Spiking Neural P-Systeme

Bakkalaureatsarbeit

zur Erlangung des akademischen Grades
Bakk.-Inf.

vorgelegt an der
Technischen Universität Dresden
Fakultät für Informatik

von

Steffen Gothan

25. März 2009

Lehrstuhl für Automatentheorie
Verantwortlicher Hochschullehrer: Prof. Dr. Franz Baader
Betreuerin: Dr. Monika Sturm

Technische Universität Dresden
Lehrstuhl für Automatentheorie
Institut für Theoretische Informatik

25.11.2008

Bakkalaureatsarbeit

Spiking Neural P-Systeme

Bearbeiter: Steffen Gothan
Studiengang Mathematik/Informatik

Im Fachgebiet DNA Computing werden als Vertreter eines „*Molekularen Rechnens in vivo*“ sogenannte „*P-Systeme*“ thematisiert. Die nach G. Păun benannten Modellierungsmöglichkeiten von Zellen und Membranen ermöglichen Forschungsarbeiten im biologischen Fokus. Einerseits kann damit ein effizientes, biologisches Rechnen modelliert werden und andererseits können mit diesen Modellierungen Prozesse, die systembiologisch als wohluntersucht gelten, als ein *biologisches* Rechnen analysiert werden. An der TU Dresden wurde bisher mit Splicing-Membran-Systemen gearbeitet, die in engem Bezug zu klassischen DNA-Computing-Modellen stehen. Ziel dieser Arbeit soll es sein, sogenannte „*Spiking Neural P-Systeme*“ zu untersuchen. Dabei soll in der Arbeit auf folgende Schwerpunkte eingegangen werden:

- Untersuchung und Darstellung unterschiedlicher Ansätze für „*Spiking Neural P-Systeme*“
- Abgrenzung von anderen bisher an der TU Dresden bekannten „*P-Systemen* “
- Definition eines allgemeinen „*Spiking Neural P-Systems* “
- Betrachtung, Beweis und Bewertung der Berechnungsstärke eines „*Spiking Neural P-Systems*“ (mit einer formal sauberen Darstellung des dazugehörigen Beweises)
- Diskussion möglicher Anwendungen (u.U. aus der Informatik hinausgehend)

Folgende Literatur wird als Pflichtliteratur angesehen und ist auszuwerten:

- G. Paun. *Membrane Computing. An Introduction.* Springer-Verlag, Berlin, 2002.
- O.H. Ibarra, A. Păun, G. Păun, A. Rodríguez-Patón, P. Sosik, and S. Woodworth. *Normal forms for spiking neural P systems.* In Miguel Angel Gutièrrez-

Naranjo, Gheorghe Păun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume II, pages 105-136. Fénix Editora, 2006.

- H. Chen, R. Freund, M. Ionescu, G. Păun, and M.J. Pérez-Jiménez. *On string languages generated by spiking neural P systems.*. In Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume I, pages 169-194. Fenix Editora, 2006.

- H. Chen, M. Ionescu, and T.O. Ishdorj. *On the efficiency of spiking neural P systems.* In Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume I, pages 195-206. Fenix Editora, 2006.

Betreuerin: Dr. M. Sturm
verantwortlicher Hochschullehrer: Prof. F. Baader
Bearbeitungszeitraum: 25.11.2008 - 25.03.2009
Abschlußleistung: schriftliche Bakkalaureatsarbeit, Verteidigung mit Vortrag

Erklärung an Eides statt

Hiermit erkläre ich an Eides statt, dass ich diese Bakkalaureatsarbeit zum Thema „Spiking Neural P-Systeme“ unter Betreuung von Dr. M. Sturm und Prof. F. Baader selbstständig erarbeitet, verfasst und Zitate kenntlich gemacht habe. Andere als die angegebenen Hilfsmittel wurden von mir nicht benutzt.

Steffen Gothan
Dresden, 25. März 2009

Inhaltsverzeichnis

1 Einleitung

Das Fachgebiet des *Natural Computing*, des biologischen Rechnens, fasst verschiedene Aspekte des Rechnens im Zusammenhang mit in der Molekularbiologie auftretenden Vorgängen zusammen. So werden einerseits in *in-vitro*-Modellen DNA-basierte Rechner konstruiert, die als Modelle des *Unconventional Computing* schwere Probleme der Informatik effizient lösen sollen. Dies geschieht unter Verwendung molekularbiologischer Verfahren zur Manipulation von DNA-Molekülen. Auf diese Weise können hilfreiche Eigenschaften der DNA-Moleküle genutzt werden, wie beispielsweise ihre hohe Speicherdichte, der geringe Energieverbrauch der DNA-Operationen und die hohe Parallelität der Verfahren, bei denen eine große Zahl von Molekülen gleichzeitig manipuliert werden kann.

In *in-vivo*-Modellen hingegen werden Konzepte verwendet, die durch Vorgänge und Prozesse aus der Natur inspiriert sind. Mit ihrer Hilfe werden neuartige Rechnermodelle *in-silico* geschaffen. Auch sie sollen Wege zur effizienten Problemlösung aufzeigen.

Ausgangspunkt für das biologische Rechnen waren theoretische Grundlagen zum Rechnen mit DNA unter Verwendung molekularbiologischer Verfahren und Operationen. Mit ihrer Hilfe konnte das Adleman-Experiment realisiert werden. In diesem Laborexperiment, das 1994 stattfand, wurde eine Instanz des Hamiltonkreis-Problems in DNA kodiert und durch DNA-Operationen gelöst. Es folgten weitere Vertreter des biologischen Rechnens, die oft DNA-Moleküle als Informationsträger verwendeten, beispielsweise das Splicing-Modell und die Insertion-Deletion-Systeme.

Einen anderen Ansatz verfolgt das *Membrane Computing* als ein Vertreter der *in-vivo*-Modelle. Es formalisiert die grundlegende Eigenschaft von Zellen, in Organismen eine meist hierarchische Organisationsstruktur auszubilden. Sie wirken als Membranen auf den Stofftransport des Organismus ein. Membranen sind halbdurchlässige Schichten, die den Transport von Molekülen in Zellen steuern. Dem Konzept der Membran wird in der Biologie ein sehr hoher Stellenwert beigemessen.

Ein sehr bekannter Vertreter des Membrane Computing ist das von Gheorghe Păun entwickelte P-System. Die erste Veröffentlichung (Vgl. [PRS98]) über P-Systeme erschien 1998. Viele weitere Arbeiten folgten, unter anderem im Buch [Pău02] veröffentlicht, in dem wichtige Vertreter der P-Systeme vorgestellt werden und Ergebnisse zu diesem Themenkomplex zusammengetragen sind. Schon zu dieser Zeit wurde von Păun vorgeschlagen, nicht nur Zellhierachien zu betrachten, sondern auch andere

Membranorganisationen in P-Systemen abzubilden. So werden in [Pău02] gewebeähnliche P-Systeme vorgestellt, in denen die Membranen eine Netzstruktur bilden. Als Vertreter für ein solches Netz sind Neuronennetze genannt. Dieses Modell wurde in [IPY05] präzisiert und führte zur Theorie der *Spiking Neural P-Systeme*. Ein wichtiger Untersuchungsgegenstand für diese Modelle ist ihre Berechnungsstärke und wie durch Vorgänge aus der Natur inspirierte Konzepte in den Modellen umgesetzt werden können.

Eine umfangreiche Darstellung dieses Themengebietes in der kürzlich erschienenen Ausgabe der Fachzeitschrift Natural Computing (Vgl. [YIP08]) unterstreicht die große Relevanz der Spiking Neural P-Systeme für das biologische Rechnen.

In dieser Arbeit wird im ersten Teil das P-System und seine geläufigen Modifikationen als Ausgangspunkt für die Entwicklung der Spiking Neural P-Systeme vorgestellt. Im zweiten Teil wird ein allgemeines Spiking Neural P-System definiert und ein Überblick über die verschiedenen Varianten dieses Modells gegeben. Im dritten Teil wird für das zuvor definierte Modell Turing-Vollständigkeit bewiesen und eine Anwendung des Modells vorgestellt.

Die in dieser Arbeit verwendeten Definitionen aus der Graphentheorie, den formalen Sprachen und zur Berechenbarkeit werden in Kapitel 2 eingeführt.
In Kapitel 3 wird ein Überblick über die P-Systeme und einige ihrer Varianten gegeben. In diesem Kapitel wird auch die formale Definition des P-Systems aus [Pău02] in einer vereinfachten Form vorgestellt. Diese enthält nur die für die Grundfunktionalität wesentlichen Elemente. Insbesondere werden Regelprioritäten und Membranauflösung nicht betrachtet. Die Definition liefert ein Gerüst für die Entwicklung einer formalen Definition der Spiking Neural P-Systeme.
In Kapitel 4 ist das Splicing-Membran-System als eine Variante der P-Systeme beschrieben. Dazu wird die Splicing-Operation definiert und anschließend eine wiederum auf der formalen Beschreibung des P-Systems aufbauende Definition des extendierten Splicing-Membran-Systems angegeben.
Im Hauptteil der Arbeit wird in Kapitel 5 die in vielen Arbeiten zu Spiking Neural P-Systemen verwendete Definition des Systems formalisiert. Dazu wird zunächst die Funktionsweise eines solchen Systems erläutert und anschließend die formale Definition angegeben. Im letzten Teil dieses Kapitels wird ein Überblick über die zahlreichen Varianten von Spiking Neural P-Systemen gegeben und erläutert, warum es nicht möglich ist, die Funktionsweise der SNP-Systeme durch P-Systeme zu realisieren.
Der Beweis der Turing-Vollständigkeit für Modelle der Klasse der SNP-systeme wird in Kapitel 6 erbracht. Es wird beschrieben, wie für eine beliebige Registermaschine eines bestimmten Typs ein äquivalentes Spiking Neural P-System konstruiert werden kann und eine formale Beschreibung der Konstruktion angegeben.
In Kapitel 7 werden Anwendungsmöglichkeiten von Spiking Neural P-systemen diskutiert.

2 Mathematische Grundlagen

2.1 Mengen und Multimengen

In dieser Arbeit werden die üblichen Definitionen für die natürlichen Zahlen $\mathbb{N}$, Mengen und Mengenoperationen, sowie Relationen und Abbildungen angenommen, wie sie beispielsweise in [Ihr94] verwendet werden.
Für eine Menge X ist ihre Potenzmenge $\mathcal{P}(X)$ diejenige Menge, die alle Teilmengen von X enthält. Die Menge $\mathcal{P}_{fin}(X)$ ist die Menge, die alle endlichen Teilmengen von X enthält. Es gilt $\mathcal{P}_{fin}(X) \subseteq \mathcal{P}(X)$.
Eine *Multimenge über X* ist eine Abbildung $M : X \to \mathbb{N}$. Für $x \in X$ wird $M(x)$ als die Vielfachheit des Elementes x in M bezeichnet und gibt die Anzahl der in der Multimenge vorhandenen Kopien von x an. $\mathbf{M}(X)$ ist die Klasse aller Multimengen über X. Für eine endliche Menge $X = \{x_1, x_2, \ldots, x_n\}$ kann die Multimenge M über X in der Form $M = \{(x_1, M(x_1)), (x_2, M(x_2)), \ldots, (x_n, M(x_n))\}$ notiert werden. Im Zusammenhang mit P-Systemen werden Multimengen über einer Menge X häufig als Wörter über dem Alphabet X interpretiert. Der *Träger* einer Multimenge M über X, notiert mit $supp(M)$, ist die Menge $supp(M) = \{x \in X | M(x) > 0\}$. Eine Multimenge M ist leer, notiert mit $M = \emptyset$, wenn ihr Träger leer ist.

Es seien M_1 und M_2 zwei Multimengen über X. M_1 ist eine Teilmenge von M_2, notiert mit $M_1 \subseteq M_2$ oder $M_1 \leq M_2$, wenn für jedes x aus X gilt, dass $M_1(x) \leq M_2(x)$. Das Skalarprodukt einer natürlichen Zahl n mit einer Multimenge M über X ist definiert als $\otimes : \mathbb{N} \times \mathbf{M}(X) \to \mathbf{M}(X)$, wobei $(n, M) \mapsto \overline{M} = n \otimes M$ mit $\overline{M}(a) = nM(a)$ für jedes $a \in X$. Der Schnitt einer Multimenge M über X mit einer Menge Y ist definiert als $M \cap Y := \{(x, M(x)) | x \in supp(M) \wedge x \in Y\}$. Die Vereinigung zweier Multimengen M und N über X ist definiert als $M \cup N = M + N = \{(x, M(x) + N(x)) | x \in X\}$. Die Mengendifferenz $M \backslash N$ für $N \leq M$ ist definiert als $M \backslash N = M - N = \{(x, M(x) - N(x)) | x \in X\}$. Die Anzahl der Elemente einer Multimenge M über X, notiert mit $|M|$, ist definiert als

$$|M| = \sum_{x \in X} M(x).$$

Beispiel 2.1. Es sei $X = \{a_1, a_2, a_3\}$ eine Menge. Die Multimenge A, die zwei Kopien des Elementes a_1, eine Kopie von a_2 und keine Kopie von a_3 enthält, wird mit $A = \{(a_1, 2), (a_2, 1), (a_3, 0)\}$ oder $A = \{(a_1, 2), (a_2, 1)\}$ notiert. Als Wort interpretiert kann M als $a_1^2 a_2$ oder auch $a_2 a_1 a_1$ dargestellt werden.

Ist die Multimenge $B \in \mathbf{M}(X)$ mit $B = \{(a_1, 3), (a_2, 1), (a_3, 0)\}$ gegeben, so gilt $A \subseteq B$.

2.2 Graphentheorie

Ein *ungerichteter Graph* G ist ein geordnetes Paar $G = (V, E)$. V ist eine Menge, deren Elemente als *Knoten* und E ist eine Menge von zweielementigen Mengen $\{u, v\}$ mit $u \in V$, $v \in V$ und $u \neq v$, deren Elemente als *Kanten* oder *Bögen* von G bezeichnet werden. Ist $\{u, v\}$ ein Bogen des Graphen G, so sind die Knoten u und v *benachbart*. Die Knotenmenge von G kann auch mit $V(G)$, die Kantenmenge mit $E(G)$ notiert werden.
Ein *Pfad p der Länge $length(p) = k$* vom Knoten u zum Knoten v im ungerichteten Graphen $G = (V, E)$ ist eine Folge von Knoten $p = (x_0, x_1, \ldots, x_k)$ mit $u = x_0$, $v = x_k$ und $\{x_j, x_{j+1}\} \in E$ für $0 \leq j < k$. Gibt es einen Pfad von u nach v, so werden Knoten u und v als *zusammenhängend in* G bezeichnet. Ein ungerichteter Graph ist zusammenhängend, wenn jede Auswahl zweier Knoten in ihm zusammenhängend ist. Ein Pfad heißt *einfach*, wenn je zwei seiner Knoten paarweise verschieden sind mit Ausnahme des ersten und des letzten Knotens des Pfades, die gleich sein dürfen. Ein *Zyklus* ist ein einfacher Pfad mit gleichem Anfangs- und Endknoten, der eine Länge von mindestens 1 hat. Ein ungerichteter Graph, in dem es keine Zyklen gibt, heißt *azyklisch*. Ein *Baum* ist ein zusammenhängender, ungerichteter, azyklischer Graph. Ein *gewurzelter Baum* ist ein Baum, mit einem speziell ausgezeichneten Knoten, der als die *Wurzel* des Baumes bezeichnet wird. Es sei u ein Knoten eines gewurzelten Baum mit Wurzel w. Es gibt genau einen einfachen Pfad von w nach u. Jeder von u verschiedene Knoten dieses Pfades wird als *echter Vorfahre* von u bezeichnet. Ist ein Knoten s ein echter Vorfahre des Knotens t, dann heißt t *echter Nachfahre* von s. Ein Knoten ohne echte Nachfahren heißt *Blatt*. Für einen Knoten u wird der eindeutig bestimmte Knoten $F(u)$, der ein echter Vorfahre von u ist und für den eine Kante $\{F(u), u\}$ existiert, als *Elternknoten* von u bezeichnet. Die Menge der *Kindknoten* $Ch(u)$ von u ist die Menge der echten Nachfahren von u, deren Elternknoten u selbst ist.

Ein *gerichteter Graph* ist ein geordnetes Paar $G = (V, E)$. V ist eine endliche Menge, deren Elemente als Knoten und E ist eine binäre, irreflexive Relation auf V, deren Elemente als *Kanten* des gerichteten Graphen G bezeichnet werden. Die Knotenmenge des gerichteten Graphen G kann auch mit $V(G)$, die Kantenmenge mit $E(G)$ notiert werden.

Knoten und Kanten eines ungerichteten oder gerichteten Graphen G können mit *Markierungen* versehen werden. Es sei X eine Menge. Eine *Knotenmarkierung mit Marken aus X* ist eine Abbildung $L : V(G) \to X$, eine *Kantenmarkierung mit Marken aus X* ist eine Abbildung $L : E(G) \to X$. Die Marke eines Knotens oder einer Kante wird auch als *Label* bezeichnet.

2.3 Formale Sprachen

Ein *Alphabet* Σ ist eine nichtleere Menge abstrakter Symbole. Ein *Wort* über Σ ist eine endliche Folge $a_1 a_2 \dots a_n$ von Symbolen aus Σ. Das leere Wort wird mit ε bezeichnet. Bei gegebenem Alphabet Σ wird mit Σ^* die Menge aller Wörter über Σ und mit Σ^+ die Menge aller Wörter über Σ, die nicht das leere Wort sind, bezeichnet. Für Wörter $r = r_1 \dots r_m$ und $s = s_1 \dots s_n$ aus Σ^* ist die *Wortverkettung*, auch bezeichnet als *Konkatenation*, definiert als $\cdot : \Sigma^* \times \Sigma^* \to \Sigma^*$ mit $r \cdot s = r_1 \dots r_m s_1 \dots s_n$. Ein Wort $a_1 \dots a_n$ hat die *Länge* n, notiert mit $|a_1 \dots a_n| = n$.

Eine beliebige Menge $L \subseteq \Sigma^*$ heißt *formale Sprache*. Jedes Element von L wird *Wort der formalen Sprache* L genannt. Die leere Sprache wird mit $\emptyset$ bezeichnet. Für zwei formale Sprachen $L_1 \subseteq \Sigma_1^*$ und $L_2 \subseteq \Sigma_2^*$ ist die Verkettung beider Sprachen definiert als $L_1 \cdot L_2 = \{r \cdot s | r \in L_1 \wedge s \in L_2\}$. Für eine formale Sprache $L \subseteq \Sigma^*$ ist für $n \in \mathbb{N}$ definiert: $L^0 := \{\varepsilon\}$ und $L^{n+1} = L \cdot L^n$. Die reflexive transitive Hülle L^* von L ist definiert als

$$L^* = \bigcup_{n \in \mathbb{N}} L^n.$$

Die Menge der *regulären Ausdrücke* über einem Alphabet Σ, bezeichnet mit REG_Σ, ist induktiv definiert: $\emptyset$, ε und a für jedes $a \in \Sigma$ sind reguläre Ausdrücke über Σ. Sind r und s reguläre Ausdrücke, dann auch $r + s$, $r \cdot s$ und r^*. Die durch einen regulären Ausdruck r definierte Sprache $L(r)$ ist induktiv wie folgt definiert: $L(\emptyset) := \emptyset$, $L(\varepsilon) := \{\varepsilon\}$, $L(a) := \{a\}$, $L(r + s) := L(r) \cup L(s)$, $L(r \cdot s) := L(r) \cdot L(s)$, $L(r^*) := (L(r))^*$.

