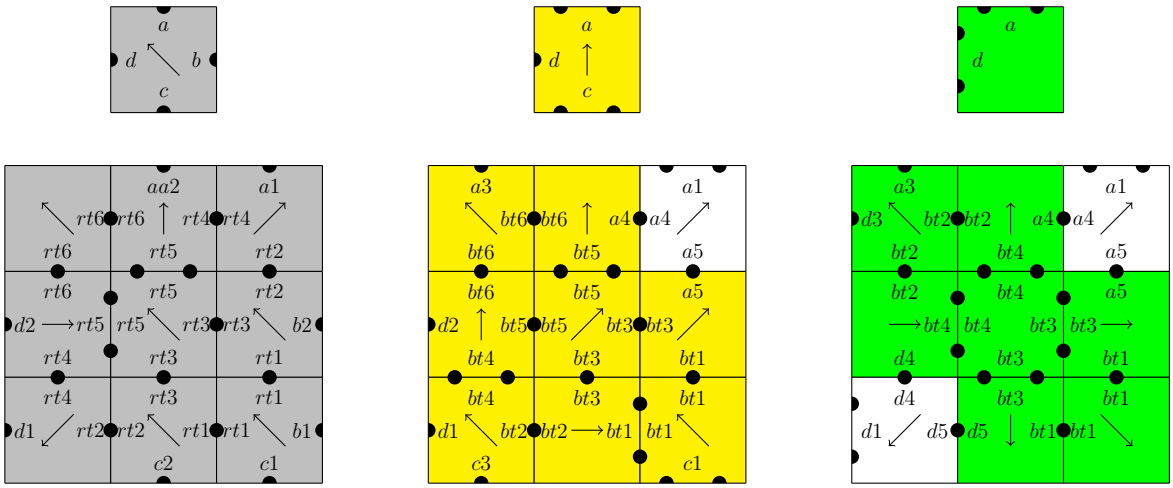
Als we een self-healing tile set willen gaan ontwerpen hebben we twee keuzes op bovenstaand probleem te verhelpen. We kunnen er enerzijds voor zorgen dat er genoeg informatie in de tiles aanwezig is zodat correcte groei mogelijk is vanuit iedere richting. Anderzijds kunnen we ervoor zorgen dat groei slechts mogelijk is in één richting.

Om achterwaartse groei tegen te gaan is het noodzakelijk dat we null bonds plaatsen op specifieke plaatsen om zo de achterwaartse groei te blockkeren. In Figuur 3.5 een 3×3 block transformatie voor een regular-tile, boundairy-tile en seed-tile. Voor iedere tile type in de originele tile set worden dus 9 nieuwe tiles geconstrueerd. De bond types die werden gebruikt voor deze nieuwe tiles zijn geïndexeerde varianten van de bond types (bv. $a5$) of tile types (bv. $t3$) van de originele tile. We noemen deze nieuwe bond types respectievelijk **bond-type bonds** en **tile-type bonds**. Nieuwe tiles die *minstens* één tile-type bond gebruiken noemen we **block tiles**. Block tiles krijgen dezelfde kleur als de originele tile. Tiles die echter uitsluitend gebruik maken van bond-type bonds noemen we **bond tiles**, deze tiles krijgen geen kleur. Het is toegestaan dat de transformaties van verschillende tiles dezelfde bond tiles construeren. Voordat we bewijzen dat deze transformaties een self-healing tile set opleveren, introduceren we eerste enkele stellingen die ze zullen gebruiken



Figuur 3.4: Deze figuur toont een assembly waarop gross damage heeft plaatsgevonden. Deze assembly werd opgebouwd met de tiles uit Figuur 2.2. De cijfers op de lege plaats duiden aan hoeveel tile types er op die plaats, op een geldige manier, kunnen worden toegevoegd aan de assembly.

in dat bewijs.



Figuur 3.5: Links in het grijs een regular-tile, in het midden in het geel een boundairy-tile en rechts in het groen een seed-tile met onderaan respectievelijk hun 3×3 block transformatie. De pijltjes geven aan, door in de andere richting te wijzen, welke zijden de inputside van een tile zijn.

***Lemma 3.1.2* [13]**

Als een tile toegevoegd kan worden op een bepaalde plaats in een assembly A, dan kan diezelfde tile ook worden toegevoegd op dezelfde positie van een grotere assembly die dezelfde tiles bevat als A plus nog enkele andere tiles. Dit op voorwaarde dat de positie nog open is.

Bewijs: Een tile mag toegevoegd worden aan een assembly in aTam indien de totale bond strength minstens τ bedraagt. Omdat aTam geen negatieve bond strength toelaat, is het onmogelijk dat het toevoegen van extra tiles aan een assembly ervoor kan zorgen dat de totale bond strength zal afnemen. Dus indien een tile t kan toegevoegd worden aan een assembly A, dan kan deze ook worden toegevoegd aan een assembly A' als deze minstens dezelfde tiles als A bevat. Want de totale bond strength van t neemt ofwel toe, of deze blijft hetzelfde, door nieuwe tiles toe te voegen aan de A'. Hierdoor mag t ook worden toegevoegd aan A'. ∎

***Lemma 3.1.3* [13]**

Beschouw een assembly A die werd geproduceerd over een tile set T, volgens de regels van aTam. Verwijder nu een willekeurige tile, verschillend van de seed-tile, uit A als een test. De test zal slagen indien er precies één tile in T

***Lemma 3.1.3* [13]**

correct kan worden toegevoegd aan de assembly, volgens de regels van aTam. T is een self-healing tile set als en slechts als deze test slaagt voor iedere tile over iedere mogelijke assembly die kan worden geproduceerd met T.

Bewijs: De eerste implicatie is makkelijk in te zien. Indien er een test bestaat die faalt, dan is de tile set niet self-healing. Dit betekend namelijk dat het mogelijk is dat een beschadigde assembly op meerdere manieren zou kunnen herstellen.

Voor de omgekeerde implicatie veronderstellen we dat de tile set niet self-healing is, we gaan nu aantonen dat er een test bestaat die faalt. Dat de tile set niet self-healing is betekend dat er een assembly A bestaat waaraan schade is toegebracht, en dat er herstel heeft plaatsgevonden waarbij t de eerste foutieve tile is die werd toegevoegd aan A tijdens dat herstel. Alle tiles die we voor t hadden toegevoegd waren echter wel correct, we noemen deze assembly A'. We breiden A' nu uit door alle tiles die verwijderd werden tijdens de schade aan A, behalve t, toe te voegen aan A'. De resulterende assembly noemen we A''. We hebben nu dus een assembly geconstrueerd die identiek is aan A, behalve op de positie van t. Wegens Lemma 3.1.2 weten we dat we t mogen toevoegen aan A'', maar de originele, juiste, tile mogen we echter ook toevoegen aan A'' op de positie van t, want A is ook een correcte assembly. We hebben dus een test gevonden die faalt. ∎

We zijn nu klaar om te bewijzen dat de 3×3 block transformatie werkt voor alle L-BCA tile set. Het bewijs van Stelling 3.1.2 gebeurd in twee stappen. Eerst tonen we aan dat de nieuwe tile set het correcte patroon genereert, maar dan 3 maal zo groot als voordien. En ten tweede dat iedere test, zoals beschreven in Lemma 3.1.3, altijd zal slagen.

***Stelling 3.1.2* [13]**

De 3×3 block transformatie, beschreven in Figuur 3.5, produceert een self-healing tile set wanneer deze wordt toegepast op een L-BCA *tile set tile set (1). Bovendien zal de nieuwe tile set het zelfde patroon construeren als de originele tile set, maar dan 3 maal zo groot. Verder zal de overheersende kleur van het 3×3 block overeenkomen met de kleur van de originele tile (2).*

Bewijs: Voor het bewijs van bewering (1) is het voldoende om een lokaal deterministische assembly sequence te identificeren. Wegens Lemma 3.1.1 weten we dat er een lokaal deterministische assembly sequence $\vec{A}$ bestaat voor de originele tile set. We moeten nu aantonen dat we $\vec{A}$ kunnen omzetten een lokaal deterministische 3×3 block tile set.

We weten dat $\vec{A}$ is opgebouwd, door tiles in een vaste volgorde toe te voegen aan $\vec{A}$, beginnend bij de seed-tile. We tonen nu aan via inductie dat we de nieuwe assembly $\vec{A}'$

kunnen construeren op een lokaal deterministische manier door de 3×3 blocks in dezelfde volgorde toe te voegen aan $\vec{A}'$ als hun overeenkomstige tile t werd toegevoegd aan $\vec{A}$.

Basisstap: De eerste tile die aan $\vec{A}$ werd toegevoegd is de seed-tile. We moeten nu op een lokaal deterministische manier het 3×3 block toevoegen aan $\vec{A}'$. In Tabel 3.1 wordt geïllustreert hoe je het seed-tile block op een lokaal deterministische manier kan opbouwen. Je kan namelijk eenvoudig nagaan dat de totale bond strength van de inputside van alle afzonderlijke tiles nooit 2 overschrijdt.

Inductiestap: We veronderstellen nu dat de assembly A^n equivalent is aan A'^n en dat we A'^n op een lokaal deterministische manier hebben kunnen opbouwen.

Inductiehypothese: Stel dat t de tile is die A^n uitbreid naar A^{n+1}. Dan is b_t het 3×3 block dat aan A'^n moet worden toegevoegd om A'^{n+1} te bekomen. Omdat de inputsides van b_t de zelfde zijn als die van zijn corresponderende tile t en omdat de originele tile set lokaal deterministisch is, kunnen we concluderen dat er maar 1 3×3 block op dezelfde positie als die van t kan geplaatst. In Tabel 3.1 wordt geïllustreert hoe je zowel een regular-tile block, als een boundairy-tile block, op een lokaal deterministische manier kan opbouwen. Je kan eenvoudig nagaan dat de totale bond strength van de inputside van alle afzonderlijke tiles nooit 2 overschrijdt. We kunnen dus concluderen dat we A'^{n+1} op een lokaal deterministische manier kunnen construeren.

Hierbij hebben we bewering (1) aangetoond.

9	6	8	9	7	8	9	4	8
5	4	3	4	5	6	3	1	5
7	2	1	3	2	1	7	2	6
regular-tile			boundairy-tile			seed-tile		

Tabel 3.1: Deze tabel illustreert in welke volgorde je de 3×3 blocken op een lokaal deterministische manier kan opbouwen, indien de input blocks reeds aanwezig zijn. Het getal 1 duid de positie aan van de eerste tile die wordt toegevoegd, ...

Voor het bewijs van bewering (2) controleren we voor iedere tile in ieder 3×3 block alle mogelijke combinaties van zijden waarvan de totale bond strength minstens 2 bedraagt. Vervolgens controleren we voor deze zijden of er precies één tile bestaat die matched aan deze zijdes. Als er 1 combinatie van zijdes wordt gevonden waarvoor dit niet geldt, dan kunnen we op 1 positie meerdere tiles toevoegen en bijgevolg geldt bewering (2) dan niet meer.

Indien er een combinaties van inputsides bestaat waarvoor de totale bond strength 2 bedraagt en waarvoor minstens 1 inputside intern in het 5×5 block, dan weten we dat deze tile uniek op die positie kan worden ingevoegd omdat alle interne bond types uniek uitgekozen woorden voor iedere tile. Hierdoor hoeven we deze combinaties in wat volgt

niet meer controleren. We maken nu een onderscheid tussen de seed-tile, regular-tiles en boundairy-tiles en testen de overige combinaties afzonderlijk:

- **regular-tile block:** In dit block is er slechts 1 tile die een totale bond strength van 2 heeft aan de buitenkant van het block, namelijk de tile rechts onderaan in de constructie. Deze tile heeft S en E als zijn inputsides. Maar omdat alle regular-tiles S en E als inputsides hebben en omdat alle combinaties van inputsides, uit de originele tile set, uniek toebehoren aan 1 tile weten we dat deze tile ook uniek is. Bijgevolg kan deze geen problemen opleveren.

- **boundairy-tile block:** In dit block zijn er 3 tiles die een totale bond strength van 2 hebben, door enkel zijden aan de buitenkant van het block in beschouwing te nemen. De eerste tile is de bond tile, maar deze tile is zijn labels zijn uniek per bond type (en dus niet per tile!). Hierdoor hebben tiles met hetzelfde bond type dus dezelfde bond tile. Hierdoor past deze bond tile uniek op deze positie. De tweede tile is de tile rechts onderaan. Maar deze is ook uniek omdat zijn inputsides overeen komt met de inputsides van de originele tile. En omdat alle combinaties van inputsides uniek toebehoren aan 1 tile weten we dat deze tile uniek is. Bijgevolg kan deze geen problemen opleveren. De derde tile is de tile links onderaan en deze levert om dezelfde reden als de vorige tile geen problemen op.

- **seed-tile block:** Dit block heeft 3 tiles wiens totale bond strength 2 bedraagt aan de buitenkant van het block. De eerste twee tiles zijn de bond tile. Maar deze zijn uniek voor per bond type en dus uniek, hierdoor leveren ze dus geen probleem op. De laatste tile is de tile links bovenaan. Maar omdat deze tile enkel aangrenst aan beide de horizontale als de verticale boundairy-tiles, die beide unieke labels hebben die enkel voor de boundairy-tiles gebruikt worden, zorgt ervoor dat ook deze tile uniek is.

