Molekulare Algorithmen

Chemische Digitalcomputermodelle für endliche Automaten und Registermaschinen sowie praktische Anwendungen

Thomas Hinze

Vorlesung und Projektübung an der Friedrich-Schiller-Universität Jena
Fakultät Mathematik und Informatik sowie
Fakultät für Biowissenschaften, Lehrstuhl Bioinformatik

thomas.hinze@uni-jena.de

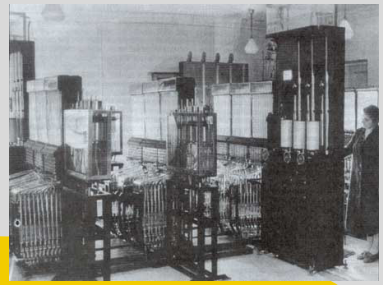
Analogcomputer sind gebaut und eingesetzt worden

150 BC

Antikythera–Mechanismus

AKAT–1

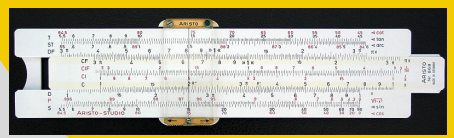
Newmark



Sowjetischer Wassercomputer

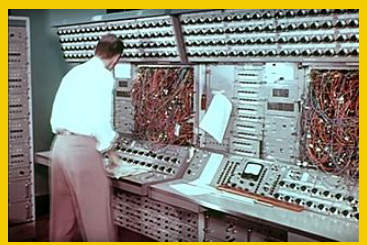
bis ca. 1980

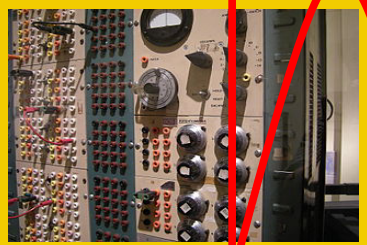
1960



Rechenschieber

1650





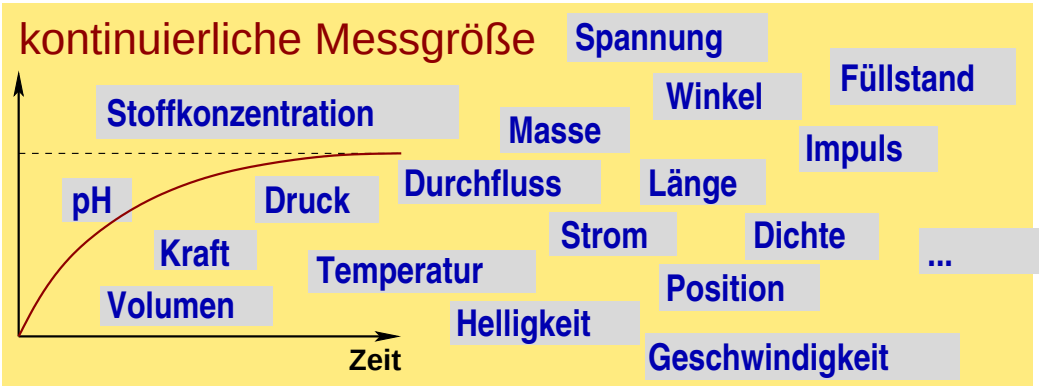
X–15

1970

Bilder: www.wikimedia.org

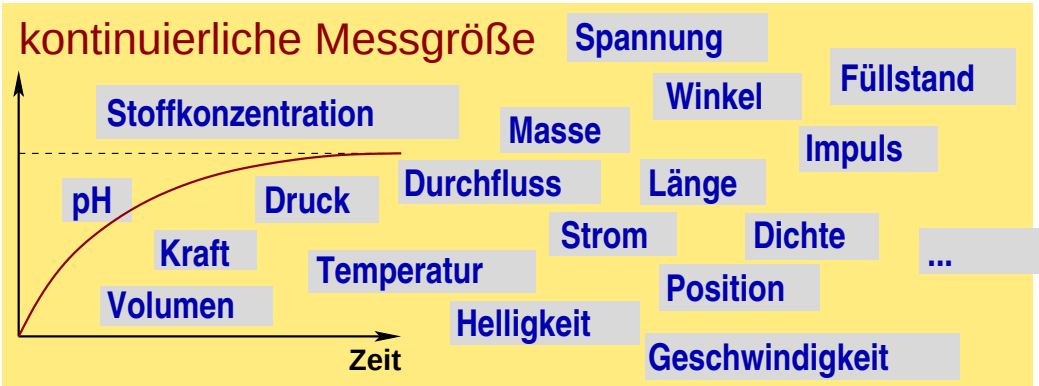
Analogcomputer mit mannigfaltigen *Funktionsprinzipien* wurden für *arithmetische Berechnungen* (z.B. Flugbahnen) und zum *numerischen Lösen von Differentialgleichungen* verwendet

Vorteile von Analogcomputern



- kontinuierliche *Signalwerte direkt als Rechengrößen* abbildbar
- enger Zusammenhang („*Analogie*“) zwischen zu *simulierendem* Originalsystem und Computermodell (*similar* $\hat{=}$ *ähnlich*)
- vergleichsweise *einfacher Aufbau*, Funktionsprinzip folgt unmittelbar den Berechnungen
- häufig *Echtzeitfähigkeit* und hohe Ausführungsparallelität
- kein Takt notwendig

Vorteile von Analogcomputern



- kontinuierliche *Signalwerte direkt als Rechengrößen* abbildbar
- enger Zusammenhang („*Analogie*“) zwischen zu *simulierendem* Originalsystem und Computermodell (*similar* $\hat{=}$ *ähnlich*)
- vergleichsweise *einfacher Aufbau*, Funktionsprinzip folgt unmittelbar den Berechnungen
- häufig *Echtzeitfähigkeit* und hohe Ausführungsparallelität
- kein Takt notwendig

Aber: *hohe Störanfälligkeit* der Signale und *geringe Genauigkeit* (selten höher als vier zuverlässige Dezimalstellen), *Berechnungsvorschrift* durch Hardwarestruktur repräsentiert

Digitaltechnik dominiert inzwischen gegenüber analog

Quelle: Branchenverband Bitkom

Analogtechnik auch außerhalb von Computern auf breiter Front *digital* ersetzt: Tonträger, Telefon, Fernsehen, . . .

Vorzüge digitaler Technik

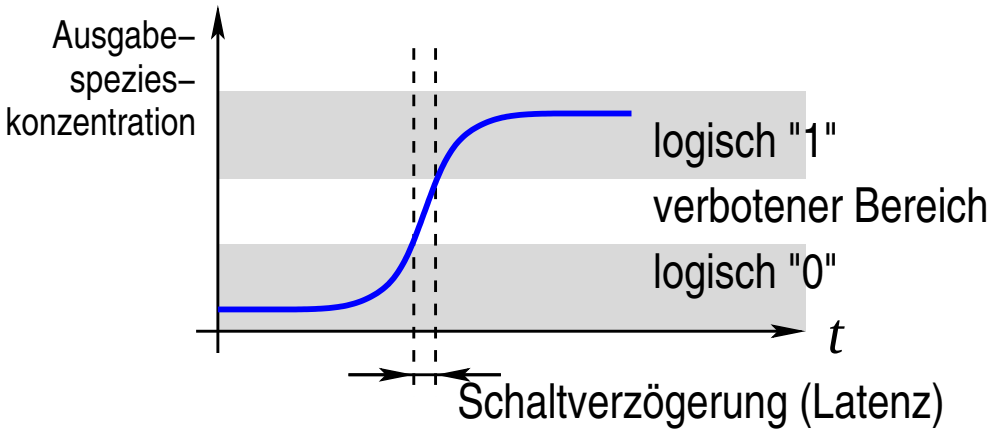
- hohe *Robustheit* gegenüber Signalstörungen und -schwankungen
- *Genauigkeit* der Zahlendarstellung kann weitgehend frei festgelegt werden auf definierbarem diskreten Wertebereich
- Wertebereichsüberschreitungen gefährden nicht die *Funktionsfähigkeit* des Gesamtsystems, sondern lassen sich abfangen
- klar voneinander abgrenzbare zeitdiskrete *Operationsschritte* in den ausgeführten Rechenvorgängen
- *Programmierbarkeit* unabhängig von Hardwarestruktur möglich und mithin algorithmisch flexibel anpassbar

$\Longrightarrow$ *Analogverarbeitung* weiterhin genutzt in vielen *Regelkreisen*, weil dort Störeinflüsse kontinuierlich minimiert werden

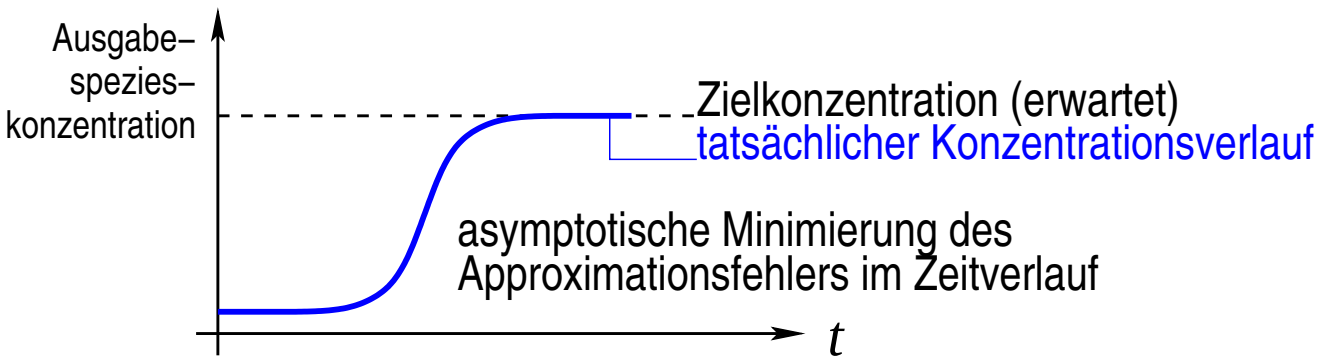
Digitale Verarbeitung erfordert Diskretisierung

zusätzliche Interpretationsebene zwischen Signalwert und Rechenwert

chemisches Digitalcomputermodell



chemisches Analogcomputermodell

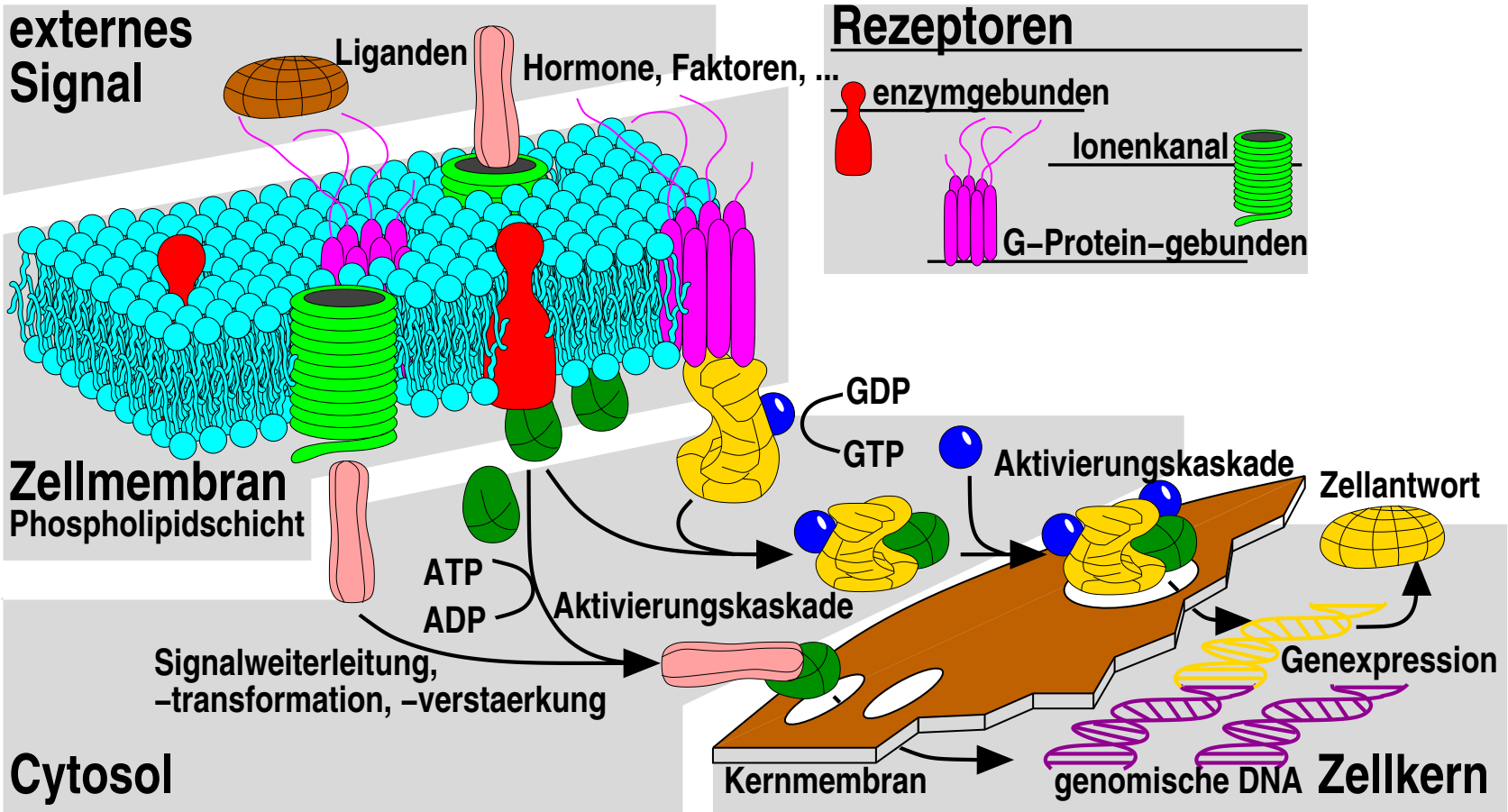


redundante *Zusammenfassung* von Signalwerten zu logischem Wert

Chemische Digitalcomputer

Grundlagen der Zellsignalverarbeitung

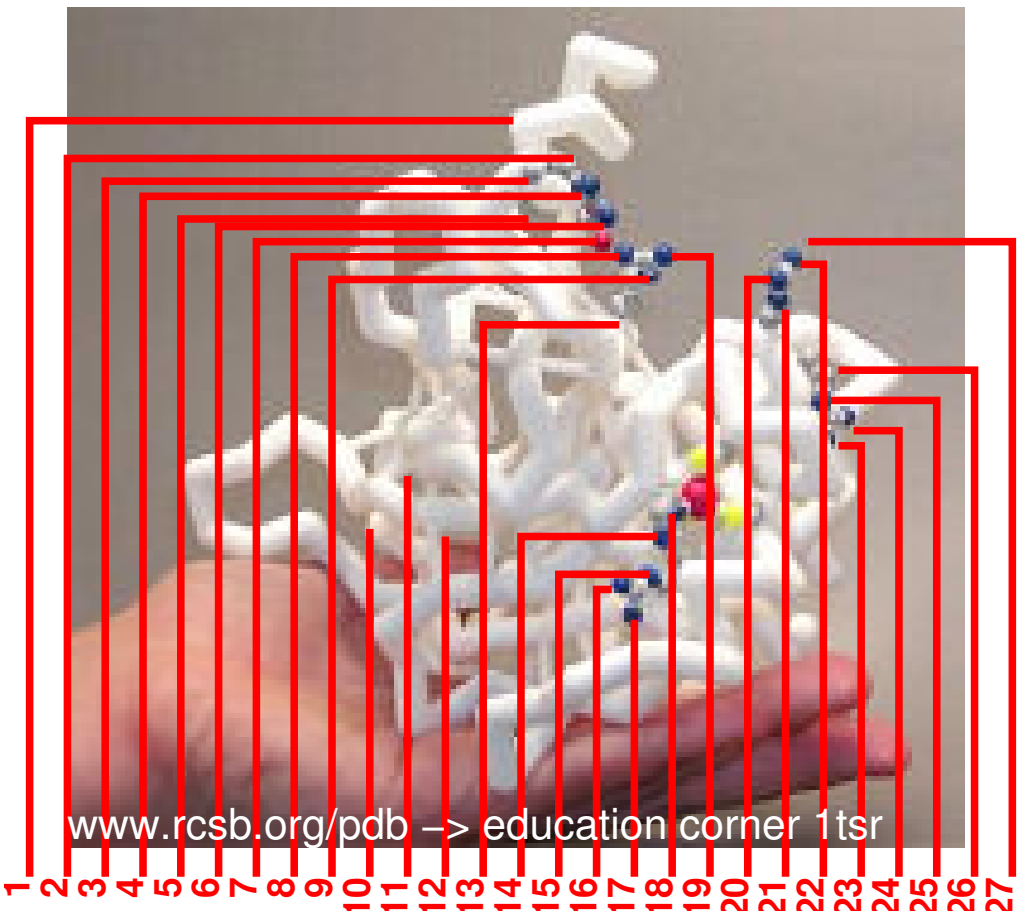
Signalverarbeitung in eukaryotischen Zellen