2.4 Berechenbarkeit und Komplexität

Es wird im Folgenden die Definition der Turing-Maschine nach [Sch08] verwendet. Eine k-stellige Funktion $f : \mathbb{N}^k \to \mathbb{N}$ heißt *Turing-berechenbar*, wenn es eine deterministische Turing-Maschine gibt, die bei Eingabe eines beliebigen Argumentes $(n_1, \dots, n_k)$ in Binärdarstellung auf dem Band nach endlich vielen Schritten eine Stoppkonfiguration erreicht und sich auf dem Band anschließend $f(n_1, \dots, n_k)$ in Binärdarstellung befindet.
Die Churchsche These besagt, dass die durch den Begriff der Turing-Berechenbarkeit erfassten Funktionen genau diejenigen Funktionen sind, die im intuitiven Sinne als die berechenbaren Funktionen gelten. Ein formales Modell wird als *Turing-vollständig* bezeichnet, wenn es genau die Turing-berechenbaren Funktionen berechnen kann.
Es werden im Folgenden die üblichen Definitionen zur Komplexitätstheorie und insbesondere des Begriffes der Komplexitätsklasse NP verwendet, wie sie in der Literatur zu diesem Themengebiet (Vgl. [Sch08], [Hop01], [AB03]) nachzulesen sind. Es seien Sprachen $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ gegeben. A ist auf B *polynomial reduzierbar*, notiert mit $A \leq_p B$, falls es eine totale und mit polynomialer Laufzeit berechenbare

Funktion $f : \Sigma^* \to \Gamma^*$ gibt, so dass für jedes $x \in \Sigma^*$ gilt: $x \in A \Rightarrow f(x) \in B$. Eine Sprache A heißt *NP-hart*, falls für alle Sprachen $L \in NP$ gilt, dass $L \leq_p A$. Die Sprache A heißt *NP-vollständig*, falls A NP-hart ist und $A \in NP$ gilt.

3 P-Systeme

P-Systeme sind ein Vertreter des Membrane-Computing. Sie modellieren den Stofftransport in einer Zellhierarchie. Eine Zelle ist als durchlässige Membran abstrahiert, die bestimmten Regeln folgend Objekte mit der Umgebung oder mit weiteren in ihr enthaltenen Zellen austauscht. Ziel einer solchen Modellierung ist nicht, biologische Vorgänge abzubilden, sondern ein von Vorgängen in der Natur inspiriertes Rechnermodell zu schaffen, das in der Lage ist, schwere Probleme der Informatik effizient zu lösen.

P-Systeme wurden in [Pău98] vorgestellt und in [PJSC02] formalisiert. Viele Ergebnisse zu diesem Themengebiet sind in [Pău02] zusammengefasst.

Ein Überblick über die Funktionsweise eines P-Systems und verschiedener Varianten wird in Abschnitt 3.1 dieses Kapitels gegeben. Die formale Definition folgt in Abschnitt 3.2.

3.1 Überblick über P-Systeme

Das Membrane Computing abstrahiert die Struktur und Wirkmechanismen von lebenden Zellen. Grundlegendes Element ist die *Membran*, die *Regionen* physisch voneinander trennt und Stofftransport nur nach festgelegten Gesetzmäßigkeiten zulässt. Die P-Systeme als Vertreter des Membrane Computing formalisieren diesen Ansatz.

Die hierarchische Anordnung von Zellen wird in P-Systemen durch die *Membranstruktur* des Systems abgebildet. Sie beschreibt, auf welche Weise Membranen in anderen Membranen enthalten sind, bildet also eine Baumstruktur. Die Wurzel dieses Baumes als höchstes Element in der Hierarchie ist die *Umgebung*, in die die *Haut*, auch bezeichnet als *Skin*, als Hauptmembran eingebettet ist, welche alle weiteren Membranen von der Umgebung abgrenzt. Die Umgebung enthält keine weitere Membran außer der *Skin*. Jede Membran ist in eine Region eingebettet und kann eine endliche Anzahl weiterer Membranen enthalten. Eine Membran, die selbst keine Membranen enthält, wird als *Elementarmembran* bezeichnet. Abbildung 3.1 zeigt, wie eine Membranstruktur in einem Venn-Diagrann dargestellt werden kann. In diesem Beispiel bildet die Membran m_1 die *Skin*, in die hierarchisch weitere Membranen eingebettet sind. Die Region außerhalb der *Skin* ist die Umgebung.

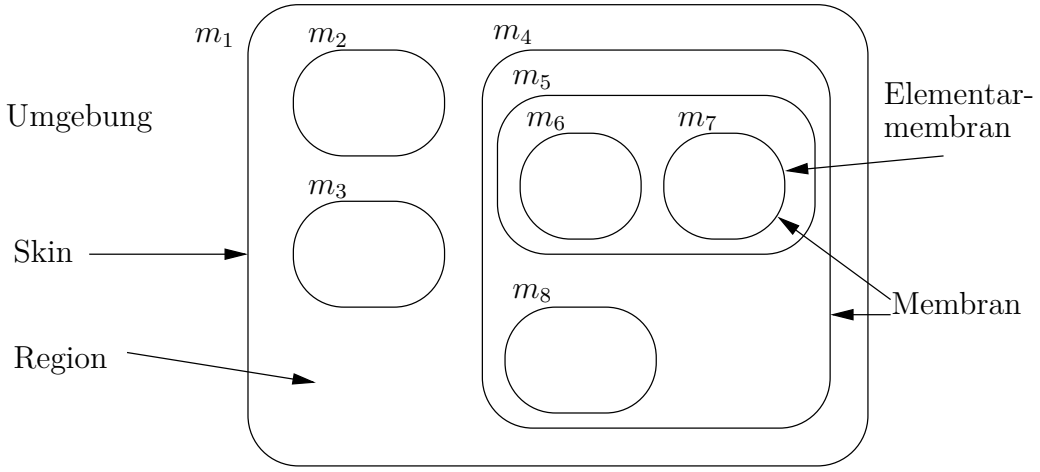

Abbildung 3.1: Beispiel für eine Membranstruktur

Der Stoffaustausch in einem solchen System wird durch Symbol-Objekte beschrieben. Dies sind in diesem Modell Elemente einer Menge, die als das Alphabet von Symbol-Objekten des P-Systems bezeichnet wird. In einer *Startkonfiguration* wird jeder Region außer der Umgebung eine Multimenge über diesem Alphabet zugewiesen, die als die Multimenge der in dieser Region beziehungsweise Membran enthaltenen Objekte bezeichnet wird. Eine Zustandsbeschreibung des Systems, das heißt eine Angabe über die in jeder Region enthaltenen Objekte, wird *Konfiguration* genannt. In den Regionen können sich beliebig viele Objekte befinden.

Von der Startkonfiguration ausgehend werden in Berechnungsschritten neue Konfigurationen erzeugt. Jeder Membran ist eine Menge von *Entwicklungsregeln*, auch bezeichnet als *Evolutionsregeln*, zugeordnet. Durch die Anwendung einer solchen Regel werden in der entsprechenden Membran Objekte konsumiert und aus der Konfiguration dieser Membran entfernt. Gleichzeitig werden neue Objekte erzeugt, die sich anschließend abhängig von der *Zielinformation* der Regel für diese Multimenge in der Membran selbst, ihrer Elternmembran oder einer ihrer Kindmembranen befinden. Die Anwendung solcher Regeln geschieht maximal, parallel und nichtdeterministisch. Maximalität bedeutet, dass stets so viele Entwicklungsregeln wie möglich angewendet werden, das heißt es verbleiben während der Anwendung aller ausgewählten Regeln in einem Berechnungsschritt nie so viele Objekte in der Membran, dass noch eine weitere Regel in ihr angewendet werden könnte. Dennoch können mehrere Möglichkeiten existieren, maximale Regelmengen zur Anwendung in einer Membran auszuwählen. Parallelität systemweit und innerhalb jeder Membran ist dadurch gegeben, dass in einem Berechnungsschritt in allen Membranen gleichzeitig jeweils die ausgewählten maximalen Regelmengen ausgeführt werden.

Der *Berechnungsbaum* des Systems gibt an, wie Konfigurationen in Berechnungs-

schritten entstehen. Dabei wird als die Wurzel des Baumes die Startkonfiguration gewählt und induktiv für jede aus einer Konfiguration in einem Berechnungsschritt entstandene Folgekonfiguration ein Kindknoten dieser Konfiguration dem Baum hinzugefügt. Eine erfolgreiche Berechnung des P-Systems ist ein endlicher Pfad im Berechnungsbaum von der Startkonfiguration bis zu einer Haltekonfiguration, die durch ein Blatt des Baumes repräsentiert ist. Die von dem P-System in dieser Berechnung generierte natürliche Zahl ist die Anzahl der Objekte, die sich in der Haltekonfiguration der Berechnung in einer festgelegten Ausgabemembran befinden. Die von dem P-System generierte Menge natürlicher Zahlen enthält genau diejenigen Zahlen, die in erfolgreichen Berechnungen des P-Systems generiert werden.

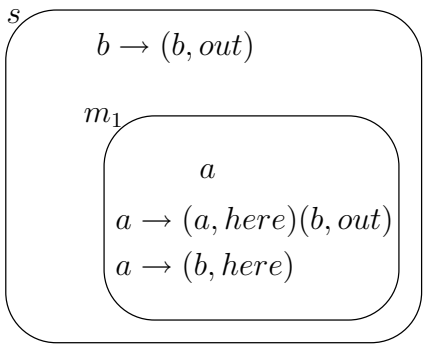

Abbildung 3.2: Beispiel für ein P-System

P-Systeme werden üblicherweise wie in dem Beispiel in Abbildung 3.2 in einem Venn-Diagramm dargestellt. Membranen erscheinen als Rechtecke mit abgerundeten Ecken und werden mit einem Bezeichner versehen. In einer Membran sind jeweils ihre Entwicklungsregeln und die Multimenge der in der Startkonfiguration in der Membran enthaltenen Objekte, sowie gegebenenfalls ihre Kindmembranenen aufgelistet. Regeln werden in der Form $w \to (w_1, tar_1) \dots (w_n, tar_n)$ notiert, wobei w_i für jedes i mit $1 \leq i \leq n$ die durch Anwendung dieser Regel erzeugte Multimenge dargestellt als Wort und tar_i die Zielinformation für das Wort w_i angibt. Die Ausgaberegion kann in dieser Darstellung nicht angegeben werden. Üblicherweise wird die Umgebung zur Ausgabe verwendet.

Das P-System in diesem Beispiel enthält in der Startkonfiguration nur in der Membran m_1 das Objekt a, die *Skin* s und die Umgebung e sind leer. Der Berechnungsbaum des Systems ist in Beispiel 3.13 angegeben. Bei Vorliegen der Startkonfiguration wird nichtdeterministisch eine der Regeln der Membran m_1 ausgewählt. Weitere Regeln können nicht angewendet werden. Wird die erste Regel $a \to (a, here)(b, out)$ ausgeführt, so wird dabei zwar das Objekt a in m_1 konsumiert, jedoch auch ein a mit Zielinformation *here* erzeugt, so dass die vorherige Konfiguration in m_1 wieder hergestellt ist. Zudem wird ein Objekt b in der *Skin* erzeugt. Nachdem die erste Regel in m_1 angewendet wurde, wird im darauf folgenden Schritt in der *Skin* stets

die Regel $b \rightarrow (b, out)$ ausgeführt, die das b in die Umgebung verschiebt.
Sobald zum ersten Mal die zweite Regel $a \rightarrow (b, here)$ in m_1 angewendet wird, bricht die Berechnung ab. In m_1 kann im darauf folgenden Schritt keine Regel mehr angewendet werden. Gegebenenfalls verschiebt die *Skin* noch ein in ihr vorhandenes b in die Umgebung, kann aber danach ebenfalls keine Regeln mehr anwenden, so dass eine Haltekonfiguration erreicht ist.

Wird im Schritt $n \in \mathbb{N}$ zum ersten Mal die zweite Regel in m_1 ausgeführt, so enthält die Umgebung bei Erreichen der Haltekonfiguration nach $n + 1$ (für $n \geq 1$) beziehungsweise n ($n = 0$) Schritten genau $n - 1$ beziehungsweise 0 Objekte. Die vom System erzeugte Menge ist die Menge, die das Ergebnis jeder möglichen Berechnung enthält. In diesem Fall ist dies die Menge der natürlichen Zahlen $\mathbb{N}$.

Oft wird die Funktionsweise eines P-Systems erweitert, um in der Realität auftretende Sachverhalte abbilden zu können oder eine höhere Berechnungsstärke des Modells zu erreichen. Die im Folgenden vorgestellten Modifikationen können in beliebiger Kombination zusammen verwendet werden. Alle Varianten wurden bereits in [Pău98] vorgestellt und können auch in [Pău02] nachgelesen werden.

Katalysatoren sind Objekte, deren Vorhandensein für die Anwendbarkeit einer Evolutionsregel zwar notwendig sein kann, die im Fall der tatsächlichen Anwendung der Regel jedoch nicht konsumiert und aus der Konfiguration der Membran entfernt werden. Die Menge der Katalysatoren wird als eine Teilmenge des Objektalphabetes definiert. Ein P-System, das Katalysatoren verwendet, wird als *katalytisches P-System* bezeichnet. Katalysatoren wurden beispielsweise in [PD99] und [Iba05] vertieft untersucht.

Als Ergebnis einer Regelanwendung kann eine *Membranauflösung* stattfinden. Dazu werden alle Entwicklungsregeln des P-Systems mit einer Information darüber versehen, ob die Membran, in der die Regel angewendet wird, nach deren Anwendung aufgelöst wird. Ist dies der Fall, so werden die in der Membran enthaltenen Objekte von der die Membran umgebenden Region aufgenommen. Die *Skin* kann nicht aufgelöst werden.

Regelprioritäten werden verwendet, um in einer Membran bei gleichzeitiger Anwendbarkeit zweier Regeln die Regelanwendung zu steuern. Es wird jeder Membran m eine strikte Partialordnung auf der Menge ihrer Entwicklungsregeln zugewiesen, die als die *Prioritätsrelation* von m bezeichnet wird. Eine Regel darf in m in einem Berechnungsschritt nur dann angewendet werden, wenn es keine zweite Regel in m mit einer höheren Priorität gibt.

3.2 Formale Definition eines P-Systems

Es wird in diesem Abschnitt zunächst eine Definition des P-Systems als 6-Tupel angegeben. Darin verwendete Begriffe werden anschließend definiert.

Definition 3.1. Ein *P-System mit Symbol-Objekten* ist ein 6-Tupel $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ mit folgenden Eigenschaften:

1. Das Objektalphabet O ist eine nichtleere, endliche Menge von *Symbol-Objekten*.
2. $\Sigma \subseteq O$ ist das Terminalalphabet von Π.
3. μ ist eine Membranstruktur gemäß Definition 3.2.
4. M_0 ist eine Konfiguration für μ über O gemäß Definition 3.3. Sie wird als die *Startkonfiguration* von Π bezeichnet. Der Umgebung e, repräsentiert durch den Wurzelknoten von μ, wird die leere Menge zugeordnet, das heißt in der Startkonfiguration befinden sich keine Objekte in der Umgebung.
5. R ist eine Regelzuweisung für μ über O gemäß Definition 3.5.
6. $i_0 \in V(\mu)$ ist die Ausgaberegion von Π.

□

Eine Membranstruktur spiegelt die Zellhierarchie des P-Systems wider. Sie ist als ein gewurzelter Baum definiert. Dessen Knoten bilden je eine Membran beziehungsweise eine Region des Systems ab. Der Wurzelknoten des Baumes repräsentiert die Umgebung des Systems.

Definition 3.2. Eine Membranstruktur ist ein gewurzelter Baum $\mu = (V(\mu), E(\mu))$, dessen Wurzelknoten genau einen Kindknoten hat. Die Knoten des Baumes $V(\mu)$ heißen *Regionen*. Knoten, die nicht die Wurzel des Baumes sind, werden auch als *Membranen* bezeichnet.

Die Wurzel des Baumes ist die *Umgebung* der Membranstruktur. Ihr einziger Kindknoten wird als *Haut* oder *Skin* und jedes Blatt des Baumes als *Elementarmembran* bezeichnet. □

Eine Konfiguration für ein P-System Π gibt einen Zustand von Π an. Dazu wird jeder Region eine Multimenge über dem Objektalphabet O des Systems zugeordnet.

Definition 3.3. Es sei ein O ein Alphabet und μ eine Membranstruktur. Eine *Konfiguration* M für μ über O ist eine Abbildung $M : V(\mu) \to \mathbf{M}(O)$, die jeder Membran eine Multimenge über O zuordnet. Eine *Konfiguration* M für μ über O eines P-Systems Π kann abgekürzt als eine Konfiguration von Π bezeichnet werden. □

Den Membranen eines P-Systems werden durch eine Regelzuweisung Entwicklungsregeln zugeordnet. Es wird zunächst der Begriff der Entwicklungsregel und anschließend die Regelzuweisung definiert. Eine Entwicklungsregel ist gegeben durch die Multimenge der bei ihrer Anwendung konsumierten Objekte und die Multimengen der durch die Regel erzeugten Objekte zusammen mit je einer Zielinformation für jede dieser Multimengen, die angibt, in welcher Region ihre Objekte erzeugt werden. Diese Zielinformation ist ein Element der Menge $V(\mu) \cup \{here, out\}$. Dabei gibt *here* an, dass die entsprechende Multimenge in der Membran selbst, im Fall *out* in der Elternmembran erzeugt wird. Gibt die Zielinformation eine Membran an, so wird die Multimenge in dieser Membran erzeugt, falls sie eine Kindmembran der Membran ist, in der die Entwicklungsregel angewendet wurde.

Definition 3.4. Es sei μ eine Membranstruktur und O ein Alphabet. Eine *Entwicklungsregel* oder *Evolutionsregel* für μ über O ist ein Paar $r = (d_r, v_r)$ mit folgenden Eigenschaften:

1. d_r ist eine Multimenge über O.
2. v_r ist eine Abbildung mit $v_r : V(\mu) \cup \{here, out\} \to M(O)$, wobei $here \notin V(\mu)$ und $out \notin V(\mu)$.

Für eine Membranstruktur μ und ein Alphabet O bezeichnet $\mathcal{R}_{(\mu,O)}$ die Klasse all ihrer Evolutionsregeln. □

Definition 3.5. Es sei μ eine Membranstruktur und O ein Alphabet. Eine *Regelzuweisung* für μ über O ist eine Abbildung $R : V(\mu) \to \mathcal{P}_{fin}(\mathcal{R}_{(\mu,O)})$, die jeder Region eine Menge von Entwicklungsregeln zuordnet. Der Umgebung $e \in V(\mu)$ werden keine Regeln zugeordnet, das heißt $R(e) = \emptyset$.
Für eine Membran $m \in V(\mu)$ wird die Menge der ihr zugeordneten Evolutionsregeln als eine endliche Folge $R(m) := (r_1^m, \ldots, r_{s_m}^m)$ notiert.

□

Beispiel 3.6. Das in Abbildung 3.2 vorgestellte P-System kann formal wie folgt beschrieben werden

$$\Pi = (O, \Sigma, \mu, M_0, R, e)$$

mit

- $O = \{a, b\}$.
- $\Sigma = \{b\}$
- $\mu = (V(\mu), E(\mu))$ mit $V(\mu) = \{e, s, m_1\}$ und $E(\mu) = \{\{e, s\}, \{s, m_1\}\}$. Die Wurzel des Baumes ist e.

- $M_0(e) = \emptyset$, $M_0(s) = \emptyset$, $M_0(m_1) = \{(a, 1)\}$.
- $$r_1^s = (\{b, 1\}, here \mapsto \{b, 1\}, \forall x \in V(\mu) \cup \{out\} : x \mapsto \emptyset)$$
$$r_1^{m_1} = (\{a, 1\}, here \mapsto \{a, 1\}, out \mapsto \{b\}, \forall x \in V(\mu) : x \mapsto \emptyset)$$
$$r_2^{m_1} = (\{a, 1\}, here \mapsto \{b, 1\}, \forall x \in V(\mu) \cup \{out\} : x \mapsto \emptyset)$$

□

Von der Startkonfiguration ausgehend werden in *Berechnungsschritten* neue Konfigurationen des P-Systems erzeugt. Dazu wird bei einer gegebenen Konfiguration für jede Membran untersucht, welche ihrer Entwicklungsregeln in welcher Häufigkeit gleichzeitig angewendet werden können. Diese Information wird in *Anwendbarkeitsvektoren* festgehalten, die angeben, welche Regeln in dieser Membran tatsächlich angewendet werden. In einer *Anwendbarkeitsmatrix*, bestehend aus einem je einem Anwendbarkeitsvektor für jede Membran, wird die Regelanwendung des gesamten Systems angegeben. Die *Ausführung* einer Anwendbarkeitsmatrix liefert eine neue Konfiguration des Systems.
Die Regelanwendung geschieht *maximal und parallel*. Anwendbarkeitsvektoren sind maximal in dem Sinne, dass es für einen Anwendbarkeitsvektor v keinen zweiten Vektor w gibt, in dem alle Regeln von v, aber noch weitere Regeln zur Anwendung festgelegt sind. Das heißt, es werden auf eine gewisse Weise immer so viele Regeln wie möglich angewendet.