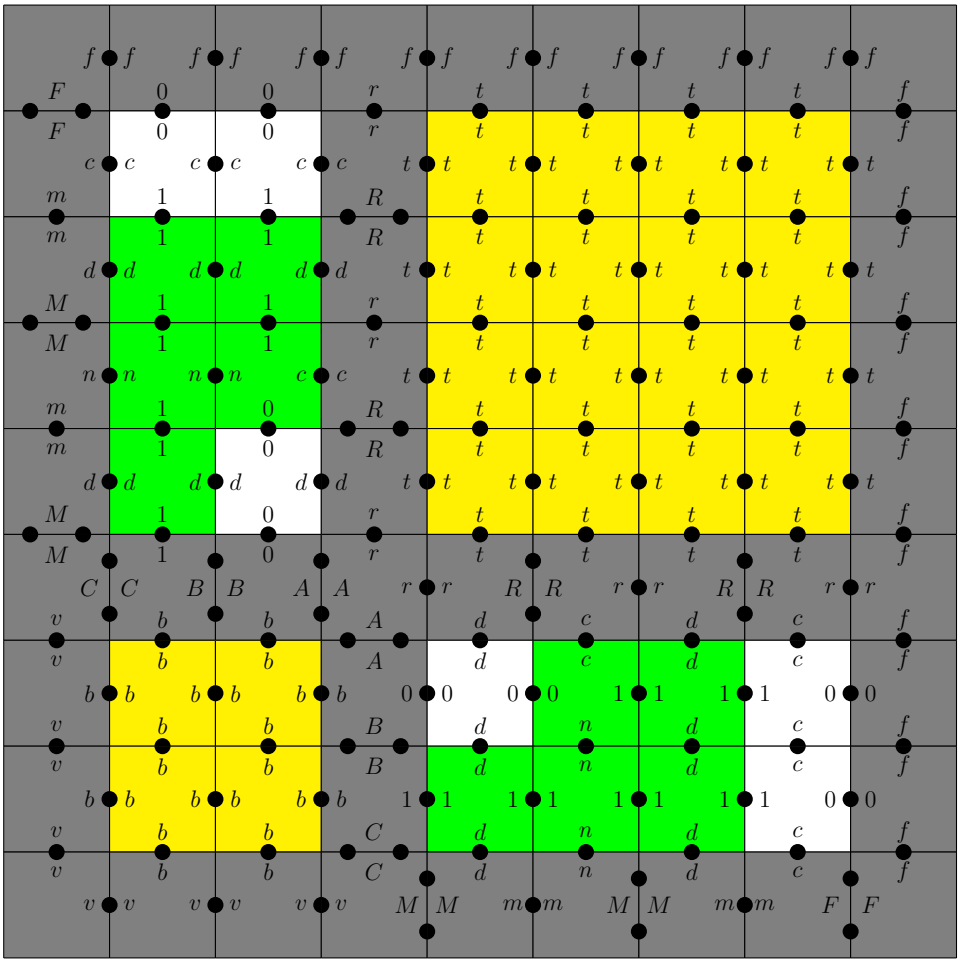
We hebben dus aangetoond dat op iedere positie een unieke tile wordt geplaatst. Hierdoor zal de test uit Lemma 3.1.3 steeds slagen. Hierbij hebben we bewering (2) aangetoond en hierbij is het bewijs afgerond.

■

3.2 Self-healing transformatie voor transformable tile sets

We hebben zonet aangetoond dat het mogelijk is om een L-BCA tile set om te zetten naar een self-healing tile set. Maar ondanks dat L-BCA tile set Turing universal zijn, valt maar een beperkt deel van alle mogelijk tile sets onder de klasse van L-BCA tile set. In Figuur 3.6 vind je een voorbeeld van een assembly die geconstrueerd is met een tile set die geen L-BCA tile set is. In deze paragraaf gaan we aantonen dat het ook mogelijk is om voor een andere klasse van tile sets een equivalente self-healing tile set te construeren. Er zijn echter twee

moeilijkheden die problemen kunnen geven bij het construeren van een self-healing tile set. Een eerste moeilijk die kan optreden bij een willekeurige tile set is dat één tile type kan voorkomen op verschillende plaatsen in een assembly, waarbij de inputsides van die tile niet steeds hetzelfde zijn. En tweede moeilijkheid is dat er mogelijk mismatchen in een assembly zitten of dat er op de buitenste rand van een finale assembly nog weak bonds kunnen voorkomen. In deze paragraaf gaan we ons concentreren op een klasse van tile sets waarin geen van bovenstaande vermelde moeilijkheden kunnen voorkomen.



Figuur 3.6: Dit is een voorbeeld van een assembly geconstrueerd van een transformable tile set. Merk op hoe de buitenste rand volledig is opgebouwd met null bonds.

Definitie 3.3: *Een **transformable tile set** is een lokaal deterministische tile set die aan volgende condities moet voldoen:*

- *ieder tile type komt altijd voor met dezelfde inputsides, propsides en termsides.*
- *alle bond types die geen null bond zijn, zijn enerzijds een inputside of anderzijds een propsides.*

De tweede conditie in bovenstaande conditie zegt intuïtief dat de finale assembly van een transformable tile set enkel null bonds mag hebben op zijn buitenste rand. Merk op dat de assembly uit Figuur 3.6 een transformable tile set is en dat deze inderdaad enkel maar null bonds op zijn buitenste rand heeft staan.

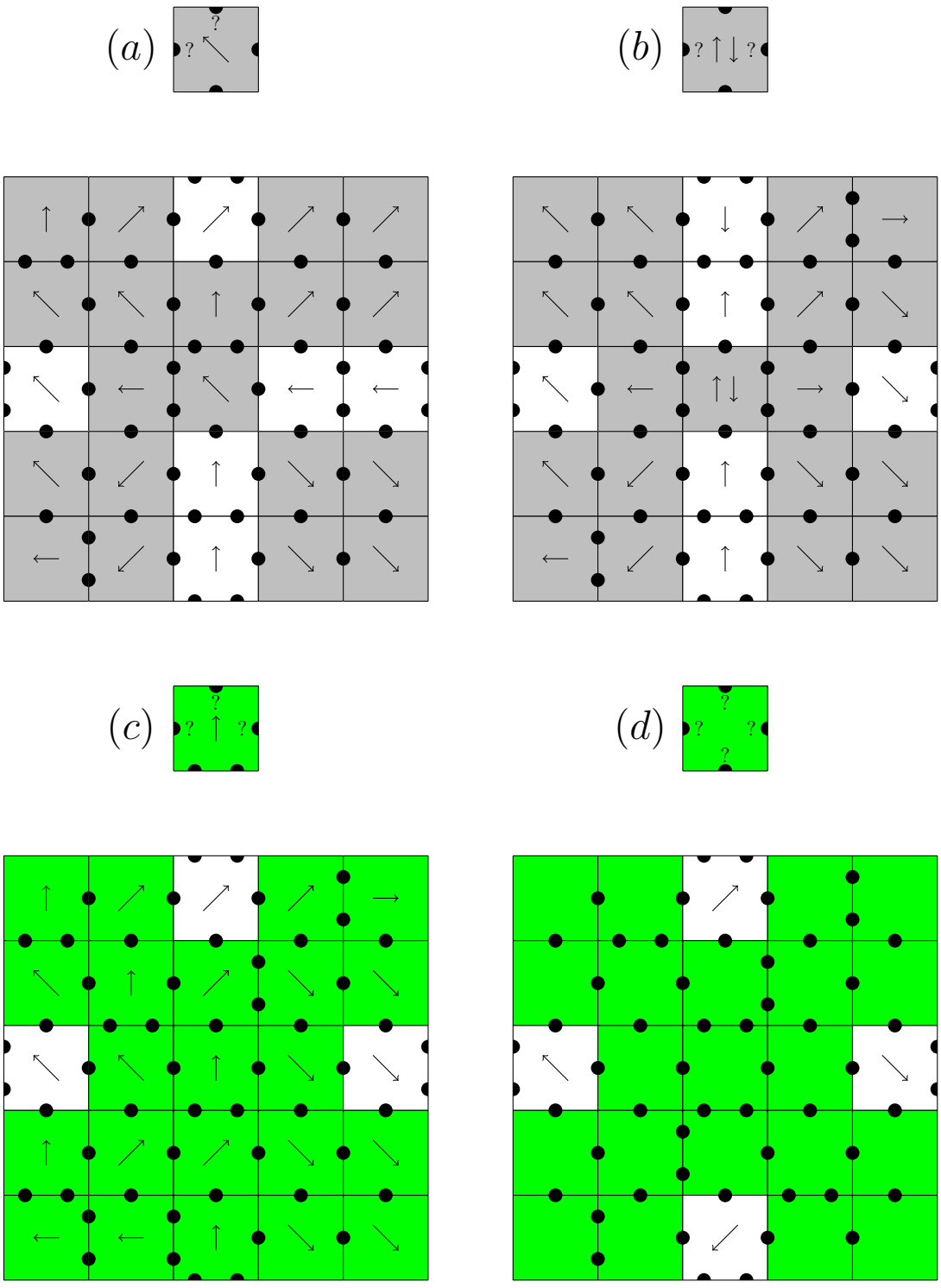
In tegenstelling tot een L-BCA tile set, waarbij er exact 3 patronen van tiles waren, zijn er hier veel meer mogelijkheden mogelijk. Het is echter niet nodig om voor iedere mogelijkheid een transformatie patroon te gaan opstellen, we kunnen namelijk veel verschillende patronen samenbrengen onder éénzelfde transformatie door geen rekening te houden bij de bond strength die wordt gebruikt voor de propsides en termsides van een tile. Concreet komt het er op neer dat er 4 verschillende combinaties van inputside mogelijk zijn, zonder de rotaties van de transformaties mee te tellen. Een eerste mogelijkheid is een tile die twee weak bonds, die langs elkaar liggen, als inputside heeft. Zijn transformatie block noemen we een **diagonaal rule-block**. De tweede mogelijkheid noemen we **convergerende rule-blocks**, deze transformaties representeren tiles die ook twee weak bonds als inputsides hebben, maar deze staan echter tegen over elkaar. De derde mogelijkheid is een tile met één strong bonds als inputside, zijn transformatie noemen we **strong-blocks**. Tot slot hebben nog een **seed-block**, deze representeert de seed-tile die geen inputsides heeft. In Figuur 3.7 vind je deze 4 combinaties terug samen met hun 5×5 self-healing transformatie block. De gekleurde tiles zijn block tiles. Tussen twee block tiles gebruiken we tile-type bonds. De witte tiles zijn bond tiles en gebruiken enkel bond-type bonds. Tussen een block tile en een bond tile gebruiken we een bond-type bond. Bovendien gebruiken bond tiles, waarvan de originele tile hetzelfde bond type hebben, dezelfde bond-type bonds. Indien de originele tile een null bond heeft, dan wordt de corresponderende bond tile vervangen door een block tile met 1 null bond en 3 tile-type bonds. Deze laatste aanpassing voor null bonds heeft als gevolg dat de getransformeerde tile set ook een transformable tile set is.

***Stelling 3.2.1* [13]**

De 5×5 block transformatie, beschreven in Figuur 3.7, produceert een self-healing tile set wanneer deze wordt toegepast op een transformable tile set (1). Bovendien zal de nieuwe tile set het zelfde patroon construeren als de originele tile set, maar dan 5 maal zo groot als voordien. Verder zal de overheersende kleur van het 5×5 block overeenkomen met de kleur van de originele tile (2).

Bewijs: Voor het bewijs van bewering (1) is het voldoende om een lokaal deterministische assembly sequence te identificeren. Gegeven een willekeurige transformable tile set T. Omdat T lokaal deterministische is, zie Definitie 3.3, weten we dat er een lokaal deterministische assembly sequence $\vec{A}$ bestaat voor de originele tile set. We moeten nu aantonen dat we $\vec{A}$ kunnen omzetten een lokaal deterministische 5×5 block tile set.

We weten dat $\vec{A}$ is opgebouwd, door tiles in een vaste volgorde toe te voegen aan $\vec{A}$, beginnend bij de seed-tile. We tonen nu aan via inductie dat we de nieuwe assembly $\vec{A}'$



Figuur 3.7: Dit zijn de 5×5 *self-healing transformaties voor transformable tile sets. Figuur (a) toont een diagonaal rule-block, (b) een convergerend rule-block, (c) een strong-block en (d) een seed-block. Dit niet-inputside, aangeduid met een ?, mogen eender welke bond strength hebben.*

kunnen construeren op een lokaal deterministische manier door de 5×5 blocks in dezelfde volgorde toe te voegen aan $\vec{A}'$ als hun overeenkomstige tile t werd toegevoegd aan $\vec{A}$.

Basisstap: De eerste tile die aan $\vec{A}$ werd toegevoegd is de seed-tile. We moeten nu op een lokaal deterministische manier het 5×5 block toevoegen aan $\vec{A}'$. In Tabel 3.2 wordt geïllustreert hoe je het seed-block op een lokaal deterministische manier kan opbouwen. Je kan namelijk eenvoudig nagaan dat de totale bond strength van de inputside van alle afzonderlijke tiles nooit 2 overschrijdt.

Inductiestap: We veronderstellen nu dat de assembly A^n equivalent is aan A'^n en dat we A'^n op een lokaal deterministische manier hebben kunnen opbouwen.

Inductiehypothese: Stel dat t de tile is die A^n uitbreid naar A^{n+1}. Dan is b_t het 5×5 block dat aan A'^n moet worden toegevoegd om A'^{n+1} te bekomen. Omdat de inputside van b_t de zelfde zijn als die van zijn corresponderende tile t en omdat de originele tile set lokaal deterministisch is, kunnen we concluderen dat er maar 1 5×5 block op dezelfde positie als die van t kan worden geplaatst. In Tabel 3.2 wordt geïllustreert hoe je een diagonaal rule-block, convergerend rule-block of strong-block op een lokaal deterministische manier kan opbouwen. Je kan eenvoudig nagaan dat de totale bond strength van de inputsides van alle afzonderlijke tiles nooit 2 overschrijdt. We kunnen dus concluderen dat we A'^{n+1} op een lokaal deterministische manier kunnen construeren.