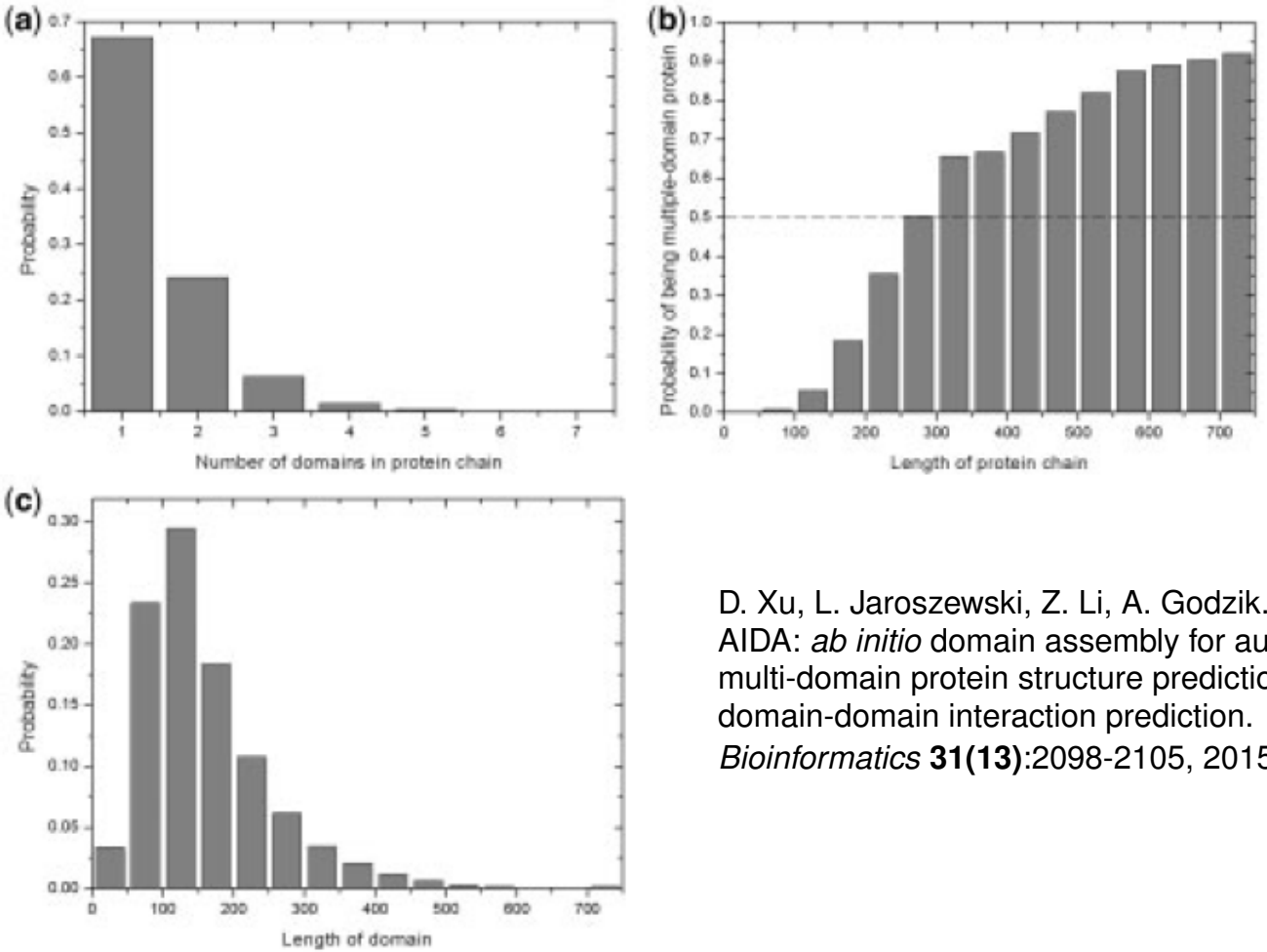
Zellsignalverarbeitung erfolgt zumeist *diskret* bzw. *digital*, indem *Proteine* schrittweise *aktiviert*, also mit *Liganden* (z.B. Phosphatgruppen) versehen werden oder *Proteinkomplexe* bilden

Proteine besitzen Bindungsdomänen für Liganden

- z.B. Tumorsuppressorprotein *p53* besitzt 27 Phosphorylierungsstellen
- damit bis zu $2^{27} = 134.217.728$ unterscheidbare Aktivierungszustände
- jeder Zustand eineindeutig durch molekulare Struktur (eigene chemische Spezies) repräsentiert vergleichbar einer Bitkette aus 27 Bits



Verteilung der Anzahl Bindungsdomänen pro Protein

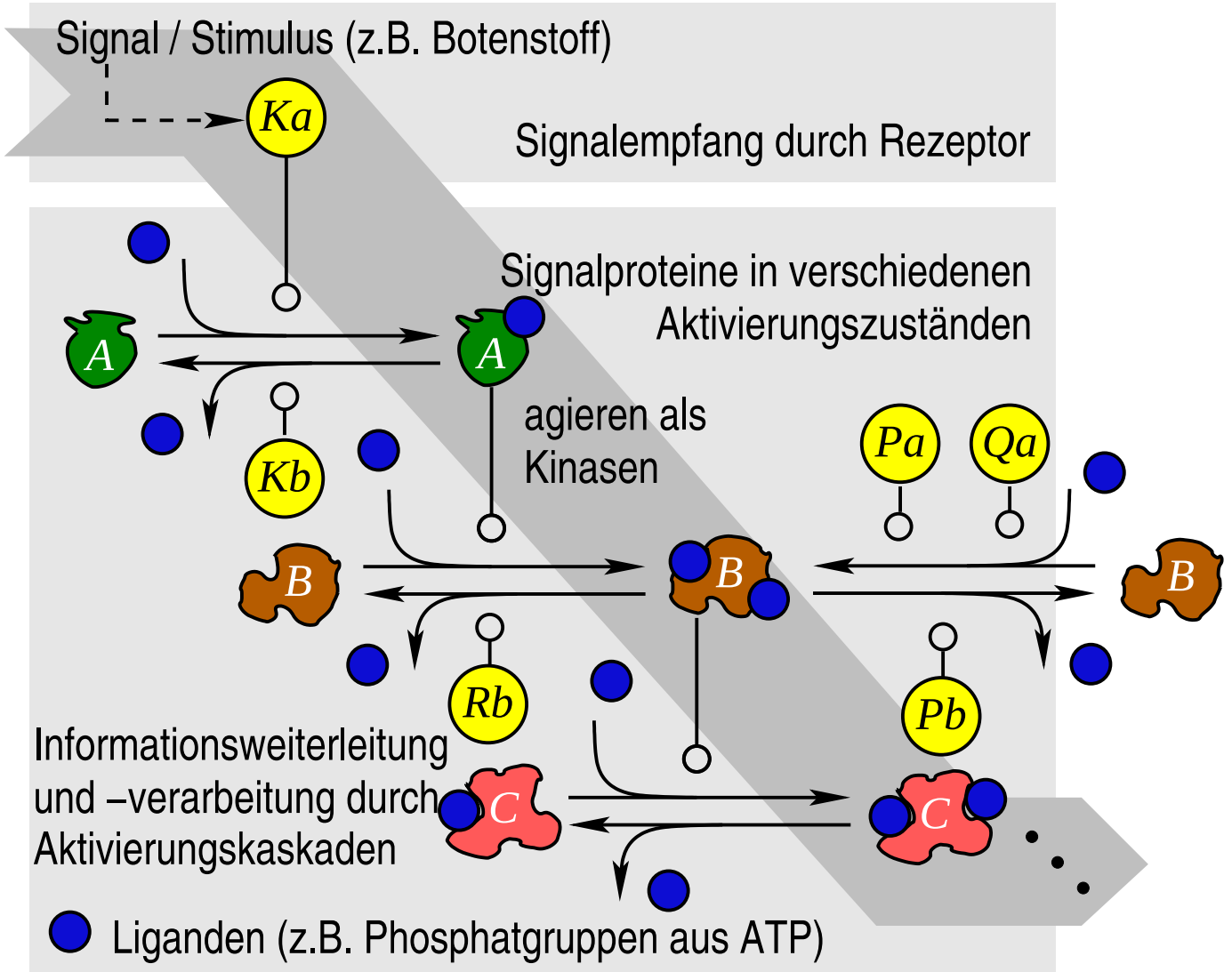


D. Xu, L. Jaroszewski, Z. Li, A. Godzik.
AIDA: *ab initio* domain assembly for automated multi-domain protein structure prediction and domain-domain interaction prediction.
Bioinformatics **31(13)**:2098-2105, 2015

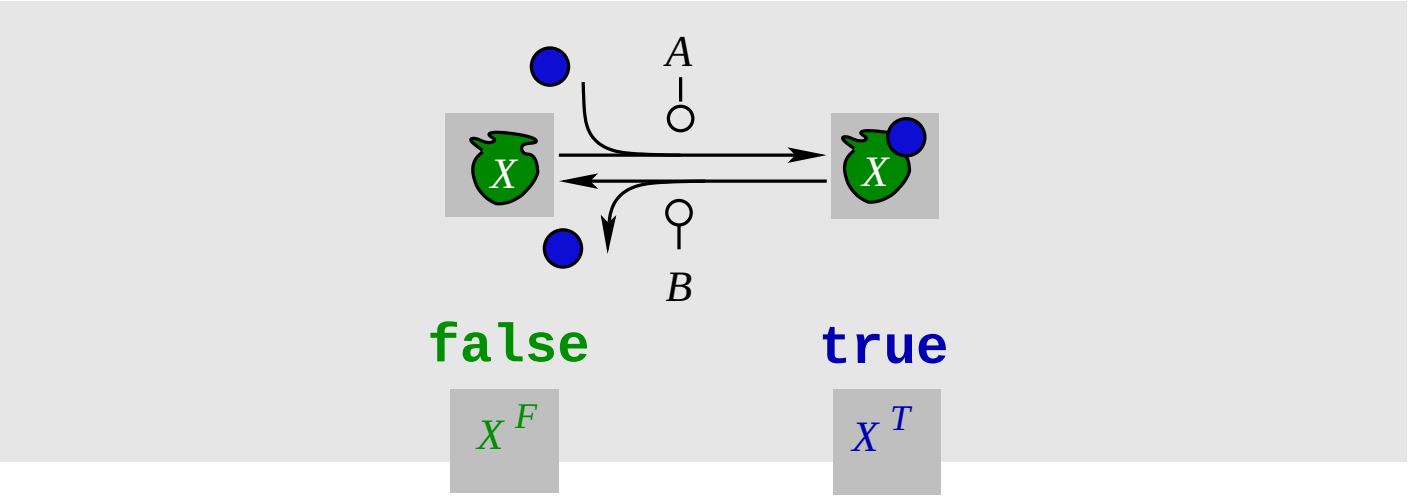
Datenbasis: ca. 130.000 in der *Protein Data Base* (PDB) strukturell erfasste Proteine (2015). **(a)**: davon ca. 66% eine Domäne, ca. 24% zwei Domänen, ca. 6% drei Domänen, ca. 4% vier oder mehr Domänen

Zellsignalkaskade aktiviert Signalproteine sukzessiv

Fiktives Beispiel dreistufige Kaskade – Informationsfluss über Aktivierungszustände



Aktivierungszustände bilden boolesche Kodierung

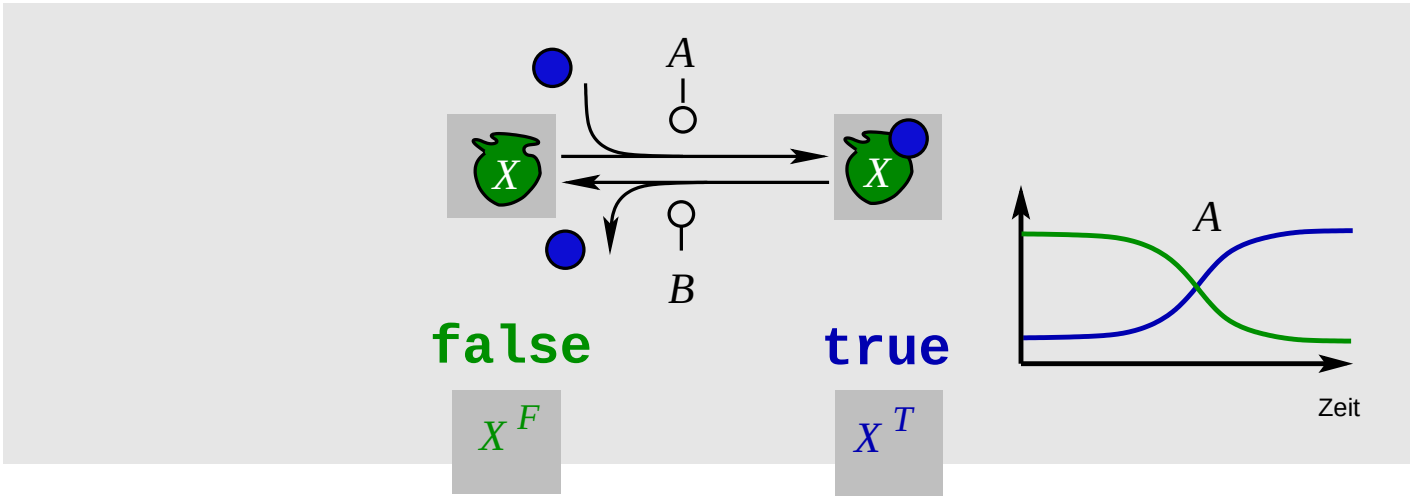


- Betrachten wir ein (fiktives) Signalprotein X.
- Deaktiviertes Protein X^F steht für den booleschen Wert **`false`**
- Aktiviertes Protein X^T steht für den booleschen Wert **`true`**
- Für die Stoffkonzentrationen gilt $[X^F] + [X^T] = \text{const.}$ sowie:

  `false` gdw. $[X^F] \gg [X^T]$
  `true` gdw. $[X^T] \gg [X^F]$

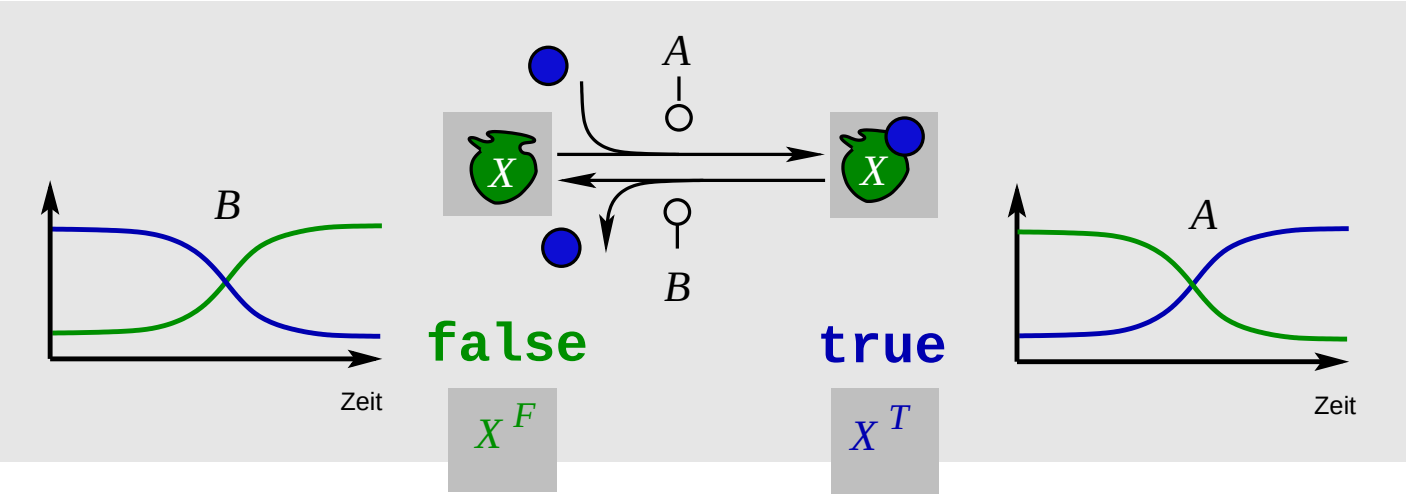
- $\gg$ bedeutet hier: mindestens zehnmal mehr

Aktivierungszustände bilden boolesche Kodierung



- Umschalten von `false` nach `true`
- ausgelöst durch Anwesenheit von Schaltenzym A
- $X^F + A \longrightarrow X^T + A$
- Liganden und mögliche Hilfsstoffe nicht explizit notiert
- Modellierung typischerweise durch eine Sättigungskinetik

Aktivierungszustände bilden boolesche Kodierung



- Umschalten von `true` nach `false`
- ausgelöst durch Anwesenheit von Schaltenzym B
- $X^T + B \longrightarrow X^F + B$
- Liganden und mögliche Hilfsstoffe nicht explizit notiert
- Modellierung typischerweise durch eine Sättigungskinetik

Chemische Digitalcomputer

Logikgatter, RS–Flip–Flops und Taktgenerator

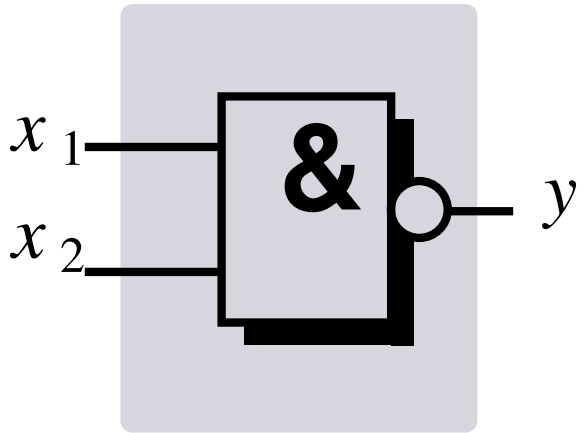
Logikgatter als Grundbausteine von Computern

"Beliebig verschaltbare NAND–Logikgatter, ein Taktgenerator und Signalwege sind alles, was man braucht, um einen Computer zu bauen, der eine beliebige algorithmisch lösbare Aufgabe erledigen kann."

Grunderkenntnis der Hardwareentwicklung

NAND-Verknüpfung als universelles Gatter

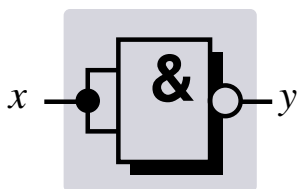
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0



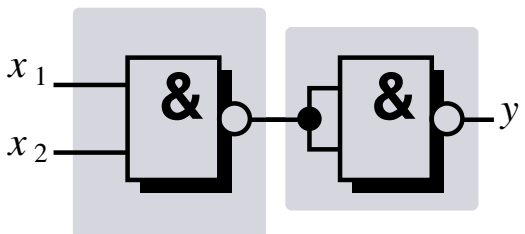
- Boolesche Notation: $y = \overline{x_1 \quad x_2} = \overline{x_1} \vee \overline{x_2}$
- Eine 0 (**`false`**) an einem Eingang setzt sich stets durch und bewirkt eine 1 (**`true`**) am Ausgang.
- Bei $x_1, x_2, y \in \{0, 1\}$ arithmetische Entsprechung: $y = 1 - x_1 \cdot x_2$
- Mithilfe von hintereinandergeschalteten NAND-Gattern lassen sich sowohl Negation als auch UND-Verknüpfung und ODER-Verknüpfung realisieren.
- NAND gilt deshalb (ebenso wie NOR) als universelles Gatter.