Aufgrund des Nichtdeterminismus bei der Regelanwendung kann es für eine Konfiguration verschiedene Anwendbarkeitsmatrizen und somit verschiedene Folgekonfigurationen geben, die in je einem Berechnungsschritt erzeugt werden.

Der *Berechnungsbaum* zeigt an, wie sich Konfigurationen des Systems durch Ausführung von Berechnungsschritten entwickeln. Eine erfolgreiche Berechnung des P-Systems wird als ein endlicher Pfad von der Wurzel bis zu einem Blatt dieses Baumes definiert, wobei das Blatt als die Haltekonfiguration der Berechnung bezeichnet wird. Das Ergebnis dieser Berechnung ist diejenige natürliche Zahl, die durch die Anzahl der in der Haltekonfiguration in der Ausgabemembran enthaltenen Symbole aus dem Terminalalphabet bestimmt wird. Die Ausgabe des P-Systems ist die Menge der durch erfolgreiche Berechnungen generierten natürlichen Zahlen.

Es wird im Folgenden zunächst angegeben, wann eine Entwicklungsregel anwendbar ist und die Begriffe Anwendbarkeitsvektor und Anwendbarkeitsmatrix definiert. Anschließend wird die Ausführung einer solchen Matrix erklärt, der Berechungsbaum konstruiert und die Ausgabe des SNP-Systems definiert.

Definition 3.7. Es sei M eine Konfiguration des P-Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ und $m \in V(\mu)$ eine Membran des Systems. Die Evolutionsregel $r \in R(m)$, $r = (d_r, v_r)$ heißt *anwendbar in der Membran m bei Konfiguration M*, wenn gilt: $d_r \leq M(m)$, das heißt, die Membran m enthält alle Objekte, die notwendig sind, um diese Regel anwenden zu können.

Der *Anwendbarkeitsindex* $N_{Ap}(r, M, m)$ von r bei der Konfiguration M in der Membran m ist definiert als die maximal mögliche Anzahl von Anwendungen der Evolutionsregel r bei der Konfiguration M in der Membran m, das heißt

$$N_{Ap}(r, M, x) = max\{n | n \otimes d_r \leq M(x)\}.$$

□

Definition 3.8. Es sei M eine Konfiguration des P-Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$, $m \in V(\mu)$ eine Membran des Systems und $R(m) = (r_1^m, \ldots, r_{s_m}^m)$ die Menge der Entwicklungsregeln, die m zugewiesen sind. Der Vektor $\mathbf{p}_m = (p_{m,1}, \ldots, p_{m,s_m})$ heißt *Anwendbarkeitsvektor* über m für die Konfiguration M, wenn gilt:

1. Jede Regel r_i^m kann bei der Konfiguration M so oft wie in p_m angegeben angewendet werden, das heißt für jedes i mit $1 \leq i \leq s_m$ gilt $\mathbf{p}_{m,i} \leq N_{Ap}(r_i^m, M, m)$.
2. Alle Regeln können gleichzeitig angewendet werden, das heißt
$$\sum_{i=1}^{s_m} \mathbf{p}_{m,i} \otimes d_{r_i^m} \leq M(m).$$
3. Der Vektor $\mathbf{p}_m$ ist maximal, das heißt es gibt keinen Vektor $\mathbf{q}_m = (q_{m,1}, \ldots, q_{m,s_m})$, der die Bedingungen (1) und (2) erfüllt und für den es eine Komponente $q_{m,i}$ gibt mit $q_{m,i} \geq p_{m,i}$.

Ist $\mathbf{p}_m$ ein Anwendbarkeitsvektor über m für M, so wird dies mit $\mathbf{p}_m \in \mathbf{Ap}(m, M)$ notiert.

□

Eine Anwendbarkeitsmatrix ordnet jeder Region einen Anwendbarkeitsvektor zu.

Definition 3.9. Es sei M eine Konfiguration des P-Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ und $V(\mu) = \{m_1, \ldots, m_t\}$ die Menge der Membranen des Systems. Eine Menge $Q = \{\mathbf{p}_{m_1}, \ldots, \mathbf{p}_{m_t}\}$ heißt *Anwendbarkeitsmatrix* für M, wenn für jedes i mit $1 \leq i \leq t$ gilt, dass $\mathbf{p}_{m_i}$ ein Anwendbarkeitsvektor über m_i für M ist. Die Menge der Anwendbarkeitsmatrizen für eine Konfiguration M wird mit $M_{Ap}(M)$ notiert. □

In einem Berechnungsschritt werden Entwicklungsregeln gemäß einer Anwendbarkeitsmatrix auf das System angewendet.

Definition 3.10. Es sei M eine Konfiguration des P-Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ und $Q = \{\mathbf{p}_{m_1}, \ldots, \mathbf{p}_{m_t}\}$ eine Anwendbarkeitsmatrix für M. Für die Membran $m \in V(\mu)$ sei $\mathbf{p}_m \in Q, \mathbf{p}_m = (p_{m,1}, \ldots, p_{m,s_m})$ ihr zugehöriger Anwendbarkeitsvektor über m für M. Die *Ausführung* von Q über M, notiert mit $Q(M)$, ist definiert als die Konfiguration M' mit

$$M'(m) = M(m) - \sum_{j=1}^{s_m} p_{m,j} \otimes d_{r_j^m} + \sum_{j=1}^{s_m} p_{m,j} \otimes v_{r_j^m}(here)$$
$$+ \sum_{j=1}^{s_{F(m)}} p_{F_m,j} \otimes v_{r_j^{F(m)}}(m) + \sum_{n \in Ch(m)} \sum_{j=1}^{s_n} p_{n,j} \otimes v_{r_j^n}(out)$$

□

Dabei beschreibt die erste Summe die Multimenge der Objekte, die aus der Konfiguration $M(m)$ der Membran m durch Anwendung von Regeln gemäß p_m entfernt werden und die zweite Summe gibt die Multimenge der Objekte an, die durch die Regelanwendung in der Membran selbst erzeugt werden. Die dritte Summe beschreibt die Multimenge, die durch Regelanwendung in der Elternmembran in m erzeugt wird, die vierte Summe ist die Multimenge, die durch Kindmembranen von m in m erzeugt wird.

Mit der Definition des Berechnungsschrittes kann der Berechnungsbaum konstruiert werden.

Definition 3.11. Eine Konfiguration M des Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ *erzeugt in einem Berechnungsschritt* eine Konfiguration M', notiert mit $M \Rightarrow M'$, wenn eine Anwendbarkeitsmatrix Q für M derart existiert, dass nicht jeder Anwendbarkeitsvektor in $Q = \{\mathbf{p}_{m_1}, \ldots, \mathbf{p}_{m_t}\}$ ein Nullvektor ist und $Q(M) = M'$ gilt.

Eine Konfiguration M *erzeugt in* $k \geq 1$ *Berechnungsschritten eine Konfiguration* M', notiert mit $M \Rightarrow^n M'$, wenn Konfigurationen $M_1, \ldots, M_{k+1}$ derart existieren, dass

1. $M = M_1$.
2. $M' = M_{k+1}$.
3. Für jedes i mit $1 \leq i \leq k$ gilt $M_i \Rightarrow M_{i+1}$.

Eine Konfiguration M *erzeugt eine Konfiguration* M', geschrieben $M \Rightarrow^* M'$, wenn M die Konfiguration M' in k Berechnungsschritten erzeugt für ein $k \geq 0$.

□

Definition 3.12. Der Berechnungsbaum **CompTree**(Π) des P-Systems Π ist ein gewurzelter und markierter Baum, der in folgender Weise induktiv definiert ist:

1. Die Wurzel des Baumes ist mit der Startkonfiguration M_0 des P-Systems Π markiert.
2. Für jede Konfiguration M gilt: Kann eine Konfiguration M' aus M in einem Berechnungsschritt erzeugt werden, das heißt $M \Rightarrow M'$, so gibt es genau einen mit M' markierten Kindknoten von M.
3. Jede Kante zwischen zwei Knoten M und M' ist mit der Anwendbarkeitsmatrix Q markiert, die M' aus M erzeugt.

Ein Pfad in **CompTree**(Π) von der mit M_0 markierten Wurzel des Baumes bis zu einem Blatt M' heißt *eine Berechnung von* Π. Eine Berechnung heißt *erfolgreich*, wenn dieser Pfad endlich ist. Eine Blatt des Baumes, das heißt eine Konfiguration M, für die alle Anwendbarkeitsvektoren $M_{Ap}(M)$ Nullvektoren sind, wird als eine *Haltekonfiguration* von Π bezeichnet.

Die Menge der Haltekonfigurationen aller erfolgreichen Berechnungen von Π wird mit $\mathbf{Comp}_{Succ}(\Pi)$ notiert.

Die Menge der von Π erzeugten natürlichen Zahlen $N(\Pi)$ ist definiert als

$$N(\Pi) = \{|M(i_0) \cap \Sigma| M \in \mathbf{Comp}_{Succ}(\Pi)\}.$$

□

Beispiel 3.13. Der Berechnungsbaum für das in Abbildung 3.2 und Beispiel 3.6 vorgestellte P-System ist in Abbildung 3.3 veranschaulicht. Es ist für jede Konfiguration jeweils das P-System als Venn-Diagramm ohne Entwicklungsregeln dargestellt. Pfeile geben an, welche Folgekonfigurationen aus jeder Konfiguration entstehen können und sind mit der Anwendbarkeitsmatrix des jeweiligen Berechnungsschrittes beschriftet. Die Anwendbarkeitsmatrix Q_{stop1} beendet die Berechnung, falls in Membran s kein Objekt vorhanden ist und Q_{stop2} beendet die Berechnung, falls s ein Objekt b enthält. Q_{inc1} setzt die Berechnung durch Anwendung der ersten Regel in m_1 fort und erhöht das Ergebnis der jeweiligen Berechnung um 1. Q_{inc2} tut das gleiche, wendet aber in s die Regel an, um das Objekt b in die Umgebung zu verschieben, falls es vorhanden ist. Die Anwendbarkeitsmatrizen sind wie folgt definiert:

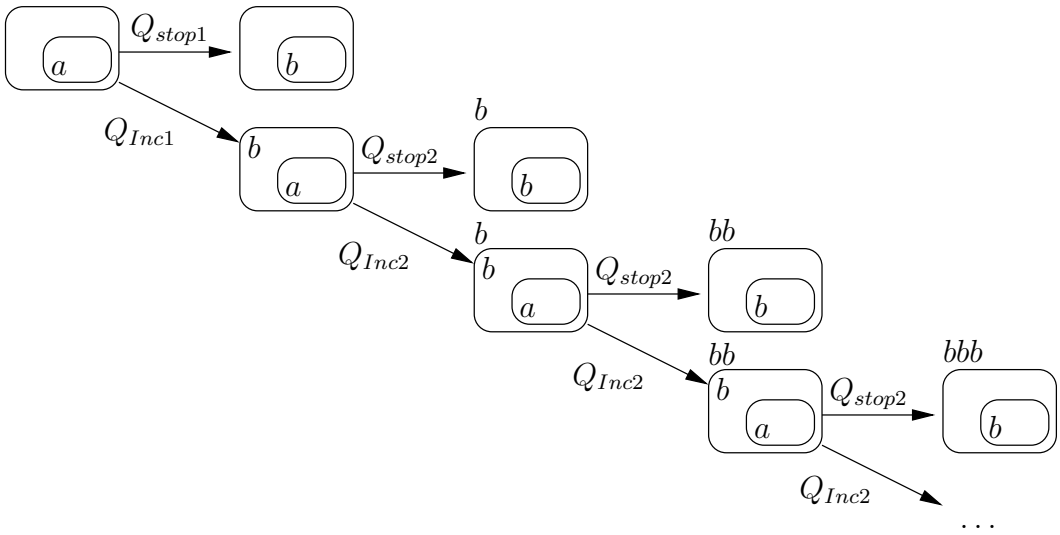

Abbildung 3.3: Der Berechnungsbaum für das P-System aus Beispiel 3.6

$$
\begin{aligned}
Q_{stop1} &= \{\mathbf{p}_{e,stop1}, \mathbf{p}_{s,stop1}, \mathbf{p}_{m_1,stop1}\} \\
\mathbf{p}_{e,stop1} &= () \\
\mathbf{p}_{s,stop1} &= (0) \\
\mathbf{p}_{m_1,stop1} &= (0,1)
\end{aligned}
$$

Der Anwendbarkeitsvektor $\mathbf{p}_{e,stop1}$ ist ein leeres Tupel, da die der Umgebung zugewiesene Regelmenge leer ist. Die einzige Regel r_1^s der *Skin* s wird nicht angewendet und erhält deshalb den Wert 0. Der Vektor $\mathbf{p}_{m_1,stop1}$ gibt an, dass in der Membran m_1 ihre erste Regel $r_1^{m_1}$ nicht und die zweite Regel $r_2^{m_1}$ einmal angewendet wird.

$$
\begin{aligned}
Q_{stop2} &= \{\mathbf{p}_{e,stop2}, \mathbf{p}_{s,stop2}, \mathbf{p}_{m_1,stop2}\} \\
\mathbf{p}_{e,stop2} &= () \\
\mathbf{p}_{s,stop2} &= (1) \\
\mathbf{p}_{m_1,stop2} &= (0,1)
\end{aligned}
$$

$$
\begin{aligned}
Q_{inc1} &= \{\mathbf{p}_{e,inc1}, \mathbf{p}_{s,inc1}, \mathbf{p}_{m_1,inc1}\} \\
\mathbf{p}_{e,inc1} &= () \\
\mathbf{p}_{s,inc1} &= (0) \\
\mathbf{p}_{m_1,inc1} &= (1,0)
\end{aligned}
$$

$$
\begin{aligned}
Q_{inc2} &= \{\mathbf{p}_{e,inc2}, \mathbf{p}_{s,inc2}, \mathbf{p}_{m_1,inc2}\} \\
\mathbf{p}_{e,inc2} &= () \\
\mathbf{p}_{s,inc2} &= (1) \\
\mathbf{p}_{m_1,inc2} &= (1,0)
\end{aligned}
$$

4 Splicing-Membran-Systeme

Die Evolutionsregeln des zuvor definierten P-Systems arbeiten nach dem Prinzip der Termersetzung. Es ist naheliegend, das aus dem DNA-Computing bekannte *Splicing* zur Definition einer neuen Art von Entwicklungsregeln zu verwenden. Die Splicing-Operation wurde unter anderem in [PRS98] zur Konstrukton eines DNA-Rechnermodells und in [Pău02] und [Hof08]) zur Beschreibung eines membranbasierten Systems verwendet. In diesem Kapitel wird in Abschnitt 4.1 die Splicing-Operation und das Splicing-Membran-System beschrieben und in Abschnitt 4.2 das extendierte Splicing-Membran-System formal definiert.

4.1 Die Splicing-Operation und Überblick über Splicing-Membran-Systeme

Das Splicing spaltet gemäß der Definition einer *Splicing-Regel* zwei Wörter auf und setzt die entstandenen Teilwörter auf eine festgelegte Weise zu zwei weiteren Wörtern zusammen. Splicing-Membran-Systeme nutzen auf dem Splicing basierende Entwicklungsregeln. Dazu werden Splicing-Regeln durch Zielinformationen ergänzt, die angeben, in welche Membran jedes der beiden durch das Splicing entstandenen Wörter kommuniziert wird.
Jeder Membran des Systems ist eine endliche Menge solcher Splicing-Regeln und eine Menge von Wörtern über dem Alphabet des Systems zugeordnet. In einem Berechnungsschritt wird in jeder Membran die *Splicing-Operation* ausgeführt. Die Splicing-Operation ist das nebenläufige Ausführen des Splicings in einer Membran gemäß der dieser Membran zugeordneten Splicing-Regeln und unter Verwendung der in der Membran vorhandenen Wörter. Dazu wird angenommen, dass jedes in einer Membran vorhandene Wort in beliebig vielen Kopien vorliegt und an Splicing-Operationen beteiligt sein kann. Wörter werden durch die Ausführung der Splicing-Operation nicht konsumiert und aus dem System entfernt, sondern sind auch nach der Operation vorhanden. Die bei der Ausführung des Splicings in der Membran entstandenen Wörter werden gemäß der Zielinformation der jeweiligen Splicing-Regel in die entsprechende Membran kommuniziert. Analog zu den P-Systemen gibt die Zielinformation *here* an, dass das entstandene Wort in der Membran verbleibt, in der es durch die Splicing-Regel erzeugt wurde. Mit *out* wird ein Wort an die umgebende Membran kommuniziert. Gibt die Zielinformation eine Membran an, so wird das Wort in diese Membran kommuniziert, falls diese eine Kindmembran ist und andernfalls aus dem System entfernt. Wörter, die aus der *Skin* heraus in die Umgebung kommuniziert werden, verlassen das System.

In Berechnungsschritten werden ausgehend von einer Startkonfiguration durch fortgesetztes Ausführen der Splicing-Operation weitere Konfigurationen des Systems erzeugt. Die Ausgabe des Splicing-Membran-Systems ist die von ihm generierte Sprache, die ein Wort genau dann enthält, wenn eine Konfiguration derart existiert, dass das Wort bei dieser Konfiguration in der als Ausgabemembran gekennzeichneten Membran enthalten ist. Im Allgemeinen werden die in der Umgebung auftretenden Wörter als die vom System erzeugte Sprache betrachtet.

Eine Splicing-Regel für Splicing-Membran-Systeme und die Splicing-Operation in einer Membran ist wie folgt definiert:

Definition 4.1. Es sei μ eine Membranstruktur, V ein Alphabet mit $\# \notin V$ und $\$ \notin V$ und es seien u_1, u_2, u_3, u_4 Wörter aus V^* mit der Eigenschaft, dass $|u_1 \cdot u_2| \geq 1$ und $|u_3 \cdot u_4| \geq 1$. Eine Splicing-Regel ist ein Paar $r = (spl, (tar_1, tar2)) \in (V^* \cdot \{\#\} \cdot V^* \cdot \{\$\} \cdot V^* \cdot \{\#\} \cdot V^*) \times (Tar \times Tar)$ mit $Tar = \{here, out\} \cup V(\mu)$. Für eine gegebene Membranstruktur μ und ein Alphabet V wird die Menge aller Splicing-Regeln mit $Spli_{\mu,V}$ notiert.

Die Splicing-Regel $r = (u_1 \# u_2 \$ u_3 \# u_4, (tar_1, tar_2))$ kann auch in folgender Form notiert werden.

$$\begin{array}{c|c|l} u_1 & u_2 & (tar_1) \\ \hline u_3 & u_4 & (tar_2) \end{array}$$

Abbildung 4.1: Notation einer Splicing-Regel

Definition 4.2. Es sei V ein Alphabet, $\varrho \subseteq (V^* \cdot \{\#\} \cdot V^* \cdot \{\$\} \cdot V^* \cdot \{\#\} \cdot V^*) \times (Tar \times Tar)$ eine endliche Menge von Splicing-Regeln und $\mathcal{L} \subseteq V^+$ eine Sprache. Die Splicing-Operation ς ist definiert als

$$\begin{aligned} \varsigma(\mathcal{L}, \varrho) = &\{(x_1 u_1 u_2 y_2, tar_1) | \exists (u_1 \# u_2 \$ u_3 \# u_4, (tar_1, tar_2)) \in \varrho. \\ &\exists x_1 x_2 y_1 y_2 \in V^*. (x_1 u_1 u_2 x_2 \in \mathcal{L} \wedge y_1 u_3 u_4 y_2 \in \mathcal{L})\} \\ \cup &\{(y_1 u_3 u_2 x_2, tar_2) | \exists (u_1 \# u_2 \$ u_3 \# u_4, (tar_1, tar_2)) \in \varrho. \\ &\exists x_1 x_2 y_1 y_2 \in V^*. (x_1 u_1 u_2 x_2 \in \mathcal{L} \wedge y_1 u_3 u_4 y_2 \in \mathcal{L})\}. \end{aligned}$$

□

4.2 Formale Definition eines Splicing-Membran-Systems

Definition 4.3. Ein *extendiertes Splicing-Membran-System* ist ein 6-Tupel $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ mit folgenden Eigenschaften

- O ist das Alphabet des Splicing-Membran-Systems Π.
- $\Sigma \subseteq O$ ist das *Terminalalphabet* von Π.
- μ ist eine Membranstruktur gemäß Definition 3.2.
- Die *Startkonfiguration* M_0 ist eine Konfiguration für μ über O gemäß Definition 4.4. Der Membran $e \in V(\mu)$, die als Wurzel des Baumes μ die Umgebung des Systems repräsentiert, wird die leere Menge zugewiesen. Für eine Membran $m \in V(\mu)$ wird die Menge der in ihr enthaltenen Wörter in der Startkonfiguration als die *Axiomenmenge von* m bezeichnet.
- R ist eine *Splicing-Regel-Zuweisung* über O für μ gemäß Definition 4.5.
- $i_0 \in V(\mu)$ ist die *Ausgabemembran*.