Hierbij hebben we bewering (1) aangetoond.

(a)

11	12	13	6	7
10	5	4	5	6
9	4	3	2	1
8	5	2	3	4
7	6	1	4	5

(b)

11	6	1	6	7
10	5	2	5	8
9	4	3	4	9
8	5	2	5	10
7	6	1	6	11

(c)

11	12	13	14	15
10	9	10	11	16
9	8	7	12	17
4	5	6	13	18
3	2	1	14	19

(d)

14	7	8	9	10
13	6	2	3	11
12	5	1	5	12
11	3	2	6	13
10	9	8	7	14

Tabel 3.2: Deze tabel illustreert in welke volgorde je de 5×5 blocken op een lokaal deterministische manier kan opbouwen, indien de input blocks reeds aanwezig zijn. Het getal 1 duid de positie aan van de eerste tile die wordt toegevoegd, ... Tabel (a) representeert een diagonaal rule-block, (b) een convergerend rule-block, (c) een strong-block en (d) een seed-block.

Voor het bewijs van bewering (2) controleren we voor iedere tile in ieder 5×5 block alle mogelijke combinaties van zijden waarvan de totale bond strength minstens 2 bedraagt. Vervolgens controleren we voor deze zijden of er precies één tile bestaat die matched aan

deze zijdes. Als er 1 combinatie van zijdes wordt gevonden waarvoor dit niet geldt, dan kunnen we op 1 positie meerdere tiles toevoegen en bijgevolg geldt bewering (2) dan niet meer.

Indien er een combinaties van inputsides bestaat waarvoor de totale bond strength 2 bedraagt en waarvoor minstens 1 inputside van een block tile is in het 5×5 block, dan weten we dat deze tile uniek op die positie kan worden ingevoegd omdat alle bond types van block tile uniek uitgekozen woorden voor iedere tile. Hierdoor hoeven we deze combinaties in wat volgt niet meer controleren. We maken nu een onderscheid tussen het diagonaal rule-block, convergerend rule-block, strong-block en seed-block en testen de overige combinaties afzonderlijk:

- **diagonaal rule-block:** De buitenste rand van dit block is opgebouwd uit null bonds behalve voor de 4 bond tiles aan de buitenzijde. Alle 4 bond tiles zijn uniek omdat hun labels uniek zijn uitgekozen op basis van het bond type van de overeenkomstige zijde van de originele tile. De twee interne bond tiles zijn ook uniek omdat hun bond types precies hetzelfde zijn als de bond types van de andere bond tiles die deze bond tiles voorzien van hun inputside. En zoals we daarnet al hadden gezegd zijn die bond tiles uniek. En omdat we steeds en unieke tile hebben die voor hun inputside zorgt, moeten zei ook uniek zijn.

- **convergerend rule-block:** De buitenste rand van dit block is opgebouwd uit null bonds behalve voor de 4 bond tiles aan de buitenzijde. Alle 4 bond tiles zijn uniek omdat hun labels uniek zijn uitgekozen op basis van het bond type van de overeenkomstige zijde van de originele tile. De twee interne bond tiles zijn ook uniek omdat hun bond types precies hetzelfde zijn als de bond types van de andere bond tiles die deze bond tiles voorzien van hun inputsides. En zoals we daarnet al hadden gezegd zijn die bond tiles uniek. En omdat we steeds en unieke tile hebben die voor hun inputsides zorgt, moeten zei ook uniek zijn.

- **strong-block:** De buitenste rand van dit block is opgebouwd uit null bonds behalve voor de 3 bond tiles aan de buitenzijde. Alle 3 bond tiles zijn uniek omdat hun labels uniek zijn uitgekozen op basis van het bond type van de overeenkomstige zijde van de originele tile. Het middelste block tile van de onderste rij is uniek omdat zijn inputsides overeenkomt met de inputsides van de originele tile. En omdat de inputsides van de originele tile uniek zijn, zie Definitie 3.3, kan enkel dit block tile dat bond type hebben gebruikt op de onderste zijde. Bijgevolg is deze block tile dus uniek. Alle overige tiles hun inputside zijn hebben minstens 1 inputside van een block tile van dit block en omdat die hun bond types uniek zijn gekozen voor dit block zijn al deze tiles dus uniek.

- **seed-block:** De middelste tile is de seed-tile, deze is altijd uniek. De buitenste rand van dit block is opgebouwd uit null bonds behalve voor de 4 bond tiles aan de buitenzijde. Alle 4 bond tiles zijn uniek omdat hun labels uniek zijn uitgekozen op basis van het bond type van de overeenkomstige zijde van de originele tile. Alle

overige tiles hun inputside zijn hebben minstens 1 inputside van een block tile van dit block en omdat die hun bond types uniek zijn gekozen voor dit block zijn al deze tiles dus uniek.

We hebben dus aangetoond dat op iedere positie een unieke tile wordt geplaatst. Hierdoor zal de test uit Lemma 3.1.3 steeds slagen. Hierbij hebben we bewering (2) aangetoond en hierbij is het bewijs afgerond.

■

Merk op dat deze transformatie weak bonds en strong bonds als hetzelfde beschouwd, beide worden namelijk vertaald naar een transformatie met een strong bond op een bond tile in het midden. Maar omdat alle andere zijden null bonds zijn, kan er enkel een andere bond tile aan deze tile hechten en niets meer. De transformaties zijn zodanig opgebouwd, dat tile die oorspronkelijk twee weak bond hadden, dat deze hun 5×5 transformatie block pas verder kan groeien vanaf het moment dat de middelste block tile is toegevoegd. En deze middelste tile heeft als inputside 2 bond tile die ieder afhankelijk zijn een andere zijde van het 5×5 block. Dus zolang de 5×5 blocks niet zijn toegevoegd aan de assembly kan er geen verdere groei ontstaan, ondanks dat de weak bond zijn vervangen door strong bonds.

3.3 Self-healing transformatie voor polyomino tile sets

In de vorige paragraaf hebben we een 5×5 transformatie gezien die een transformable tile set self-healing maakt. Eigen aan deze constructie is dat ze zeer veel strong bonds introduceert, ondanks dat de originele tile set er mogelijk zeer weinig had. Een nadeel van deze strong bonds is dat ze de groei van nieuwe assemblies, zonder de aanwezigheid van een seed-tile, groter maakt. Vooral de strong-block zorgen voor problemen omdat in dat block een block tile aanwezig is met een strong bond aan de buitenkant van het block. Die block tile heeft dus de potentie om op zichzelf, zonder dat er een inputside aanwezig moet zijn, een volledig 5×5 block op te bouwen. Omdat bond tile identiek zijn voor alle blocken met hetzelfde bond type in hun originele tile, leveren deze tiles minder snel een probleem op. Zei hebben namelijk hulp van andere tiles nodig om een volledig block reproduceren. In deze paragraaf beschrijven we een self-healing transformatie die de kans op nucleation errors probeert te verhelpen.

Het basis idee van deze nieuwe constructie is om strong bonds zo veel mogelijk proberen te beperken. De oplossing bestaat erin om een block tile met strong bonds enkel nog intern in het block te laten voorkomen. Hierdoor krijgen de strong bonds van block tile steeds unieke bond types voor één specifiek block. Het resultaat is dat block tile met strong bonds steeds aan elkaar kunnen hechten zonder tussenkomst van andere tiles. Op deze manier creëren we een 'grotere' tile die ze een **polymino** noemen. De buitenste zijden van een polymino bevatten enkel null bonds en weak bonds. Merk op dat het toegestaan is dat een polymino uit meer dan twee tiles in opgebouwd.

Definitie 3.4: *Gegeven een lokaal deterministische tile set $\mathcal{T}$ in aTam die een assembly A produceert, dan construeren we een polymino-safe tile set P als volg:*

1. *voeg alle tiles, behalve de seed-tile, toe aan P*
2. *Indien er twee polymino $p_1, p_2 \in P$ bestaat zodat hun totale bond strength minstens 2 bedraagt, voeg dan de resulterende assembly toe aan p. Herhaal deze stap zolang P groeit.*
3. *Iedere polymino $p \in P$ moet een deel-assembly van A zijn, anders is P geen polymino-safe tile set.*

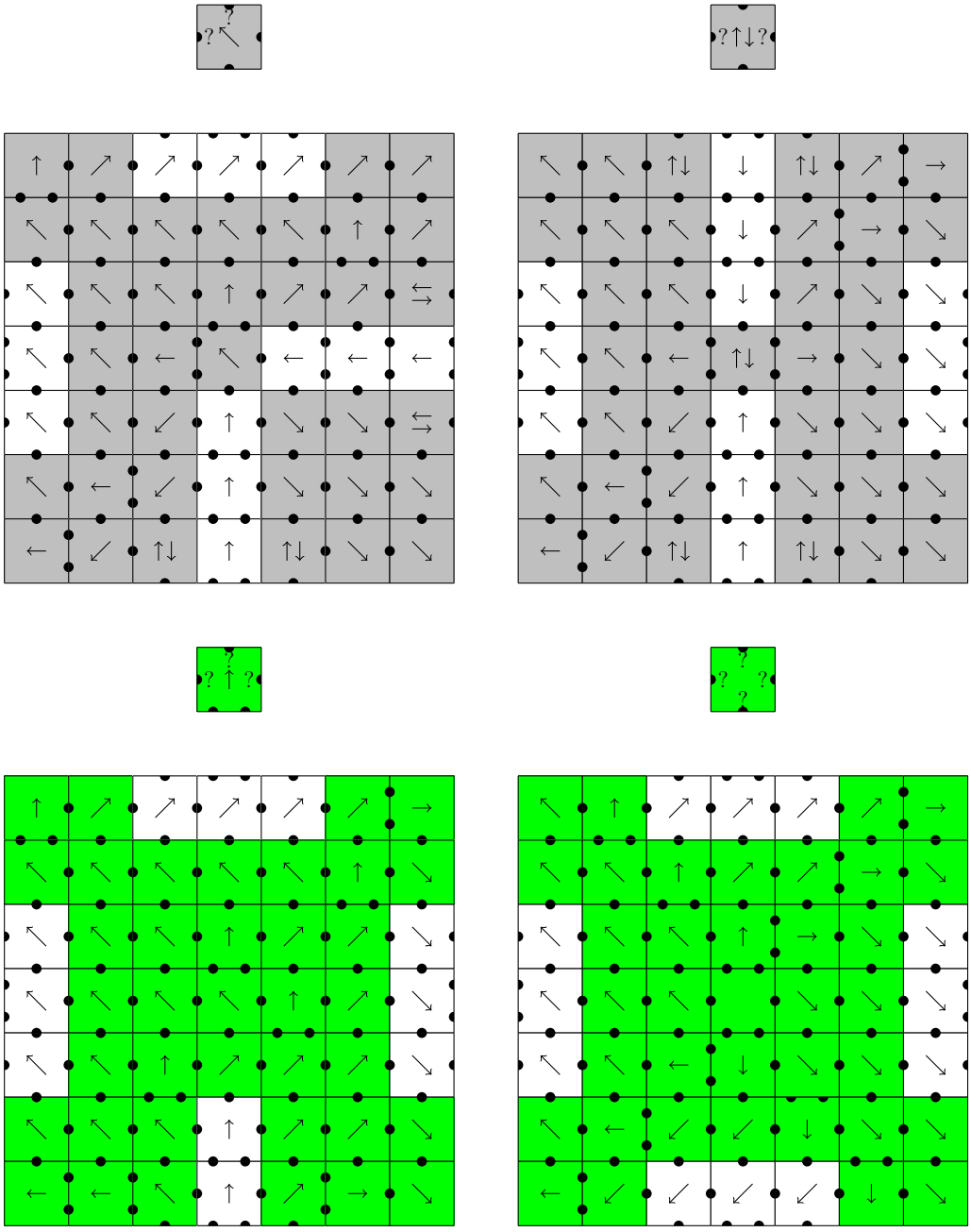
Omdat aTam gedefinieerd is op tiles, en niet op polymino's, definiëren we **polymino aTam** niet zoals aTam, enkel wordt de tile set vervangen door een polymino-safe tile set. Een geldige toevoegen tussen twee polymino is wanneer hun totale bond strength minstens 2 bedraagt.

Definitie 3.5: *We noemen een tile system $\mathcal{T}$* ***polymino-safe self-healing*** *(in polymino aTam) als voor iedere assembly A over $\mathcal{T}$ volgende eigenschap geldt: Indien een willekeurig aantal polymino's verwijderd worden uit A zodat de overgebleven polymino's allemaal nog verbonden zijn aan de seed-tile, dan wordt herstel van de assembly gegarandeerd zodat uiteindelijk iedere polymino die was verwijderd terug toegevoegd wordt aan de assembly zonder dat er fouten zullen optreden.*

Om aan te tonen dat een tile set aan deze eigenschap kan voldoen, maken we gebruik van de transformaties in Figuur 3.8. De gekleurde polymino's zijn block polymino's . Tussen twee block polymino's gebruiken we tile-type bonds. De witte polymino's zijn bond polymino's en gebruiken enkel bond-type bonds. Tussen een block polymino en een bond polymino gebruiken we een bond-type bond. Bovendien gebruiken bond polymino's , waarvan de originele tile hetzelfde bond type hebben, dezelfde bond-type bonds. Indien de originele tile een null bond heeft, dan wordt de corresponderende bond tile vervangen door een block tile met 1 null bond en 3 tile-type bonds. Deze laatste aanpassing voor null bonds heeft als gevolg dat de getransformeerde tile set ook een transformable tile set is. Merk op dat een bond polymino altijd is opgebouwd met alleen maar bond tiles en dat een block polymino enkel is opgebouwd met behulp van block tiles. We passen nu enkele stellingen, die we eerder al hadden gezien, aan zodat ze ook gelden voor polymino's.