Simulation von Negation, UND, ODER durch NAND

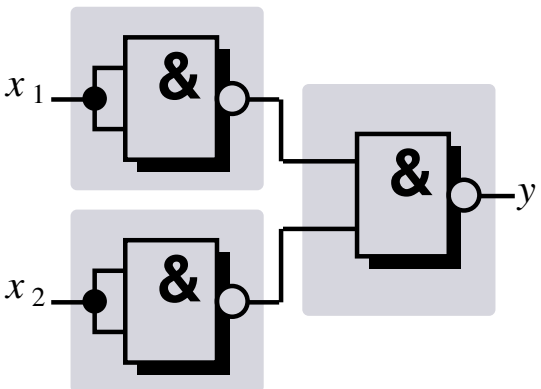
Negation	**UND**	**ODER**
$y = \overline{x}$ $= \overline{x \quad x}$	$y = x_1 \quad x_2$ $= \overline{\overline{x_1 \quad x_2}}$	$y = x_1 \vee x_2$ $= \overline{\overline{x_1 \vee x_2}}$ $= \overline{\overline{x_1} \quad \overline{x_2}}$



NICHT



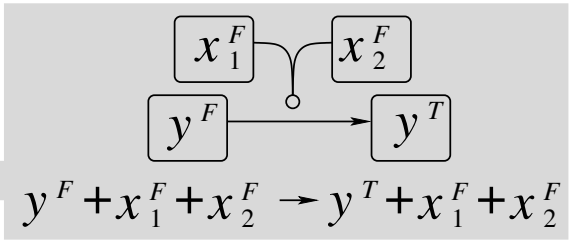
UND

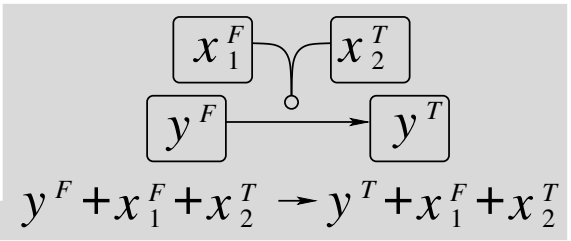


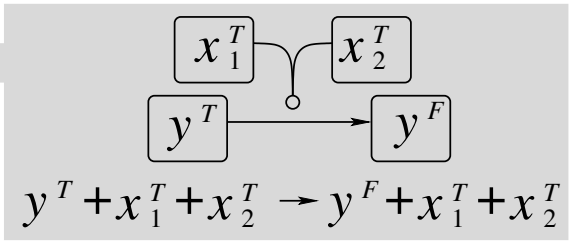
ODER

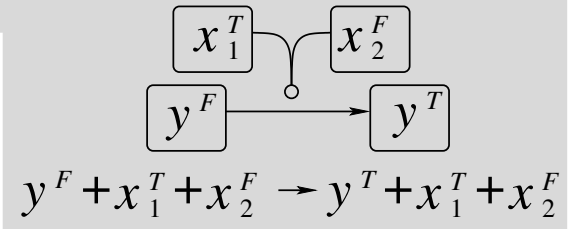
Chemisches NAND-Gatter (mit Katalysatoren)

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0





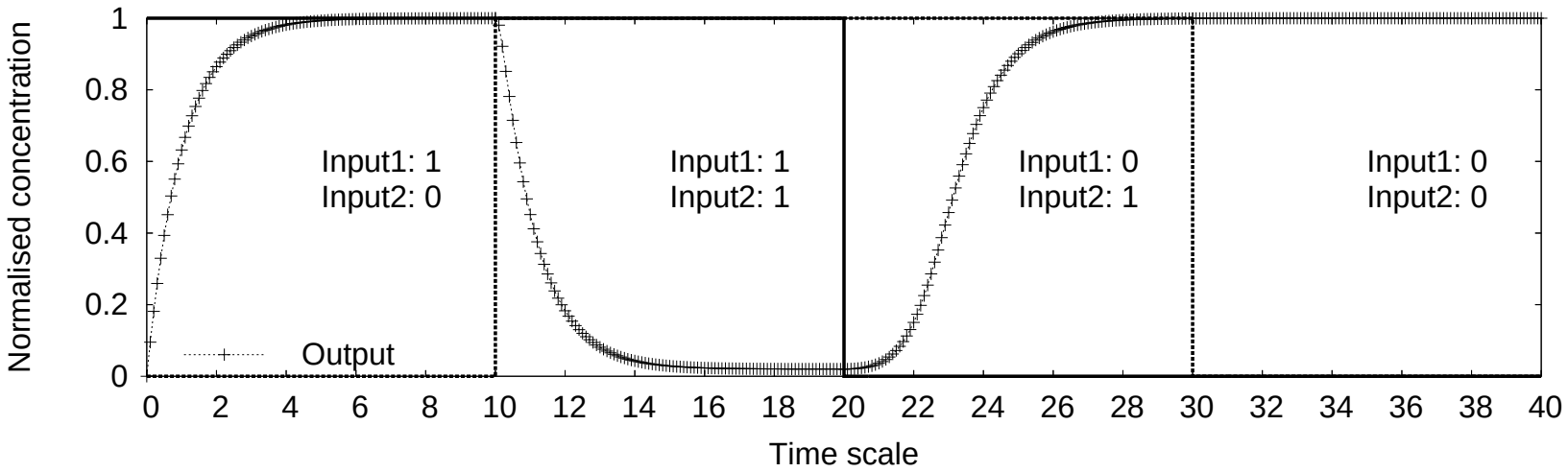




- Eingänge bilden Schaltenzyme für Ausgabe
- zulässige Initialisierungen gemäß Belegungstabelle
- schnelles Schaltverhalten über hohe Konstanten k
- Eingabe x_1, x_2 bleibt erhalten (nicht chemisch abgebaut)
- alle binären Funktionen nach diesem Prinzip realisierbar

J. Decraene, T. Hinze. A Multidisciplinary Survey of Computational Techniques for the Modelling, Simulation and Analysis of Biochemical Networks. *Journal of Universal Computer Science* **16(9)**:1152-1175, 2010

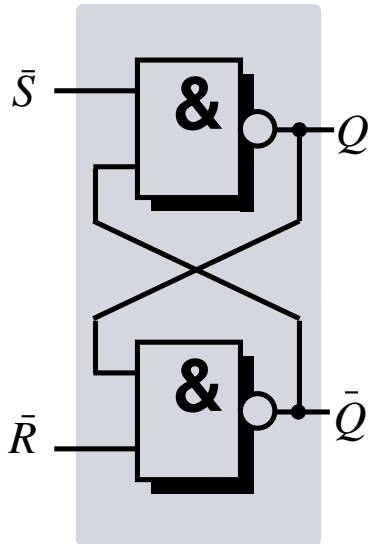
Schaltverhalten des chemischen NAND-Gatters



- Modellierung der enzymatischen Reaktionen mit Hill-Kinetik ($n = 2$)
- Bei $t = 10, 20, 30$ Eingangskonzentrationen *sprunghaft* verändert
- Beobachtung, wie Ausgangssignal darauf reagiert (*Sprungantwort*)
- logischer Wert 0 (`false`) *bis* 10% des maximal erreichbaren Signalwerts
- logischer Wert 1 (`true`) *mindestens* 90% des maximal erreichbaren Signalwerts
- *Gatter-Latenzzeit* in Abhängigkeit der Ratenkonstanten und Reaktionsparameter experimentell bestimmbar (hier: 5 Zeiteinheiten)

NAND-basiertes RS-Flip-Flop als 1-Bit-RAM-Speicher

$\overline{S}$	$\overline{R}$	Q



NAND

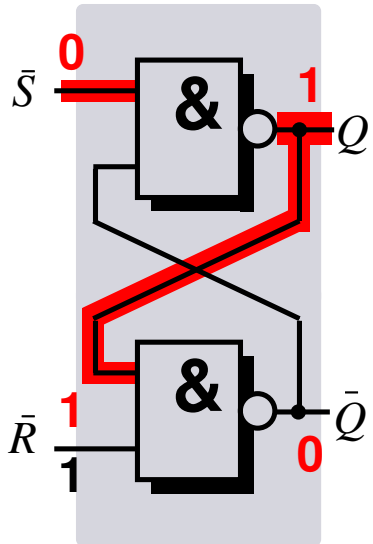
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

0 am Eingang setzt sich durch

NAND-basiertes RS-Flip-Flop als 1-Bit-RAM-Speicher

$\overline{S}$	$\overline{R}$	Q	
0	1	1	(Set)

- Signal 0 am Eingang $\overline{S}$ setzt das Flip-Flop auf $Q = 1$ (Set)



NAND

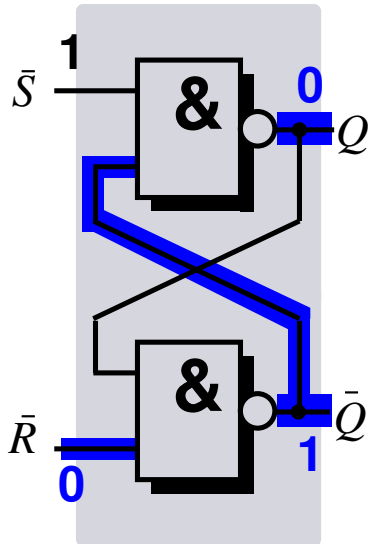
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

0 am Eingang setzt sich durch

NAND-basiertes RS-Flip-Flop als 1-Bit-RAM-Speicher

$\overline{S}$	$\overline{R}$	Q	
0	1	1	(Set)
1	0	0	(Reset)

- Signal 0 am Eingang $\overline{S}$ setzt das Flip-Flop auf $Q = 1$ (Set)
- Signal 0 am Eingang $\overline{R}$ setzt das Flip-Flop auf $Q = 0$ (Reset)



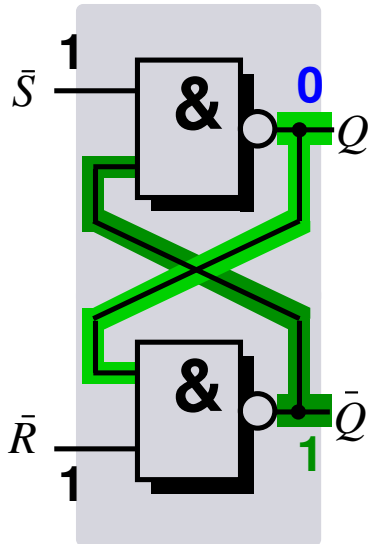
NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

0 am Eingang setzt sich durch

NAND-basiertes RS-Flip-Flop als 1-Bit-RAM-Speicher

$\overline{S}$	$\overline{R}$	Q	
0	1	1	(Set)
1	0	0	(Reset)
1	1	Q	(Hold)



- Signal 0 am Eingang $\overline{S}$ setzt das Flip-Flop auf $Q = 1$ (Set)
- Signal 0 am Eingang $\overline{R}$ setzt das Flip-Flop auf $Q = 0$ (Reset)
- Beide Eingänge 1: Flip-Flop speichert zuvor gesetztes Bit, bis erneutes Set oder Reset erfolgt

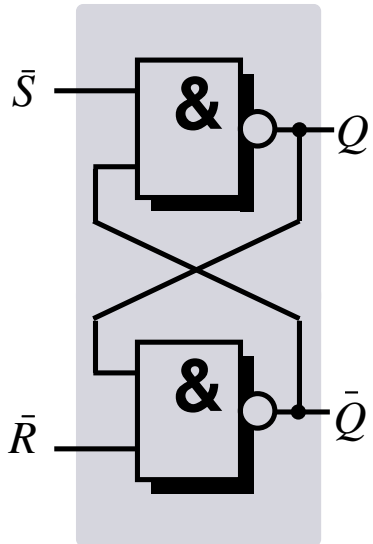
NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

0 am Eingang setzt sich durch

NAND-basiertes RS-Flip-Flop als 1-Bit-RAM-Speicher

$\overline{S}$	$\overline{R}$	Q	
0	1	1	(Set)
1	0	0	(Reset)
1	1	Q	(Hold)
0	0	verboten	



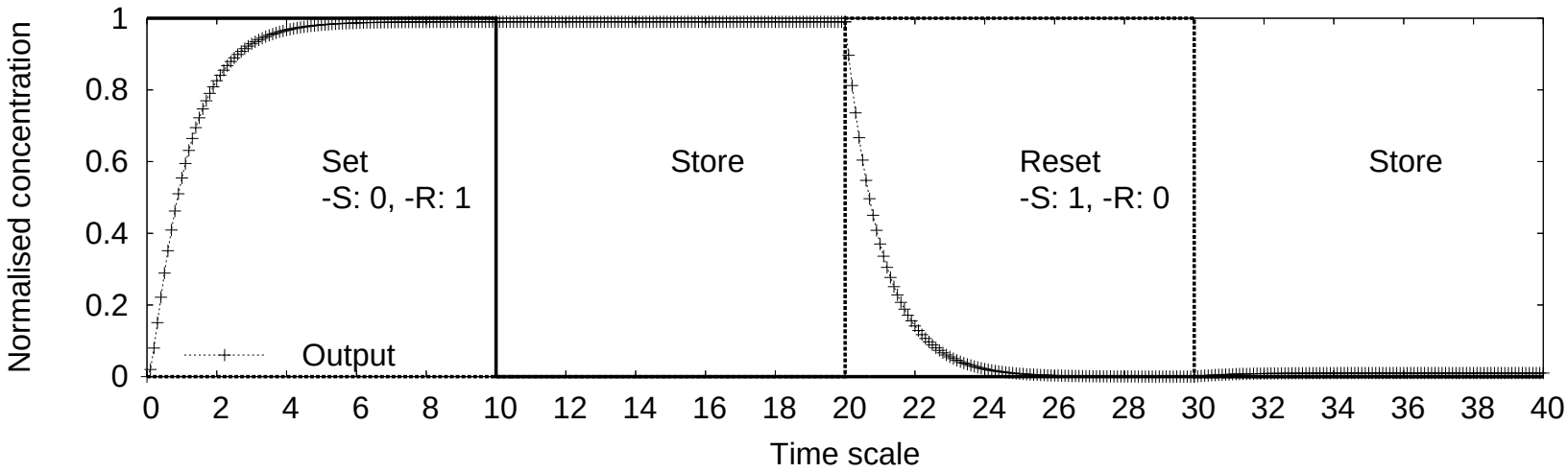
- Signal 0 am Eingang $\overline{S}$ setzt das Flip-Flop auf $Q = 1$ (Set)
- Signal 0 am Eingang $\overline{R}$ setzt das Flip-Flop auf $Q = 0$ (Reset)
- Beide Eingänge 1: Flip-Flop speichert zuvor gesetztes Bit, bis erneutes Set oder Reset erfolgt
- Beide Eingänge 0: Bit nicht gleichzeitig auf 1 und auf 0 setzbar, daher verboten ($Q = 1$ und $\overline{Q} = 1$)

NAND

x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0

0 am Eingang setzt sich durch

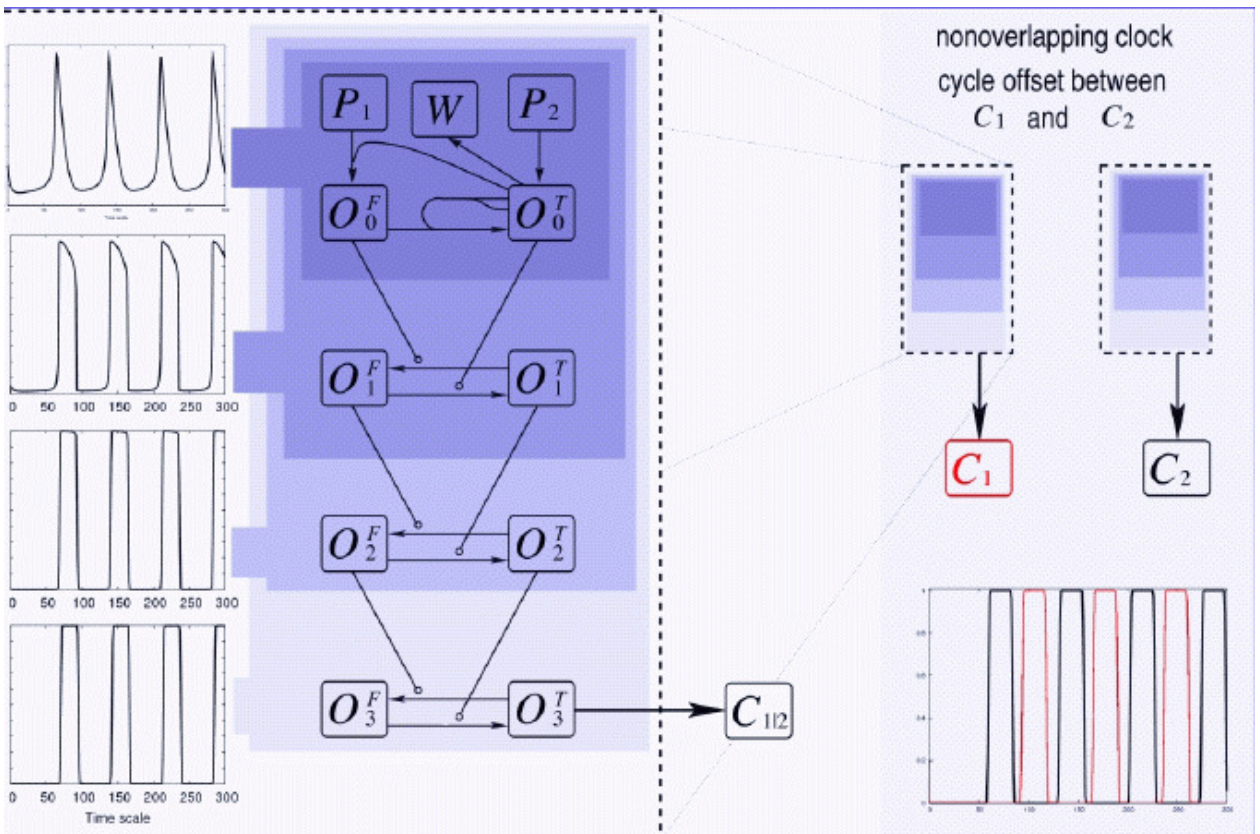
Schaltverhalten des chemischen RS-Flip-Flops



- Modellierung der enzymatischen Reaktionen mit Hill-Kinetik ($n = 2$)
- Bei $t = 0$ bis 10 Setzen, bei $t = 20$ bis 30 Rücksetzen durch *sprunghafte* Änderung der betreffenden Eingangskonzentrationen
- während des Speicherns Ausgangsspezieskonzentration gehalten
- logischer Wert 0 (`false`) *bis* 10% des maximal erreichbaren Signalwerts
- logischer Wert 1 (`true`) *mindestens* 90% des maximal erreichbaren Signalwerts
- *Latenzzeit* (worst case) in Abhängigkeit der Ratenkonstanten und Reaktionsparameter experimentell bestimmbar (hier: 7 Zeiteinheiten)