□

Die Konfiguration eines extendierten Splicing-Membran-Systems wird in Analogie zur Konfiguration eines P-Systems definiert. In den Membranen sind hier keine Multimengen von Symbol-Objekten, sondern Wortmengen enthalten.

Definition 4.4. Es sei O ein Alphabet und $\mu = (V(\mu), E(\mu))$ eine Membranstruktur. Eine *Konfiguration* M für μ über O ist eine Abbildung $M : V(\mu) \to \mathcal{P}(O^+)$, die jeder Membran $m \in V(\mu)$ eine Menge von Wörtern aus O^+ zuordnet. Eine Konfiguration für μ über O eines Splicing-Membran-Systems $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ wird verkürzt als eine Konfiguration von Π bezeichnet. □

Entwicklungregeln werden den Membranen wie folgt zugewiesen:

Definition 4.5. Es sei O ein Alphabet und $\mu = (V(\mu), E(\mu))$ eine Membranstruktur. Eine Splicing-Regel-Zuweisung R für μ über O ist eine Abbildung $R : \mu \to \mathcal{P}_{fin}(Spli_{\mu,O})$. Der Membran $e \in V(\mu)$, die als Wurzel des Baumes μ die Umgebung repräsentiert, wird die leere Menge zugewiesen, das heißt der Umgebung sind keine Regeln zugeordnet. □

Mit diesen Definitionen kann nun der Begriff eines Berechnungsschrittes definiert werden. In einem solchen Berechnungsschritt wird von einer Konfiguration M ausgehend für jede Membran $m \in V(\mu)$ mit Splicing-Regel-Menge $R(m)$ die Splicing-Operation $\varsigma(M(m), R(m))$ ausgeführt und so eine neue Konfiguration M' erzeugt. So kann die Menge derjenigen Konfigurationen bestimmt werden, die durch eine Folge von Berechnungsschritten ausgehend von der Startkonfiguration M_0 erzeugt werden kann. Die Ausgabe von Π ist die vom System generierte Sprache, die alle Wörter enthält, die sich bei einer erzeugten Konfiguration in der Ausgabemembran befinden und die Wörter über dem Terminalalphabet Σ sind.

Definition 4.6. Es sei $\Pi = (O, \Sigma, \mu, M_0, R, i_0)$ ein extendiertes Splicing-Membran-System, M eine Konfiguration von Π und es sei $e \in V(\mu)$ die Umgebung und $s \in Ch(e)$ die *Skin* des Systems. Ein *Berechnungsschritt* erzeugt aus der Konfiguration M die Konfiguration M', notiert mit $M \Rightarrow M'$. Dabei ergibt sich M' als

$$\begin{aligned} M'(e) =& \{w|(w, out) \in \varsigma(M(s), R(s))\} \\ M'(m) =& M(m) \cup \\ & \{w|(w, here) \in \varsigma(M(m), R(m)) \cup \\ & \{w|(w, out) \in \varsigma(M(F(m)), R(F(m)))\} \cup \\ & \bigcup_{m' \in Ch(m)} \{w|(w, m) \in \varsigma(M(m'), R(m'))\} \end{aligned}$$

für jede Membran $m \in V(\mu) \backslash \{e\}$.

Eine Konfiguration M erzeugt in k Berechnungsschritten eine Konfiguration M', notiert mit $M \Rightarrow^k M'$, wenn Konfigurationen $(M_1, \ldots, M_{k+1})$ derart existieren, dass

1. $M = M_1$
2. $M' = M_{k+1}$
3. Für jedes i mit $1 \leq i \leq k$ gilt: $M_i \Rightarrow M_{i+1}$.

Eine Konfiguration M erzeugt eine Konfiguration M', notiert mit $M \Rightarrow^* M'$, wenn M die Konfiguration M' in k Berechnungsschritten erzeugt für ein $k \geq 0$.

Die von Π *erzeugte Sprache* $L(\Pi)$ ist definiert als

$$L(\Pi) = \Sigma^+ \cap \{w|w \in M'(i_0) \wedge M_0 \Rightarrow^* M'\}.$$

□

Beispiel 4.7. Ein Splicing-Membran-System kann wie in Abbildung 4.2 in einem Venn-Diagramm notiert werden. In diesem Beispiel gibt es nur eine Membran, die *Skin* s, der zwei Splicing-Regeln zugeordnet sind und in der sich in der Startkonfiguration die Axiome a^3 und a^4 befinden.

Dieses extendierte Splicing-Membran-System wird formal definiert als

$$\Pi_{a^+} = (\{a\}, \{a\}, \mu, M_0, R, e)$$

mit

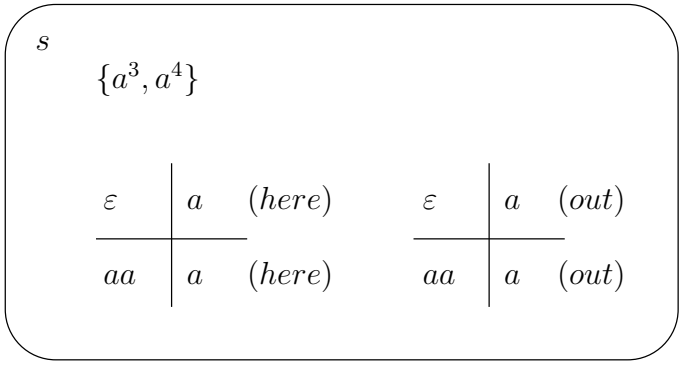

Abbildung 4.2: Beispiel für ein Splicing-Membran-System

- $V(\mu) = \{e, s\}$ und $E(\mu) = \{\{e, s\}\}$.
- $M_0(e) = \emptyset$ und $M_0(s) = \{a^3, a^4\}$.
- $R(s) = \{(e\#a\$aa\#a, (here, here)), (e\#a\$aa\#a, (out, out))\}$.

Die von diesem System erzeugte Sprache ist $L(\Pi_{a^+}) = \{a\}^+$.

5 Spiking Neural P-Systeme

Spiking Neural P-Systeme, auch als SNP-Systeme bezeichnet, modifizieren den membranbasierten Ansatz der P-Systeme, um statt eines Stofftransportes in einer Zellhierarchie die Ausbreitung von elektrischen Impulsen, sogenannten *Spikes*, in Netzen von Neuronen zu modellieren. Der Austausch von Spikes zwischen Neuronen wird durch zwei Arten von Evolutionsregeln, *Spiking-Rules* und *Forgetting-Rules*, gesteuert. Ziel ist es auch hier, ein unkonventionelles Rechnermodell zu schaffen, um schwer lösbare Probleme der Informatik effizienter lösen zu können.
Tissue-like P Systems und *Neural-like P Systems* wurden in [Pău02] als Vorläufer der Spiking Neural P-Systeme vorgestellt. Als Weiterentwicklung dieser Systeme wurden die Spiking Neural P-Systeme in [IPY05] zusammen mit einem Beweis der Turing-Vollständigkeit der SNP-Systeme veröffentlicht.

In diesem Kapitel wird in Abschnitt 5.1 die Idee der Spiking Neural P-Systeme dargestellt und ein Überblick über ihren Aufbau und ihre Funktionsweise gegeben. In Abschnitt 5.2 wird der bekannteste Typ, das generierende SNP-System, formal definiert. Abschnitt 5.3 stellt einige Variationen von SNP-Systemen aus aktuellen Arbeiten zu diesem Themengebiet vor. In Abschnitt 5.4 werden Unterschiede und Gemeinsamkeiten von P-Systemen und SNP-Systemen kurz diskutiert.

5.1 Überblick über Spiking Neural P-Systeme

Der elektrische Impuls als Kern des Modells wird durch das Element a eines einelementigen Objektalphabetes $O = \{a\}$ definiert. Das Element a wird als *Spike* bezeichnet. Es wird vereinfachend angenommen, dass alle elektrischen Impulse, die in einem solchen System auftreten, gleichförmig sind. Das bedeutet, dass Information in dem System nicht durch verschiedene Symbole kodiert wird, sondern allein durch die Anzahl von Spikes.
Spikes können sich in beliebiger Anzahl in *Neuronen* befinden und über Kontaktstellen, den *Synapsen*, zu anderen Neuronen gelangen. Neuronen und Synapsen werden als ein gerichteter Graph $\nu = (V(\nu), E(\nu))$ modelliert, der als die *Neuronenstruktur* des SNP-Systems bezeichnet wird. Neuronen werden durch die Knoten, Synapsen durch die Bögen des Graphen dargestellt. Eine hierarchische Struktur wie bei den membranbasierten P-Systemen ist bei SNP-Systemen nicht vorhanden, stattdessen formen Neuronen und Synapsen ein Netz. Dieser Ansatz wurde bereits in [Pău02] verwendet, um Gewebestrukturen („tissue-like systems“) zu modellieren.

Die Ausbreitung von Spikes im System wird durch *Evolutionsregeln* gesteuert, die ein Neuron dazu veranlassen können, in Abhängigkeit von den bereits in dem Neuron vorhandenen Spikes weitere Spikes auszusenden oder die bereits vorhandenen zu vergessen, das heißt ohne weitere Aktion zu löschen. Dazu werden vom System *Berechnungsschritte* ausgeführt. In einem Berechnungschritt wird ein neuer Zustand des Systems erzeugt, indem in jedem Neuron eine der ihm zugewiesenen Regeln angewendet wird, falls dies gemäß der Definition der Regeln möglich ist.

Es werden die folgenden zwei Arten von Regeln unterschieden: Eine *Spiking-Rule* hat die Form $E\backslash a^c \rightarrow a^d; e$ und veranlasst bei ihrer Anwendung in einem Neuron n das Aussenden von Spikes. Dabei ist E ein regulärer Ausdruck über dem Alphabet O. Ist die Anzahl der in dem Neuron vorhandenen Spikes, interpretiert als ein Wort aus O^*, in der von E erzeugten Sprache enthalten, so kann die Regel angewendet werden. Im Fall der tatsächlichen Anwendung in n werden c Spikes konsumiert, das heißt aus dem Neuron entfernt. Nach Ablauf einer Wartezeit von e Berechnungsschritten werden d Spikes von dem Neuron ausgesendet und befinden sich sofort in allen Neuronen, zu denen von n aus eine Synapse führt. Diese Verzögerungszeit e spiegelt die in der Neurobiologie bekannte Refraktärphase wider, während der ein Neuron nach Auslösen eines elektrischen Impulses auf keine weiteren Reize reagieren kann. Während dieser Phase können in dem Neuron keine Regeln angewendet werden und es kann keine Spikes aufnehmen, die von anderen Neuronen an n ausgesendet werden. Ein Neuron, das sich nicht in der Refraktärphase befindet, wird als *offen* bezeichnet. Befindet es sich in der Refraktärphase, so heißt es *geschlossen* oder *blockiert*.
Forgetting-Rules modellieren das „Vergessen“ von Information durch das Löschen von Spikes in Neuronen. Eine *Forgetting-Rule* ist eine Regel der Form $a^s \rightarrow \lambda$. Enthält das Neuron genau s Spikes, so werden diese gelöscht, das heißt aus dem Neuron entfernt. Sind weniger oder mehr als s Spikes in dem Neuron vorhanden, so kann die *Forgetting-Rule* nicht angewendet werden.

In einem Berechnungsschritt werden gleichzeitig in allen Neuronen Regeln angewendet, falls das in den einzelnen Neuronen möglich ist. Wenn in einem Neuron mindestens eine Regel anwendbar ist, so muss in diesem Berechnungsschritt in dem Neuron eine Regel angewendet werden. Die Regeln dürfen den Neuronen nur derart zugewiesen sein, dass es nicht möglich ist, gleichzeitig eine *Spiking-Rule* und eine *Forgetting-Rule* anzuwenden: In keinem Neuron darf es eine *Forgetting-Rule* $a^s \rightarrow \lambda$ und eine *Spiking-Rule* $E\backslash a^c \rightarrow a^d; e$ geben mit der Eigenschaft, dass $a^s \in L(E)$. Es ist möglich, dass in einem Neuron mit mehreren *Spiking-Rules* die von den regulären Ausdrücken dieser Regeln generierten Sprachen nicht disjunkt und somit mehrere Regeln gleichzeitig anwendbar sind. In diesem Fall wird nichtdeterministisch eine Regel ausgewählt. Regeln werden also in Neuronen sequentiell, systemweit jedoch parallel angewendet.

Eine *Berechnung* eines SNP-Systems ist eine Folge von Konfigurationen und beginnt

mit einer Startkonfiguration, in der alle Neuronen offen sind und eine bestimmte Anzahl Spikes enthalten. In Berechnungsschritten werden schrittweise neue Konfigurationen des Systems erzeugt. Aufgrund der nichtdeterministischen Arbeitsweise können aus einer Konfiguration mehrere Folgekonfigurationen entstehen, was dazu führt, dass für ein SNP-System im Allgemeinen mehrere Berechnungen existieren. Um die *Ausgabe* des Systems zu erhalten, wird untersucht, ob und in welchem Berechnungsschritt in einem als *Ausgabeneuron* ausgezeichneten Neuron Spikes eintreffen. Die ersten beiden im Ausgabeneuron eintreffenden Spikes während einer Berechnung definieren das *Ergebnis* der Berechnung: Es treffe im Berechnungsschritt k_1 das erste und im Schritt k_2 das zweite Spike im Ausgabeneuron ein. Das Ergebnis der Berechnung ist die Differenz $k_2 - k_1$. Die Ausgabe des SNP-System ist die Menge, welche die Ergebnisse aller möglichen Berechnungen des Systems enthält.
Da ein solches SNP-System eine Menge von Zahlen erzeugt, wird es als *generierendes Spiking Neural P-System* bezeichnet.

SNP-Systeme werden üblicherweise wie in dem Beispiel in Abbildung 5.1 dargestellt. Neuronen erscheinen als Rechtecke mit abgerundeten Ecken und werden mit einem Bezeichner versehen, Synapsen werden durch Pfeile wiedergegeben. In einem Neuron sind jeweils seine *Spiking-Rules* und *Forgetting-Rules* und die Anzahl der in der Startkonfiguration in dem Neuron enthaltenen Spikes notiert. Das Spikesymbol ergibt sich implizit aus der Regeldefinition. Die *Spiking-Rules* sind in diesem Beispiel in einer Variante ohne regulären Ausdruck dargestellt. In dieser Form wird als der reguläre Ausdruck der Regel das Wort der konsumierten Spikes angenommen, das heißt die Darstellung $a^c \to a^d; e$ wird als die *Spiking-Rule* $a^c \backslash a^c \to a^d; e$ interpretiert.

Das SNP-System in Abbildung 5.1 besitzt die Neuronen $n_1, \ldots, n_5, out$, wobei *out* das Ausgabeneuron des Systems ist. Das Spikesymbol ist a. Regeln sind in die einzelnen Neuronen eingetragen. In der Startkonfiguration befindet sich im gesamten System nur ein Spike in Neuron n_1. Im ersten Schritt kann n_1 mit seiner einzigen Regel feuern. Dabei wird das vorhandene Spike konsumiert und es wird jeweils ein Spike an die Neuronen n_2, n_3 und n_5 ausgesendet. Im zweiten Schritt kann die Regel $a \to a; 1$ in n_5 angewendet werden. Das Neuron sendet das Spike erst im nächsten Schritt aus, so dass in Schritt drei zum ersten Mal ein Spike das Ausgabeneuron *out* erreicht. Währenddessen befindet sich nach dem ersten Schritt in n_2 und n_3 jeweils ein Spike und beide Neuronen feuern nun solange jeweils mit der Regel $a \to a; 0$, bis in n_2 zum ersten Mal die Regel $a \to a; 1$ angewendet wird. Solange beide Neuronen jeweils gleichzeitig ihre Spikes aussenden, enthalten sie nach dem Schritt wieder je ein Spike. Zudem befinden sich nach jedem Schritt in n_4 genau zwei Spikes, die im darauf folgenden Schritt durch Anwendung der *Forgetting-Rule* $a^2 \to \lambda$ sofort gelöscht werden.
Nachdem in einem beliebigen Schritt k zum ersten Mal die Regel $a \to a; 1$ in n_2 angewendet wurde, befindet sich im darauf folgenden Schritt in n_4 nur ein Spike, so dass dieses Neuron nun in diesem Schritt $k + 1$ mit seiner *Spiking-Rule* ein Spike an das Ausgabeneuron *out* aussendet. In out ist damit ein zweites Spike eingetroffen,

wodurch vom System die Zahl $k+1-3$ ausgegeben wurde. Da n_2 frühestens in Schritt zwei feuern kann, gilt $k \geq 2$. Die von diesem SNP-System generierte Menge ist also die Menge der natürlichen Zahlen $\mathbb{N}$. Das System arbeitet auch nach Eintreffen des zweiten Spikes in *out* weiter, was für die Ausgabe jedoch nicht mehr berücksichtigt wird.

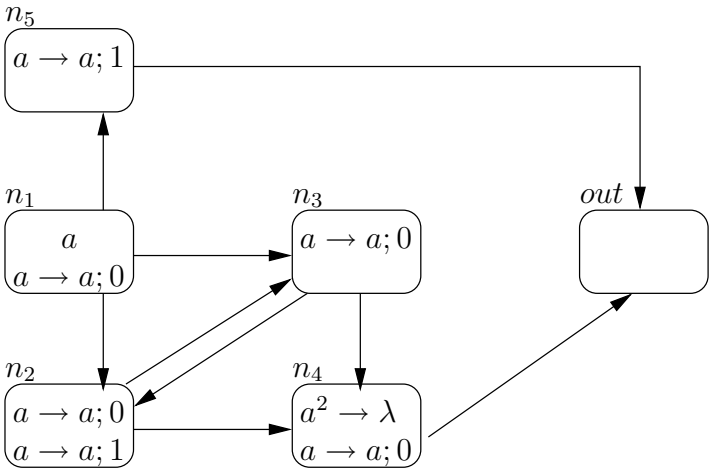

Abbildung 5.1: Beispiel für ein SNP-System

5.2 Formale Definition eines Spiking Neural P-Systems

Es wird zunächst eine Definition des generierenden Spiking Neural P-Systems als 5-Tupel angegeben. Darin verwendete Begriffe werden anschließend definiert.

Definition 5.1. Ein *generierendes Spiking Neural P-System* („SNP-System“) ist ein 5-Tupel $\Pi = (O, \nu, C_0, R, out)$ mit folgenden Eigenschaften:

1. $O = \{a\}$ ist das Objektalphabet. O enthält genau ein Element, welches als *Spike* bezeichnet wird.

2. ν ist eine *Neuronenstruktur*. Eine Neuronenstruktur ist ein gerichteter Graph $\nu = (V(\nu), E(\nu))$. Die Knoten des Graphen werden als *Neuronen*, die Kanten als *Synapsen* bezeichnet.

3. $C_0 = (M_0, T_0, F_0)$ ist eine Konfiguration für ν über O gemäß Definition 5.2. Für jedes Neuron $n \in V(\nu)$ ist $T_0(n) = 0$ und $F_0(n) = \varepsilon$. C_0 wird als die *Startkonfiguration* von Π bezeichnet.

4. $R = (R_{Sp}, R_{Fo})$ ist eine Regelzuweisung für ν über O gemäß Definition 5.5.

5. Das Neuron $out \in V(\nu)$ ist das *Ausgabeneuron.*

□

Eine Konfiguration $C = (M, T, F)$ eines Spiking Neural P-Systems ist bestimmt durch die Anzahl der in jedem Neuron $n \in V(\nu)$ vorhandenen Spikes $M(n)$, die Angabe der Zeitspanne $T(n)$, nach der das Neuron feuert und wieder aktiv sein kann und die Anzahl der Spikes $F(n)$, die nach Ablauf dieser Zeitspanne ausgesendet werden. Ein Neuron n wird als *offen* bezeichnet, wenn $T(n) = 0$ gilt. Im offenen Zustand kann es Spikes aufnehmen und es können in dem Neuron *Spiking-* und *Forgetting-Rules* angewendet werden.