Lemma 3.3.1 **[13]**

Als een polymino toegevoegd kan worden op een bepaalde plaats in een assembly A, dan kan diezelfde polymino ook worden toegevoegd op dezelfde positie van een grotere assembly die dezelfde tiles bevat als A plus nog enkele andere tiles. Dit op voorwaarde dat de positie nog open is.



Figuur 3.8: Dit zijn de 7 × 7 self-healing transformaties die een polymino-safe self-healing tile set opleveren wanneer ze worden toegepast op een transformable tile sets. Figuur (a) toont een polymino diagonaal rule-block, (b) een polymino convergerend rule-block, (c) een polymino strong-block en (d) een polymino seed-block. Dit niet-inputsides, aangeduid met een ?, mogen eender welke bond strength hebben.

Bewijs: Een polymino mag toegevoegd worden aan een assembly in polymino aTam indien de totale bond strength minstens τ bedraagt. Omdat polymino aTam geen negatieve bond strength toelaat, is het onmogelijk dat het toevoegen van extra polymino's aan een assembly ervoor kan zorgen dat de totale bond strength zal afnemen. Dus indien een polymino p kan toegevoegd worden aan een assembly A, dan kan deze ook worden toegevoegd aan een assembly A' als deze minstens dezelfde polymino's als A bevat en de plaats nog niet opgevuld is. Want de totale bond strength van p neemt ofwel toe, of deze blijft hetzelfde, door nieuwe polymino's toe te voegen aan de A'. Hierdoor mag p ook worden toegevoegd aan A'. ∎

***Lemma 3.3.2* [13]**

Beschouw een assembly A die werd geproduceerd over een polymino-safe tile set T, volgens de regels van polymino aTam . Kies nu een polymino uit T en kies vervolgens een plaats waar dat de polymino overlapt met andere tiles, maar hij mag niet overlappen met de seed-tile. Verwijder nu alle overlappende tiles uit A als een test. De test zal slagen als de polymino enkel maximaal één weak bond heeft met de overgebleven tiles in de assembly, of als alle tiles in de polymino identiek zijn aan de tiles die je hebt weggenomen. De tile set is polymino-safe self-healing als en slechts als de test slaagt voor iedere mogelijk geval.

Bewijs: De eerste implicatie is makkelijk in te zien. Indien er een test bestaat die faalt, dan is de tile set niet polymino-safe self-healing . Dit betekend namelijk dat het mogelijk is dat een beschadigde assembly op meerdere manieren zou kunnen herstellen.

Voor de omgekeerde implicatie veronderstellen we dat de polymino-safe self-healing tile set niet self-healing is, we gaan nu aantonen dat er een test bestaat die faalt. Dat de polymino-safe self-healing tile set niet self-healing is betekend dat er een assembly A bestaat waaraan schade is toegebracht, en dat er herstel heeft plaatsgevonden waarbij p de eerste foutieve polymino is die werd toegevoegd aan A tijdens dat herstel. Alle polymino's die we voor p hadden toegevoegd waren echter wel correct, we noemen deze assembly A'. We breiden A' nu uit door alle polymino's die verwijderd werden tijdens de schade aan A, behalve p, toe te voegen aan A'. De resulterende assembly noemen we A''. We hebben nu dus een assembly geconstrueerd die identiek is aan A, behalve op de positie van p. Wegens Lemma 3.3.1 weten we dat we p mogen toevoegen aan A'', maar de originele, juiste, polymino mogen we echter ook toevoegen aan A'' op de positie van p, want A is ook een correcte assembly. We hebben dus een test gevonden die faalt. ∎

***Stelling 3.3.1* [13]**

De 7×7 block transformatie, beschreven in Figuur 3.8, produceert een polymino-safe self-healing tile set wanneer deze wordt toegepast op een transformable tile set (1). Bovendien zal de nieuwe polymino-safe self-healing tile set het zelfde patroon construeren als de originele transformable tile set, maar dan 7 maal zo groot. Verder zal de overheersende kleur van het 7×7 block overeenkomen met de kleur van de originele polymino (2).

Bewijs: Voor het bewijs van bewering (1) is het voldoende om een lokaal deterministische assembly sequence te identificeren. Gegeven een willekeurige transformable tile set T. Omdat T lokaal deterministische is, zie Definitie 3.3, weten we dat er een lokaal deterministische assembly sequence $\vec{A}$ bestaat voor de originele tile set. We moeten nu aantonen dat we $\vec{A}$ kunnen omzetten een lokaal deterministische 7×7 block tile set.

We weten dat $\vec{A}$ is opgebouwd, door tiles in een vaste volgorde toe te voegen aan $\vec{A}$, beginnend bij de seed-tile. We tonen nu aan via inductie dat we de nieuwe assembly $\vec{A'}$ kunnen construeren op een lokaal deterministische manier door de 7×7 blocks in dezelfde volgorde toe te voegen aan $\vec{A'}$ als hun overeenkomstige tile t werd toegevoegd aan $\vec{A}$.

Basisstap: De eerste tile die aan $\vec{A}$ werd toegevoegd is de seed-tile. We moeten nu op een lokaal deterministische manier het 7×7 block toevoegen aan $\vec{A'}$. In Tabel 3.3 wordt geïllustreert hoe je het polymino seed-block op een lokaal deterministische manier kan opbouwen. Je kan namelijk eenvoudig nagaan dat de totale bond strength van de inputsides van alle afzonderlijke tiles nooit 2 overschrijdt.

Inductiestap: We veronderstellen nu dat de assembly A^n equivalent is aan A'^n en dat we A'^n op een lokaal deterministische manier hebben kunnen opbouwen.

Inductiehypothese: Stel dat p de polymino is die A^n uitbreid naar A^{n+1}. Dan is b_p het 7×7 block dat aan A'^n moet worden toegevoegd om A'^{n+1} te bekomen. Omdat de inputsides van b_p de zelfde zijn als die van zijn corresponderende tiles p en omdat de originele tile set lokaal deterministisch is, kunnen we concluderen dat er maar 1 7×7 block op dezelfde positie als die van p kan worden geplaatst. In Tabel 3.3 wordt geïllustreert hoe je een polymino diagonaal rule-block, polymino convergerend rule-block of polymino strong-block op een lokaal deterministische manier kan opbouwen. Je kan eenvoudig nagaan dat de totale bond strength van de inputsides van alle afzonderlijke polymino's nooit 2 overschrijdt. We kunnen dus concluderen dat we A'^{n+1} op een lokaal deterministische manier kunnen construeren.

Hierbij hebben we bewering (1) aangetoond.

Voor het bewijs van bewering (2) controleren we voor iedere polymino in ieder 7×7 block alle mogelijke combinaties van zijden waarvan de totale bond strength minstens 2

(a)

12	13	14	15	16	17	18
12	9	8	7	6	5	7
11	8	4	3	4	5	6
10	7	3	2	1	1	1
9	6	4	1	3	4	5
8	5	5	1	4	5	6
7	7	6	1	5	6	7

(b)

13	7	6	1	6	7	7
12	9	5	1	5	5	8
11	8	6	1	4	6	9
10	7	3	2	3	7	10
9	6	4	1	4	8	11
8	5	5	1	5	9	12
7	7	6	1	6	7	13

(e)

13	14	15	16	17	18	18
13	12	11	10	9	8	19
10	1	8	6	7	8	20
9	8	7	6	5	7	21
6	5	3	4	5	6	22
5	4	3	1	3	4	23
2	2	2	1	2	2	24

(d)

19	9	10	11	12	13	13
18	9	3	4	5	5	14
17	8	3	1	1	6	15
16	7	2	1	2	7	16
15	6	1	1	3	8	17
14	5	5	4	3	9	18
13	13	12	11	10	9	19

Tabel 3.3: Deze tabel illustreert in welke volgorde je de 7×7 blocken op een lokaal deterministische manier kan opbouwen, indien de input blocks reeds aanwezig zijn. Het getal 1 duid de positie aan van de eerste polymino die wordt toegevoegd, ... Tabel (a) representeert een polymino diagonaal rule-block, (b) een polymino convergerend rule-block, (c) een polymino strong-block en (d) een polymino seed-block.

bedraagt. Vervolgens controleren we voor deze zijden of er precies één polymino bestaat die matched aan deze zijdes. Als er 1 combinatie van zijdes wordt gevonden waarvoor dit niet geldt, dan kunnen we op 1 positie meerdere polymino toevoegen en bijgevolg geldt bewering (2) dan niet meer.

Indien er een combinaties van inputsides bestaat waarvoor de totale bond strength 2 bedraagt en waarvoor minstens 1 inputside van een block polymino is in het 7×7 block, dan weten we dat deze polymino uniek op die positie kan worden ingevoegd omdat alle bond types van block polymino uniek uitgekozen woorden voor iedere tile. Hierdoor hoeven we deze combinaties in wat volgt niet meer controleren.

Voor zowel de polymino diagonaal rule-block, polymino convergerend rule-block, polymino strong-block en polymino seed-block geldt dat de block polymino's aan de buitenste rand hebben allemaal null bonds, hierdoor zijn ze volledig afhankelijk van de tile-type bonds van het block. Omdat deze uniek zijn voor ieder block zijn alle block polymino's ook uniek. Alle bond polymino's zijn uniek omdat hun labels uniek zijn uitgekozen op basis van het bond type van de overeenkomstige zijde van de originele tile.

We hebben dus aangetoond dat op iedere positie een unieke polymino wordt geplaatst. Hierdoor zal de test uit Lemma 3.3.2 steeds slagen. Hierbij hebben we bewering (2) aan-

getoond en hierbij is het bewijs afgerond.

■

Hoofdstuk 4

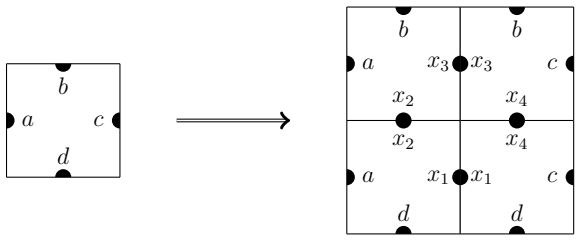
Proofreading

In dit hoofdstuk bespreken we een techniek, namelijk proofreading, die het error rate ε in kTam kan verlagen voor een willekeurige tile set. Het basis idee van proofreading is om ervoor te zorgen dat een fout nooit alleen kan optreden. Hiermee bedoelen we dat, als er een fout optreed, dat er vervolgens minstens nog een fout moet optreden voordat er weer self assembly zonder fouten kan plaatsvinden. Op deze manier vertragen we de groeisnelheid van een assembly, want het duurt langer voordat een foutieve tile zich hecht aan een assembly t.o.v. een correcte tile. Deze vertraging geeft de foutieve tile dan de kans om zich terug los te koppelen van de assembly waardoor die positie weer vrij komt zodat de correcte tile zich kan plaatsen op die positie. Proofreading is dus een techniek die growth error zal proberen tegen te gaan.

We beginnen dit hoofdstuk door 2×2 proofreading tile set voor te stellen. Vervolgens zullen we een meer algemene $k \times k$ proofreading tile set bespreken.

4.1 $K \times K$ proofreading tile sets

De constructie van een 2×2 proofreading tile set is eenvoudig. Je moet simpelweg iedere tile uit de originele tile set vervangen door 4 nieuwe tiles, dit wordt geïllustreert in Figuur 4.1. Belangrijk aan deze constructie is dat de interne labels van de nieuwe tiles uniek moeten zijn voor ieder 2×2 blok dat we gaan construeren. Het resultaat zal zijn dat we de originele assembly nu twee maal zo groot als voordien zullen construeren.



Figuur 4.1: Deze figuur illustreert hoe je een willekeurige tile kan omzetten naar een 2×2 proofreading blok.