Ein chemischer Taktgenerator

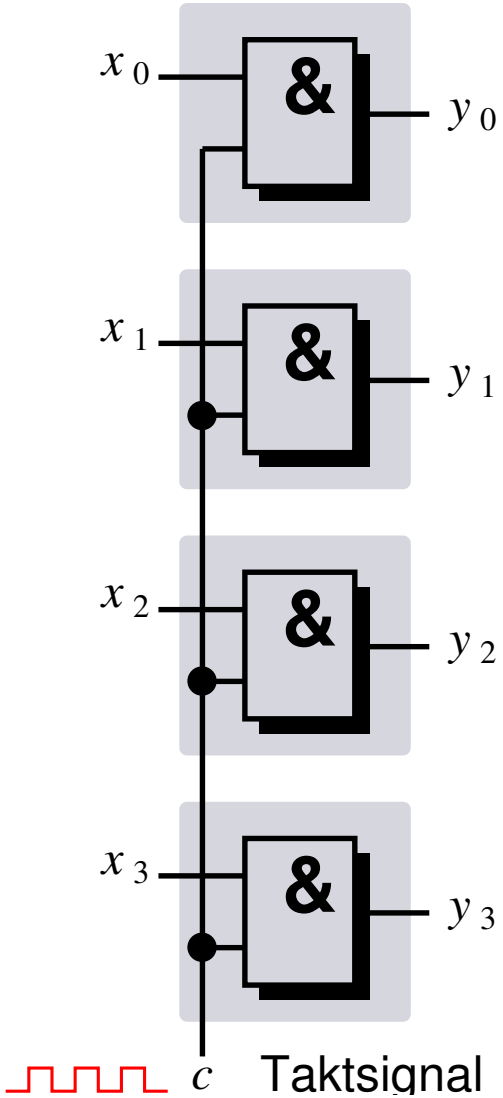
- basierend auf *Brusselator* (spikeförmige Oszillation)
- Kaskade von Hilfsreaktionen als *binärer Signalseparator*
- zwei versetzte Oszillationen liefern Taktsignale C_1 und C_2



S. Hayat, T. Hinze. Toward integration of in vivo molecular computing devices: successes and challenges. *HFSP Journal* **2(5)**:239-243, 2008

Tor aus UND-Gattern

Bei Takt $= 0$ logische Berechnungen, bei Takt $= 1$ gültige Ergebniswerte freigegeben



Chemische Digitalcomputer

Endliche Automaten und praktische Anwendungen

Nichtdeterministischer endlicher Automat (NEA)

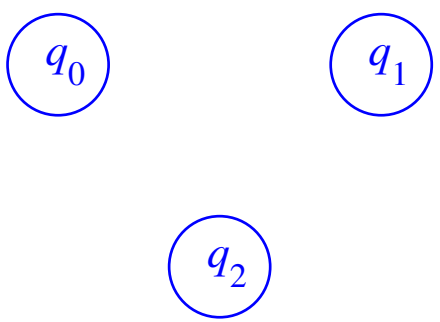
zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$

nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph

$$Q = \{q_0, q_1, q_2\}$$



Nichtdeterministischer endlicher Automat (NEA)

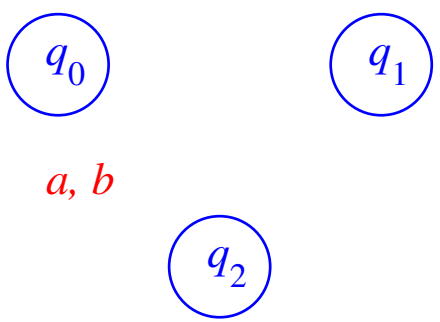
zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$$

Eingabealphabet Σ, wobei $\Sigma \cap Q = \emptyset$

nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph



$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{a, b\}$$

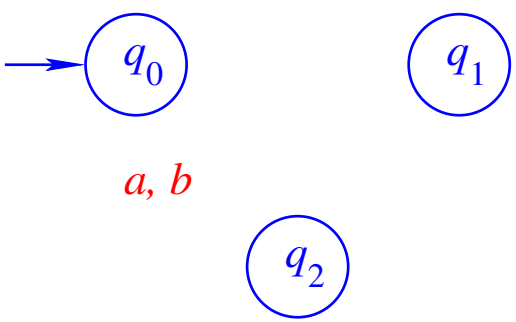
Nichtdeterministischer endlicher Automat (NEA)

zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$

- q_0: Anfangszustand $q_0 \in Q$
- Σ: Eingabealphabet Σ, wobei $\Sigma \cap Q = \emptyset$
- Q: nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph



$$Q = \{q_0, q_1, q_2\}$$
$$\Sigma = \{a, b\}$$

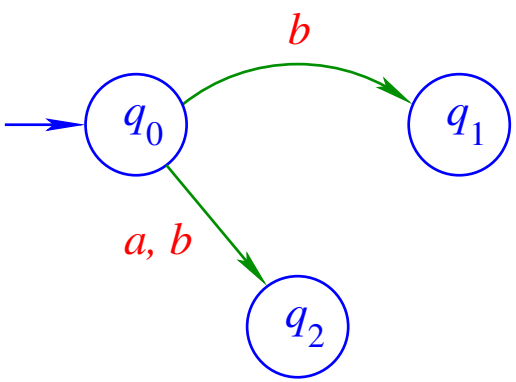
Nichtdeterministischer endlicher Automat (NEA)

zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$

- Δ: Übergangsrelation $\Delta \subseteq Q \times \Sigma \times Q$
- q_0: Anfangszustand $q_0 \in Q$
- Σ: Eingabealphabet Σ, wobei $\Sigma \cap Q = \emptyset$
- F: nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph



$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b\} \\
\Delta &= \{(q_0, a, q_2), (q_0, b, q_1), (q_0, b, q_2),
\end{aligned}
$$

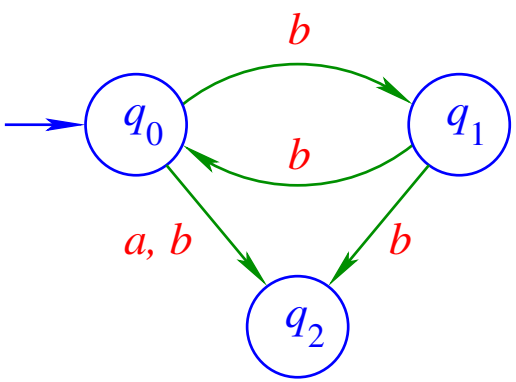
Nichtdeterministischer endlicher Automat (NEA)

zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$

- Δ: Übergangsrelation $\Delta \subseteq Q \times \Sigma \times Q$
- q_0: Anfangszustand $q_0 \in Q$
- Σ: Eingabealphabet Σ, wobei $\Sigma \cap Q = \emptyset$
- F: nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph



$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b\} \\
\Delta &= \{(q_0, a, q_2), (q_0, b, q_1), (q_0, b, q_2), \\
&\quad (q_1, b, q_0), (q_1, b, q_2)\}
\end{aligned}
$$

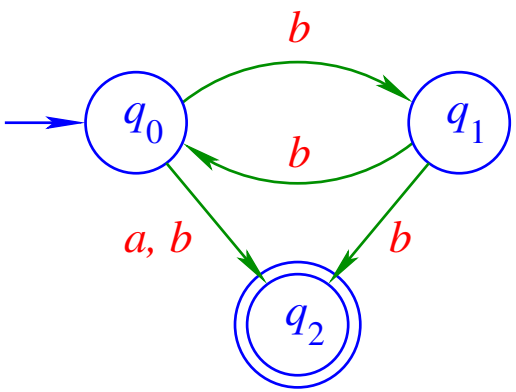
Nichtdeterministischer endlicher Automat (NEA)

zustandsbasiertes Berechnungsmodell für reguläre Sprachen

$\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$

- F: Menge der Finalzustände $F \subseteq Q$
- Δ: Übergangsrelation $\Delta \subseteq Q \times \Sigma \times Q$
- q_0: Anfangszustand $q_0 \in Q$
- Σ: Eingabealphabet Σ, wobei $\Sigma \cap Q = \emptyset$
- Q: nichtleere endliche Menge von Zuständen Q

Beispiel und Notation als Graph

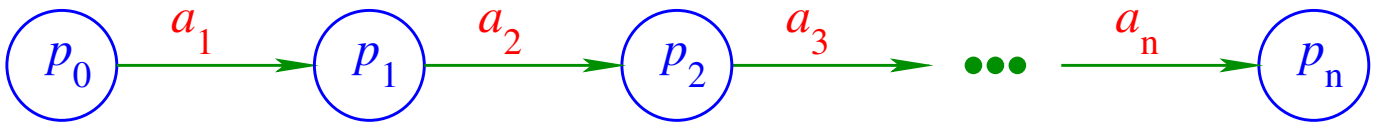


$$
\begin{aligned}
Q &= \{q_0, q_1, q_2\} \\
\Sigma &= \{a, b\} \\
\Delta &= \{(q_0, a, q_2), (q_0, b, q_1), (q_0, b, q_2), \\
&\quad\ (q_1, b, q_0), (q_1, b, q_2)\} \\
F &= \{q_2\}
\end{aligned}
$$

Ausführung von Berechnungen auf NEAs

Pfad

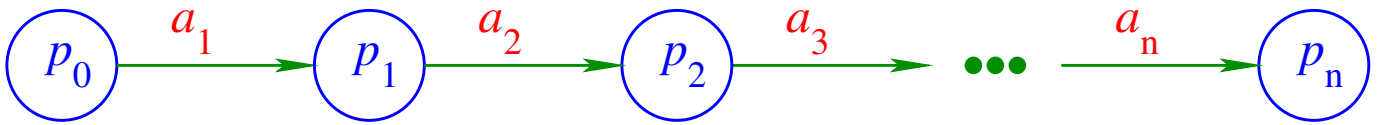
Ein *Pfad* in einem NEA $\mathcal{N}$ ist eine Folge
$p_0 \xrightarrow{a_1 \ldots a_n} p_n = (p_0, a_1, p_1)(p_1, a_2, p_2) \ldots (p_{n-1}, a_n, p_n)$ mit
$(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \ldots, n-1$.
Die Beschriftung des Pfades ist das Wort $a_1 \ldots a_n$.



Ausführung von Berechnungen auf NEAs

Pfad

Ein *Pfad* in einem NEA $\mathcal{N}$ ist eine Folge
$p_0 \xrightarrow{a_1 \ldots a_n} p_n = (p_0, a_1, p_1)(p_1, a_2, p_2) \ldots (p_{n-1}, a_n, p_n)$ mit
$(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \ldots, n-1$.
Die Beschriftung des Pfades ist das Wort $a_1 \ldots a_n$.



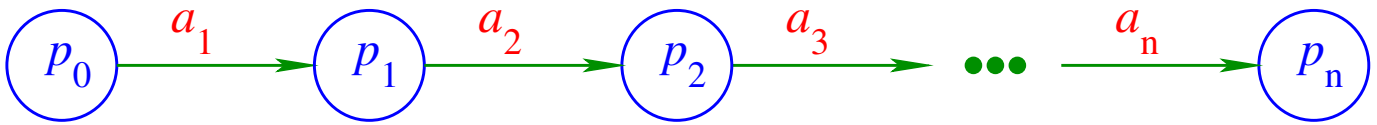
Akzeptierte Sprache

Ein NEA $\mathcal{N}$ *akzeptiert* das Wort $w \in \Sigma^*$ gdw. es einen Pfad $q_0 \xrightarrow{w} q_F$ gibt, wobei q_0 der Startzustand ist und $q_F \in F$ ein Finalzustand.

Ausführung von Berechnungen auf NEAs

Pfad

Ein *Pfad* in einem NEA $\mathcal{N}$ ist eine Folge
$p_0 \xrightarrow{a_1 \ldots a_n} p_n = (p_0, a_1, p_1)(p_1, a_2, p_2) \ldots (p_{n-1}, a_n, p_n)$ mit
$(p_i, a_{i+1}, p_{i+1}) \in \Delta$ für $i = 0, \ldots, n-1$.
Die Beschriftung des Pfades ist das Wort $a_1 \ldots a_n$.



Akzeptierte Sprache

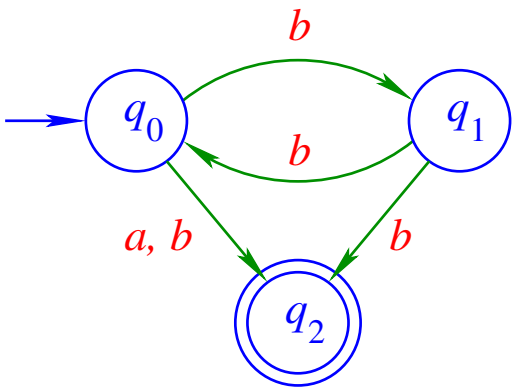
Ein NEA $\mathcal{N}$ *akzeptiert* das Wort $w \in \Sigma^*$ gdw. es einen Pfad $q_0 \xrightarrow{w} q_F$ gibt, wobei q_0 der Startzustand ist und $q_F \in F$ ein Finalzustand.

Die von $\mathcal{N}$ *akzeptierte (erkannte) Sprache* ist
$L(\mathcal{N}) = \{w \mid \mathcal{N} \text{ akzeptiert } w\}$.

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

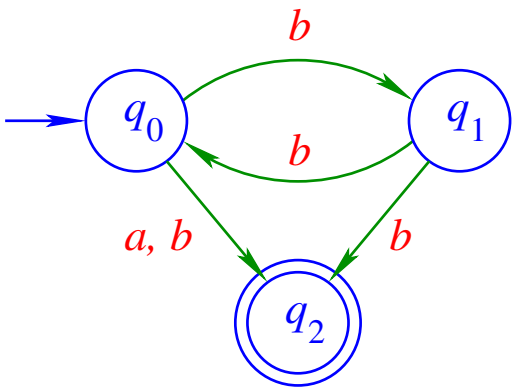


$$L(\mathcal{N}) \quad = \quad \{a, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

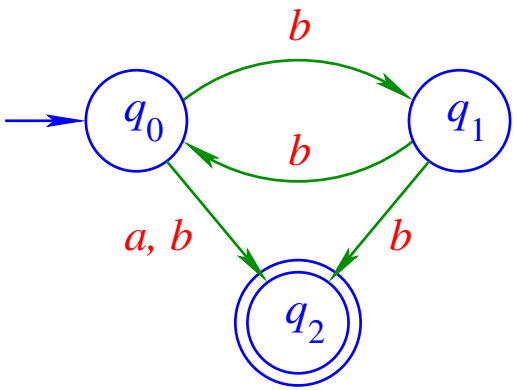


$$L(\mathcal{N}) \quad = \quad \{a, b, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

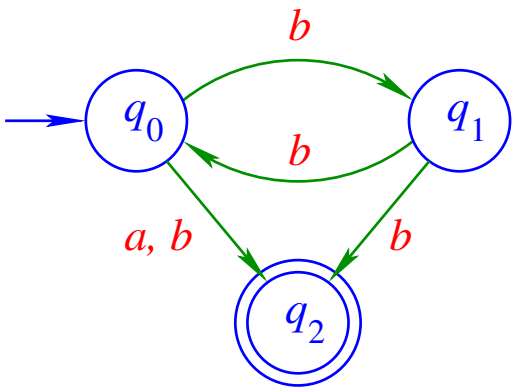


$$L(\mathcal{N}) \quad = \quad \{a, b, bb, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

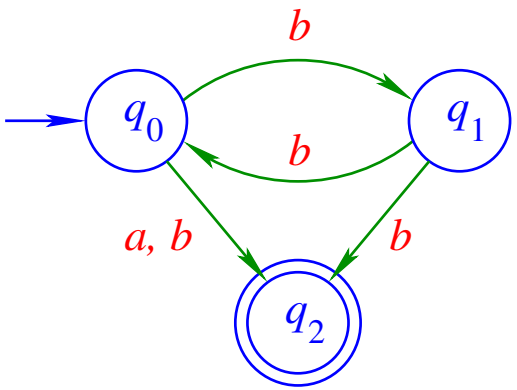


$$L(\mathcal{N}) \quad = \quad \{a, b, bb, bbb, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

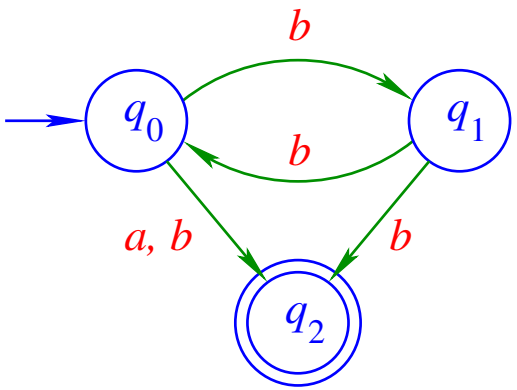


$$L(\mathcal{N}) \quad = \quad \{a, b, bb, bbb, b^4, b^5, b^6, b^7, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

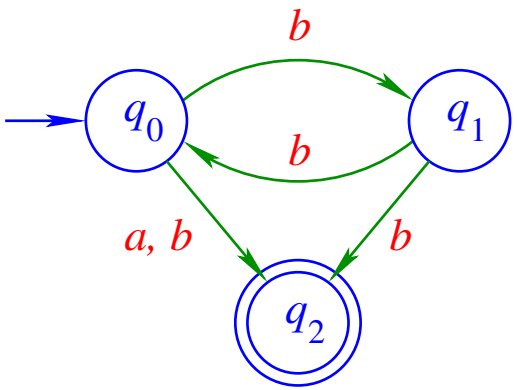


$$L(\mathcal{N}) \quad = \quad \{a, b, bb, bbb, b^4, b^5, b^6, b^7, \ldots, bba, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

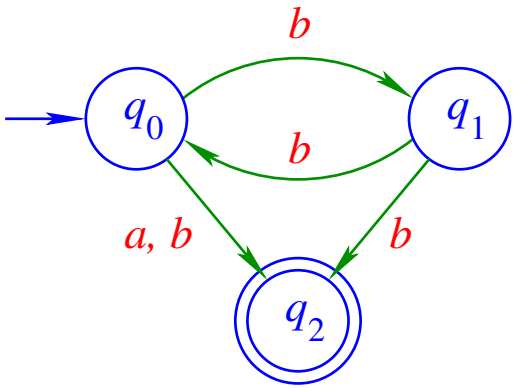


$$L(\mathcal{N}) \quad = \quad \{a, b, bb, bbb, b^4, b^5, b^6, b^7, \ldots, bba, b^4a, b^6a, \ldots\}$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$

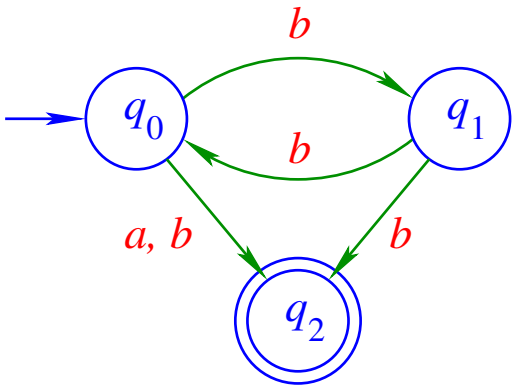


$$
\begin{aligned}
L(\mathcal{N}) &= \{a, b, bb, bbb, b^4, b^5, b^6, b^7, \ldots, bba, b^4a, b^6a, \ldots\} \\
&= \{a\} \cup \{\text{Wörter aus mindestens einem oder beliebig vielen } b\} \\
&\quad \cup \{\text{Wörter aus gerader Anzahl } b \text{ und Schluss-}a\}
\end{aligned}
$$

Ausführung von Berechnungen auf NEAs

Beispiel

Gegeben: NEA $\mathcal{N}$



$$
\begin{aligned}
L(\mathcal{N}) &= \{a, b, bb, bbb, b^4, b^5, b^6, b^7, \ldots, bba, b^4a, b^6a, \ldots\} \\
&= \{a\} \cup \{\text{Wörter aus mindestens einem oder beliebig vielen } b\} \\
&\quad \cup \{\text{Wörter aus gerader Anzahl } b \text{ und Schluss-}a\} \\
&= (\{b\} \cdot \{b\})^* \cdot \{a\} \cup \{b\} \cdot \{b\}^*
\end{aligned}
$$

Ausführung von Berechnungen auf NEAs

- Die Gesamtheit aller Wörter, die vom Startzustand q_0 zeichenweise eingelesen in einen Finalzustand $\in F$ führen, bilden die formale Sprache $L(\mathcal{N})$, die der NEA $\mathcal{N}$ erkennt.