Definition 5.2. Es sei $O = \{a\}$ ein Alphabet und ν eine Neuronenstruktur. Eine *Konfiguration* für ν über O ist ein Tripel $C = (M, T, F)$ von Abbildungen. M ist eine Abbildung $M : V(\nu) \to \{a\}^*$, T ist eine Abbildung $T : V(\nu) \to \mathbb{N}$ und F ist eine Abbildung $F : V(\nu) \to \{a\}^*$, wobei gelte, dass $F(n) = \varepsilon$, falls $T(n) = 0$. □

Die Regelzuweisung $R = (R_{Sp}, R_{Fo})$ ordnet jedem Neuron *Spiking-Rules* und *Forgetting-Rules* zu. Es werden zunächst beide Typen von Regeln definiert und anschließend die Definition der Regelzuweisung angegeben.

Definition 5.3. Es sei $O = \{a\}$ ein Alphabet. Eine *Spiking-Rule* über O ist ein 4-Tupel $r_{Sp} = (E, a^k, a^l, i)$. E ist ein regulärer Ausdruck über O, k und l sind natürliche Zahlen mit der Eigenschaft $k \geq l \geq 1$ und i ist ebenfalls eine natürliche Zahl.

Die Menge aller *Spiking-Rules* über einem Alphabet $O = \{a\}$ wird mit $\mathcal{R}_{Sp}^O$ bezeichnet. □

Der reguläre Ausdruck E beschreibt, bei welcher Konfiguration eines Neurons es möglich ist, diese Regel anzuwenden. Es werden bei Anwendung der *Spiking-Rule* k Spikes aus dem Neuron entfernt und nach i Schritten l Spikes ausgesendet. Für eine *Spiking-Rule* $r = (E, a^k, a^l, i)$ bezeichnet $R(r) = E$ den regulären Ausdruck der Regel, $D(r) = k$ die Anzahl der durch Anwendung der Regel konsumierten Spikes, $S(r) = l$ die Anzahl der ausgesendeten Spikes und $T(r) = i$ die Zeit, nach der diese Spikes ausgesendet werden.

Definition 5.4. Es sei $O = \{a\}$ ein Alphabet. Eine *Forgetting-Rule* über O ist ein Paar $r_{Fo} = (a^k, \lambda)$. Dabei ist k eine von Null verschiedene natürliche Zahl.

Die Menge aller *Forgetting-Rules* über einem Alphabet $O = \{a\}$ wird mit $\mathcal{R}_{Fo}^O$ bezeichnet. □

Eine *Forgetting-Rule* ist beschrieben durch die Anzahl der Spikes, die bei Anwendung dieser Regel in einem Neuron aus dem Neuron entfernt werden. Da eine *Forgetting-Rule* $s = (a^k, \lambda)$ nur dann anwendbar sein soll, wenn das Neuron genau k Spikes enthält, wird kein regulärer Ausdruck angegeben. Für die Regel s bezeichnet $D(s) = k$ die Anzahl der durch diese Regel gelöschten Spikes.

Definition 5.5. Es sei $O = \{a\}$ ein Alphabet und ν eine Neuronenstruktur. Eine *Regelzuweisung* für ν über O ist ein Paar von Abbildungen $R_{\nu,O} = (R_{Sp}, R_{Fo})$ mit folgenden Eigenschaften:

1. R_{Sp} ist eine Abbildung $R_{Sp} : V(\nu) \to \mathcal{P}(\mathcal{R}^O_{Sp})$, die jedem Neuron eine endliche Menge von *Spiking-Rules* über O zuordnet. Die dem Neuron $n \in V(\nu)$ zugeordnete Menge von *Spiking-Rules* wird mit $r^n = \{r^n_1, \ldots, r^n_{u_n}\}$ notiert.
2. R_{Fo} ist eine Abbildung $R_{Fo} : V(\nu) \to \mathcal{P}(\mathcal{R}^O_{Fo})$, die jedem Neuron eine endliche Menge von *Forgetting-Rules* über O zuordnet. Die dem Neuron $n \in V(\nu)$ zugeordnete Menge von *Forgetting-Rules* wird mit $s^n = \{s^n_1, \ldots, s^n_{v_n}\}$ notiert.

Dabei sind keinem Neuron *Spiking-Rules* und *Forgetting-Rules* derart zugeordnet, dass es in diesem Neuron eine *Forgetting-Rule* $s = (a^k, \lambda)$ und eine *Spiking-Rule* $r = (E, a^l, a^m, i)$ gibt mit $a^k \in L(E)$. □

Beispiel 5.6. Das in Abbildung 5.1 vorgestellte SNP-System wird formal wie folgt notiert:

$$\Pi_{\mathbb{N}} = (O, \nu, C_0, R, out)$$

mit

- $O = \{a\}$.
- $\nu = (V(\nu), E(\nu))$ mit
$$V(\nu) = \{n_1, \ldots, n_5, out\}$$
und
$$E(\nu) = \{(n_1, n_2), (n_1, n_3), (n_1, n_5), \ldots, (n_4, out), (n_5, out)\}.$$
- $C_0 = (M_0, T_0, F_0)$ mit $M_0(n_1) = a$ und $M_0(n) = \varepsilon$ für jedes $n \in V(\nu)\backslash\{n_1\}$. T und F haben gemäß Definition 5.1 für jedes Neuron den Wert Null.
- Die Regeln werden wie folgt zugewiesen:
$r^{n_1} = \{(a, a, a, 0)\}, s^{n_1} = \{\}$
$r^{n_2} = \{(a, a, a, 0), (a, a, a, 1)\}, s^{n_2} = \{\}$
$r^{n_3} = \{(a, a, a, 0)\}, s^{n_3} = \{\}$
$r^{n_4} = \{(a, a, a, 0)\}, s^{n_4} = \{(a^2, \lambda)\}$
$r^{n_5} = \{(a, a, a, 1)\}, s^{n_3} = \{\}$.

□

Von der Startkonfiguration ausgehend werden in *Berechnungsschritten* neue Konfigurationen des SNP-Systems erzeugt. Dazu wird bei einer gegebenen Konfiguration für jedes Neuron untersucht, welche seiner *Spiking-Rules* und *Forgetting-Rules* angewendet werden können und diese Information in Anwendbarkeitsvektoren festgehalten, die angeben, welche Regel in diesem Neuron tatsächlich angewendet wird. In einer *Anwendbarkeitsmatrix*, bestehend aus einem je einem Anwendbarkeitsvektor für jedes Neuron, wird die Regelanwendung des gesamten Systems angegeben. Die Ausführung einer Anwendbarkeitsmatrix liefert eine neue Konfiguration des Systems. Aufgrund des Nichtdeterminismus bei der Regelanwendung kann es für eine Konfiguration verschiedene Anwendbarkeitsmatrizen und somit verschiedene Folgekonfigurationen geben, die in je einem Berechnungsschritt erzeugt werden.

Es wird im Folgenden zunächst angegeben, wann eine *Spiking-Rule* oder *Forgetting-Rule* als anwendbar bezeichnet wird und die Begriffe Anwendbarkeitsvektor und Anwendbarkeitsmatrix definiert. Anschließend wird die Ausführung einer solchen Matrix erklärt, der Berechungsbaum konstruiert und die Ausgabe des SNP-Systems definiert.

Damit eine *Spiking-Rule* in einem Neuron anwendbar ist, muss das in dem Neuron vorhandene Wort von Spikes in der durch den regulären Ausdruck der Regel generierten Sprache enthalten sein.

Definition 5.7. Es sei $\Pi = (O, \nu, C_0, R, out)$ ein generierendes Spiking Neural P-System. Eine einem Neuron $n \in V(\nu)$ zugeordnete *Spiking-Rule* $r = (E, a^k, a^l, i)$ ist *anwendbar bei der Konfiguration* $C = (M, T, F)$ von Π, wenn gilt:

1. $T(n) = 0$. Das Neuron n ist offen.
2. $M(n) \in L(E)$. Das Konfigurationswort des Neurons n ist in der von E generierten Sprache enthalten.

□

Damit eine *Forgetting-Rule* in einem Neuron anwendbar ist, muss das betreffende Neuron offen sein und die Anzahl der Spikes in diesem Neuron mit der in der Regel angegebenen Anzahl der zu konsumierenden Spikes übereinstimmen.

Definition 5.8. Es sei $\Pi = (O, \nu, C_0, R, out)$ ein generierendes Spiking Neural P-System. Eine einem Neuron $n \in V(\nu)$ zugeordnete *Forgetting-Rule* $s = (a^m, \lambda)$ ist *anwendbar bei der Konfiguration* $C = (M, T, F)$ von Π, wenn gilt:

1. $T(n) = 0$. Das Neuron n ist offen.

2. $M(n) = a^m$. Das Neuron enthält genau soviele Spikes, wie die *Forgetting-Rule* entfernt.

□

Ein Anwendbarkeitsvektor für ein Neuron n bei einer Konfiguration C gibt eine Möglichkeit an, in diesem Neuron eine Regel anzuwenden. Es dürfen nicht mehrere Regeln gleichzeitig angewendet werden. Falls es anwendbare Regeln gibt, muss eine Regel ausgewählt sein.

Definition 5.9. Es sei $\Pi = (O, \nu, C_0, R, out)$ ein generierendes Spiking Neural P-System und $n \in V(\nu)$ ein Neuron. Ein Anwendbarkeitsvektor über n für die Konfiguration $C = (M, T, F)$ von Π ist ein Paar von Vektoren $\mathbf{p}_n = (\mathbf{sp}_n, \mathbf{fo}_n)$ mit folgenden Eigenschaften:

- $\mathbf{sp}_n := (p_{n,1}, \ldots, p_{n,u_n}) \in \{0,1\}^{u_n}$ ordnet jeder dem Neuron n zugeordneten *Spiking-Rule* aus $R_{Sp}(n) = r^n = (r_1^n, \ldots, r_{u_n}^n)$ entweder die Zahl 0 oder 1 zu. Ist der Regel r_i^n der Index 1 zugeordnet, so ist diese Regel anwendbar bei C.
- $\mathbf{fo}_n := (q_{n,1}, \ldots, q_{n,v_n}) \in \{0,1\}^{v_n}$ ordnet jeder dem Neuron n zugeordneten *Forgetting-Rule* aus $R_{Fo}(n) = s^n = (s_1^n, \ldots, s_{v_n}^n)$ entweder die Zahl 0 oder 1 zu. Ist der Regel s_i^n der Index 1 zugeordnet, so ist diese Regel anwendbar bei C.
- In $\mathbf{sp}_n$ und $\mathbf{fo}_n$ hat höchstens eine Komponente den Wert 1. Wenn es mindestens eine Regel gibt, die angewendet werden kann, so muss eine Komponente von $\mathbf{sp}_n$ oder eine Komponente von $\mathbf{fo}_n$ den Wert 1 erhalten.

□

Eine Anwendbarkeitsmatrix beschreibt, welche Regeln in einem Berechnungsschritt in jedem Neuron angewendet werden:

Definition 5.10. Es sei $\Pi = (O, \nu, C_0, R, out)$ ein generierendes Spiking Neural P-System mit $V(\nu) = \{n_1, \ldots, n_t\}$ und $C = (M, T, F)$ eine Konfiguration des Systems. Eine Menge $Q = \{\mathbf{p}_{n_1}, \ldots, \mathbf{p}_{n_t}\}$ heißt *Anwendbarkeitsmatrix* für C, wenn für jedes $1 \leq i \leq t$ $\mathbf{p}_{n_i}$ ein Anwendbarkeitsvektor über n_i für C ist. Die Menge der Anwendbarkeitsmatrizen für eine Konfiguration C wird mit $M_{Ap}(C)$ notiert. □

Mit den Definitionen zur Anwendbarkeit von Regeln kann wie folgt ein Berechnungsschritt des Systems als Ausführung einer Anwendbarkeitsmatrix definiert werden:

Definition 5.11. Es sei $C = (M, T, F)$ eine Konfiguration des generierenden SNP-Systems $\Pi = (O, \nu, C_0, R, out)$ und $Q = \{\mathbf{p}_{n_1}, \ldots, \mathbf{p}_{n_t}\}$ eine Anwendbarkeitsmatrix für C. Für das Neuron n_i sei $\mathbf{p}_{n_i} = (\mathbf{sp}_{n_i}, \mathbf{fo}_{n_i})$ mit $\mathbf{sp}_{n_i} = (p_{n_i,1}, \ldots, p_{n_i,u_{n_i}})$ und $\mathbf{fo}_{n_i} = (q_{n_i,1}, \ldots, q_{n_i,v_{n_i}})$ der Anwendbarkeitsvektor über n_i für C. Die *Ausführung* von Q über C, notiert mit $Q(C)$, ist definiert als die Konfiguration $C' = (M', T', F')$.

- Die Anzahl der Spikes $|M(n_i)|$ in n_i in der Konfiguration M' ergibt sich als

$$|M'(n_i)| = \begin{cases} |M(n_i)| & \text{falls } T(n_i) \geq 1 \\ |M(n_i)| + A_1 + A_2 - S_{sp} - S_{fo} & \text{falls } T(n_i) = 0 \end{cases}$$

Die Anzahl der Spikes in n_i ändert sich nur, falls das Neuron offen ist. Ist das Neuron nicht offen, so können keine Spikes ausgesendet und aufgenommen werden und es werden keine Regeln angewendet.

Ist das Neuron offen, so erhöht sich die Anzahl der Spikes in n_i um die Spikes, die von anderen Neuronen an n_i gesendet werden. A_1 ist dabei die Zahl der Spikes, die von Neuronen durch *Spiking-Rules* ausgesendet werden, die keine Verzögerungszeit besitzen und A_2 ist die Zahl der Spikes, die von anderen Neuronen nach Ablauf ihrer Verzögerungszeit ausgesendet werden. Es sei $\widetilde{N} = \{n' \in V(\nu) | (n', n) \in E(\nu)\}$ die Menge der Neuronen, von denen aus eine Synapse zu n_i führt. A_1 ergibt sich dann als

$$A_1 = \sum_{n' \in \{\widetilde{n} \in \widetilde{N} | T(\widetilde{n}) = 0\}} \sum_{j \in \{1, \ldots, u_{n'} | T(r_j^{n'}) = 0\}} p_{n',j} S(r_j^{n'})$$

und A_2 ergibt sich als

$$A_2 = \sum_{n' \in \{\widetilde{n} \in \widetilde{N} | T(\widetilde{n}) = 1\}} F(n').$$

Die Anzahl der Spikes in n_i verringert sich um die Spikes, die durch Anwendung von *Spiking-Rules* oder *Forgetting-Rules* aus dem Neuron entfernt werden. Die Menge S_{sp} gibt die von *Spiking-Rules* konsumierten, die Menge S_{fo} die von *Forgetting-Rules* entfernten Spikes an:

$$S_{sp} = \sum_{j=1}^{u_{n_i}} p_{n_i,j} D(r_j^{n_i})$$

$$S_{fo} = \sum_{j=1}^{v_{n_i}} q_{n_i,j} D(s_j^{n_i}).$$

- Die Zeit, nach der n_i feuert, ergibt sich als

$$T'(n_i) = \begin{cases} T(n_i) - 1 & \text{falls } T(n_i) > 1 \\ \sum_{j=1}^{u_{n_i}} p_{n_i,j} T(r_j^{n_i}) & \text{falls } T(n_i) = 0 \end{cases}$$

Im ersten Fall ist das Neuron n_i blockiert und die Zeitangabe $T(n_i)$ wird um eins verringert. Im zweiten Fall ist das Neuron offen und es können möglicherweise *Spiking-Rules* angewendet werden. Da höchstens eine Regel angewendet wird, kann nicht mehr als eine Komponente von $\mathbf{sp}_{n_i}$ den Wert 1 haben, die restlichen Komponenten haben den Wert 0. Sollte eine *Spiking-Rule* r für n_i ausgewählt sein, so ergibt sich $T'(n_i)$ als die Zeit $T(r)$, nach der n_i gemäß der ausgewählten Regel feuert. Ist keine Regel ausgewählt, so ist jedes Produkt in der Summe und die Summe selbst Null und das Neuron bleibt offen.

- Die Anzahl der Spikes, die ausgesendet werden, wenn der Wert $T(n_i) = 0$ erreicht wird, ergibt sich als

$$|F'(n_i)| = \begin{cases} |F(n_i)| & \text{falls } T(n_i) > 1 \\ 0 & \text{falls } T(n_i) = 1 \\ \sum_{j=1}^{u_{n_i}} p_{n_i,j} F(r_j^{n_i}) & \text{falls } T(n_i) = 0 \end{cases}$$

Im ersten Fall $T(n_i) > 1$ bleibt $F(n_i)$ unverändert. Im zweiten Fall $T(n_i) = 1$ sendet n_i nun Spikes aus und $F(n_i)$ wird auf Null zurückgesetzt. Im dritten Fall ist n_i offen. Dieser Fall wird analog zum zweiten Fall für die Berechnung von $T'(n_i)$ gehandhabt.

□

Damit kann der Begriff des Berechnungsschrittes eines SNP-Systems definiert und der Berechnungsbaum des Systems konstruiert werden.

Definition 5.12. Eine Konfiguration C des SNP-Systems Π erzeugt in einem Berechnungsschritt eine Konfiguration C', notiert mit $C \Rightarrow C'$, wenn eine Anwendbarkeitsmatrix Q für C derart existiert, dass $Q(M) = M'$ gilt.

Eine Konfiguration C erzeugt in $k \geq 1$ Berechnungsschritten eine Konfiguration C', notiert mit $C \Rightarrow^k C'$ wenn Konfigurationen $C_1, \ldots, C_{k+1}$ derart existieren, dass

1. $C = C_1$.
2. $C' = C_{k+1}$.
3. Für jedes i mit $1 \leq i \leq k$ gilt: $C_i \Rightarrow C_{i+1}$.

Eine Konfiguration C erzeugt eine Konfiguration C', geschrieben $C \Rightarrow^* C'$, wenn C die Konfiguration C' in k Schritten erzeugt für ein $k \geq 0$. □

Definition 5.13. Der Berechnungsbaum **CompTree**(Π) des SNP-Systems $\Pi = (O, \nu, C_0, R, out)$ ist ein gewurzelter und markierter Baum, der in folgender Weise induktiv definiert ist:

1. Die Wurzel des Baumes ist mit der Startkonfiguration C_0 markiert.
2. Für jede Konfiguration C gilt: Kann eine Konfiguration C' aus C in einem Berechnungsschritt erzeugt werden, so gibt es genau einen mit C' markierten Kindknoten von C.
3. Jede Kante zwischen zwei Knoten C und C' ist mit der Anwendbarkeitsmatrix Q markiert, die C' aus C erzeugt.

Ein Pfad p in $\mathbf{CompTree}(\Pi)$ mit $p = (C_0, C_1, \ldots)$, wobei C_{i+1} Kindknoten von C_i ist für jedes $i \in \{0, \ldots, length(p)\}$, heißt *Berechnung*. Eine Berechnung ist nicht notwendig endlich. Eine Berechnung *hält*, wenn sie endlich ist. Die Menge aller Berechnungen des SNP-Systems Π wird mit $Comp(\Pi)$ notiert.

Für eine Berechnung $p = (C_0, C_1, \ldots)$ gelte $C_0 \Rightarrow^{n_1} C_{n_1+1}$ und $C_0 \Rightarrow^{n_2} C_{n_2+1}$. Es treffe im Berechnungsschritt $C_{n_1-1} \Rightarrow C_{n_1}$ zum ersten Mal und im Berechnungsschritt $C_{n_2-1} \Rightarrow C_{n_2}$ zum zweiten Mal während der gesamten Berechnung ein Spike im Ausgabeneuron ein. Eine Berechnung heißt *erfolgreich*, wenn für sie n_1 und n_2 existiert. Für eine erfolgreiche Berechnung p ist $Res(p) = n_2 - n_1$ das *Ergebnis* der Berechnung. Die Menge der erfolgreichen Berechnungen des SNP-Systems Π wird mit $\mathbf{Comp}_{Succ}(\Pi)$ notiert. □

Die durch Ergebnisse von Berechnungen erzeugte Menge von natürlichen Zahlen ist die Ausgabe $N_2(\Pi)$ des SNP-Systems. Der Index 2 in N_2 gibt an, wie viele eintreffende Spikes im Neuron *out* zur Ermittlung des Ergebnisses der Berechnung verwendet werden. Diese Definition wird in Abschnitt 5.3 verallgemeinert.