Als we de constructie nader bekijken, dan kunnen we vaststellen dat als er een fout optreed, dat er nog een tweede fout moet optreden voordat er opnieuw normale assembly kan plaatsvinden. In Figuur 4.2 vind je alle mogelijke manieren waarop je een 2×2 blok kan construeren. De rode bond types duiden aan waar een mismatch heeft plaatsgevonden. In de bovenste rij staan alle finale blokken. Je kan duidelijk zien dat er precies één finaal blok is waar er geen fouten inzitten, in alle overige blokken zitten steeds twee fouten. Dit valt eenvoudig te verklaren omdat iedere nieuwe tile steeds aan één zijde een intern label heeft, en aan de overstaande zijde een label uit van de originele tile. Stel dat er een net een foutieve tile t was toegevoegd aan een assembly van een blok, dan kunnen we twee situaties onderscheiden:

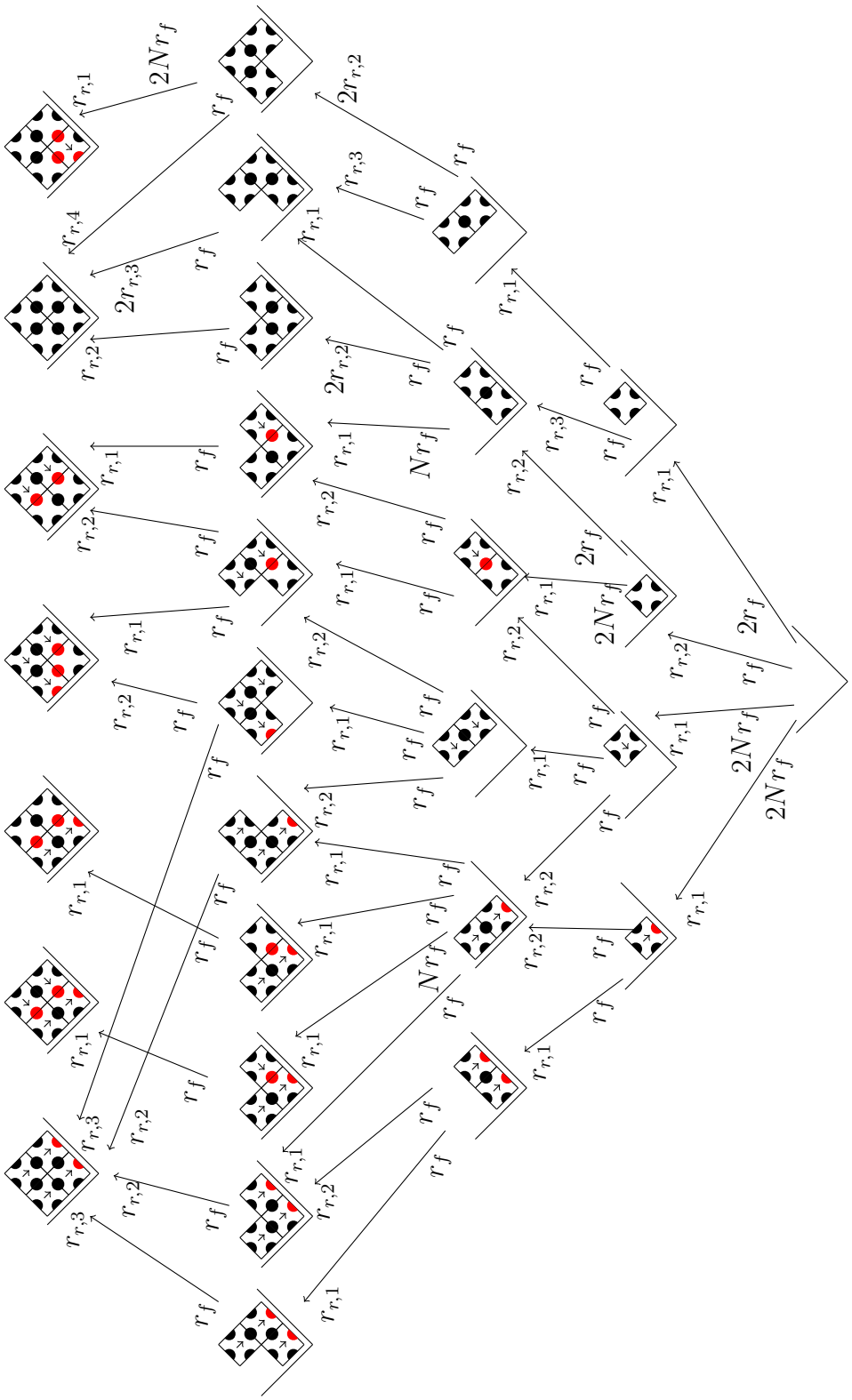
- De eerste situatie is dat t een tile is dat toebehoord aan een ander blok. Dit betekend dat zijn interne labels verschillen van de interne labels van de correcte tiles. Hierdoor dat de tile die zich zal hechten aan een intern label van t automatisch ook een fout opleveren en bijgevolg dus een tweede fout introduceren.

- De tweede situatie is dat t tot het juiste blok toebehoord, maar dat t op een verkeerde plaats in geplaatst in de assembly. Indien t één plaats teveel naar links staat in de assembly, dan heeft t een intern label op zijn linker zijde en een label van de originele tile op zijn rechter zijde. De correcte tile op die positie heeft namelijk de omgekeerde situatie voor, dus een intern label op zijn rechter zijde en een label van de originele tile op zijn linker zijde. Indien t één plaats teveel naar rechts is geplaatst geldt er een gelijkaardige redenering, ook hier zullen de labels op de linker er rechter zijde van het verkeerde type zijn. Indien t één positie veel naar boven of onder staat, dan zullen de types van de labels van de boven en onder zijde omgekeerd zijn.

We kunnen bovenstaand voorbeeld echter makkelijk veralgemenen naar een $k \times k$ blok. We vervangen dus iedere tile uit een tile set door k^2 tiles. Net als voordien hebben deze nieuwe tiles unieke interne labels en hergebruiken ze de labels van de originele tile aan de buitenkant. In een $k \times k$ blok moeten er k fouten achter elkaar optreden voordat de assembly weer zonder nieuwe fouten kan worden opgebouwd.

De hamvraag die we nu willen beantwoorden is: kunnen we met deze constructie een assembly opbouwen,die minder fouten bevat, binnen een aanvaardbare tijd? In Hoofdstuk 2.3 hebben we besproken wat het growth rate r is, dit is namelijk het verband tussen het error rate en de groeisnelheid van een tile set. We zullen nu uitrekenen hoeveel r bedraagt in de originele tile set om deze waarde vervolgens te vergelijken met het growth rate van de $k \times k$ proofreading tile set. Om optimale groei te verkrijgen kiezen we $G_{mc} = 2G_{se}$. Onder deze condities is het error rate $\varepsilon = e^{-G_(se)} = e^{-\frac{1}{2}G_(mc)}$. Omdat het growth rate ook afhangt van de concentratie tiles, kunnen we schrijven dat $r = \beta\lceil t \rceil = \beta e^{-G_{mc}} = \beta\varepsilon^2$ bedraagt.

Voor het growth rate van de $k \times k$ proofreading blokken te bepalen veronderstellen we ook dat $G_{mc} = 2G_{se}$. Het error rate dat één tile een mismatch maakt is niet als voordien $\varepsilon = e^{-G_(se)}$. Maar omdat in een $k \times k$ blok steeds k errors achter elkaar moeten plaatsvinden, is het error rate van zo een blok $\varepsilon = e^{-kG_(se)}$. Hierdoor wordt het growth rate $r = \beta e^{-kG_{mc}} = \beta\varepsilon^{\frac{2}{k}}$.

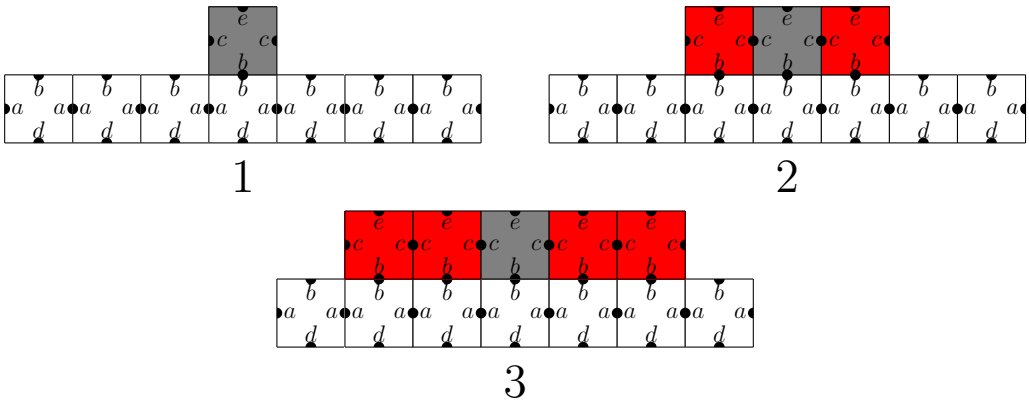


Figuur 4.2: Deze figuur toont alle mogelijke manieren waarop je een 2×2 proofreading tile kan construeren. De rode bolletjes duiden een mismatch aan. De pijlen duiden aan langs welke zijde er een fout is opgetreden. Je kan duidelijk zien dat de volledige assemblies ofwel geen, ofwel minstens 2 fouten in zich hebben zitten.

4.1.1 Bespreking van simulaties

De resultaten die we zonet hebben uitgerekend zijn veelbelovend. Daaruit blijkt namelijk dat er een verband bestaat tussen het error rate van een tile set en de blok grootte k die we gebruiken. Dit zou betekenen dat we k kunnen gebruiken als parameter om het error rate van een tile set te bepalen.

Winfree en Bekbolatov[14] hebben een reeks van simulaties uitgevoerd om te controleren of deze resultaten ook effectief waar zijn in de praktijk. Uit deze resultaten is gebleken dat voor $k = 2$ inderdaad het verwachte growth rate werd behaald. Maar voor $k = 3$ en $k = 4$ was dit echter niet het geval. Dit gedrag valt te verklaren omdat proofreading enkel growth error aanpakt, dit zijn errors waar dat een foutieve tile de plaats van een correcte tile invult. Een andere type error dat kan voorkomen zijn facet roughening errors, dit wil er op een willekeurige plaats een tile zich met precies éń weak bond hecht aan een assembly. Indien deze tile lang genoeg blijft hangen zodat er andere tile beginnen verder te groeien rondom deze foutieve tile is het mogelijk dat de assembly op die plaats een verkeerd patroon begin te construeren, waardoor de assembly niet correct kan worden geproduceerd, zie Figuur 4.3 voor een voorbeeld.



Figuur 4.3: Deze figuur illustreert hoe een facet roughening error kan optreden bij een 2×2 proofreading tile set. De grijze tile illustreert de inadequate hechting, de rode tiles zijn tiles die kunnen worden toegevoegd dankzij de inadequate hechting.

Een mogelijke oplossing voor sommige tile sets, maar zeker niet allemaal, is om volgende aanpassingen aan te brengen:

1. Sommige, maar zeker niet alle, facet roughening errors komen voor langs de termsides van tiles. Om te voorkomen dat er zich hier foutieve tiles aan de assembly gaan hechten, is het mogelijk om **capping** tiles te introduceren. Een capping tile heeft 1 zijde, dat een label van een termsides heeft, en alle overige zijden hebben null bonds. Hierdoor zal een capping tile zich hechten aan de termsides van een assembly en door hun null bonds is het onmogelijk dat er op die plaats nog fouten kan plaatsvinden omdat er daar geen groei meer mogelijk is.

2. Een tweede verbetering bestaat erin om rekening te houden met de groeirichting van een tile en ervoor te zorgen dat iedere tile slechts gebruikt wordt voor 1 groeirichting. Indien het er meerdere zijn introduceer je een nieuwe tile voor iedere extra groeirichting en pas je de labels van de eerste tile aan zodat deze tile enkel nog correct kan toegevoegd worden aan een assembly in 1 groeirichting.

3. Het is gunstig om labels op zo weinig mogelijk verschillende tiles te laten voorkomen. Dus indien een label heel vaak hergebruikt wordt, dan kan je overwegen om nieuwe labels te introduceren. Hierdoor verminder je namelijk het aantal tiles dat op een bepaalde plaats minstens één weak bond heeft. Bijgevolg zijn er dus minder potentiële kandidaat tiles die een facet roughening error kunne introduceren.

Winfree en Bekbolatov[14] hebben simulaties uitgevoerd op een tile set die een $M \times M$ vierkant produceert. Ze voerde 31 pogingen uit per tile set. Ze probeerde om een 26×26 vierkant te produceren met de origine tile set, van hun 31 pogingen was er geen enkele succesvol. In een tweede test probeerde ze een 49×49 vierkant te produceren met een aangepaste tile set die zowel nieuwe labels als capping tiles bevatte. In deze test werd er in 65% van de testgevallen het correcte patroon geconstrueerd. Tot slot probeerde ze een 98×98 vierkant te produceren met een proofreading tile set met capping tiles. In deze test werd het gewenste patroon 87% keer geproduceerd. Deze laatste test geeft een veelbelovende indruk over proofreading tile sets, maar diezelfde tile set produceerde in vergelijking slechts in 26% van de testgevallen een correct 99×99 vierkant.

Hieruit kunnen we besluiten dat proofreading tile sets, uit deze sectie, de potentie hebben om errors te verminderen, maar ze kunnen het echter niet garanderen. Het bewijs om aan te tonen hoe effectief een proofreading tile set is t.o.v zijn origine tile set in kTam is tot op vandaag nog steeds een open probleem.

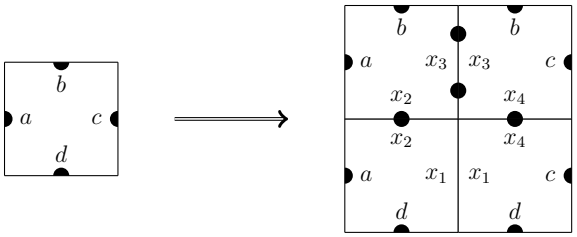
4.2 Snaked Proofreading

In de vorige paragraaf hebben we een $k \times k$ proofreading constructie gezien die growth error tegenging door te forceren dat errors steeds in groep moeten optreden om terug normale groei toe te laten. Het probleem bij die constructie was echter dat ze geen rekening hield met facet roughening errors. Hierdoor waren de resultaten voor vele tile sets nog steeds teleurstellend. In deze paragraaf stellen we een andere proofreading constructie voor, namelijk **Snaked proofreading**. We zullen aantonen dat deze constructie zowel growth errors als facet roughening errors tegengaat.