Ausführung von Berechnungen auf NEAs

- Die Gesamtheit aller Wörter, die vom Startzustand q_0 zeichenweise eingelesen in einen Finalzustand $\in F$ führen, bilden die formale Sprache $L(\mathcal{N})$, die der NEA $\mathcal{N}$ erkennt.
- $L(\mathcal{N})$ kann (abzählbar) unendlich viele Wörter enthalten, aber auch endlich oder sogar leer sein.

Ausführung von Berechnungen auf NEAs

- Die Gesamtheit aller Wörter, die vom Startzustand q_0 zeichenweise eingelesen in einen Finalzustand $\in F$ führen, bilden die formale Sprache $L(\mathcal{N})$, die der NEA $\mathcal{N}$ erkennt.
- $L(\mathcal{N})$ kann (abzählbar) unendlich viele Wörter enthalten, aber auch endlich oder sogar leer sein.
- Man kann konstruktiv zeigen, dass durch NEAs alle regulären Sprachen beschrieben werden können, aber keine weiteren.

NEAs als Berechnungsmodell

Einfache algorithmische Aufgaben, z.B. Pattern Matching mit regulären Ausdrücken, lassen sich gut durch NEAs beschreiben.

NEAs als Berechnungsmodell

Einfache algorithmische Aufgaben, z.B. Pattern Matching mit regulären Ausdrücken, lassen sich gut durch NEAs beschreiben.

Vorteile von NEAs

- bieten eine kompakte, gut verständliche Darstellung
- kommen oft mit wenigen Zuständen aus

NEAs als Berechnungsmodell

Einfache algorithmische Aufgaben, z.B. Pattern Matching mit regulären Ausdrücken, lassen sich gut durch NEAs beschreiben.

Vorteile von NEAs

- bieten eine kompakte, gut verständliche Darstellung
- kommen oft mit wenigen Zuständen aus

Nachteile von NEAs

- Nichtdeterminismus (gleichzeitiger Übergang in *mehrere* Zustände) erschwert technische Implementierung
- Verfolgung aller Berechnungspfade mitunter aufwendig
- Berechnung endet abrupt, wenn es mit der gelesenen Eingabe am aktuellen Zustand keinen Folgezustand gibt

Deterministischer endlicher Automat (DEA)

Ein NEA $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), wenn es für alle $q \in Q$ und für alle $a \in \Sigma$ stets **genau einen** Folgezustand $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Deterministischer endlicher Automat (DEA)

Ein NEA $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), wenn es für alle $q \in Q$ und für alle $a \in \Sigma$ stets **genau einen** Folgezustand $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Anstelle der Übergangsrelation Δ verwenden wir dann die *Übergangsfunktion* $\delta : Q \times \Sigma \to Q$ mit $\delta(q, a) = q'$ gdw. $(q, a, q') \in \Delta$. DEAs werden in der Form $(Q, \Sigma, q_0, \delta, F)$ geschrieben.

Deterministischer endlicher Automat (DEA)

Ein NEA $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), wenn es für alle $q \in Q$ und für alle $a \in \Sigma$ stets **genau einen** Folgezustand $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Anstelle der Übergangsrelation Δ verwenden wir dann die *Übergangsfunktion* $\delta : Q \times \Sigma \to Q$ mit $\delta(q, a) = q'$ gdw. $(q, a, q') \in \Delta$. DEAs werden in der Form $(Q, \Sigma, q_0, \delta, F)$ geschrieben.

Beachte

- Aus dem aktuellen Zustand q und dem gelesenen Zeichen a ist der Folgezustand q' eineindeutig bestimmt, es kann keine gleichzeitigen Übergänge zu mehreren Folgezuständen geben.

Deterministischer endlicher Automat (DEA)

Ein NEA $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), wenn es für alle $q \in Q$ und für alle $a \in \Sigma$ stets **genau einen** Folgezustand $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Anstelle der Übergangsrelation Δ verwenden wir dann die *Übergangsfunktion* $\delta : Q \times \Sigma \to Q$ mit $\delta(q, a) = q'$ gdw. $(q, a, q') \in \Delta$. DEAs werden in der Form $(Q, \Sigma, q_0, \delta, F)$ geschrieben.

Beachte

- Aus dem aktuellen Zustand q und dem gelesenen Zeichen a ist der Folgezustand q' eineindeutig bestimmt, es kann keine gleichzeitigen Übergänge zu mehreren Folgezuständen geben.
- Wir fordern, dass die Übergangsfunktion δ eine *totale Funktion* ist, also für *jedes* Paar $(q, a) \in Q \times \Sigma$ einen Folgezustand liefert. Man kann dies immer durch Hinzunahme eines „Papierkorbzustands“, der kein Finalzustand ist, erreichen.

Deterministischer endlicher Automat (DEA)

Ein NEA $\mathcal{N} = (Q, \Sigma, q_0, \Delta, F)$ heißt *deterministisch* (DEA), wenn es für alle $q \in Q$ und für alle $a \in \Sigma$ stets **genau einen** Folgezustand $q' \in Q$ gibt mit $(q, a, q') \in \Delta$.

Anstelle der Übergangsrelation Δ verwenden wir dann die *Übergangsfunktion* $\delta : Q \times \Sigma \to Q$ mit $\delta(q, a) = q'$ gdw. $(q, a, q') \in \Delta$. DEAs werden in der Form $(Q, \Sigma, q_0, \delta, F)$ geschrieben.

Beachte

- Aus dem aktuellen Zustand q und dem gelesenen Zeichen a ist der Folgezustand q' eineindeutig bestimmt, es kann keine gleichzeitigen Übergänge zu mehreren Folgezuständen geben.
- Wir fordern, dass die Übergangsfunktion δ eine *totale Funktion* ist, also für *jedes* Paar $(q, a) \in Q \times \Sigma$ einen Folgezustand liefert. Man kann dies immer durch Hinzunahme eines „Papierkorbzustands“, der kein Finalzustand ist, erreichen.

$\Longrightarrow$ Jeder NEA über Potenzmengenkonstruktion in DEA überführbar

Chemische Digitalcomputermodelle nach dem Vorbild der *Zellsignalverarbeitung* implementieren *deterministische endliche Automaten* (DEAs).

Kodierung der Alphabetsymbole

Jeder DEA $(Q, \Sigma, q_0, \delta, F)$ besitzt endliche Menge Σ an *Alphabetsymbolen*, von denen in jedem Berechnungsschritt eines als Eingabe eingelesen wird. Sei $\Sigma = \{a_1, a_2, \ldots, a_m\}$.

Elegante Kodierung

Verwende ein *Signalprotein* X, das mindestens soviele Aktivierungszustände besitzt wie Anzahl Symbole in Σ und ordne zu $X^{a_1}, X^{a_2}, \ldots, X^{a_m}$

Kodierung der Alphabetsymbole

Jeder DEA $(Q, \Sigma, q_0, \delta, F)$ besitzt endliche Menge Σ an *Alphabetsymbolen*, von denen in jedem Berechnungsschritt eines als Eingabe eingelesen wird. Sei $\Sigma = \{a_1, a_2, \ldots, a_m\}$.

Elegante Kodierung

Verwende ein *Signalprotein X*, das mindestens soviele Aktivierungszustände besitzt wie Anzahl Symbole in Σ und ordne zu $X^{a_1}, X^{a_2}, \ldots, X^{a_m}$

Binäre Kodierung

Ordne jedem Alphabetsymbol eineindeutig eine *Bitkette* der Länge $\lceil \log_2(|\Sigma|) \rceil$ zu und ersetze jeden Zustandsübergang $q_i \xrightarrow{a_k} q_j$ durch den Pfad $q_i \xrightarrow{b_1} q_{i,2} \xrightarrow{b_2} q_{i,3} \xrightarrow{b_3} \cdots \xrightarrow{b_h} q_j$, wobei die Bitkette $b_1 b_2 \ldots b_h$ das Symbol a_k repräsentiert. Dabei werden zusätzliche Hilfszustände $q_{i,2}, q_{i,3}, \ldots$ eingeführt. Damit ist $\Sigma = \{0, 1\}$

Kodierung der Zustände

Jeder DEA $(Q, \Sigma, q_0, \delta, F)$ besitzt endliche Menge Q an *Zuständen*. In jedem Berechnungsschritt erfolgt ein Zustandsübergang. Sei $Q = \{q_1, q_2, \ldots, q_n\}$.

Elegante Kodierung

Verwende ein *Signalprotein* Z, das mindestens soviele Aktivierungszustände besitzt wie die Anzahl Zustände in Q und ordne zu $Z^{q_1}, Z^{q_2}, \ldots, Z^{q_n}$

Kodierung der Zustände

Jeder DEA $(Q, \Sigma, q_0, \delta, F)$ besitzt endliche Menge Q an *Zuständen*. In jedem Berechnungsschritt erfolgt ein Zustandsübergang. Sei $Q = \{q_1, q_2, \ldots, q_n\}$.

Elegante Kodierung

Verwende ein *Signalprotein* Z, das mindestens soviele Aktivierungszustände besitzt wie die Anzahl Zustände in Q und ordne zu $Z^{q_1}, Z^{q_2}, \ldots, Z^{q_n}$

Kodierung als Bitkette

Ordne jedem Zustand eineindeutig eine *Bitkette* der Länge $\lceil \log_2(|Q|) \rceil$ zu und ersetze die Einträge in der Zustandsübergangsfunktion entsprechend. Sei $b_1 b_2 \ldots b_h$ die Bitkette für Zustand q_i und $c_1 c_2 \ldots c_h$ die Bitkette für Zustand q_j, so wird aus dem Zustandsübergang $q_i \xrightarrow{a_k} q_j$ die Form $b_1 b_2 \ldots b_h \xrightarrow{a_k} c_1 c_2 \ldots c_h$.

Abbildung DEA in Zellsignal-Reaktionsnetzwerk

unter Verwendung der eleganten Kodierungen für Zustände und Alphabetsymbole

- Aus einem DEA $(Q, \Sigma, q_0, \delta, F)$, dessen Zustände $q_1, \ldots, q_n \in Q$ durch Aktivierungszustände $Z^{q_1}, \ldots, Z^{q_n}$ des Signalproteins Z
- und dessen Alphabetsymbole $a_1, \ldots, a_m \in \Sigma$ durch Aktivierungszustände $X^{a_1}, \ldots, X^{a_m}$ des Signalproteins X kodiert sind,
- wird unter Zuhilfenahme eines *Taktsignals* $[C]$ ein Reaktionsgleichungssystem,

Abbildung DEA in Zellsignal-Reaktionsnetzwerk

unter Verwendung der eleganten Kodierungen für Zustände und Alphabetsymbole

- Aus einem DEA $(Q, \Sigma, q_0, \delta, F)$, dessen Zustände $q_1, \ldots, q_n \in Q$ durch Aktivierungszustände $Z^{q_1}, \ldots, Z^{q_n}$ des Signalproteins Z
- und dessen Alphabetsymbole $a_1, \ldots, a_m \in \Sigma$ durch Aktivierungszustände $X^{a_1}, \ldots, X^{a_m}$ des Signalproteins X kodiert sind,
- wird unter Zuhilfenahme eines *Taktsignals* $[C]$ ein Reaktionsgleichungssystem,

indem jeder Eintrag

$$(q_i, a_k, q_j) \qquad \text{bzw.} \qquad q_i \xrightarrow{a_k} q_j$$

der Zustandsübergangsfunktion $\delta : Q \times \Sigma \longrightarrow Q$ überführt wird in die zugehörige Reaktionsgleichung

$$Z^{q_i} + X^{a_k} + C \longrightarrow Z^{q_j} + X^{a_k} + C$$

Abbildung DEA in Zellsignal-Reaktionsnetzwerk

unter Verwendung der eleganten Kodierungen für Zustände und Alphabetsymbole

- Aus einem DEA $(Q, \Sigma, q_0, \delta, F)$, dessen Zustände $q_1, \ldots, q_n \in Q$ durch Aktivierungszustände $Z^{q_1}, \ldots, Z^{q_n}$ des Signalproteins Z
- und dessen Alphabetsymbole $a_1, \ldots, a_m \in \Sigma$ durch Aktivierungszustände $X^{a_1}, \ldots, X^{a_m}$ des Signalproteins X kodiert sind,
- wird unter Zuhilfenahme eines *Taktsignals* $[C]$ ein Reaktionsgleichungssystem,

indem jeder Eintrag

$$(q_i, a_k, q_j) \qquad \text{bzw.} \qquad q_i \xrightarrow{a_k} q_j$$

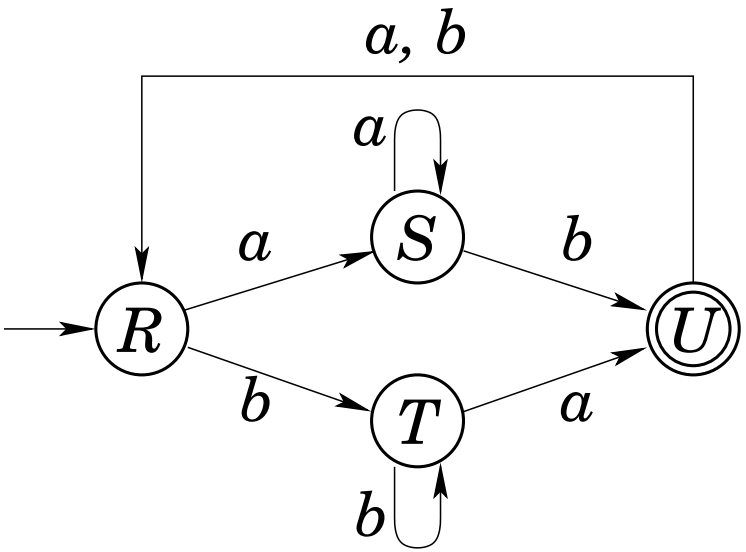
der Zustandsübergangsfunktion $\delta : Q \times \Sigma \longrightarrow Q$ überführt wird in die zugehörige Reaktionsgleichung

$$Z^{q_i} + X^{a_k} + C \longrightarrow Z^{q_j} + X^{a_k} + C$$

Die Eingabekonzentrationsverläufe $[X^{a_1}]$ bis $[X^{a_m}]$ sind auf den Verlauf des Taktsignals $[C]$ abgestimmt und ändern sich sprunghaft, während $[C] \approx 0$.