Definition 5.14. Für ein SNP-System $\Pi = (O, \mu, M_0, R, out)$ ist die von Π generierte Menge von natürlichen Zahlen $N_2(\Pi)$ definiert als

$$N_2(\Pi) = \{Res(p) | p \in \mathbf{Comp}_{Succ}(\Pi)\}.$$

□

Spiking Neural P-Systeme können nach [PPJR06] gemäß der folgenden Definition klassifiziert werden.

Definition 5.15. $Spik_2P_m(rule_k, cons_p, for_q, dley_r, outd_s)$ mit $m \geq 1$, $k \geq 1$, ist die Familie aller Mengen $N_2(\Pi)$, die von SNP-Systemen Π generiert werden, die höchstens m Neuronen enthalten, in denen jedem Neuron höchstens k Regeln zugeordnet sind, in denen für jede *Spiking-Rule* $E \backslash a^u \rightarrow a^v; t$ gilt, dass $u \leq p$ und für jede *Forgetting-Rule* $a^u \rightarrow \lambda$ gilt, dass $u \leq q$. Regeln haben eine Verzögerungszeit von höchstens r und kein Neuron $n \in \nu$ hat mehr als s ausgehende Synapsen. Wird einer der Parameter durch das Symbol * ersetzt, so ist dieser Parameter unbeschränkt.

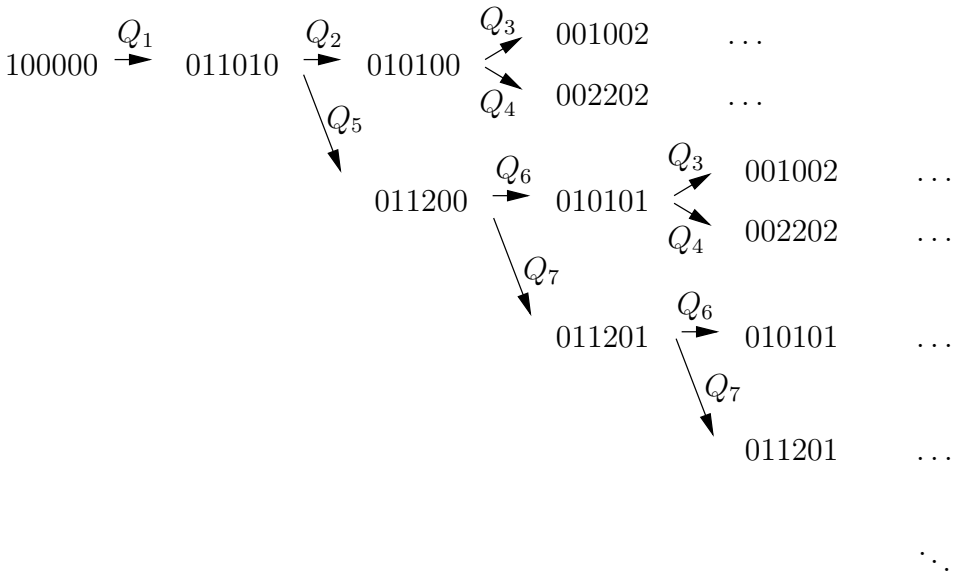

Abbildung 5.2: Der Berechnungsbaum für das SNP-System aus Beispiel 5.6

Beispiel 5.16. Der Berechnungsbaum für das in Abbildung 5.1 und Beispiel 5.6 vorgestellte SNP-System ist in Abbildung 5.2 schematisch dargestellt. Es ist für jede Konfiguration nur die Anzahl der in den Neuronen vorhandenen Spikes in Form einer Zahlenfolge notiert. Die i-te Ziffer gibt die Anzahl der Spikes in Neuron n_i, die letzte Ziffer die Anzahl der Spikes im Ausgabeneuron *out* an. Pfeile geben an, welche Folgekonfigurationen aus jeder Konfiguration entstehen können und sind mit der Anwendbarkeitsmatrix des jeweiligen Berechnungsschrittes beschriftet. Die Anwendbarkeitsmatrix Q_1 startet die Berechnung des Systems durch Ausführung der einzigen *Spiking-Rule* in n_1. Die Anwendbarkeitsmatrizen Q_2 und Q_6 leiten jeweils das Senden eines Spikes an das Ausgabeneuron ein, Q_5 und Q_7 setzen die Berechnung fort. Q_3 und Q_4 sind Matrizen, bei deren Ausführung ein Spike in *out* eintrifft. Als Beispiel für eine solche Anwendbarkeitsmatrix wird Q_7 angegeben.

$$
\begin{aligned}
Q_7 &= \{\mathbf{p}_{n_1}, \mathbf{p}_{n_2}, \mathbf{p}_{n_3}, \mathbf{p}_{n_4}, \mathbf{p}_{n_5}, \mathbf{p}_{out}\} \\
\mathbf{p}_{n_1} &= (\mathbf{sp}_{n_1}, \mathbf{fo}_{n_1}) \text{ mit } \mathbf{sp}_{n_1} = (0), \mathbf{fo}_{n_1} = () \\
\mathbf{p}_{n_2} &= (\mathbf{sp}_{n_2}, \mathbf{fo}_{n_2}) \text{ mit } \mathbf{sp}_{n_2} = (1,0), \mathbf{fo}_{n_2} = () \\
\mathbf{p}_{n_3} &= (\mathbf{sp}_{n_3}, \mathbf{fo}_{n_3}) \text{ mit } \mathbf{sp}_{n_3} = (1), \mathbf{fo}_{n_3} = () \\
\mathbf{p}_{n_4} &= (\mathbf{sp}_{n_4}, \mathbf{fo}_{n_4}) \text{ mit } \mathbf{sp}_{n_4} = (0), \mathbf{fo}_{n_4} = (1) \\
\mathbf{p}_{n_5} &= (\mathbf{s}p_{n_5}, \mathbf{fo}_{n_5}) \text{ mit } \mathbf{sp}_{n_5} = (0), \mathbf{fo}_{n_5} = () \\
\mathbf{p}_{out} &= (\mathbf{sp}_{out}, \mathbf{fo}_{out}) \text{ mit } \mathbf{sp}_{out} = (), \mathbf{fo}_{out} = ()
\end{aligned}
$$

Bei der Ausführung von Q_7 wird in den Neuronen n_2 und n_3 jeweils die *Spiking-Rule* $a \to a; 0$ und in n_4 die *Forgetting-Rule* $a^2 \to \lambda$ ausgeführt.

5.3 Varianten von Spiking Neural P-Systemen

In der in dieser Arbeit eingeführten Definition der SNP-Systeme sind *Spiking-Rules* in der Form $E\backslash a^u \rightarrow a^v; t$ zulässig. Diese Regeldefinition folgt der in [CIPPJ06] verwendeten. Feuert ein Neuron, so kann es gleichzeitig mehrere Spikes über jede seiner Synapsen aussenden. In der Literatur wird im Allgemeinen eine eingeschränkte Form der Regeldefinition verwendet, bei der ein feuerndes Neuron nur ein Spike über jede seiner Synapsen zu anderen Neuronen aussendet. Die meisten Ergebnisse, insbesondere der Beweis der Turing-Vollständigkeit für Modelle der Klasse der SNP-Systeme in Kapitel 6, sind auch bei Verwendung dieser eingeschränkten Form der *Spiking-Rules* gültig.

Es konnte in [IPP$^+$06] und [GAPRPS07] gezeigt werden, dass auch bei Einschränkung der Funktionsweise der SNP-Systeme die Turing-Vollständigkeit erhalten bleibt. So kann auf *Spiking-Rules*, die nicht sofort feuern, sondern eine Verzögerungszeit $t > 0$ besitzen, und auf *Forgetting-Rules* verzichtet und dennoch Turing-Vollständigkeit erreicht werden. Auch kann die maximale Anzahl von Synapsen, die ein Neuron zu anderen Neuronen besitzt auf zwei beschränkt werden und es ist möglich, im System ausschließlich *Spiking-Rules* zu verwenden, deren regulärer Ausdruck E die Form $E = a^+$ oder $E = a^i$ für $i \geq 1$ besitzt. Manche Modifikationen sind auch in Kombinationen noch Turing-vollständig. So kann beispielsweise die ausschließliche Verwendung von *Spiking-Rules* ohne Verzögerungszeit jeweils mit dem Verzicht auf *Forgetting-Rules* oder der Verwendung vereinfachter regulärer Ausdrücke kombiniert werden.

In der ursprünglichen Definition der SNP-Systeme von Paun wurde als Ergebnis einer Berechnung p nur die natürliche Zahl $n = Res(p)$ betrachtet. Dieser Ansatz wurde in einigen Arbeiten verallgemeinert (Vgl. [IPP$^+$06], [PPJR06]).
So kann der *Spike Train*, die Folge der im Ausgabeneuron eintreffenden Spikes, zur Ausgabe mehrerer natürlicher Zahlen verwendet oder selbst als Ergebnis der Berechnung definiert werden. Es sei $p = (C_0, C_1, \ldots)$ eine Berechnung des Spiking Neural P-Systems Π, wobei $C_{i-1} \Rightarrow C_i$ der i-te Berechnungsschritt dieser Berechnung ist. Die Folge $st(p)$ enthält die Indizes der Berechnungsschritte, in denen ein Spike im Ausgabeneuron eintrifft und wird mit $st(p) = (t_1, t_2, \ldots)$ notiert, wobei $1 < t_1 < t_2 < \ldots$ gelte. Diese Folge kann unendlich sein oder endlich, falls die Berechnung hält oder nur eine endliche Zahl von Spikes im Ausgabeneuron eintrifft. Die Menge der Folgen $st(p)$ aller Berechnungen p von Π wird mit $ST(\Pi)$ notiert.

Als vom SNP-System Π generierte Zahlenmengen können nun definiert werden:

- Die bereits vorgestellte Definition der von Π generierten Menge natürlicher Zahlen.

$$N_2(\Pi) := \{t_2 - t_1 | st(p) = (t_1, t_2, \ldots) \wedge st(p) \in ST(\Pi)\}.$$

- Eine Verallgemeinerung der vorherigen Definition für die ersten $k \geq 2$ Spikes.

$$N_k(\Pi) = \{n | n = t_i - t_{i-1}, 2 \leq i \leq k, st(p) = (t_1, t_2, \ldots), \\ st(p) \in ST(\Pi), st(p) \text{ enthält mindestens k Elemente}\}.$$

- Für eine unendliche Folge von Spikes im Ausgabeneuron.

$$N_\omega(\Pi) = \{n | n = t_i - t_{i-1}, 2 \leq i, st(p) = (t_1, t_2, \ldots), \\ st(p) \in ST(\Pi), st(p) \text{ ist eine unendliche Folge}\}.$$

Ist ein SNP-System so konstruiert, dass in einem Berechnungsschritt mehrere Spikes im Ausgabeneuron eintreffen können, so kann durch die Anzahl der eintreffenden Spikes je ein Element eines Alphabetes kodiert und so für endliche Berechnungen Wörter über diesem Alphabet als Ausgabe definiert werden. Dieser Ansatz wurde in [CIPJ+07] und [ZZP08] untersucht.

SNP-Systeme können wie in [IPY05] beschrieben auch als *akzeptierende* Systeme arbeiten. Hierfür wird die Angabe eines Ausgabeneurons ersetzt durch die Definition eines Eingabeneurons $in \in V(\nu)$, über das natürliche Zahlen in das System eingegeben werden. Die Eingabe erfolgt analog zur Ausgabe als Länge der Teilfolge einer Folge von Konfiguration $p = (C_0, \ldots, C_{n_1}, \ldots, C_{n_2})$ des Systems, bei der im ersten Berechnungsschritt der Teilfolge $C_{n_1} \Rightarrow C_{n_1+1}$ das erste Spike und im Berechnungsschritt $C_{n_2} \Rightarrow C_{n_2+1}$ das zweite Spike im Eingabeneuron platziert und dadurch die natürliche Zahl $n = n_2 - n1$ eingegeben wird. Das System *akzeptiert* die eingegebene Zahl, wenn die Berechnung hält. Damit diese Definition sinnvoll ist, wird verlangt, dass sich das System deterministisch verhält, das heißt in jedem Berechnungsschritt in jedem Neuron höchstens eine *Spiking-Rule* anwendbar ist.

5.4 Diskussion von P-Systemen und Spiking Neural P-Systemen

Die Definition des P-Systems unterscheidet sich in vielerlei Hinsicht von der Definition des SNP-Systems. Insbesondere können SNP-Systeme nicht als ein Spezialfall der P-Systeme angesehen werden oder ihre Funktionsweise durch P-Systeme simuliert werden. In diesem Abschnitt werden verschiedene Aspekte diskutiert.

Ein grundlegender Unterschied zwischen beiden Systemen besteht in der Definition des Alphabetes. Während P-Systeme ein Objektalphabet besitzen, deren Elemente Informationen kodieren, wird auf der Seite der SNP-Systeme ein einelementiges Alphabet verwendet. Information wird hier nicht durch die Elemente des Alphabetes selbst, sondern durch die Anzahl der in den Neuronen vorhandenen Spikes kodiert. Die Verwendung eines einelementigen Alphabetes bedeutet eine Einschränkung der

Funktionalität eines Systems.

Dies könnte durch flexibleren Objekttransport kompensiert werden. P-Systeme ermöglichen den Transport der Objekte in der Zell- oder Umgebungshierarchie nur in Kindmembranen hinein oder aus einer Membran hinaus in die sie umgebende Region. SNP-Systeme implementieren den *gewebeähnlichen* („tissue-like“) Ansatz. Objekte, im Fall der SNP-Systeme Spikes, können entlang der Synapsen der Neuronenstruktur bewegt werden. So kann das System derart gestaltet werden, dass Spikes in ein beliebiges Neuron gelangen können. Dieser Sachverhalt äußert sich darin, dass für P-Systeme die Membranhierarchie ein gewurzelter Baum ist, während für SNP-Systeme lediglich ein gerichteter Graph gefordert wird. P-Systeme ermöglichen jedoch einen gezielten Transport der Objekte, während Neuronen in SNP-Systemen ein Spike stets an alle mit ihnen durch Synapsen verbundenen Neuronen aussenden.

Ein wesentlicher Unterschied zwischen den beiden Systemen ist die Definition und Anwendung der Regeln. Stets konsumieren Regeln bei ihrer Anwendung Elemente des Objektalphabetes. Neu erzeugte Objekte oder Spikes werden gemäß der Regeldefinition in der Membranhierarchie oder entlang Synapsen verschoben. Dabei ist die Regeldefinition in P-Systemen insofern komplexer, dass explizit angegeben wird, in welchen Membranen durch Regeln Objekte erzeugt werden. Diese Information ist Bestandteil der Regeldefinition. In SNP-Systemen wird diese Angabe nicht benötigt. Zu welchen Neuronen Spikes ausgesendet werden, ist durch die Neuronenstruktur definiert.
Während Regeln in P-Systemen maximal parallel angewendet werden, kann in einem SNP-System jeweils nur eine Regel in jedem Neuron in einem Berechnungsschritt angewendet werden.

Die im Kapitel zu P-Systemen diskutierten Erweiterungen Membranauflösung und Prioritätsrelationen, wurden in der Forschung zu SNP-Systemen noch nicht betrachtet oder sind wie im Falle des Splicing nicht umsetzbar. Es stellt sich dabei jedoch stets die Frage, ob das Einbringen einer solchen Erweiterung immer noch ein sinnvolles Modell für tatsächliche biologische Vorgänge ist oder die Berechnungsstärke des Systems erhöht. So werden Neuronen nicht durch das Empfangen elektrischer Impulse aufgelöst und es ist anzunehmen, dass der Nichtdeterminismus die Realität besser abbildet als festgelegte Regelprioritäten.

6 Beweis der Turing-Vollständigkeit der Spiking Neural P-Systeme

Gemäß der Churchschen These sind die intuitiv berechenbaren Funktionen genau die Turing-berechenbaren Funktionen. Formale Modelle, die ebenfalls all diese Funktionen berechnen können, werden als *Turing-vollständig* bezeichnet. Für Modelle der Klasse der Registermaschinen wurde in [Min67] gezeigt, dass sie Turing-vollständig sind.

In diesem Kapitel wird ein Verfahren angegeben, welches für eine beliebige Registermaschine ein äquivalentes SNP-System konstruiert. Damit soll gezeigt werden, dass auch die Klasse der SNP-Systeme Turing-vollständig ist.

Eine solche Konstruktion wurde in [IPY05] beschrieben und in verschiedenen Arbeiten (Vgl. [IPP$^+$06], [CIPPJ06], [Pău07]) weiterentwickelt. Im Folgenden wird eine Konstruktion mit einem vereinfachten Ausgabenetz und für den Fall einer deterministisch arbeitenden Registermaschine gezeigt.

Es wird in den Abschnitten 6.1 und 6.2 zunächst die verwendete Registermaschine definiert. Die Konstruktion des SNP-Systems ist in 6.3 und 6.4 beschrieben.

6.1 Überblick über Registermaschinen und formale Definition

Die Registermaschine ist ein formales Modell der theoretischen Informatik und dient dazu, den Begriff zur Berechenbarkeit zu formalisieren.

Eine Registermaschine besitzt eine beliebig große Anzahl von *Registern*, in denen beliebig große natürliche Zahlen gespeichert werden. Auf den Registern können bestimmte Operationen ausgeführt werden, beispielsweise das Inkrementieren eines Registers. Die Registermaschine besitzt ein Programm, das aus einer Folge von fortlaufend mit *Labels* nummerierten Anweisungen besteht. Anweisungen haben die Form:

- $ZERO(r)$: Setze den Wert des Registers r auf Null.
- $INC(r)$: Inkrementiere das Register r, setze das Programm mit der nächsten Anweisung fort.

- $DEC(r), l$: Falls das Register r nicht leer ist, so dekrementiere es und setze das Programm mit der nächsten Anweisung fort. Ist r leer, so springe zur Anweisung mit dem Label l.
- $HALT$: Beende die Ausführung des Programmes.

Die Abarbeitung eines Programmes wird durch einen Befehlszeiger gesteuert, der jeweils die als nächstes auszuführende Anweisung anzeigt. Dazu speichert der Befehlszeiger jeweils das Label dieser Anweisung. Die Registermaschine führt eine Folge von Berechnungsschritten aus. In einem Berechnungsschritt wird die durch den Befehlszeiger bestimmte Anweisung ausgeführt und anschließend der Befehlszeiger auf das Label der im nächsten Schritt abzuarbeitenden Anweisung gesetzt. Erreicht das Programm eine $HALT$-Anweisung, so wird die Abarbeitung beendet und das Ergebnis der Berechnung liegt in dem zuvor als Ausgaberegister definierten Register vor. Die Startbelegung der Register kann als Eingabe frei gewählt werden.

Es wird zunächst die Registermaschine und anschließend der Begriff der Konfiguration definiert.

Definition 6.1. Eine Registermaschine ist ein 5-Tupel $M = (R, L, C_0, I, r_{out})$.

- $R = \{r_1, \ldots, r_g\}$ ist die endliche Menge der Register der Registermaschine.
- $L = \{l_0, l_1, \ldots, l_h\}$ ist die endliche Menge von *Anweisungsmarken* oder *Labels*.
- $C_0 = (\mathcal{R}_0, l_0)$ ist eine Konfiguration gemäß Definition 6.2. Sie wird als die *Startkonfiguration* der Registermaschine bezeichnet. Zu Beginn der Abarbeitung des Programmes zeigt der Befehlszeiger auf die Anweisungsmarke l_0. Die Startbelegung der Register kann als die *Eingabe* der Registermaschine interpretiert werden.
- I ist eine Abbildung, die jedem Label eine Anweisung der Form $ZERO(r)$, $INC(r)$, $DEC(r), l$ oder $HALT$ zuordnet. Das Register $r \in R$ gibt an, auf welchem Register die Operation ausgeführt wird. Das Label $l \in L$ gibt an, zu welchem Label gesprungen wird, falls das Register r der Dekrementierungsanweisung leer ist. I wird als das *Programm* der Registermaschine bezeichnet.
- $r_{out} \in R$ ist das Ausgaberegister von M.

□

Eine Konfiguration ist eine Zustandsbeschreibung der Registermaschine und besteht aus einer Angabe der Belegung der Register und der Position des Befehlszeigers.