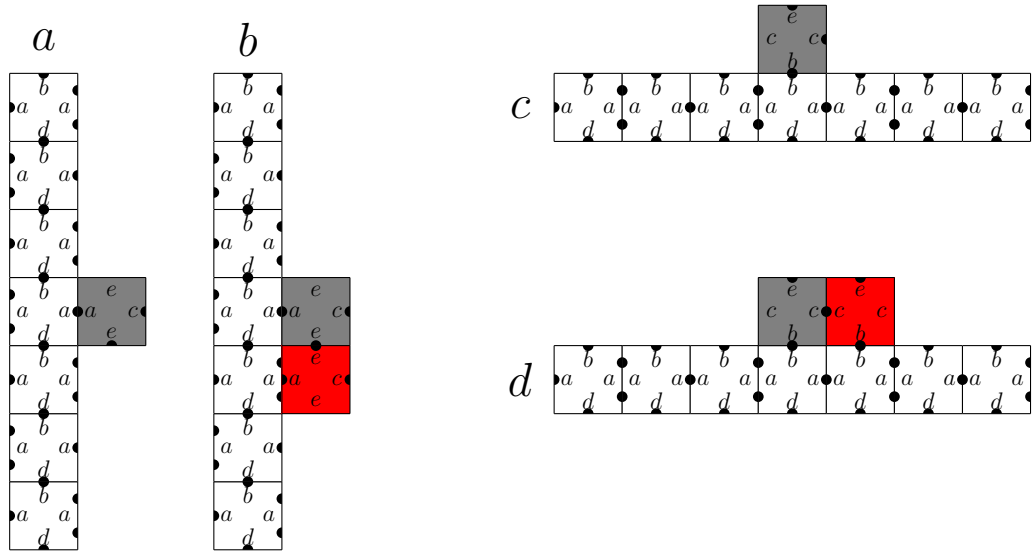
4.2.1 2×2 Snaked proofreading

In deze paragraaf proberen we de verschillen tussen $k \times k$ proofreading en snaked proofreading uit te leggen. Dit doen we aan de hand van een 2×2 snaked proofreading constructie, zie Figuur 4.4. Net zoals bij $k \times k$ proofreading vervangen we iedere tile door een $k \times k$ blok van tiles. In Figuur 4.1 kan je ter vergelijking zien hoe een 2×2 proofreading blok

eruit ziet. De labels gebruikt in beide constructies zijn hetzelfde. Het verschil tussen de twee constructies zit hem namelijk in de bond strength van de interne zijden van het blok. Het 2×2 proofreading blok gebruikt uitsluiten weak bonds, het 2×2 snaked proofreading blok daarentegen maakt gebruik van zowel een strong bond als een null bond. Dankzij deze nieuwe bond strengths kan een facet roughening error niet meer plaatsvinden als er slechts één inadequate hechting plaatsvindt, dit wordt geïllustreerd in Figuur 4.5.



Figuur 4.4: Deze figuur illustreert hoe je een willekeurige tile kan omzetten naar een 2×2 snaked proofreading blok.



Figuur 4.5: Links zie je hoe dat er maar maximaal 2 foutieve tiles kunnen worden toegevoegd bij 2×2 snaked proofreading tile set als er een facet roughening error optreed aan de rechterkant van een assembly. Rechts zie je dat dit ook geldt als er een facet roughening error optreed aan de bovenkant van een assembly.

Definitie 4.1: *Een **inadequate hechting**, met $\tau = 2$, is een proces waarbij er eerst een tile een hechting aangaat bij precies één weak bond. Vervolgens, voordat t_1 terug loskoppelt, hecht een andere tile t_2 zich aan de assembly langs de positie van t_1 zodat beide tiles ieder een totale bond strength van minstens 2 hebben.*

***Lemma 4.2.1* [3]**

De kans dat een inadequate hechting kan plaatsvinden op een willekeurige plaats in een groeiende assembly is $\mathcal{O}(e^{-3G_{se}})$.

Bewijs: Opdat er een inadequate hechting kan plaatsvinden, in kTam, moet er eerst een tile t_1 zich hechten aan de assembly met een weak bond. Dit gebeurt volgens zijn forward rate $r_{f_1} = k_f e^{G_{mc}} = k_f e^{2G_{se}}$. Deze laatste gelijkheid schrijven we onder de assumptie dat $G_{mc} = 2G_{se}$, omdat onder deze condities optimale groei kan plaatsvinden, zie Sectie 2.3. Omdat t_1 zich vast hechte met een weak bond is zijn reverse rate $r_{r,b} = r_{r,1} = k_f e^{1G_{se}}$. Om de inadequate hechting volledig te maken moet er nog een tweede tile t_2 zich hechten aan de assembly, zijn forward rate r_{f_2} is gelijk aan dat van t_1. Hierdoor is het totale ratio dat een inadequate hechting kan plaatsvinden gelijk aan $r_{f_1} \times r_{r,1} \times r_{f_2} = k_f e^{-3G_{se}} = \mathcal{O}(e^{-3G_{se}})$. ∎

***Lemma 4.2.2* [3]**

De kans dat een facet roughening error kan plaatsvinden in het snaked proofreading model is $\mathcal{O}(e^{-4G_{se}})$.

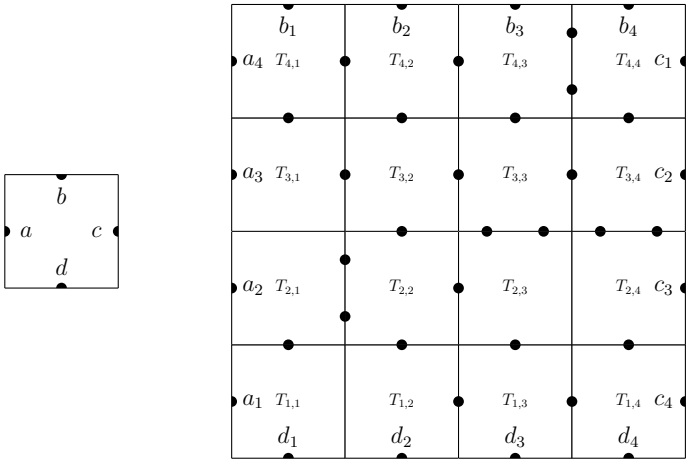
Bewijs: In het snaked proofreading model moeten er minstens 2 inadequate hechtingen plaatsvinden, voordat er facet roughening error kan optreden. Dit valt de verklaren door de null bonds die worden gebruikt in de onderste rij van het 2×2 blok. Deze verhinderen verdere groei in een horizontale richting, op voorwaarde dat er geen tiles boven hun aanwezig zijn. Indien er bovenop één van deze twee tiles nog eens een inadequate hechting plaatsvind, dan kan een 2×2 blok zich wel volledig ontwikkelen. Dit omdat de bovenste tiles van een blok gebruik maken van strong bonds. En nadat er zich een volledig blok heeft gevormd kan er facet roughening error optreden. Dit wordt geïllustreert in Figuur 4.5.

Wegens Stelling 4.2.1 weten we dat de kans dat de eerste inadequate hechting plaatsvind, gelijk is aan $\mathcal{O}(e^{-3G_{se}})$. Ook de tweede inadequate hechting vindt plaats met dezelfde kans. Maar ditmaal moeten we ook rekening houden met het reverse rate van de eerste inadequate hechting. Omdat deze vast hangt met een totale bond strength van 2 weten we dat deze terug kan loskomen met een kans van $\mathcal{O}(2G_{se})$. De kans dat een facet roughening error kan plaatsvinden is dan gelijk aan $e^{-3G_{se}} \times e^{2G_{se}} \times e^{-3G_{se}} = \mathcal{O}(e^{-4G_{se}})$. ∎

4.2.2 Het algemene Snaked Proofreading

In de vorige paragraaf hebben we een model voorgesteld dat enkel facet roughening errors tegenhoudt in één richting, namelijk van het westen naar het oosten. In deze paragraaf beschrijven we een model in facet roughening errors tegenhoudt twee groeirichtingen, namelijk wel van west naar oost, als van zuid naar het noord. Definitie 4.2 beschrijft hoe je

een willekeurige tile kan omvormen naar een $2k \times 2k$ snaked proofreading, zie Figuur 4.6 voor een voorbeeld. In de rest van deze paragraaf zullen we aantonen dat de kans dat er facet roughening errors optreed, steeds kleiner wordt naarmate we een grotere waarde voor k kiezen. Bovendien tonen we ook aan dat de constructietijd van de nieuwe assembly slechts lineair stijgt in functie van k. Hiervoor introduceren we eerst enkele begrippen, namelijk een k-bottleneck en een block error.



Figuur 4.6: Deze figuur illustreert hoe je een willekeurige tile kan omzetten naar een $2k \times 2k$ snaked proofreading blok, met $k = 2$.

Definitie 4.2: *Een $2k \times 2k$ snaked proofreading blok, met $k \geq 2$, van een tile t is een blok van $2k \times 2k$ tiles $T_{1,1}, T_{1,2}, \ldots, T_{2k,2k}$, waarbij alle zijden van deze tiles weak bonds hebben, behalve voor volgende uitzonderingen:*

- *De oost-zijden van alle tiles $T_{1,2i-1}$, en de west-zijden van alle tiles $T_{1,2i}$ hebben null bonds, met $i = 1, 2, \ldots, k-1$.*
- *De noord-zijden van alle tiles $T_{2i,1}$, en de zuid-zijden van alle tiles $T_{1,2i+1}$ hebben null bonds, met $i = 1, 2, \ldots, k-1$.*
- *De noord-zijden van alle tiles $T_{2i,2i+1}$, en de zuid-zijden van alle tiles $T_{2i+1,2i+1}$ hebben strong bonds, met $i = 1, 2, \ldots, k-1$.*
- *De oost-zijden van alle tiles $T_{2i,2i-1}$, en de west-zijden van alle tiles $T_{2i,2i}$ hebben strong bonds, met $i = 1, 2, \ldots, k-1$.*
- *De oost-zijde van de tile $T_{2k-2,2k-1}$, en de west-zijde van de tile $T_{2k-2,2k}$ hebben een null bond.*
- *De noord-zijde van de tile $T_{2k-2,2k}$, en de zuid-zijde van de tile $T_{2k-1,2k}$ hebben een strong bond.*

Hierbij is $T_{1,1}$ de tile in de linker beneden hoek van het blok en $T_{2k,2k}$ de tile in de rechter bovenhoek van het blok. De interne labels zijn uniek en komen dus niet voor in een ander $2k \times 2k$ snaked proofreading blok binnen eenzelfde tile system. De labels van de buitenste zijden van een $2k \times 2k$ snaked proofreading blok zijn geïndexeerde varianten van het label van de overeenkomstige zijde van t. Ook de bond strength van de overeenkomstige zijde van t wordt overgenomen.

Definitie 4.3: *Een k-**bottleneck** is een assembly die enkel kan gevormd worden als er minstens k inadequate hechting hebben plaatsgevonden.*

Definitie 4.4: *Een **block error** neemt plaats indien er een $2k \times 2k$ blok wordt gevormd waarvan alle tiles foutief zijn t.o.v. de tiles in het originele correcte blok.*

Voor de bewijzen van onderstaande stellingen en lemma's maken we volgende assumpties:

- Indien een tile vast hangt aan een assembly met een totale bond strength die minimaal 3 bedraagt, dan kan diezelfde tile **nooit** meer loskomen van de assembly.
- Het forward rate gelijk is aan het reverse rate, dus $r_f = r_r$.
- We maken steeds gebruik van een L-BCA tile set. Bovendien gaan we er vanuit dat de boundairies initieel aanwezig zijn.

Deze veronderstellingen zijn noodzakelijk om de bewijzen te laten kloppen. Maar, zoals ook blijkt uit de simulaties op het einde van dit hoofdstuk, werkt de snaked proofreading constructie ook in situaties die niet voldoen aan bovenstaande criteria. Er is echter tot op heden nog geen bewijs gevonden om de correcte werking van snaked proofreading aan te tonen onder mildere condities.

De intuïtie achter Lemma 4.2.3 en 4.2.4 is dat een inadequate hechting slechts voor een beperkte groei van een $2k \times 2k$ snaked proofreading blok kan zorgen. Deze lemma's zullen we later gebruiken om aan te tonen dat minstens k inadequate hechtingen nodig hebben om foutief blok volledig te construeren.

***Lemma 4.2.3* [3]**

Als de breedte van een structuur X, voordat er zich een inadequate hechting heeft voorgedaan, w bedraagt, dan is de breedte van X maximaal $w + 2$ nadat er zich een inadequate hechting heeft voorgedaan.

Bewijs: Veronderstel dat $2i - 2 \leq w < 2i$, voor een willekeurige i. Onmiddellijk nadat er een inadequate hechting heeft plaatsgevonden is de breedte maximaal $w + 1 \leq 2i$, er werd namelijk maar 1 tile aan X toegevoegd dus de breedte kan maximaal met 1 verhogen. Omdat er geen strong bonds aanwezig zijn op de zijden tussen de $2i^{de}$ en $2i + 1^{de}$ kolom van een blok, zie Definitie 4.2, kan de breedte niet meer dan $2i$ $(= w + 2)$ bedragen zonder dat er eerst nog een inadequate hechting plaatsvind. ∎

Merk op dat uit Lemma 4.2.3 volgt dat een block error een voorbeeld is van een k-bottleneck.

***Lemma 4.2.4* [3]**

Als de hoogte van een structuur X, voordat er zich een inadequate hechting heeft voorgedaan, h bedraagt, dan is de hoogte van X maximaal $h + 2$ nadat er zich een inadequate hechting heeft voorgedaan.

Bewijs: Veronderstel dat $2i - 2 \leq h < 2i$, voor een willekeurige i. Onmiddellijk nadat er een inadequate hechting heeft plaatsgevonden is de breedte maximaal $h + 1 \leq 2i$, er werd namelijk maar 1 tile aan X toegevoegd, dus de hoogte kan maximaal met 1 verhogen. Omdat er geen strong bonds aanwezig zijn op de zijden tussen de $2i - 1^{de}$ en $2i^{de}$ rij van een blok, zie Definitie 4.2, kan de hoogte niet meer dan $2i$ $(= h + 2)$ bedragen zonder dat er eerst nog een inadequate hechting plaatsvind. ∎

Definitie 4.5: *Een **begin-positie** op de oost-boundary bestaat uit de twee onderste rijen van een blok. Een begin-positie op de noord-boundary bestaat uit de twee linker kolommen van een blok.*

De intuïtie achter Lemma 4.2.5 is dat we minstens k inadequate hechtingen nodig hebben om verdere groei in de assembly mogelijk te maken. Indien er geen k inadequate hechtingen plaatsvinden, dan zal er een onvolledig blok geconstrueerd worden waaraan geen correcte groei aan kan plaatsvinden. Indien er geen nieuwe inadequate hechtingen plaatsvinden, dan zal de foutieve structuur na eindige tijd afvallen van assembly zodat er op die plaats weer normale groei kan plaatsvinden.