Beispiel zur chemischen Modellierung eines DEA

Zustände durch Protein Z mit 4, Symbole durch Protein X mit 2 Aktivierungszuständen

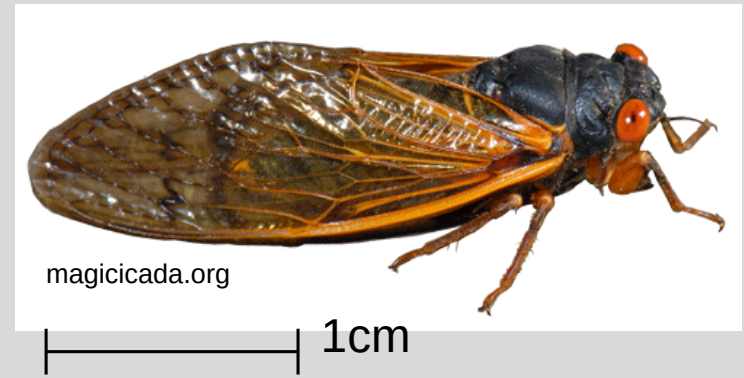


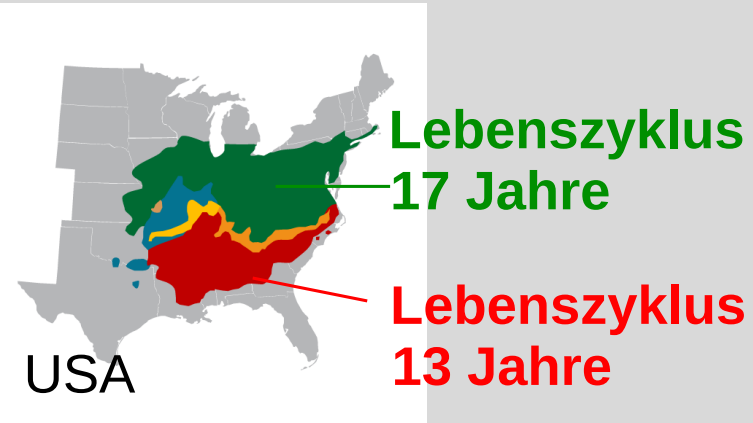
$$Z^R + X^a + C \longrightarrow Z^S + X^a + C$$
$$Z^R + X^b + C \longrightarrow Z^T + X^b + C$$
$$Z^S + X^a + C \longrightarrow Z^S + X^a + C$$
$$Z^S + X^b + C \longrightarrow Z^U + X^b + C$$
$$Z^T + X^a + C \longrightarrow Z^U + X^a + C$$
$$Z^T + X^b + C \longrightarrow Z^T + X^b + C$$
$$Z^U + X^a + C \longrightarrow Z^R + X^a + C$$
$$Z^U + X^b + C \longrightarrow Z^R + X^b + C$$

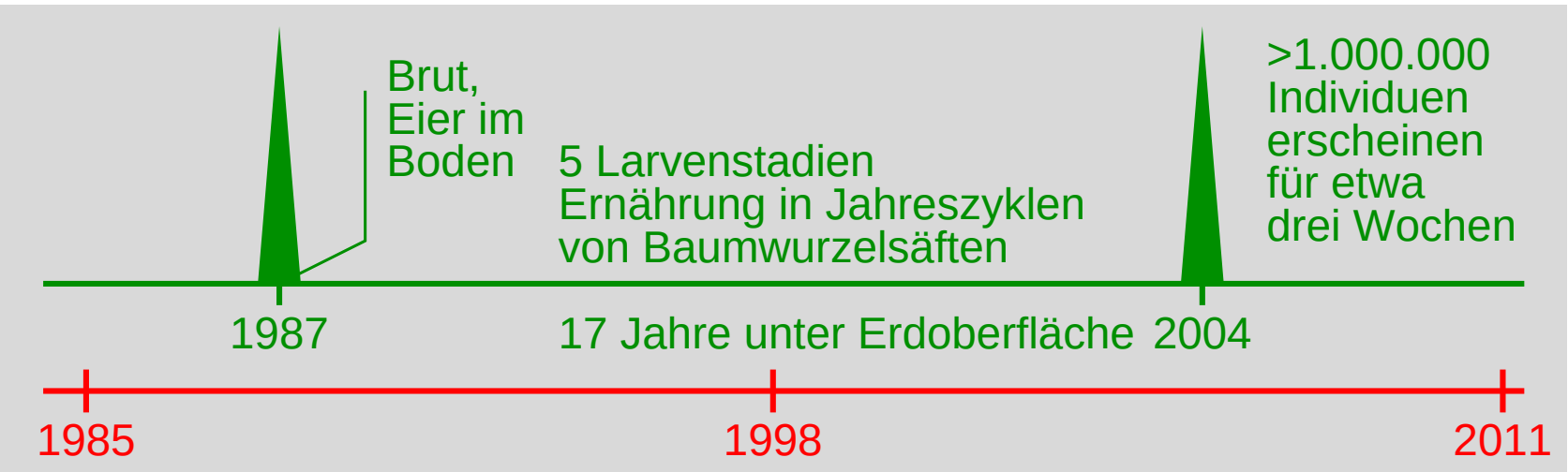
Anfangskonzentrationen: $[Z^R] = 1, [Z^S] = [Z^T] = [Z^U] = 0$

Laufbedingungen: $[X^a] + [X^b] = 1$ und $[Z^R] + [Z^S] + [Z^T] + [Z^U] = 1$

Zikaden-inspirierter chemischer Frequenzteiler 1:17







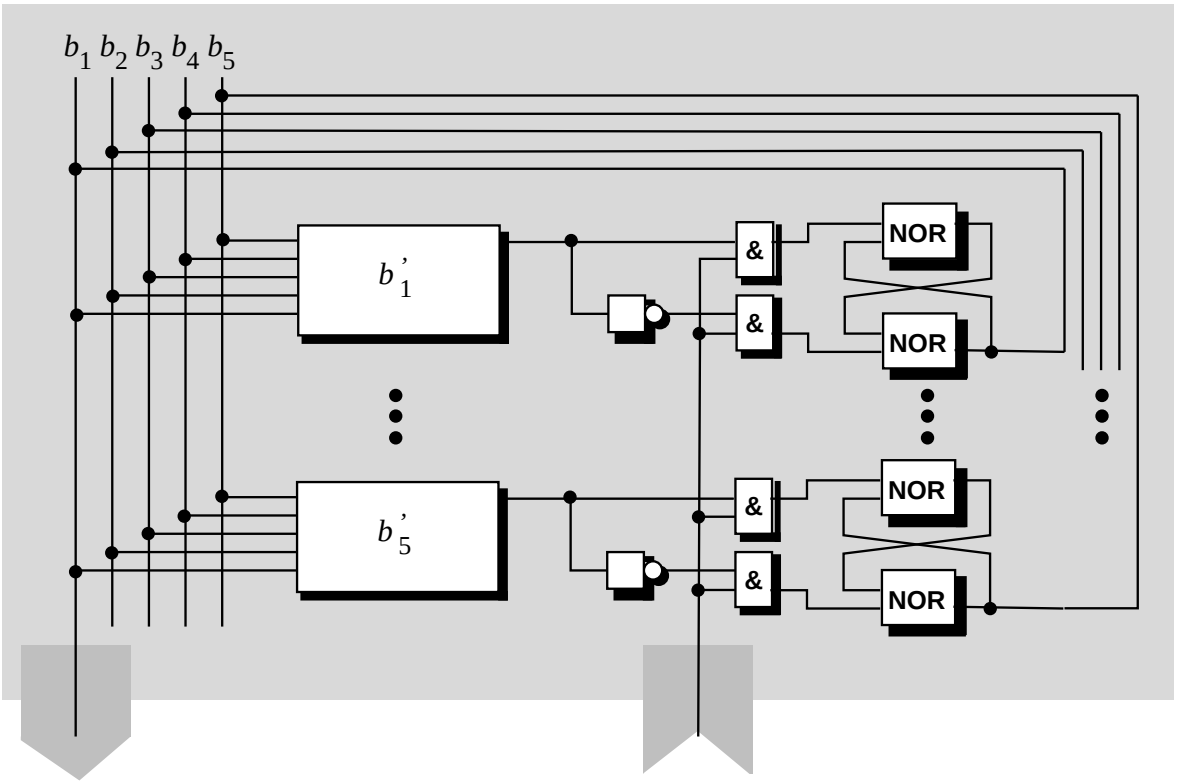
deterministischer endlicher Automat als 5-Bit-Binärzähler modulo 17

Binärrepräsentation der Zustandsübergänge

5-Bit-Gray-Code ändert pro Zustandsübergang lediglich 1 Bit (bis auf letzten)

Zählwert	b_1	b_2	b_3	b_4	b_5	b'_1	b'_2	b'_3	b'_4	b'_5
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	1	0	0	0	1	1
3	0	0	0	1	1	0	0	0	1	0
4	0	0	0	1	0	0	0	1	1	0
5	0	0	1	1	0	0	0	1	1	1
6	0	0	1	1	1	0	0	1	0	1
7	0	0	1	0	1	0	0	1	0	0
8	0	0	1	0	0	0	1	1	0	0
9	0	1	1	0	0	0	1	1	0	1
10	0	1	1	0	1	0	1	1	1	1
11	0	1	1	1	1	0	1	1	1	0
12	0	1	1	1	0	0	1	0	1	0
13	0	1	0	1	0	0	1	0	1	1
14	0	1	0	1	1	0	1	0	0	1
15	0	1	0	0	1	0	1	0	0	0
16	0	1	0	0	0	1	1	0	0	0
17	1	1	0	0	0	0	0	0	0	0

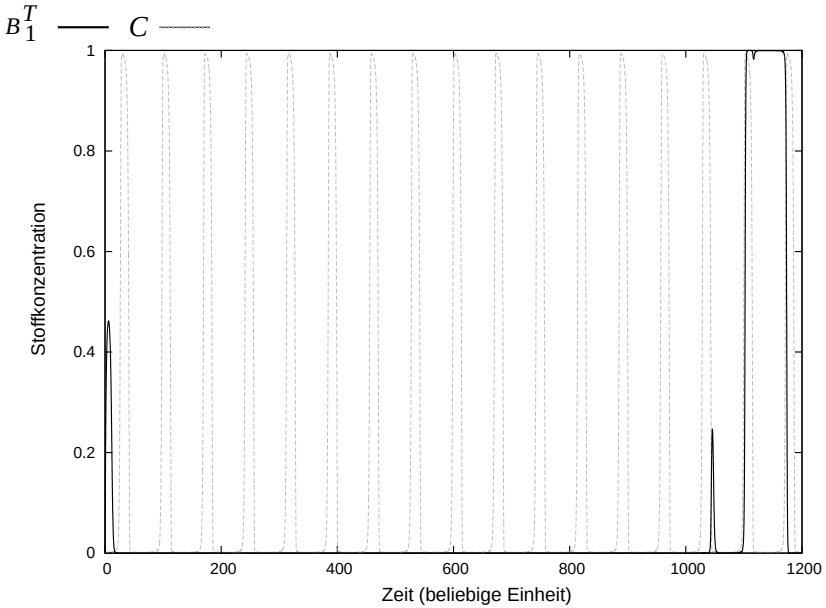
Binärzähler mod 17 – Funktionen Zustandsübergänge

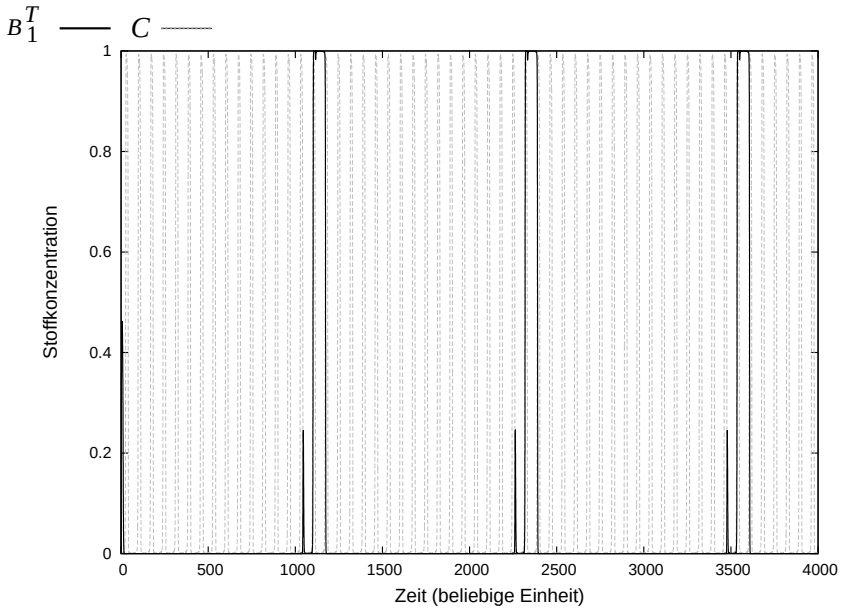


$$
\begin{aligned}
b'_1 &= \bar{b}_1 b_2 \bar{b}_3 \bar{b}_4 \bar{b}_5 \\
b'_2 &= \bar{b}_1 b_2 \vee \bar{b}_1 b_2 \bar{b}_3 \bar{b}_4 \bar{b}_5 \\
b'_3 &= \bar{b}_1 \bar{b}_2 b_3 \vee \bar{b}_1 b_3 \bar{b}_4 \vee \\
&\quad \bar{b}_1 b_2 b_3 b_5 \vee \bar{b}_1 \bar{b}_2 b_4 \bar{b}_5 \\
b'_4 &= \bar{b}_1 b_4 \bar{b}_5 \vee \bar{b}_1 b_2 b_3 b_5 \vee \\
&\quad \bar{b}_1 \bar{b}_2 \bar{b}_3 b_5 \vee \bar{b}_1 \bar{b}_2 \bar{b}_3 b_4 \vee \\
&\quad \bar{b}_1 b_2 b_3 b_4 \\
b'_5 &= \bar{b}_1 \bar{b}_2 \bar{b}_3 \bar{b}_4 \vee \bar{b}_1 \bar{b}_2 b_3 b_4 \vee \\
&\quad \bar{b}_1 b_2 b_3 \bar{b}_4 \vee \bar{b}_1 b_2 \bar{b}_3 b_4
\end{aligned}
$$

- *5-Bit-Gray-Code* zur binären Kodierung der Zustände
- 1-Impuls des Taktsignals $[C]$ erhöht den Zähler um 1 (Eingabe)
- jeweils nach 17 Zählimpulsen wieder Beginn von vorn

Verhalten des chemischen 1:17-Frequenzteilers





- Ausgabesignal: Konzentrationsverlauf von B_1^T korrespondierend zum Zustandsbit b_1, numerisch simuliert mit COPASI
- Komponenten: Taktgenerator (Brusselator und binärer Signalseparator), 76 Logikgatter mit je zwei Eingängen, 5 NOR-basierte RS-Flip-Flops
- insgesamt 145 *chemische Spezien* und 416 *Einzelreaktionen*

T. Hinze, B. Schell, M. Schumann, C. Bodenstein. Maintenance of Chronobiological Information by P System Mediated Assembly of Control Units for Oscillatory Waveforms and Frequency.
Lecture Notes in Computer Science **7762**:208-227, 2013

Rucksackproblem

NP-vollständiges Problem

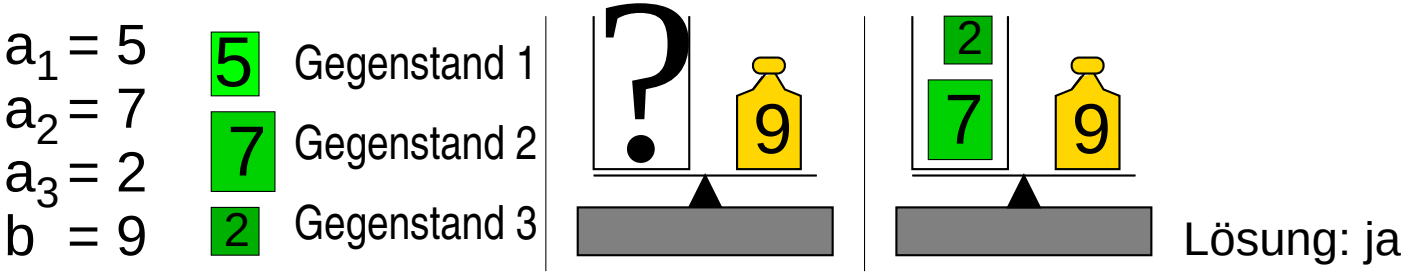
Definition

gegeben: n natürliche Zahlen $a_1, \ldots, a_n$ sowie eine nat. Zahl b
gesucht: Gibt es eine Teilmenge $I \subseteq \{1, \ldots, n\}$ mit $\sum_{i \in I} a_i = b$?

Veranschaulichung

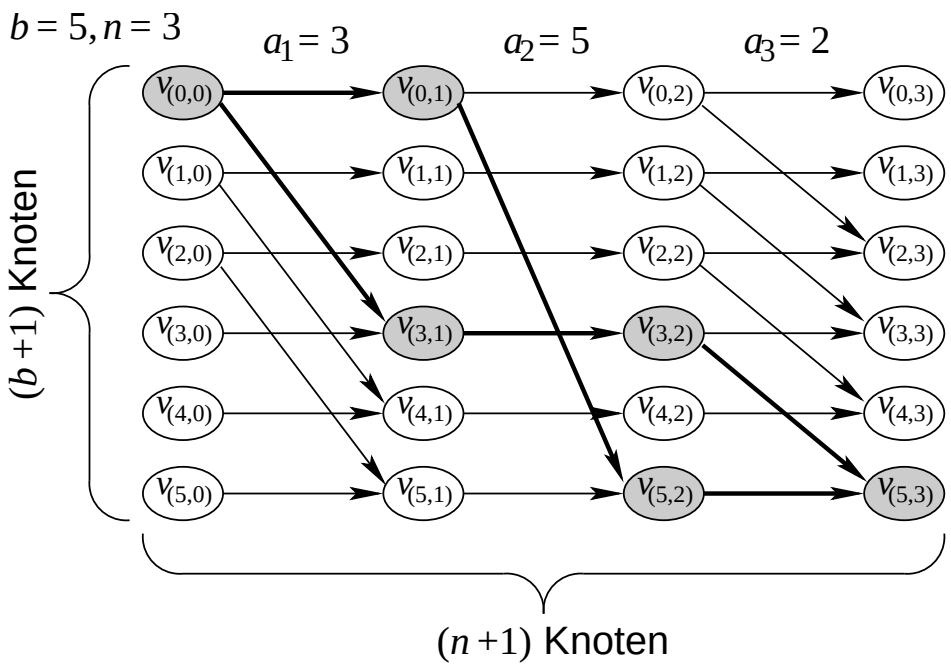
a_1 bis a_n: Gewichte der Gegenstände 1 bis n.
Gibt es eine Packmöglichkeit mit einer Auswahl aus diesen Gegenständen, so dass das Referenzgewicht b entsteht?