Definition 6.2. Es sei R eine Menge von Registern und L eine Menge von Labels. Eine Konfiguration einer Registermaschine ist ein Paar $C = (\mathcal{R}, l)$. $\mathcal{R}$ ist eine Abbildung $\mathcal{R} : R \to \mathbb{N}$ und $l \in L$ ist ein Label.

Bemerkung 6.3. Das Programm I kann vereinfacht wie in Tabelle 6.1, die Konfigurationsfolge für einen Programmablauf wie in Tabelle 6.2 notiert werden.

$$\begin{array}{l} l_1 : Op_1 \\ l_2 : Op_2 \\ \vdots \end{array}$$

Tabelle 6.1: Programmnotation

C	l	r_1	r_2	$\ldots$
C_1	l_1	$\mathcal{R}_1(r_1)$	$\mathcal{R}_1(r_2)$	$\ldots$
C_2	l_2	$\mathcal{R}_2(r_1)$	$\mathcal{R}_2(r_2)$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Tabelle 6.2: Konfigurationsfolge

□

Die Berechnung der Registermaschine, das heißt die Abarbeitung ihres Programmes, wird durch die folgende Definition beschrieben:

Definition 6.4. Die *Berechnung* der Registermaschine $M = (R, L, C_0, I, r_{out})$ ist eine Folge $\mathcal{C} = (C_0, C_1, \ldots,)$ von Konfigurationen mit folgenden Eigenschaften:

- Das erste Element der Folge ist die Startkonfiguration C_0.
- Ist die Folge endlich, so ist ihr letztes Element eine Konfiguration, deren Label auf eine $HALT-$Anweisung zeigt. Es gibt außer der letzten keine weitere Konfiguration, deren Label auf eine $HALT-$Anweisung zeigt.
- Von je zwei aufeinanderfolgenden Konfigurationen ist die Nachfolgende $C_{i+1} = (\mathcal{R}_{i+1}, l_j)$ aus der Vorhergehenden $C_i = (\mathcal{R}_i, l_i)$ in folgender Weise in einem *Berechnungsschritt* entstanden:
  - Falls die Anweisungsmarke l_i auf eine $ZERO(r)$-Anweisung zeigt, so setze $\mathcal{R}_{i+1}(r) = 0$ und $\mathcal{R}_{i+1}(s) = \mathcal{R}_i(s)$ für jedes $s \in R\backslash\{r\}$. Setze $l_j := l_{i+1}$. Das Register r erhält den Wert Null und der Befehlszeiger zeigt auf die nächste Anweisungsmarke.
  - Falls die Anweisungsmarke l_i auf eine $INC(r)$-Anweisung zeigt, so setze $\mathcal{R}_{i+1}(r) = \mathcal{R}_i(r)+1$ und $\mathcal{R}_{i+1}(s) = \mathcal{R}_i(s)$ für jedes $s \in R\backslash\{r\}$. Setze $l_j := l_{i+1}$. Der Inhalt des Registers r wird inkrementiert und der Befehlszeiger zeigt auf die nächste Anweisungsmarke.
  - Falls die Anweisungsmarke l_i auf eine $DEC(r), l_k$-Anweisung zeigt und falls das Register r nicht leer ist, so setze $\mathcal{R}_{i+1}(r) = \mathcal{R}_i(r) - 1$ und

$\mathcal{R}_{i+1}(s) = \mathcal{R}_i(s)$ für jedes $s \in R \backslash \{r\}$. Setze $l_j := l_{i+1}$. Falls das Register r leer ist, so setze $\mathcal{R}_{i+1}(s) = \mathcal{R}_i(s)$ für jedes $s \in R$. Setze $l_j := l_k$. Der Inhalt des Registers r wird dekrementiert, falls es nicht leer ist und der Befehslzeiger zeigt auf die nächste Anweisungsmarke. Falls r leer ist, wird zur Anweisungsmarke l_k gesprungen.

Falls die Berechnung endlich ist, so wird die letzte Konfiguration $C_{HALT} = (\mathcal{R}_{HALT}, l_{HALT})$ als die *Haltekonfiguration von M* bezeichnet. Für eine endliche Berechnung ist das *Ergebnis der Berechnung* der Registermaschine M definiert als $N(M) := \mathcal{R}_{HALT}(r_{out})$. □

Beispiel 6.5. Es sei die Registermaschine $M_{ADD} = (R, L, C_0, I, c)$ gegeben mit

- $R = \{a, b, c, h_1, h_2\}$.
- $L = l_0, l_1, \ldots, l_{14}$.
- $C_0 = (\mathcal{R}_0, l_0)$.
- I gemäß Tabelle 6.3.

$l_0 : DEC(a), l_4$	$l_8 : INC(c)$
$l_1 : INC(c)$	$l_9 : INC(h_1)$
$l_2 : INC(h_1)$	$l_{10} : DEC(h_2), l_7$
$l_3 : DEC(h_2), l_0$	$l_{11} : DEC(h_1), l_{14}$
$l_4 : DEC(h_1), l_7$	$l_{12} : INC(b)$
$l_5 : INC(a)$	$l_{13} : DEC(h_2), l_{11}$
$l_6 : DEC(h_2), l_4$	$l_{14} : HALT$
$l_7 : DEC(b), l_{11}$	

Tabelle 6.3: Programm zur Addition zweier Zahlen

Dieses Programm addiert den Inhalt der Register a und b und legt das Ergebnis im Register c ab.

Tabelle 6.4 zeigt die ersten fünf und die letzte Konfiguration der Berechnung von M für die Registerbelegung $\mathcal{R}_0(a) = 2$, $\mathcal{R}_0(b) = 1$, $\mathcal{R}_0(c) = \mathcal{R}_0(h_1) = \mathcal{R}_0(h_2) = 0$.
Das Ergebnis der Berechnung von M ist $N(M) = 3$.

□

6.2 Modifikation der Registermaschine

Um für eine gegebene Registermaschine ein SNP-System mit gleicher Ausgabe konstruieren zu können, werden nur Registermaschinen mit folgenden Einschränkungen betrachtet:

C	l	a	b	c	h_1	h_2
C_0	l_0	2	1	0	0	0
C_1	l_1	1	1	0	0	0
C_2	l_2	1	1	1	0	0
C_3	l_3	1	1	1	1	0
C_4	l_0	0	1	1	1	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
C_{26}	l_{14}	2	1	3	0	0

Tabelle 6.4: Teil der Konfigurationsfolge für das Additionsprogramm

- Das Programm einer Registermaschine kann mehrere HALT-Anweisungen beinhalten. Im Folgenden werden ohne Einschränkung der Allgemeinheit nur Registermaschinen mit Programmen betrachtet, die genau eine HALT-Anweisung besitzen. Um dies zu erreichen, kann eine HALT-Anweisung stets durch ein Teilprogramm ersetzt werden, das einen Sprung zu einer anderen im Programm vorkommenden HALT-Anweisung ausführt. Hierfür ist im Allgemeinen das Einführen eines Hilfsregisters h notwendig.

- Es werden nur Registermaschinen betrachtet, deren Programme keine ZERO-Anweisung besitzen. Dies bedeutet keine Einschränkung für die Berechnungsstärke der Registermaschine. Eine Anweisung der Form $l_i : ZERO(r)$ kann im Programm stets durch das folgende Teilprogramm ersetzt werden. Auch dafür ist es im Allgemeinen erforderlich, ein zusätzliches Hilfsregister zu definieren.

l_{i-1}	$:\ldots$
l_i	$: DEC(r), l_{i+2}$
l_{i+1}	$: DEC(h), l_i$
l_{i+2}	$: \ldots$

- Es wird im Folgenden vorausgesetzt, dass während der Abarbeitung des Programmes einer Registermaschine nur durch Inkrementierunganweisungen auf das Ausgaberegister zugegriffen wird. Für jede Registermaschine kann dies bei gleichem Ergebnis der Berechnung durch Hinzufügen eines Hilfsregisters und Modifikation des Programmes erreicht werden. Dazu wird die $HALT$-Anweisung durch ein Teilprogramm ersetzt, das zunächst mit einer Schleife den Inhalt des Ausgaberegisters in das Hilfsregister kopiert und dann die $HALT$-Anweisung ausführt. Das Ausgaberegister r_{out} wird durch das Hilfsregister ersetzt.

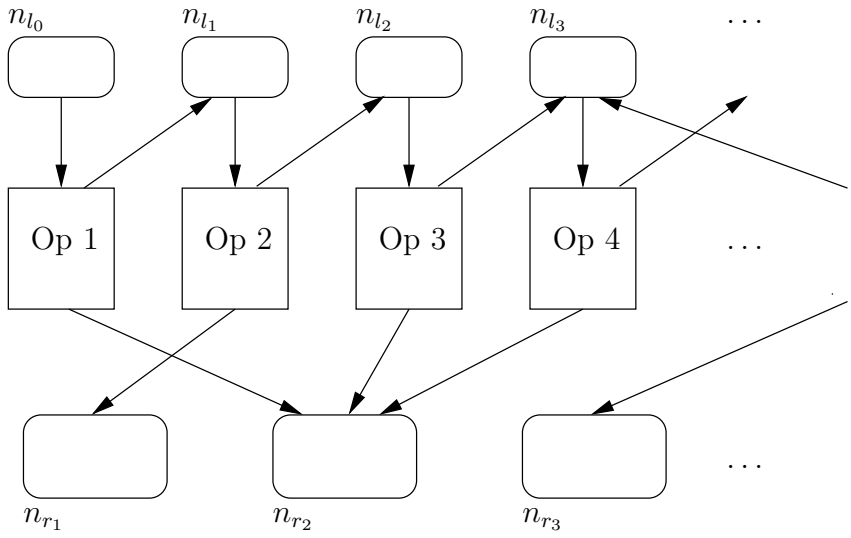

Abbildung 6.1: Schematische Darstellung des zu konstruierenden SNP-Systems

6.3 Vorbetrachtungen zur Konstruktion eines SNP-Systems zu einer gegebenen Registermaschine M

Es wird nun ein Verfahren angegeben, mit dem aus einer Registermaschine M gemäß Definition 6.1 mit den zuvor genannten Modifikationen ein Spiking Neural P-System konstruiert werden kann, das bei auf eine geeignete Weise stattfindender Eingabe der Registerinhalte der Registermaschine in das SNP-System die gleiche Ausgabe liefert wie M.

Jedes Register der Registermaschine wird im SNP-System durch ein Neuron dargestellt. Das Vorhandensein von $2n$ Spikes in einem solchen Registerneuron repräsentiert eine Belegung des entsprechenden Registers mit der natürlichen Zahl n. Es werden entsprechend der Startkonfiguration der Registermaschine Spikes in der Startkonfiguration des SNP-Systems definiert.
Jedem Label wird ein Neuron zugeordnet. Ein solches Labelneuron ist durch Synapsen mit einem Teil-SNP-System verbunden, welches als Modul die Operation dieses Labels umsetzt. Die Module selbst können Spikes an andere Labelneuronen senden und so die Berechnung des Moduls der nächsten Operation starten. In der Startkonfiguration des SNP-Systems sind alle Labelneuronen leer mit Ausnahme des Neurons, welches das Startlabel des Registermaschinen-Programmes repräsentiert. Dieses Neuron enthält zu Beginn zwei Spikes und startet die Berechnung des Systems durch Aktivierung des mit ihm verbundenen Moduls.

Der Aufbau des zu konstruierenden SNP-Systems ist in Abbildung 6.1 schematisch dargestellt. In der Abbildung sind die Labelneuronen mit $n_{l_0}, n_{l_1}, \ldots$, Teil-Systeme

mit Op 1, Op 2, ... und Registerneuronen mit $n_{r_1}, n_{r_2}, \ldots$ beschriftet. Pfeile zeigen an, welche Neuronen und Module miteinander durch Synapsen verbunden sind.
Enthält ein Labelneuron genau zwei Spikes, so feuert es und sendet Spikes an die mit ihm verbundenen Neuronen des Teil-Systems. Dieses Modul wird dadurch aktiviert und realisiert entweder eine Inkrementierungs- oder Dekrementierungsoperation durch Senden von Spikes an das mit dem Teil-System verbundene Registerneuron. Nach Ausführung der Operation werden zwei Spikes an das Labelneuron gesendet, dessen Teil-System als nächstes aktiviert werden soll. In der Abbildung sind Synapsen stets zu dem Labelneuron der nächsten Anweisung dargestellt. Im Falle des Dekrementierungsoperators exisitiert zusätzlich eine Synapse zu einem beliebigen Labelneuron. Wird das Labelneuron der HALT-Anweisung erreicht, so wird das Modul zur Ausgabe des Ergebnisses der Berechnung ausgeführt.

Die Anzahl der Register, die Anzahl der Labels und das Programm der Registermaschine spiegelt sich in der Neuronenstruktur und den Regeln wider, die den Neuronen des SNP-Systems zugeordnet sind. Die Startkonfiguration der Registermaschine wird durch die Anzahl der Spikes in den Registerneuronen abgebildet.

Es wird zunächst gezeigt, wie die Module der Operationen INC, DEC und $HALT$ als Teil-SNP-Systeme realisiert werden können. Anschließend wird in 6.4 daraus das vollständige SNP-System konstruiert.

6.3.1 Der Inkrementierungsoperator $l_i : INC(r)$

Das SNP-Teilsystem für eine Anweisung der Form $l_i : INC(r)$ wird gemäß Abbildung 6.2 konstruiert. Die Neuronen n_{l_i} und $n_{l_{i+1}}$ repräsentieren die Labels l_i und l_{i+1}. Dabei ist l_i das Label, dessen Operation durch dieses Teilsystem simuliert wird und l_{i+1} das Label der darauf folgenden Operation. Das Neuron n_r bildet den Inhalt des Registers r ab, das inkrementiert wird. Die Neuronen n_{l_i,h_1} und n_{l_i,h_2} sind Hilfsneuronen für dieses Teilsystem.

Sobald in einer Konfiguration des SNP-Systems das Labelneuron n_{l_i} genau zwei Spikes enthält und damit das Teilsystem aktiviert, muss in n_{l_i}die *Spiking-Rule* $a^2 \rightarrow a; 0$ angewendet werden, wodurch jeweils ein Spike an die Neuronen n_{l_i,h_1} und n_{l_i,h_2} ausgesendet wird. Diese feuern im nächsten Schritt mit ihrer einzigen Regel und senden jeweils ein Spike an das Registerneuron n_r und das Labelneuron $n_{l_{i+1}}$. Schließlich ist die Anzahl der Spikes im Neuron n_r um genau zwei erhöht und das Labelneuron $n_{l_{i+1}}$ enthält zwei Spikes, um das Teilsystem für die nächste Operation zu aktivieren.

6.3.2 Der Dekrementierungsoperator $l_i : DEC(r), l_j$

Das SNP-Teilsystem für eine Anweisung der Form $l_i : INC(r)$ wird gemäß Abbildung 6.3 konstruiert. Die Neuronen n_{l_i}, $n_{l_{i+1}}$ und n_{l_j} repräsentieren die Labels l_i, l_{i+1} und l_j. Dabei ist l_i das Label, dessen Operation durch dieses Teilsystem simuliert wird, l_{i+1} das Label der darauf folgenden Anweisung, die ausgeführt wird, falls

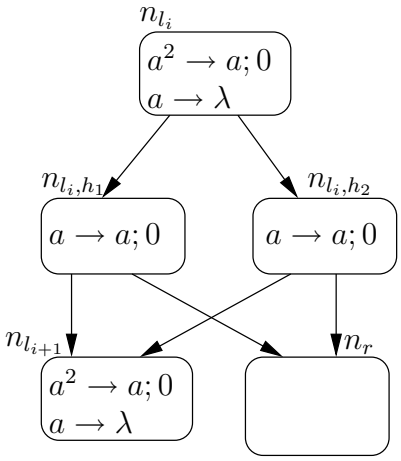

Abbildung 6.2: Teil-System für $l_i : INC(r)$

das Register r nicht leer ist und l_j das Label der Anweisung, zu der gesprungen wird, falls das Register r leer ist. Das Neuron n_r bildet den Inhalt des Registers r ab, das dekrementiert wird. Die Neuronen n_{l_i,h_1} und n_{l_i,h_2} sind Hilfsneuronen für dieses Teilsystem.

Sobald das Labelneuron n_{l_i} genau zwei Spikes enthält und damit das mit ihm verbundene Teilsystem aktiviert wird, muss im ersten Schritt in n_{l_i} die *Spiking-Rule* $a^2 \to a; 0$ angewendet werden, die sofort jeweils ein Spike an die Neuronen n_{l_i,h_1}, n_{l_i,h_2} und n_r aussendet. In den Neuronen n_{l_i,h_1} und n_{l_i,h_2} wird im zweiten Schritt jeweils ihre einzige *Spiking-Rule* angewendet, so dass n_{l_i,h_1} sofort ein Spike an $n_{l_{i+1}}$ und n_{l_i,h_2} im dritten Schritt ein Spike an n_{l_j} aussendet.
Enthält n_r anfänglich kein Spike, repräsentiert also die Zahl Null, so wird in diesem Neuron, nachdem es im ersten Schritt ein Spike von n_{l_i} erhalten hat, im zweiten Schritt die *Spiking-Rule* $a \to a; 1$ angewendet und im dritten Schritt ein Spike jeweils an $n_{l_{i+1}}$ und n_{l_j} ausgesendet. Im vierten Schritt enthält n_{l_j} genau zwei Spikes und aktiviert durch die Anwendung der *Spiking-Rule* das mit ihm verbundene Teilsystem, das die Anweisung mit dem Label l_j umsetzt. Das in $n_{l_{i+1}}$ nach dem zweiten Schritt vorhandene Spike wird im dritten Schritt durch die *Forgetting-Rule* $a \to \lambda$ gelöscht. Das Neuron n_r enthält auch nach dem Abarbeiten des Teilsystems kein Spike.
Repräsentiert n_r anfänglich eine von Null verschiedene Zahl n, enthält also $2n$ Spikes, so wird im zweiten Schritt die *Spiking-Rule* $a(aa)^+/a^3 \to a; 0$ angewendet und sofort ein Spike jeweils an $n_{l_{i+1}}$ und n_{l_j} ausgesendet. Im dritten Schritt entält $n_{l_{i+1}}$ zwei Spikes, feuert sofort und aktiviert so das mit ihm verbundene Teilsystem, das die Anweisung mit dem Label l_j umsetzt. Das in n_{l_j} nach dem dritten Schritt vorhandene Spike wird im vierten Schritt durch die *Forgetting-Rule* $a \to \lambda$ gelöscht. Das Neuron n_r enthält nun $2(n-1)$ Spikes, repräsentiert also die Zahl $n-1$.