***Lemma 4.2.5* [3]**

Een structuur X die geen k-bottleneck is overdekt maximaal 2 blokken.

Bewijs: We delen dit bewijs op in twee delen. In het deel bevindt X zich aan de oost-boundary, in het tweede geval aan de noord-boundary.

Deel 1: X bevindt zich aan de oost-boundary. Zij B_1, B_2, B_3 drie aaneengesloten blokken van noord naar zuid en zij S_1, S_2, S_3 respectievelijk hun begin-posities. We veronderstellen dat X een structuur is die geen k-bottleneck is en tiles heeft op de positie van S_2. Zij w de breedte van X. Wegens Lemma 4.2.3 weten we dat w maximaal $2k-2$ bedraagt, want anders zou X een k-bottleneck zijn.

Veronderstel nu dat er tiles aanwezig zijn in X in de $(2\lceil\frac{w}{2}\rceil)^{de}$ rij van blok B_3. Voor $1 \leq r \leq k - \lfloor\frac{w}{2}\rfloor$, beschouw het gebied gedefinieerd door $(2k-2r)^{de}$ en $(2k-2r+1)^{de}$ rijen van B_3. Alle tiles van X die zich in dit gebied bevinden hebben weak bonds op al hun buitenste zijden, zie Definitie 4.2. Hierdoor kunnen de tiles in dit gebied zich hechten niet aan X, tenzij er een inadequate hechting plaatsvind. Beschouw nu voor ieder van deze gebieden de eerste inadequate hechting die hierin plaatsvind. Omdat er geen strong bonds aanwezig zijn in deze gebieden, kan deze adequate hechting er niet toe lijden dat de breedte van dit gebied wordt verhoogt op voorwaarde dat het gebied al minstens 1 breed was. We hebben eveneens één inadequate hechting nodig voor de tiles die zich hechten aan het zuiden van de $(2\lceil\frac{w}{2}\rceil)^{de}$ rij van B_3, dit om wille van de null bond die hier aanwezig. Een andere inadequate hechting is nodig voor de tiles die zich hechten aan het noorden van de $(2k-2)^{de}$ rij van B_3, ook hier om wille van de aanwezigheid van een null bond. De eerste inadequate hechting die optreed in deze gebieden kan de breedte niet verhogen als de breedte minstens 2 bedroeg voor deze inadequate hechting plaatsvond. Herinner je dat X niet k-bottleneck is, dit wil zeggen dat er maximaal $k-1$ inadequate hechtingen mogen plaatsvinden om X volledig op te bouwen. Naast de $k - \lfloor\frac{w}{2}\rfloor + 1$ inadequate hechtingen die we zonet hebben opgenoemd, zijn er dus nog maximaal $\lceil\frac{w}{2}\rceil - 2$ andere inadequate hechtingen. Wegens Lemma 4.2.3 kunnen deze $\lceil\frac{w}{2}\rceil - 2$ inadequate hechtingen de breedte met maximaal $w-4$ verhogen. Hierdoor kan de totale breedte van X maar maximaal $w-2$ bedragen, en dit is een contradictie want X is w breed. Hieruit volgt dat onze aanname niet waar is, X kan dus geen tiles kan hebben in de $2\lceil\frac{w}{2}\rceil^{de}$ rij van B_3. Bijgevolg kan X ook geen tiles in S_3 hebben. Het bewijs voor B_1 is volledig analoog aan dat van B_3 en toont aan dat X geen tiles kan hebben in het gebied van S_1.

Deel 2: X bevindt zich aan de noord-boundary. Zij B_1, B_2, B_3 drie aaneengesloten blokken van west naar oost en zij S_1, S_2, S_3 respectievelijk hun begin-posities. We veronderstellen dat X een structuur is die geen k-bottleneck is en tiles heeft op de positie van S_2. Zij h de hoogte van X. Wegens Lemma 4.2.4 weten we dat h maximaal $2k-2$ bedraagt, want anders zou X een k-bottleneck zijn.

Veronderstel nu dat er tiles aanwezig zijn in X in de $(2\lceil\frac{h}{2}\rceil)^{de}$ kolom van blok B_3. Voor $1 \leq r \leq k - \lfloor\frac{h}{2}\rfloor$, beschouw het gebied gedefinieerd door $(2k-2r)^{de}$ en $(2k-2r-1)^{de}$ kolommen van B_3. Alle tiles van X die zich in dit gebied bevinden hebben weak bonds op al hun buitenste zijden, zie Definitie 4.2. Hierdoor kunnen de tiles in dit gebied zich hechten niet aan X, tenzij er een inadequate hechting plaatsvind. Beschouw nu voor ieder van deze gebieden de eerste inadequate hechting die hierin plaatsvind. Omdat er geen strong bonds aanwezig zijn in deze gebieden, kan deze adequate hechting er niet toe lijden dat de hoogte van dit gebied wordt verhoogt op voorwaarde dat het gebied al minstens 1

hoog was. We hebben eveneens één inadequate hechting nodig voor de tiles die zich hechten aan het westen van de $(2\lceil\frac{h}{2}\rceil)^{de}$ kolom van B_3, dit om wille van de aanwezigheid van een null bond. Een andere inadequate hechting is nodig voor de tiles die zich hechten aan het oosten van de $(2k-1)^{de}$ kolom van B_3, ook hier omdat er een null bond aanwezig is. De eerste inadequate hechting die optreed in deze gebieden kan de hoogte niet verhogen als de hoogte minstens 2 bedroeg voor deze inadequate hechting plaatsvond. Herinner je dat X niet k-bottleneck is, dit wil zeggen dat er maximaal $k-1$ inadequate hechtingen mogen plaatsvinden om X volledig op te bouwen. Naast de $k-\lfloor\frac{h}{2}\rfloor+1$ inadequate hechtingen die we zonet hebben opgenoemd, zijn er dus nog maximaal $\lceil\frac{h}{2}\rceil-2$ andere inadequate hechtingen. Wegens Lemma 4.2.4 kunnen deze $\lceil\frac{h}{2}\rceil-2$ inadequate hechtingen de hoogte met maximaal $h-4$ verhogen. Hierdoor kan de totale hoogte van X maar maximaal $h-2$ bedragen, en dit is een contradictie want X is h breed. Hieruit volgt dat onze aanname niet waar is, X kan dus geen tiles kan hebben in de $2\lceil\frac{h}{2}\rceil^{de}$ rij van B_3. Bijgevolg kan X ook geen tiles in S_3 hebben. Het bewijs voor B_1 is volledig analoog aan dat van B_3 en toont aan dat X geen tiles kan hebben in het gebied van S_1. ∎

Definitie 4.6: *Een (x,y)-**gedeeltelijke rechthoek** is een sequentie van x gehele getallen $(i_1,i_2,\ldots,i_x)$, waarbij $0\leq i_1\leq i_2\leq\ldots\leq i_x\leq y$. Een (x,y)-gedeeltelijke rechthoek R_1 is **omvat** door een andere (x,y)-gedeeltelijke rechthoek R_2 indien ieder element van R_1 kleiner of gelijk is dan het overeenkomstige element uit R_2. Twee (x,y)-gedeeltelijke rechthoek R_1,R_2 zijn **opeenvolgend** indien R_1 omvat is door R_2 en $\|R_1\|_1=\|R_2\|_1-1$, of omgekeerd.*

Definitie 4.7: *Een **willekeurige (x,y)-wandeling met snelheid r** is een willekeurige wandeling vertrekkend bij een (x,y)-gedeeltelijke rechthoek R_1 naar een (x,y)-gedeeltelijke rechthoek R_2, waarbij R_1 en R_2 opeenvolgend zijn. Dit gebeurd tegen een snelheid r.*

Definitie 4.8: *Een **hitting time** voor een willekeurige (x,y)-wandeling is verwachte tijd dat een willekeurige (x,y)-wandeling, beginnend bij $(0,0,\ldots,0)$, $(y,y,\ldots,y)$ zal bereiken.*

Lemma 4.2.6 toont aan dat een blok relatief snel kan worden opgebouwd.

***Lemma 4.2.6* [3]**

De hitting time voor een willekeurige $(2,k)$-wandeling is $\mathcal{O}(\frac{k^4}{r})$, hier is r de snelheid van de willekeurige $(2,k)$-wandeling.

Bewijs: Beschouw een willekeurige wandeling W_1 doe over een 1 dimensionaal raster. De tijd die nodig is om naar een willekeurig punt k te gaan wordt beschreven door de differentievergelijking $t_k = 1 + \frac{1}{2}(t_{k-1} + t_{k+1})$. De algemene oplossing voor deze vergelijking is $t_k = -k^2 + mk + n$. We kunnen dus in $\mathcal{O}(k^2)$ tijd naar een punt k gaan in een 1 dimensionaal raster. We zijn echter op zoek naar de tijd die nodig is om van $(0, 0)$ naar (k, k) te gaan in een 2 dimensionaal raster. We kunnen dit probleem echter modelleren aan de hand van twee wandelingen over een 1 dimensionaal raster, namelijk 1 horizontale wandeling en 1 verticale wandeling. Hierdoor wordt de totale tijd van de wandeling over een 2 dimensionaal raster gegeven door $\mathcal{O}(k^4)$. Deze tijd is echter opgebouwd onder de assumptie dan een wandeling gebeurd tegen een snelheid 1. Als we over gaan naar een snelheid r, dan wordt de totale tijd gelijk aan $\mathcal{O}(\frac{k^4}{r})$. ∎

Lemma 4.2.7 en 4.2.8 zeggen simpelweg dat een foutieve structuur zich binnen een aanvaardbare tijd terug zal loskoppelen van een assembly. Dit betekend dat, als er een foutieve onvolledig blok aanwezig is, dat deze er niet voor kan zorgen dat de constructietijd van een assembly enorm vertraagt zal worden.

***Lemma 4.2.7* [3]**

Beschouw een structuur X, aan de oost-boundary, die geen k-bottleneck is. Zonder dat er een nieuwe inadequate hechting zal plaatsvinden op X of op zijn naburige blokken, dan zal X terug afvallen in een verwachte tijd $\mathcal{O}(\frac{k^5}{r_r})$, hierbij is r het reverse rate. Herinner je dat het forward rate gelijk is aan het reverse rate.

Bewijs: Zij w de breedte van X. Wegens Lemma 4.2.3 is $w \leq 2k-2$. Wegens Lemma 4.2.5 weten we dat X maximaal 2 blokken B_2 en B_3 overlapt. Bovendien kunnen er ook geen tiles voorkomen in de $2\lceil \frac{w}{2} \rceil)^{de}$ rij van B_3. Hierdoor kunnen de tiles uit X enkel de bovenste $2k - w$ rijen van B_3 invullen. We verdelen de tiles van X in B_3 in $\frac{2k-w}{2}$ delen, hierbij is ieder deel een $2(rij) \times w(kolom)$ rechthoek. Deze rechthoeken kunnen zich loskoppelen van onder naar boven, en nadat ze volledig zijn losgekoppeld kunnen ze enkel terug worden toegevoegd als er eerst een inadequate hechting plaatsvind. De tijd die nodig is om een $2 \times k$ rechthoek te laten loskoppelen komt overeen met de hitting time van een willekeurige $(2, k)$-wandeling. Wegens Lemma 4.2.6 weten we dan dat iedere rechthoek zal afvallen aan een verwachte tijd van $\mathcal{O}(\frac{w^4}{r_r})$. Hierdoor is de verwachte tijd dat alle tiles uit de onderste $2k - 2$ rijen van het onderste blok afvallen $\mathcal{O}(\frac{kw^4}{r_r})$. Nadat deze tiles allemaal zijn afgevallen, zijn er nog $\mathcal{O}(k^2)$ tiles over. Deze zullen in een soort van ketting reactie één voor één afvallen, wegens Lemma 4.2.6 weten we dat dit in $\mathcal{O}(\frac{k^4}{r_r})$ tijd gebeurd. ∎

***Lemma 4.2.8* [3]**

Beschouw een structuur X, aan de noord-boundary, die geen k-bottleneck is. Zonder dat er een nieuwe inadequate hechting zal plaatsvinden op X of op zijn naburige blokken, dan zal X terug afvallen in een verwachte tijd $\mathcal{O}(\frac{k^5}{r_r})$, hierbij is r het reverse rate. Herinner je dat het forward rate gelijk is aan het reverse rate.