Beispiel



Lösungsidee: Darstellung als Graph für DEA

$$
\begin{aligned}
\mathcal{G} &= (V, E) \text{ mit } E \subset V \times V \qquad \text{endlicher gerichteter Graph} \\
V &= \{v_{(i,k)} \mid \forall i = 0, \ldots, b \; \forall k = 0, \ldots, n\} \\
E &= \{\left(v_{(i,k)}, v_{(i,k+1)}\right) \mid \forall i = 0, \ldots, b \; \forall k = 0, \ldots, n-1\} \; \cup \\
&\quad \{\left(v_{(i,k)}, v_{(i+a_i,k+1)}\right) \mid \forall i = 0, \ldots, b \; . \; (i + a_i \leq b) \; \forall k = 0, \ldots, n-1\}
\end{aligned}
$$

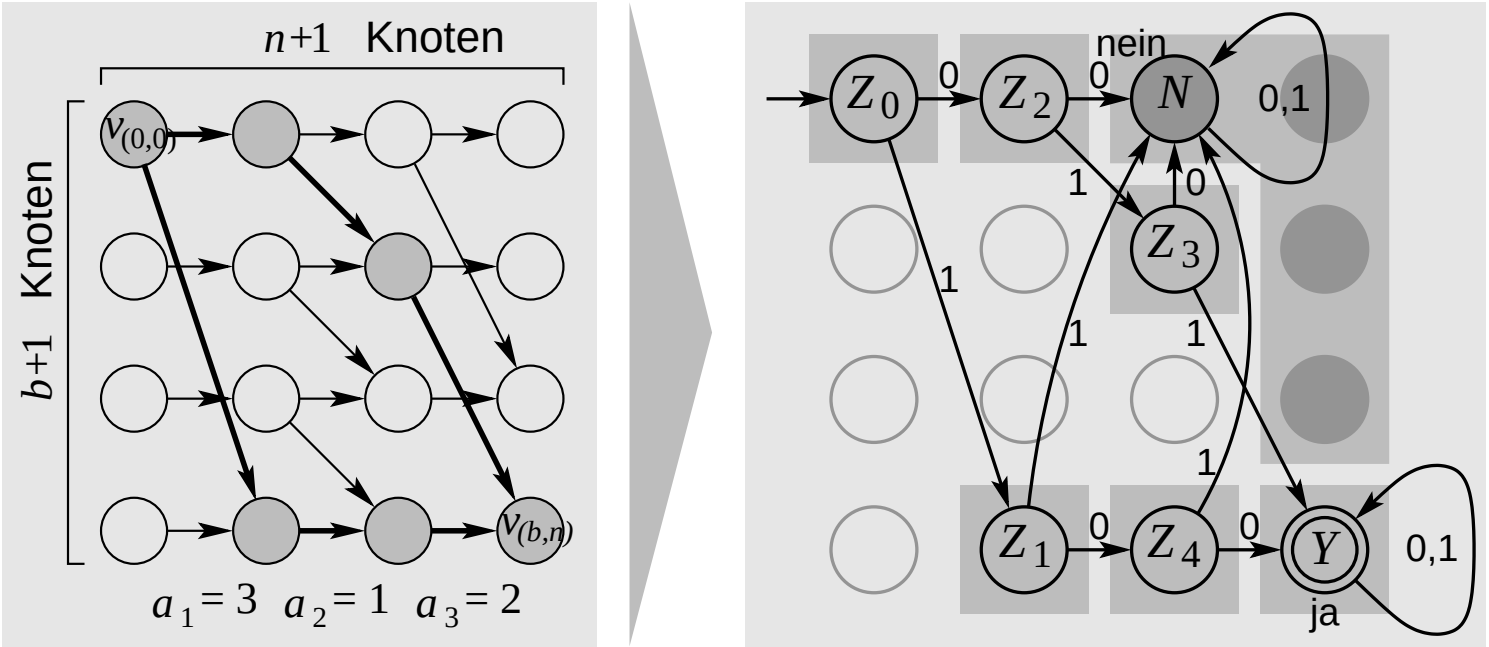


Gibt es in $\mathcal{G}$ einen Pfad von $v_{(0,0)}$ nach $v_{(b,n)}$? Falls ja, besitzt das korrespondierende Rucksackproblem die Lösung „ja“, anderenfalls „nein“.

E. Baum, D. Boneh. Running dynamic programming algorithms on a DNA Computer. *DIMACS* **44**:77-86, 1999

Darstellung Probleminstanz als endlicher Automat

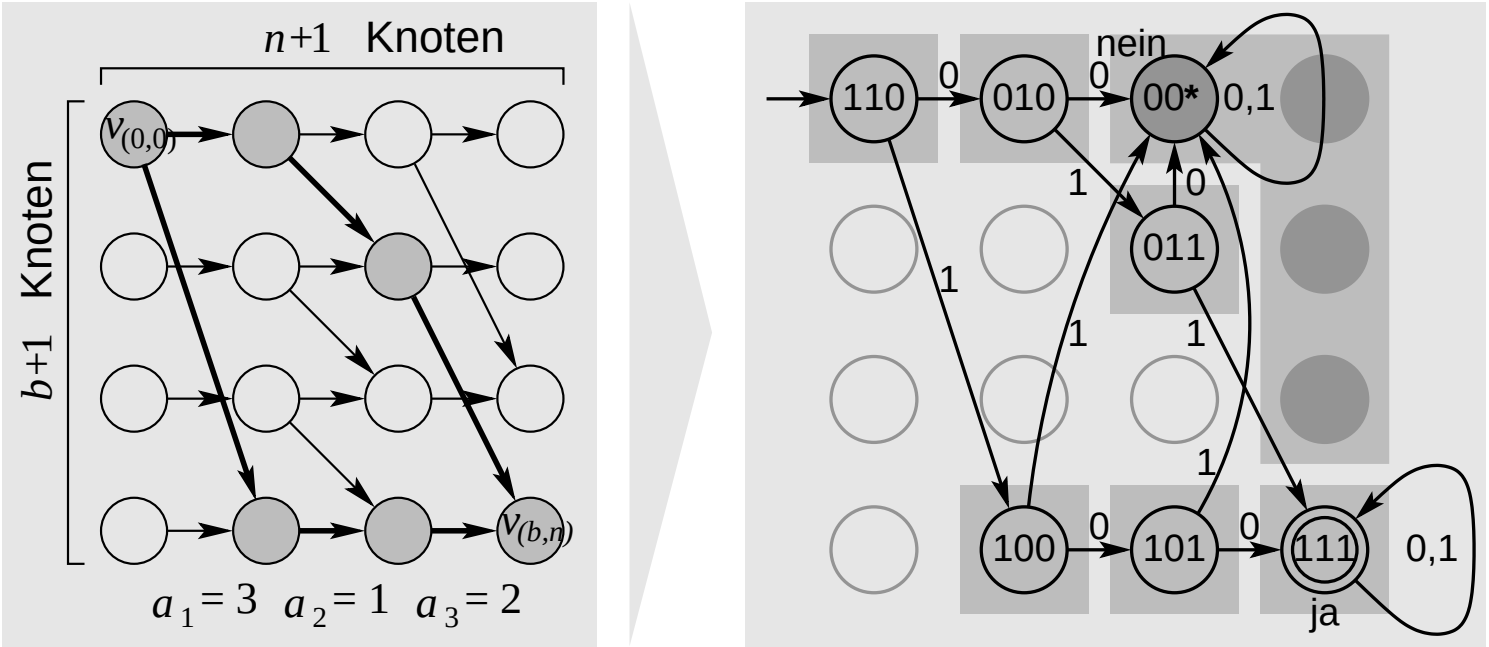
Beispielinstanz: $n = 3, a_1 = 3, a_2 = 1, a_3 = 2, b = 3$



- Jede Probleminstanz führt zu einem eigenen DEA
- unerreichbare Zustände eliminiert
- alle Zustände, die die Lösung „nein“ verkörpern, zusammengefasst
- Alle bitweise eingelesenen Bitketten der Länge $n = 3$, die in den Finalzustand münden, bilden die akzeptierte Sprache und stehen für die gesuchten Rucksackbelegungen (Lösung „ja“)

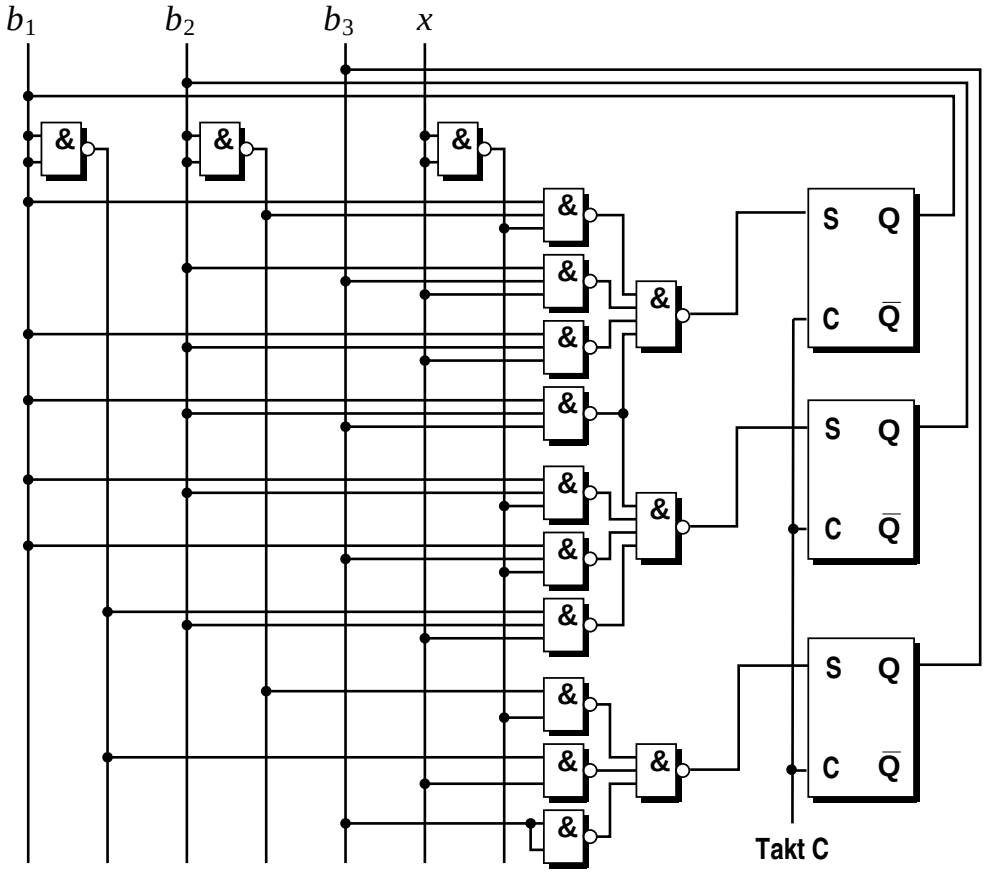
Binärkodierung der Zustände des DEA

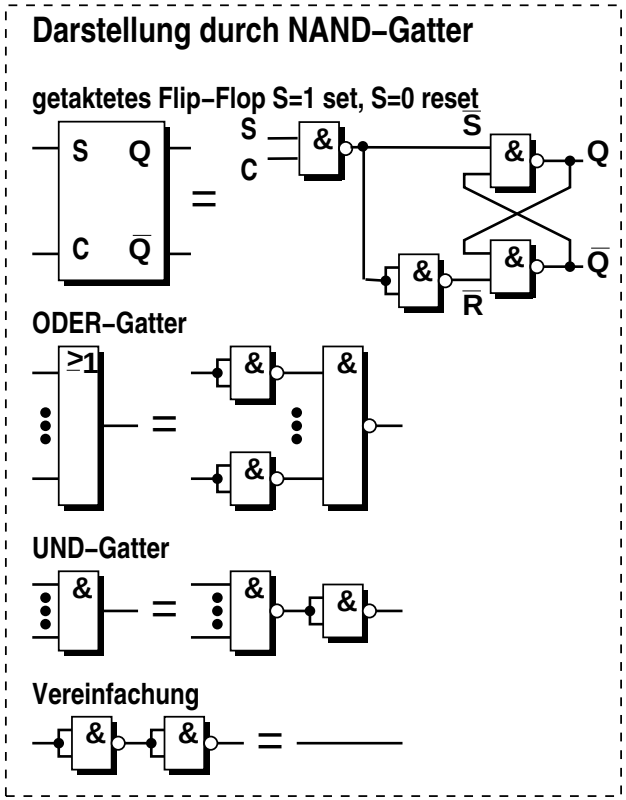
Beispielinstanz: $n = 3, a_1 = 3, a_2 = 1, a_3 = 2, b = 3$

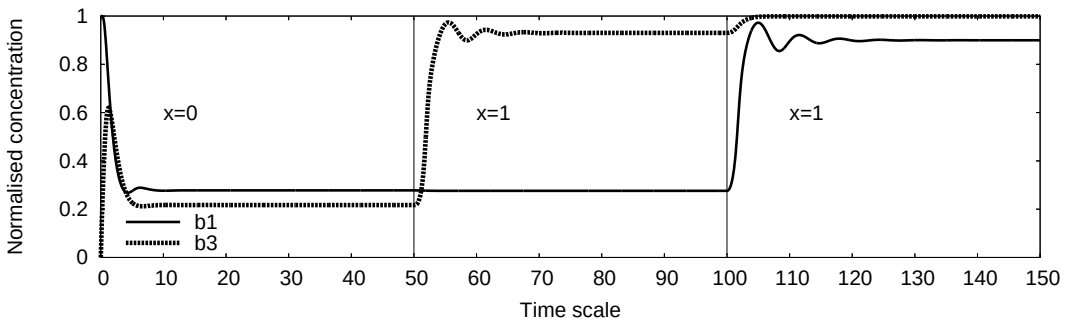


- Bei 7 Zuständen sind 3 Bit pro Zustand notwendig
- Jeder Zustandsübergang sollte möglichst wenige Bits der zustandskodierenden Bitketten ändern, idealerweise nur eines.
- dadurch Schaltlogik effizient, da wenig Gatter benötigt und geringe Anzahl hintereinandergeschalteter Gatter
- 3-Bit-Gray-Code herangezogen und Bitketten bestmöglich verteilt

Gatterschaltung und chemische Simulation







b_1	b_2	b_3	x	b'_1	b'_2	b'_3
1	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	1	1	1	1

Finalzustand erreicht

Chemische NEAs

- Die getaktet arbeitenden chemischen Modelle deterministischer endlicher Automaten (DEAs) folgen eng dem Vorbild sequentieller elektronischer Implementierungen, aber mittels Signalproteinen mit mehreren Aktivierungszuständen.

Chemische NEAs

- Die getaktet arbeitenden chemischen Modelle deterministischer endlicher Automaten (DEAs) folgen eng dem Vorbild sequentieller elektronischer Implementierungen, aber mittels Signalproteinen mit mehreren Aktivierungszuständen.
- Durch die Simulation lässt sich leicht nachvollziehen, *welche* Folge von Eingabesymbolen zu einem Finalzustand führt. Praktisch heißt das, dass damit eine konkrete Problemlösung sichtbar wird.

Chemische NEAs

- Die getaktet arbeitenden chemischen Modelle deterministischer endlicher Automaten (DEAs) folgen eng dem Vorbild sequentieller elektronischer Implementierungen, aber mittels Signalproteinen mit mehreren Aktivierungszuständen.
- Durch die Simulation lässt sich leicht nachvollziehen, *welche* Folge von Eingabesymbolen zu einem Finalzustand führt. Praktisch heißt das, dass damit eine konkrete Problemlösung sichtbar wird.
- Reicht es aus zu wissen, *ob* der Finalzustand durch irgendeine zulässige Eingabesymbolfolge erreicht werden kann, so lässt sich das chemische Modell vereinfachen und auch auf NEAs anwenden. Jeder Zustandsübergang $q_i \xrightarrow{a_k} q_j$ wird dann einfach als Reaktion $q_i + a_k \longrightarrow q_j$ geschrieben und die Anfangskonzentrationen *aller* symbolkodierenden Spezies auf Werte > 0 gesetzt. Ein Takt ist nicht mehr notwendig.

Chemische Digitalcomputer

Registermaschine als frei programmierbares Modell

Registermaschine (RAM)

Ein befehlsorientiertes Berechnungsmodell, Turing-vollständig

- Definition der Komponenten

$RAM = (R, L, P, I_1)$

- I_1: Sprungmarke Startbefehl
- P: Programm (endlicher Befehlssatz)
- L: endl. Menge von Sprungmarken $L = \{I_1, \ldots, I_c\}$
- R: endl. Menge von Registern $R = \{R_1, ..., R_m\}$, $R_k \in \mathbb{N}$

Registermaschine (RAM)

Ein befehlsorientiertes Berechnungsmodell, Turing-vollständig

- Definition der Komponenten
  $RAM = (R, L, P, I_1)$
  - I_1: Sprungmarke Startbefehl
  - P: Programm (endlicher Befehlssatz)
  - L: endl. Menge von Sprungmarken $L = \{I_1, \ldots, I_c\}$
  - R: endl. Menge von Registern $R = \{R_1, \ldots, R_m\}$, $R_k \in \mathbb{N}$
- verfügbare Befehle
  - I_i : `INC` R_k I_j Register R_k inkrementieren, gehe zu I_j
  - I_i : `DEC` R_k I_j Register R_k dekrementieren, gehe zu I_j
  - I_i : `IFZ` R_k I_j I_p Falls $R_k = 0$ gehe zu I_j sonst zu I_p
  - I_i : `HALT` Programmende

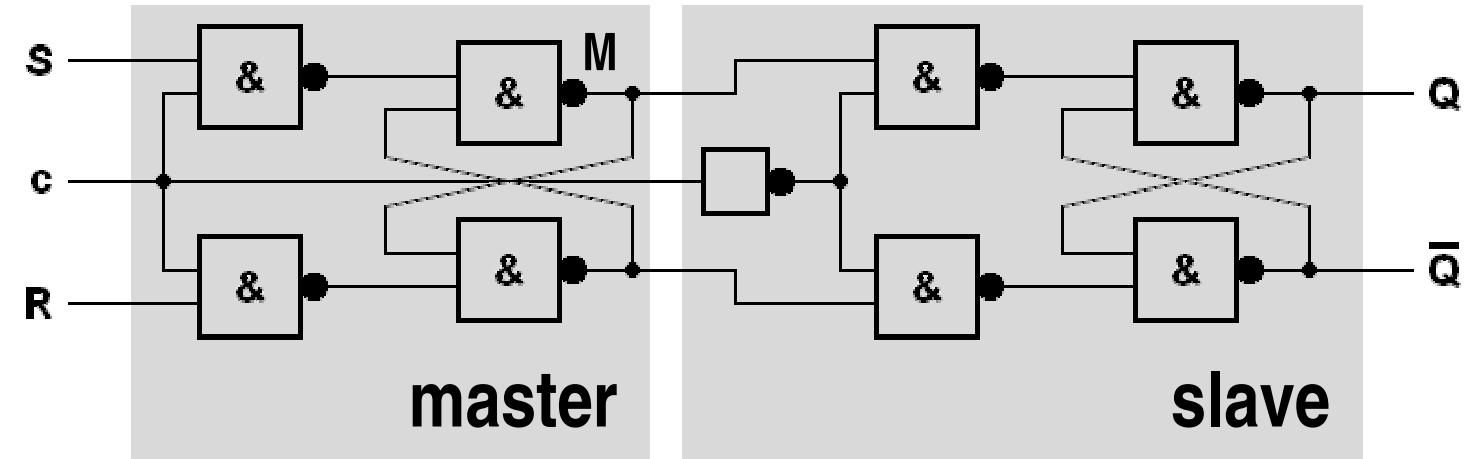
Registermaschine (RAM)