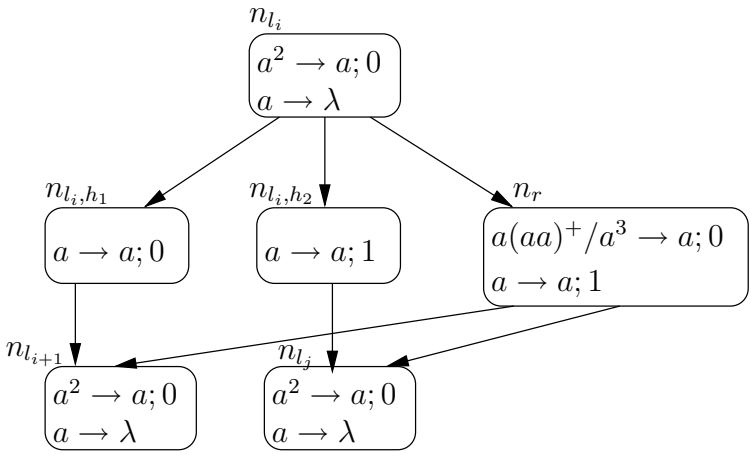

Abbildung 6.3: Teil-System für $l_i : DEC(r), l_j$

6.3.3 Die HALT-Anweisung und Ausgabe des Ergebnisses der Berechnung

Das Teilsystem beendet die Berechnung des SNP-Systems und gibt das Ergebnis aus. Sind in dem Neuron $n_{r_{out}}$, welches das Ausgaberegister repräsentiert, $2n$ Spikes vorhanden, so wird die Zahl n ausgegeben.
Das SNP-Teilsystem für die Anweisung der Form $l_i : HALT$ wird gemäß Abbildung 6.4 konstruiert. Das Neuron n_{l_i} repräsentiert das Label der HALT-Anweisung des Registermaschinen-Programmes. Das Ausgaberegister r_{out} wird durch $n_{r_{out}}$ repräsentiert. Das Neuron *out* ist das Ausgabeneuron des SNP-Systems, $n_{l_i,h_1}, \ldots, n_{l_i,h_4}$ sind Hilfsneuronen für dieses Teilsystem. Da für $n_{r_{out}}$ vorausgesetzt wird, dass es nicht an Dekrementierungsoperationen beteiligt ist, enthält es nur die in der Abbildung aufgeführten Regeln.
Enthält das Neuron n_{l_i} zwei Spikes, so wird das Teilsystem aktiviert. Im ersten Schritt feuert n_{l_i} und sendet sofort jeweils ein Spike an n_{l_i,h_1} und $n_{r_{out}}$. Im zweiten Schritt feuert n_{l_i,h_1} und in Schritt drei sendet n_{l_i,h_3} ein Spike an das Ausgabeneuron *out*.
Währenddessen feuert das Registerneuron $n_{r_{out}}$ ab dem zweiten Schritt, solange es mindestens drei Spikes in ungerader Anzahl enthält, jeweils ein Spike an n_{l_i,h_2} und n_{l_i,h_4}. Es werden beim Feuern jeweils zwei Spikes konsumiert, so das in jedem Schritt die von $n_{r_{out}}$ repräsentierte Zahl dekrementiert wird. Das Neuron n_{l_i,h_2} feuert seinerseits sofort ein Spike an n_{l_i,h_4}, so dass es nach dem Schritt j ($j \geq 2$) stets genau zwei Spikes enthält und diese im nächsten Schritt mit seiner *Forgetting-Rule* löscht. Im Schritt $n + 1$ feuert das Registerneuron $n_{r_{out}}$ nicht, da es keine Spikes mehr enthält. Das Neuron n_{l_i,h_4} enthält in Schritt $n + 2$ folglich nur ein Spike, feuert und sendet im Schritt $n + 3$ ein Spike an das Ausgabeneuron *out*. Damit ist die natürliche Zahl $n + 3 - 3$ das Ergebnis der Berechnung des SNP-Systems Π, das heißt $N_2(\Pi) = \{n\}$.

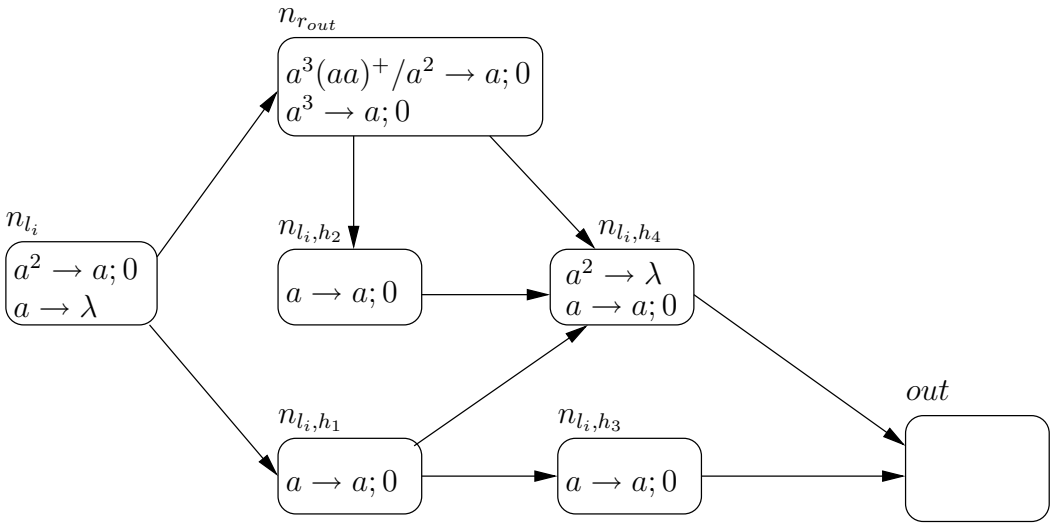

Abbildung 6.4: Teil-System für $l_i : HALT$

6.4 Formale Beschreibung der Konstruktion eines SNP-Systems zu einer gegebenen Registermaschine M

Durch die Konstruktion wird die folgende Aussage bewiesen, aus der sofort die Turing-Vollständigkeit für Modelle der Klasse der SNP-Systeme folgt:

Satz 6.6. Eine Registermaschine M sei mit $M = (R, L, C_0, I, r_{out})$ gegeben, wobei $C_0 = (\mathcal{R}_0, l_0)$. Es kann ein generierendes Spiking Neural P-System Π konstruiert werden, das bei geeigneter Eingabe der Startbelegung der Register $\mathcal{R}_0$ in das System Π das gleiche Ergebnis wie M ausgibt, das heißt $N_2(\Pi) = \{N(M)\}$.

Beweis. $M = (R, L, C_0, I, r_{out})$ sei eine Registermaschine mit $R = \{r_1, \ldots, r_g\}$, $L = \{l_0, \ldots, l_h\}$ und $C_0 = (\mathcal{R}_0, l_0)$. Für das im Folgenden konstruierte SNP-System

$$\Pi = (O, \nu, C_0, R, out)$$

gilt $N_2(\Pi) = \{N(M)\}$. Dabei ist $O = \{a\}$ das Objektalphabet von Π. Das Ausgabeneuron *out* wird als ein Neuron des Teilsystems für die Realisierung der HALT-Anweisung definiert.

Die **Neuronenstruktur** $\nu = (V(\nu), E(\nu))$ von Π wird wie folgt konstruiert:

$$\begin{aligned} V(\nu) &= N_{Reg} \cup N_{Lab} \cup N_{INC} \cup N_{DEC} \cup N_{HALT} \\ E(\nu) &= Syn_{INC} \cup Syn_{DEC} \cup Syn_{HALT} \end{aligned}$$

- Für jedes Register $r_i \in R = \{r_1, \ldots, r_g\}$ wird ein Neuron n_{r_i} der Neuronenmenge $V(\mu)$ hinzugefügt. Die Menge aller Neuronen, die ein Register der Registermaschine repräsentieren, wird mit N_{Reg} bezeichnet.

- Für jedes Label $l_j \in L = \{l_0, l_1, \ldots, l_h\}$ wird ein Neuron n_{l_j} der Neuronenmenge hinzugefügt. Die Menge aller Neuronen, die ein Label der Registermaschine repräsentieren, wird mit N_{Lab} bezeichnet.

- Für jede Anweisung vom Typ $l_i : INC(r)$ wird das in 6.3.1 angegebene Teilsystem der Neuronenstruktur hinzugefügt. Es werden zwei Neuronen n_{l_i,h_1} und n_{l_i,h_2} und die Synapsen (n_{l_i}, n_{l_i,h_1}), (n_{l_i}, n_{l_i,h_2}), (n_{l_i,h_1}, n_r), (n_{l_i,h_2}, n_r), $(n_{l_i,h_1}, n_{l_{i+1}})$, $(n_{l_i,h_2}, n_{l_{i+1}})$ erzeugt. Die Menge der Neuronen, die für das Realisieren von INC-Anweisungen der Neuronenstruktur hinzugefügt werden, wird mit N_{INC}, die Menge der hinzugefügten Synapsen mit Syn_{INC} bezeichnet.

- Für jede Anweisung vom Typ $l_i : DEC(r), l_j$ wird das in 6.3.2 angegebene Teilsystem der Neuronenstruktur hinzugefügt. Es werden zwei Neuronen n_{l_i,h_1} und n_{l_i,h_2} und die Synapsen (n_{l_i}, n_{l_i,h_1}), (n_{l_i}, n_{l_i,h_2}), (n_{l_i}, n_r), $(n_r, n_{l_{i+1}})$, (n_r, n_{l_j}), $(n_{l_i,h_1}, n_{l_{i+1}})$, (n_{l_i,h_2}, n_{l_j}) erzeugt. Die Menge der Neuronen, die für das Realisieren von DEC-Anweisungen der Neuronenstruktur hinzugefügt werden, wird mit N_{DEC}, die Menge der hinzugefügten Synapsen mit Syn_{DEC} bezeichnet.

- Für die HALT-Anweisung $l_i : HALT$ wird das in 6.3.3 angegebene Teilsystem der Neuronenstruktur hinzugefügt. Es hält bei Aktivierung durch das Vorhandensein von zwei Spikes in n_{l_i} die Berechnung an und gibt das Ergebnis über das Ausgabeneuron aus.
  Dazu werden Neuronen $n_{l_i,h_1}, \ldots, n_{l_i,h_4}, out$ und Synapsen (n_{l_i}, n_{l_i,h_1}), $(n_{l_i}, n_{r_{out}})$, $(n_{l_i,h_1}, n_{l_i,h_3})$, $(n_{l_i,h_1}, n_{l_i,h_4})$, $(n_{l_i,h_2}, n_{l_i,h_4})$, $(n_{r_{out}}, n_{l_i,h_4})$, $(n_{r_{out}}, n_{l_i,h_2})$, (n_{l_i,h_4}, out) und (n_{l_i,h_3}, out) definiert, wobei out das Ausgabeneuron des SNP-Systems ist. Die Menge der Neuronen, die für das Realisieren der HALT-Anweisung der Neuronenstruktur hinzugefügt werden, wird mit N_{HALT}, die Menge der hinzugefügten Synapsen mit Syn_{HALT} bezeichnet.

In der **Startkonfiguration** $C_0 = (M_0, T_0, F_0)$ des SNP-Systems Π wird jedem Registerneuron $n_r \in N_{Reg}$ die entsprechende Anzahl Spikes zugewiesen. Enthält ein Register r in der Startkonfiguration der Registermaschine die natürliche Zahl k_r, so wird dies im zugehörigen Neuron n_r durch $2k_r$ Spikes repräsentiert.

$$\forall r \in R : M(n_r) := a^{2k_r}.$$

Dem Labelneuron n_{l_0} werden genau zwei Spikes zugewiesen, damit es das Teilsystem aktivieren kann, welches die erste Operation des Registermaschinenprogrammes realisiert.
T_0 und F_0 werden gemäß der Definition des SNP-Systems für jedes Neuron $n \in V(\nu)$ mit $T_0(n) = 0$ und $F_0(n) = \varepsilon$ definiert.

Die **Regeln** werden den Neuronen des SNP-Systems Π wie folgt zugewiesen:

- Labelneuronen $n_l \in N_{Lab}$ feuern, wenn sie genau zwei Spikes enthalten und aktivieren dadurch das Teilsystem, mit dem sie verbunden sind. Enthält ein Neuron in einer Konfiguration genau ein Spike, so wird dieses durch eine *Forgetting-Rule* gelöscht.

$$r^{n_l} = \{(a^2, a^2, a, 0)\}$$

$$s^{n_l} = \{(a, \lambda)\}.$$

- Registerneuronen $n_r \in N_{Reg}$, die für die Realisierung einer DEC-Anweisung verwendet werden, enthalten folgende Regeln:

$$r^{n_r} = \{(a(aa)^+, a^3, a, 0), (a, a, a, 1)\}$$

- Das Registerneuron $n_{r_{out}} \in N_{Reg}$, dessen Inhalt ausgegeben wird, enthält folgende Regeln:

$$r^{n_{r_{out}}} = \{(a^3(aa)^+, a^2, a, 0), (a^3, a^3, a, 0)\}$$

- Den Neuronen, die in Teilsystemen als Hilfsneuronen verwendet werden, werden Regeln gemäß der Definition der jeweiligen Teilnsysteme aus 6.3.1, 6.3.2 und 6.3.3 zugewiesen.

Damit ist das SNP-System Π vollständig beschrieben. □

Folgerung 6.7. Die Klasse der SNP-Systeme ist Turing-vollständig.

Beweis. Die Aussage folgt sofort aus der Turing-Vollständigkeit der Registermaschine und Satz 6.6. □

7 SAT als Anwendungsbeispiel für Spiking Neural P-Systeme

Ziel der Entwicklung der Spiking Neural P-Systeme war nicht, ein Modell für biologische Anwendungen zu finden, das heißt aus dem Modell Aussagen zu Sachverhalten der Biologie oder Medizin zu erhalten. Es war stets der Wunsch, ein Rechnermodell zu schaffen, mit dem Probleme der Informatik effizient lösbar sind.

In [CII06] wurde eine Konstruktion vorgestellt, die für eine belibige Instanz des SAT-Problems ein SNP-System generiert, welches das Problem in konstanter Zeit mit vier Berechnungsschritten löst. Das SAT-Problem ist NP-vollständig. Seine Bedeutung ergibt sich daraus, dass für viele weitere Probleme konstruktive Verfahren bekannt sind, die das Problem auf SAT zurückführen.

Das SAT-Problem ist ein Entscheidungsproblem. Für eine beliebige aussagenlogische Formel soll entschieden werden, ob diese erfüllbar ist. Es seien n aussagenlogische Variablen $x_1, \ldots, x_n$, $n \geq 1$ und eine aussagenlogische Formel der Form $\gamma = C_1 \wedge \ldots \wedge C_m$ gegeben, wobei jede Klausel C_i, $1 \leq i \leq m$ von der Form $C_i = y_{1,1} \vee \ldots \vee y_{i,k_i}$, $k_i \geq 1$ ist. Dabei gilt stets, dass $y_{i,j} \in \{x_k, \neg x_k | 1 \leq k \leq n\}$. Jede aussagenlogische Formel lässt sich in dieser Form darstellen. Die Menge aller Instanzen von SAT mit n Variablen und m Klauseln wird mit $SAT(\langle n, m \rangle)$ notiert.

Für jedes Paar $\langle n, m \rangle$ wird ein SNP-System konstruiert, das alle Instanzen mit diesen Parametern durch Anpassung der Startkonfiguration, das heißt durch Belegen der Neuronen mit einer bestimmten Anzahl Spikes in Abhängigkeit von der konkreten Instanz, lösen kann. Das für eine Instanz von $SAT(\langle n, m \rangle)$ konstruierte SNP-System besitzt $2^n(m+1) + 2nm + 1$ Neuronen und $2^n(3m+1)$ Synapsen. Es gibt also einen exponentiellen Zusammenhang zwischen der Anzahl der aussagenlogischen Variablen n und Klauseln m und der Anzahl der Neuronen und Synapsen des SNP-Systems, so dass große Instanzen von SAT zwar schnell, jedoch nur mit hohem Platzaufwand gelöst werden können. Dies ist als Verschiebung von Zeitaufwand zu Speicheraufwand zu interpretieren und ist im biologischen Rechnen häufig anzutreffen. Auch muss berücksichtigt werden, dass zusätzlich zum Rechenaufwand des SNP-Systems auch Zeit- und Speicheraufwand benötigt wird, um das SNP-System zu konstruieren.

8 Ausblick

In dieser Arbeit wurde auf der Definition der P-Systeme in [Pău02] aufbauend eine formale Definition der Spiking Neural P-Systeme erarbeitet. Im Beweis der Turing-Vollständigkeit dieses Modells in [IPY05] wird erläutert, wie für jede Registermaschine ein äquivalentes SNP-System konstruiert werden kann. Auch diese Konstruktion wurde in dieser Arbeit formalisiert und die Konstruktion vollständig beschrieben.

Zukünftige Arbeiten zu SNP-Systemen könnten darauf aufbauend die Konstruktion eines universellen SNP-Systems, das in [CIPPJ06] vorgestellt wurde, beschreiben. Eine solche Darstellung enthält eine vollständige Definition der Struktur des Systems und seiner Abarbeitung und ermöglicht einerseits eine einfache Umsetzung von SNP-Systemen in Software, beispielsweise zur Simulation des Verhaltens dieser Systeme. Es lassen sich damit aber auch Verfikationsverfahren zur Untersuchung von Eigenschaften der SNP-Systeme anwenden, wie dies in [Hof08] bereits für Splicing-Membran-Systeme getan wurde. Um das auch für die zahlreichen Varianten von SNP-Systemen zu ermöglichen, ist deren Integration in die formale Definition des Modells erforderlich.

Literaturverzeichnis

[AB03] Alexander Asteroth and Christel Baier. *Theoretische Informatik*. Pearson Studium, München, 2003.

[CII06] Haiming Chen, Mihai Ionescu, and Tseren-Onolt Ishdorj. On the efficiency of spiking neural P systems. In Miguel Angel Gutiérrez-Naranjo, Gheorghe Paun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume I*, pages 195–206. Fénix Editora, 2006.

[CIPJ+07] H. Chen, M. Ionescu, M.J. Pérez-Jiménez, R. Freund, and Gh. Păun. On string languages generated by spiking neural p systems. *Fundamenta Informaticae*, 75:141–162, 2007.

[CIPPJ06] Haiming Chen, Tseren-Onolt Ishdorj, Gheorghe Păun, and Mario-Jesús Pérez-Jiménez. Spiking neural P systems with extended rules. In Miguel Angel Gutiérrez-Naranjo, Gheorghe Paun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume I*, pages 241–266. Fénix Editora, 2006.

[GAPRPS07] M. Garcia-Arnau, D. Perez, A. Rodriguez-Paton, and P. Sosik. Spiking neural p systems: Stronger normal forms. In M.A. Gutiérrez-Naranjo, Gh. Păun, A.Romero-Jiménez, and A. Núñez, editors, *Proceedings of the Fifth Brainstorming Week on Membrane Computing*, pages 157–178, Sevilla (Spain), January 29th - February 2 2007.

[Hof08] Christian Hofmann. *Formale Analyse- und Verifikationsparadigmen für ausgewählte verteilte Splicing-Systeme*. Dissertation, Technische Universität Dresden, Dresden, Deutschland, 2008.

[Hop01] Ullman Hopcroft, Motwani. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Boston, 2001.

[Iba05] Oscar H. Ibarra. Some computational issues in membrane computing. *Mathematical Foundations of Computer Science 2005*, 3618:39–51, 2005.

[Ihr94] Thomas Ihringer. *Diskrete Mathematik : Eine Einführung in Theorie und Anwendungen*. Teubner, Stuttgart, 1994.

[IPP+06] Oscar-H. Ibarra, Andrei Păun, Gheorghe Păun, Alfonso Rodríguez-Patón, Petr Sosik, and Sara Woodworth. Normal forms for spiking neural P systems. In Miguel Angel Gutiérrez-Naranjo, Gheorghe Păun, Agustín Riscos-Núñez, and Francisco José Romero-Campero, editors, *Fourth Brainstorming Week on Membrane Computing, Sevilla, January 30 - February 3, 2006. Volume II*, pages 105–136. Fénix Editora, 2006.

[IPY05] M. Ionescu, Gh. Păun, and T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae*, 71(2-3):279–308, February 2005.

[Min67] Marvin Minksy. *Computation: Finite and Infinite Machines*. Prentice-Hall, London, 1967.

[Pău98] Gheorghe Păun. Computing with membranes. Technical Report 208, Turku Center for Computer Science-TUCS, 1998. (www.tucs.fi).

[Pău02] Gheorghe Păun. *Membrane Computing. An Introduction*. Springer-Verlag, Berlin, 2002.

[Pău07] Gheorghe Păun. Spiking neural P systems. A tutorial. *Bulletin of the EATCS*, February 2007.

[PD99] Gh Păun and J. Dassow. On the power of membrane computing. *Journal of Universal Computer Science*, 5(2):33–49, 1999.

[PJSC02] M.J. Pérez-Jiménez and F. Sancho-Capparini. A formalization of basic p systems. *Fundamenta Informaticae*, 49(1-3):261–272, 2002.

[PPJR06] Gh. Păun, M.J. Pérez-Jiménez, and G. Rozenberg. Spike trains in spiking neural P systems. *Intern. J. Found. Computer Sci.*, 17(4):975–1002, 2006.

[PRS98] Păun, Rozenberg, and Salomaa. *DNA computing : New computing paradigms*. Springer, Berlin, 1998.

[Sch08] Uwe Schöning. *Theoretische Informatik - kurz gefasst*. Spektrum Akademischer Verlag, Heidelberg, 2008.

[YIP08] T. Yokomori, M. Ionescu, and Gh. Păun. Special issue on spiking neural p systems: Preface. *Natural Computing*, 7(4):451–452, 2008.

[ZZP08] X. Zhang, X. Zeng, and L. Pan. On string languages generated by spiking neural p systems with exhaustive use of rules. *Nat. Comput.*, 7:535–549, 2008.