Bewijs: Zij h de hoogte van X. Wegens Lemma 4.2.4 is $h \leq 2k-2$. Wegens Lemma 4.2.5 weten we dat X maximaal 2 blokken B_2 en B_3 overlapt. Bovendien kunnen er ook geen tiles voorkomen in de $2\lceil\frac{h}{2}\rceil)^{de}$ kolom van B_3. Hierdoor kunnen de tiles uit X enkel de rechter $2k-h$ kolommen van B_3 invullen. We verdelen de tiles van X in B_3 in $\frac{2k-h}{2}$ delen, hierbij is ieder deel een $2(kolom) \times h(rij)$ rechthoek. Deze rechthoeken kunnen zich loskoppelen van rechts naar links, en nadat ze volledig zijn losgekoppeld kunnen ze enkel terug worden toegevoegd als er eerst een inadequate hechting plaatsvind. De tijd die nodig is om een $2 \times k$ rechthoek te laten loskoppelen komt overeen met de hitting time van een willekeurige $(2, k)$-wandeling. Wegens Lemma 4.2.6 weten we dan dat iedere rechthoek zal afvallen aan een verwachte tijd van $\mathcal{O}(\frac{h^4}{r_r})$. Hierdoor is de verwachte tijd dat alle tiles uit de onderste $2k-2$ rijen van het onderste blok afvallen $\mathcal{O}(\frac{kh^4}{r_r})$. Nadat deze tiles allemaal zijn afgevallen, zijn er nog $\mathcal{O}(k^2)$ tiles over. Deze zullen in een soort van ketting reactie één voor één afvallen, wegens Lemma 4.2.6 weten we dat dit in $\mathcal{O}(\frac{k^4}{r_r})$ tijd gebeurd. ∎

***Stelling 4.2.1* [3]**

Veronderstel dat we een $2k \times 2k$ snaked tile system gebruiken met $G_{mc} = 2G_{se}$. De kans dat er geen k-bottleneck optreed op een specifieke plaats binnen een tijd t is minstens e^{-tR_E}, hierbij is $R_E = r_f e^{-G_{se}}(\frac{e^{-G_{se}}}{e^{-G_{se}}+\frac{1}{ck^6}})^{k-1}$, c een constante onafhankelijk van k en G_{se}

Bewijs: Per definitie moet er k inadequate hechtingen plaatsvinden voordat er zich een k-bottleneck voordoet. Na $i < k-1$ inadequate hechtingen hebben plaatsgevonden, moet een van onderstaande gevallen gebeuren:

- Nog een inadequate hechting: Beschouw een maximale structuur X die werd opgebouwd met i inadequate hechtingen. Wegens Lemma 4.2.5 weten we dat X maximaal 2 blokken overlapt, hierdoor weten we dat er maximaal $6k$ inadequate hechtingen nodig zijn om X te laten groeien. Dus de kans dat de i^{de} inadequate hechting plaatsvind is maximaal $6kr_f e^{-Gse}$.

- Alle tiles komen terug los: Wegens Lemma 4.2.7 weten we dat de verwachte tijd dat alle tiles afvallen gelijk is aan $\mathcal{O}(c\frac{k^5}{r_r})$, voor een constante c onafhankelijk van k.

Dus, nadat de i^{de} inadequate hechting heeft plaatsgevonden, is de kans dat de $i+1^{de}$ inadequate hechting plaatsvind, voordat alle tiles terug afvallen, gelijk aan $\frac{kr_f e^{-G_{se}}}{kr_f e^{-G_{se}}+\frac{r}{ck^5}} = \frac{e^{-G_{se}}}{e^{-G_{se}}+\frac{1}{ck^6}}$. Hieruit volgt dat, nadat de eerste inadequate hechting heeft plaatsgevonden, de kans dat een k-bottleneck optreed voordat alle tiles afvallen gelijk is aan $(\frac{e^{-G_{se}}}{e^{-G_{se}}+\frac{1}{ck^6}})^{k-1}$. Wegens Lemma 4.2.1 weten we dat de kans dat een eerste inadequate hechting plaatsvind gelijk is aan $r_f e^{-G_{se}}$. Hieruit volgt dat de kans dat een block error kan optreden gelijk is aan $r_f e^{-G_{se}}(\frac{e^{-G_{se}}}{e^{-G_{se}}+\frac{1}{ck^6}})^{k-1}$. ∎

***Stelling 4.2.2* [3]**

Als we er vanuit gaan dat er dat er geen k-bottlenecks zijn, en dat kans op een inadequate hechting maximaal $\mathcal{O}(\frac{r_f}{k^6})$ bedraagt, dan kan er een $N \times N$ vierkant van $2k \times 2k$ snaked proofreading blokken gevormd worden in $\mathcal{O}(\frac{k^5 N}{r_f})$ tijd.

Bewijs: In het snaked tile system zullen de alle tiles van een blok vasthangen met een totale bond strength van minstens 3 vanaf het moment dat het blok volledig samengesteld is. Hierdoor zullen blokken dus nooit meer loskomen. Deze aanname hadden we namelijk gemaakt aan het begin van deze paragraaf.

Omdat er geen k-bottlenecks kunnen optreden, kunnen de foutieve tiles die de plaats opvullen van het blok dat we willen gaan toevoegen hoogstens $k-1$ inadequate hechtingen bevatten. Wegens Lemma 4.2.7 en 4.2.8 weten we dat deze tiles zich terug zullen loskoppelen na maximaal $\mathcal{O}(\frac{k^5}{r_f})$ tijd waarna het blok zich kan hechten aan het $N \times N$ vierkant in $\mathcal{O}(\frac{k^4}{r_f})$ tijd wegens Lemma 4.2.6. We hebben aangenomen dat de kans op een inadequate hechting gelijk is aan $\mathcal{O}(\frac{r_f}{k^6})$. Bovendien zijn er slechts $\mathcal{O}(k)$ plaatsen waar een inadequate hechting kan plaatsvinden opdat ze da constructie van het huidige blok in gedrang kunnen brengen. Hierdoor wordt de kans dat er geen inadequate hechting kan plaatsvinden gelijk aan $\frac{1}{k\frac{r_f}{k^6}} = \mathcal{O}(\frac{k^5}{r_f})$, dit is eveneens de tijd die nodig is om 1 blok samen te stellen.

Volgens de running time analyse van Adleman et al.[1] zal een tile system een assembly van grootte N in $\mathcal{O}(N \times T_B)$ tijd geconstrueerd worden, hierbij is T_B de tijd die nodig is om 1 blok te construeren. We hebben zonet berekend dat $T_B = \mathcal{O}(\frac{k^5}{r_f})$ in onze situatie, hierdoor wordt het $N \times N$ vierkant geproduceerd in $\mathcal{O}(\frac{k^5 N}{r_f})$ tijd. ∎

4.2.3 Bespreking van simulaties

De bewijzen uit de vorige paragraaf gelden enkel onder strikte voorwaarden. Dit wil echter niet zeggen dat het snaked proofreading model daarom niet werkt voor andere tile sets onder andere condities. Ho-Lin en Ashish[3] hebben een reeks van simulaties uitgevoerd om te controleren hoe goed hun snaked proofreading model nu effectief werkt vergeleken met andere methodes.

	$G_{mc} = 15$, $G_{se} = 7.8$			$G_{mc} = 15$, $G_{se} = 8.0$		
	Origineel	Proofreading	Snaked	Origineel	Proofreading	Snaked
Constructie tijd:	550	2230	6020	350	1750	3780
Fouten percentage:	52%	24%	0%	63%	75%	0%
Tijd dat de assembly stabiel bleef na constructie:	0	0	> 400000	0	0	5700

Tabel 4.1: Dit zijn de resultaten van de simulaties die Ho-Lin en Ashish[3] hebben uitgevoerd om een 20×20 *Sierpinski blok te construeren.*

In Tabel 4.1 worden de resultaten van de simulaties getoond. De simulaties werden uitgevoerd op de originele tile set, een 4×4 proofreading tile set zoals beschreven in Paragraaf 4.1 en een 4×4 snaked proofreading tile set. Uit deze resultaten kunnen we afleiden dat de snaked proofreading tile set duidelijk een veel lager error percentage heeft ten opzichte van de overige twee tile sets. Merk ook op dat de snaked proofreading tile set er in slaagt om veel langer stabiel te blijven en dat de constructietijd slechts 3 maal trager is vergeleken met de gewone proofreading tile set. Gezien de drastische vermindering op vlak van fouten is dit zeker een aanvaardbare vertraging.

Hoofdstuk 5

Conclusie

Ik wil dit proefschrift beëindigen met het bespreken van enkele conclusies. In de eerste sectie wil ik enkele bemerkingen formuleren over DNA computing, vervolgens wil ik enkele onderwerpen aanhalen die ik niet behandeld hebt in dit proefschrift. In de daarna volgende sectie geef ik mijn algemene conclusies.

5.1 Bemerkingen en verder onderzoek

Ik heb in dit proefschrift aangetoond dat DNA zeer veel mogelijkheden biedt op computationeel vlak. Ik zou er graag de nadruk op leggen dat ook DNA computing zijn grenzen heeft en dat het dus geen magisch drankje is dat NP-complete problemen plots kan oplossen in polynomiale tijd. Je kan bijvoorbeeld wel alle mogelijke oplossingen parallel berekenen in 1 testbuis, maar het zoeken naar de juiste oplossing tussen al deze oplossing vergt nog steeds zeer veel tijd.

Ik heb in Hoofdstuk 3 en 4 respectievelijk besproken hoe je een self-healing en proofreading tile set kan construeren. Het is echter mogelijk om een tile set te construeren die zowel self-healing als proofreading is [9]. Merk op dat het moeilijker is dan simpelweg eerst de self-healing transformatie toe te passen en vervolgens de proofreading transformatie op uit te voeren het resultaat van de self-healing transformatie, of omgekeerd. Dit komt omdat beide transformaties zijn gedefinieerd op tiles die enkel weak bonds hebben op hun 4 zijden, maar ze resulteren wel in een tile sets die gebruik maakt van zowel null bonds, weak bonds als strong bonds.

Een andere techniek waar ik in dit proefschrift geen aandacht heb besteed is het gebruik van negatie bond strengths. Zo kan je de negatieve bond strengths bijvoorbeeld gebruiken om tiles terug los te maken van een assembly om op die manier tiles te kunnen hergebruiken. Doty et. al. [4] tonen echter aan dat je tiles niet oneindig lang kan blijven hergebruiken, er zal namelijk een moment komen dat ze zich permanent zullen vasthechten. Een ander toepassing voor negatieve bond strengths kan zijn dat je ze gebruikt om te voorkomen dat

een foutieve tile zich zal hechten aan de assembly.

Tot slot heb ik mij enkel geconcentreerd op tile systems die werken bij een vaste temperatuur gelijk aan 2. Twee vragen komen dan in mij op. Wat zijn de mogelijkheden indien we een temperatuur verschillend van 2 beschouwen? En biedt een temperatuur die veranderd naarmate de tijd vordert meer mogelijkheden dan een vaste temperatuur?

5.2 Algemene conclusie

Ik ben aan dit proefschrift begonnen met een beperkte kennis over DNA en geen kennis over DNA computing of DNA self assembly. Na mij het afgelopen jaar in dit onderwerp verdiept te hebben, kan ik zeggen dat ik enorm veel bijgeleerd heb. In eerste instantie heb ik veel geleerd over DNA en over DNA self assembly. Ik heb ook veel ervaring opgedaan bij het lezen van wetenschappelijke artikels en papers om deze daarna kritisch te verwerken. Het schrijven van deze masterproef heeft me veel bijgebracht over hoe je een groter werk aanpakt en welke schrijfstijl je best hanteert.

Persoonlijk ben ik tevreden over het resultaat dat ik bereikt heb. Ik heb op een begrijpbare manier kunnen neerschrijven hoe DNA self assembly werkt. Natuurlijk heb ik slechts een klein aspect behandeld uit het veel grotere topic van DNA computing, maar desondanks heb ik toch enkele transformaties voor tile sets kunnen voorstellen om zo verschillende types van errors te kunnen verhelpen.

DNA computing, en dus ook DNA self assembly, is een recent onderzoeksdomein dat nog in de kinderschoenen staat, maar waarin de laatste jaren reeds veel vooruitgang is geboekt. Het zal steeds spannend afwachten zijn wanneer men nieuwe theorieén zal uittesten in laboratoria's, hopend op een nieuwe doorbraak. Maar zoals in ieder opkomend onderzoeksdomein maakt de onwetendheid van wat de toekomst ons te bieden heeft het er alleen maar spannender op.

Bibliografie

[1] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 740–748. ACM, 2001.

[2] Leonard M Adleman et al. Molecular computation of solutions to combinatorial problems. *Science-AAAS-Weekly Paper Edition*, 266(5187):1021–1023, 1994.

[3] Ho-Lin Chen and Ashish Goel. Error free self-assembly using error prone tiles. In *DNA computing*, pages 62–75. Springer, 2005.

[4] David Doty, Lila Kari, and Benoît Masson. Negative interactions in irreversible self-assembly. In *DNA Computing and Molecular Programming*, pages 37–48. Springer, 2011.

[5] Urmi Majumder, Sudheer Sahu, Thomas H LaBean, and John H Reif. Design and simulation of self-repairing dna lattices. In *DNA Computing*, pages 195–214. Springer, 2006.

[6] Gheorghe Paun, Grzegorz Rozenberg, and Arto Salomaa. *DNA Computing: New Computing Paradigms (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[7] Joel L Schiff. *Cellular automata: a discrete view of the world*, volume 45. Wiley-Interscience, 2011.

[8] Michael Sipser. *Introduction to the Theory of Computation*. Course Technology, Cengage Learning, 2006.

[9] David Soloveichik, Matthew Cook, and Erik Winfree. Combining self-healing and proofreading in self-assembly. *Natural Computing*, 7(2):203–218, 2008.

[10] David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36(6):1544–1569, 2007.

[11] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, 1998.

[12] Erik Winfree. Simulations of computing by self-assembly. 1998.

[13] Erik Winfree. Self-healing tile sets. In *Nanotechnology: science and computation*, pages 55–78. Springer, 2006.

[14] Erik Winfree and Renat Bekbolatov. Proofreading tile sets: Error correction for algorithmic self-assembly. In *DNA computing*, pages 126–144. Springer, 2004.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Spontaan formerende structuren uit DNA tegels

Richting: **master in de informatica-databases**
Jaar: **2013**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vandesbosch, Niki

Datum: **6/06/2013**