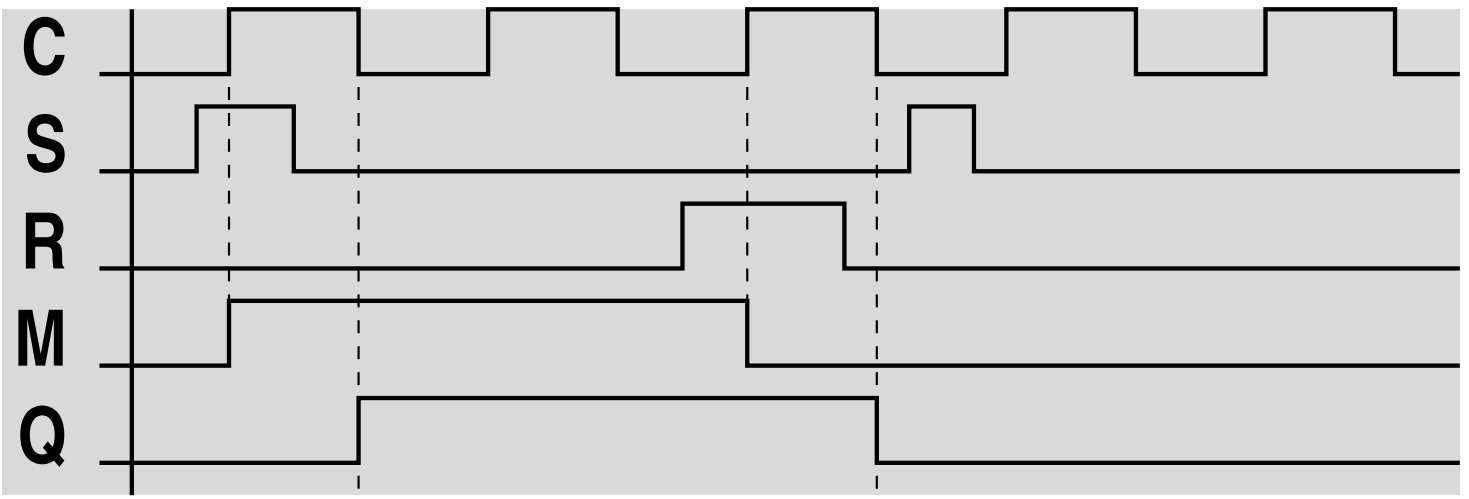
Ein befehlsorientiertes Berechnungsmodell, Turing-vollständig

- Definition der Komponenten
  $RAM = (R, L, P, I_1)$
  - I_1: Sprungmarke Startbefehl
  - P: Programm (endlicher Befehlssatz)
  - L: endl. Menge von Sprungmarken $L = \{I_1, \ldots, I_c\}$
  - R: endl. Menge von Registern $R = \{R_1, ..., R_m\}$, $R_k \in \mathbb{N}$
- verfügbare Befehle
  - I_i : `INC` R_k I_j Register R_k inkrementieren, gehe zu I_j
  - I_i : `DEC` R_k I_j Register R_k dekrementieren, gehe zu I_j
  - I_i : `IFZ` R_k I_j I_p Falls $R_k = 0$ gehe zu I_j sonst zu I_p
  - I_i : `HALT` Programmende
- Festlegungen
  - jedes Register speichert eine natürliche Zahl
  - bei Start alle Register mit Eingabewerten initialisiert
  - dediziertes Ausgaberegister festgelegt
  - deterministische Arbeitsweise

Master-Slave Flip-Flop (MSFF)

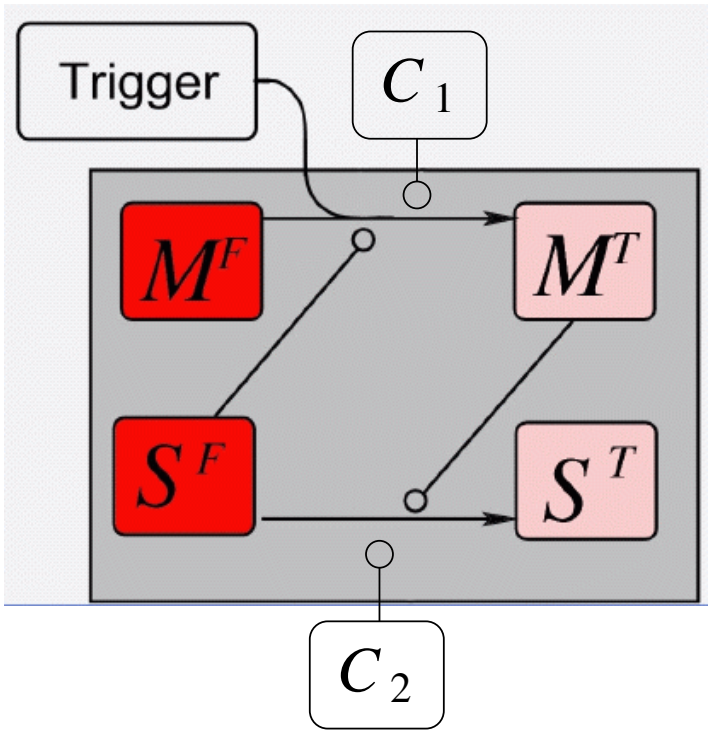
zuverlässiger 1-bit-Speicher, gut untersucht





Chemische MSFF Implementierung

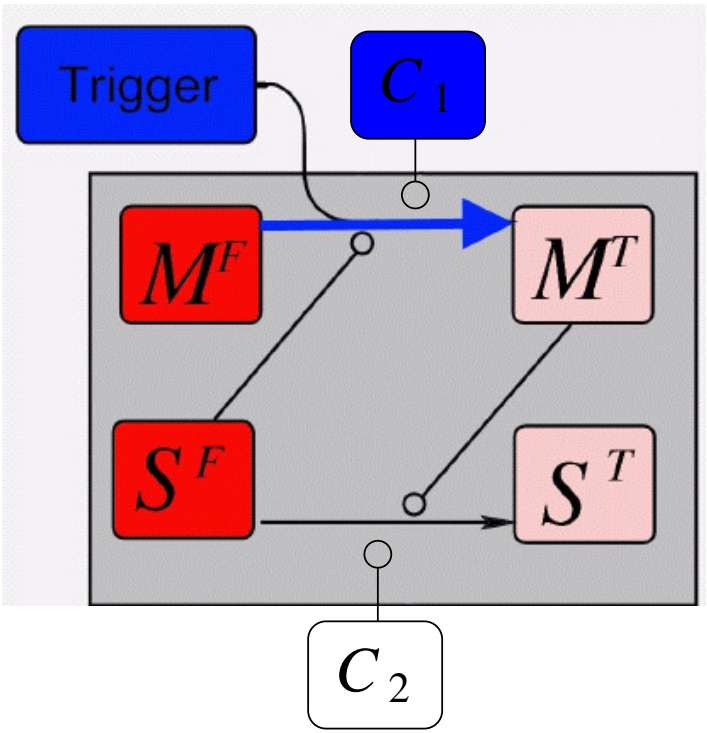
zweistufiges Schalten von **FALSE** auf **TRUE** mittels Trigger-Spezien und versetzten Taktsignalen C_1 und C_2



Spezies M^F, M^T: Bitwert im Master-Teil
Spezies S^F, S^T: Bitwert im Slave-Teil

Chemische MSFF Implementierung

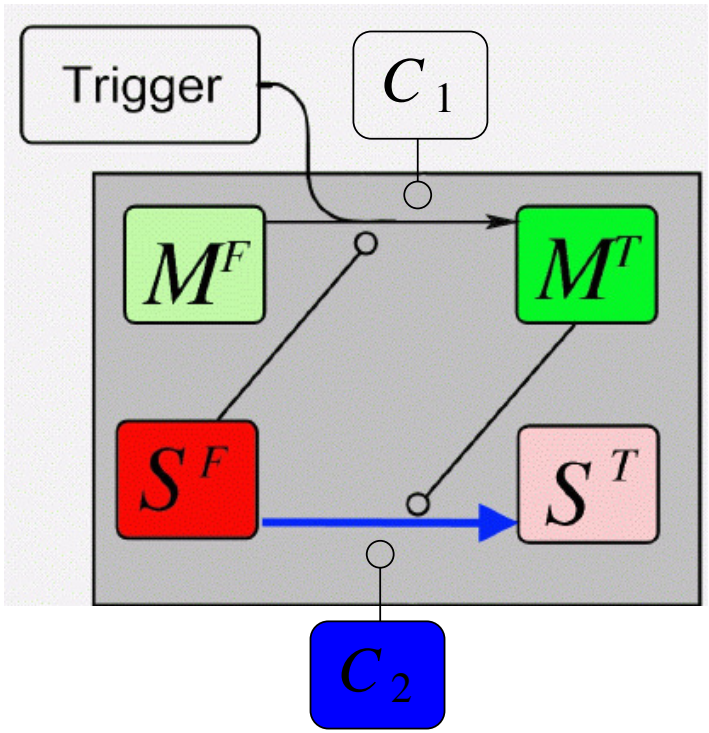
zweistufiges Schalten von **FALSE** auf **TRUE** mittels Trigger-Spezien und versetzten Taktsignalen C_1 und C_2



Spezies M^F, M^T: Bitwert im Master-Teil
Spezies S^F, S^T: Bitwert im Slave-Teil

Chemische MSFF Implementierung

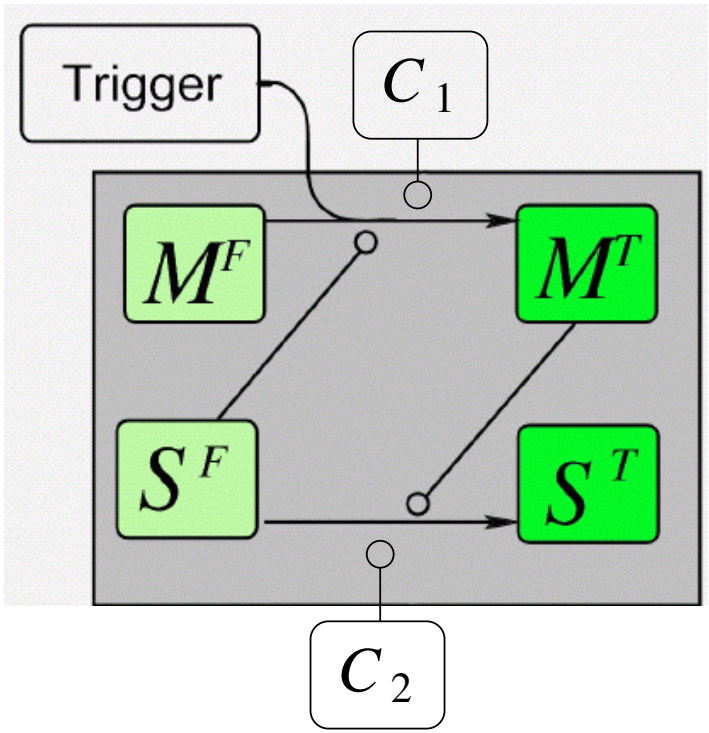
zweistufiges Schalten von **FALSE** auf **TRUE** mittels Trigger-Spezien und versetzten Taktsignalen C_1 und C_2



Spezies M^F, M^T: Bitwert im Master-Teil
Spezies S^F, S^T: Bitwert im Slave-Teil

Chemische MSFF Implementierung

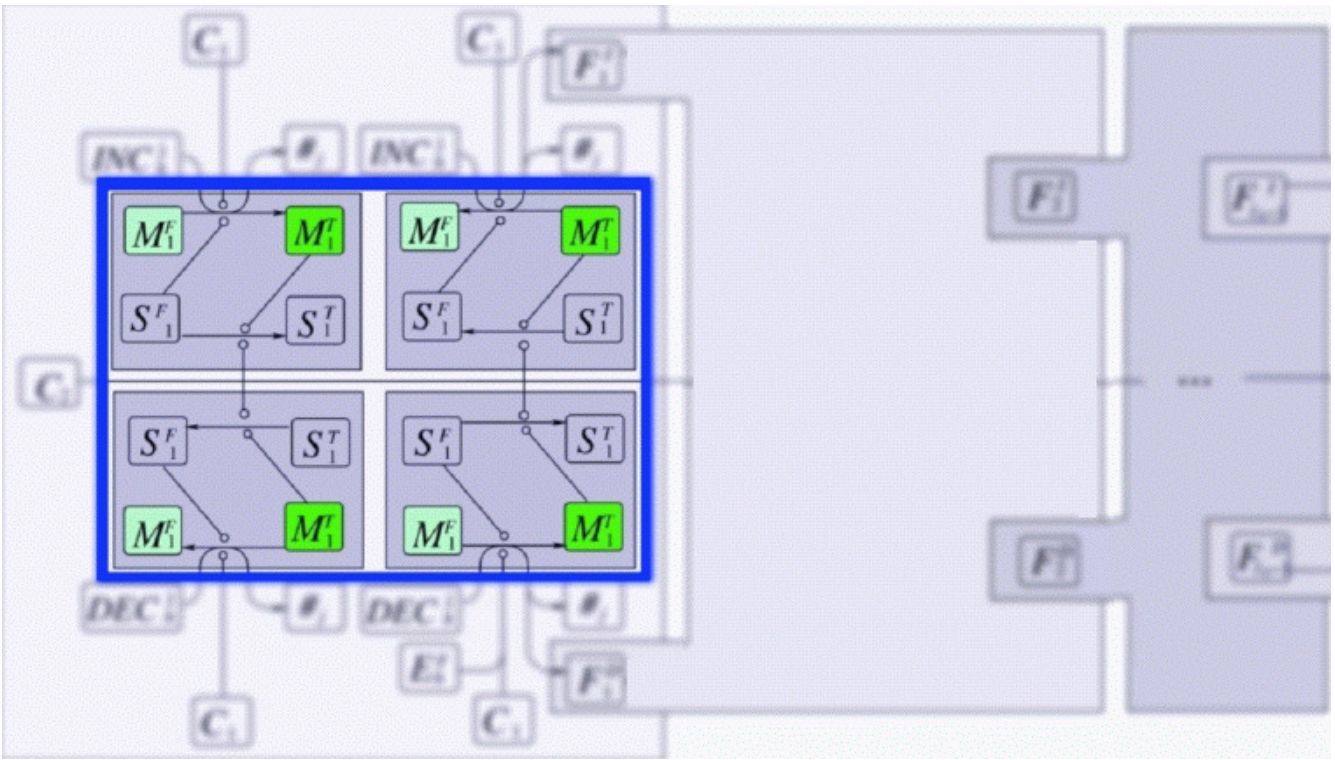
zweistufiges Schalten von **FALSE** auf **TRUE** mittels Trigger-Spezien und versetzten Taktsignalen C_1 und C_2



Spezies M^F, M^T: Bitwert im Master-Teil
Spezies S^F, S^T: Bitwert im Slave-Teil

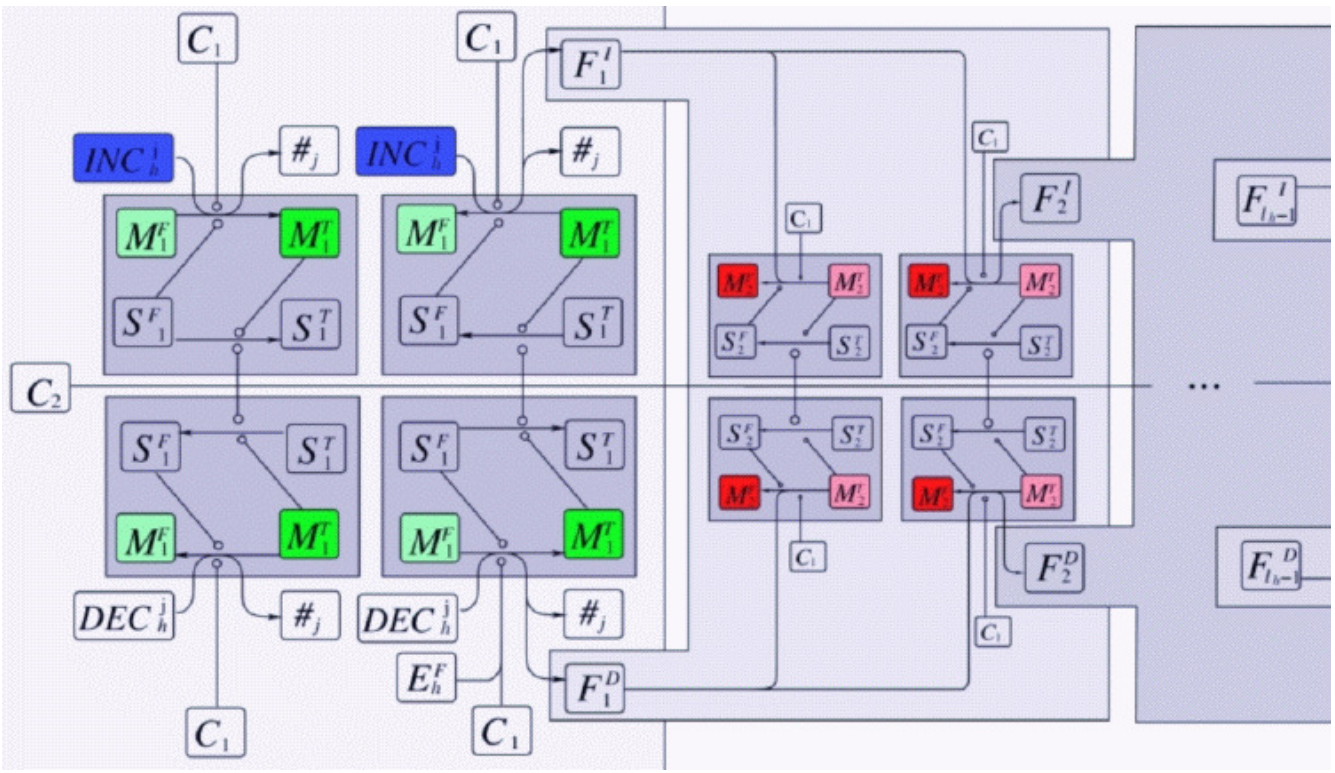
Vom MSFF zum Register

- vier Netzwerkmotive (alle Schaltszenarien) bilden MSFF
- Verkettung von MSFFs zu Registern beliebiger Bitlänge
- Annahme für Universalität: MSFF als selbstreplizierbare modulare Einheit



Vom MSFF zum Register

- vier Netzwerkmotive (alle Schaltszenarien) bilden MSFF
- Verkettung von MSFFs zu Registern beliebiger Bitlänge
- Annahme für Universalität: MSFF als selbstreplizierbare modulare Einheit



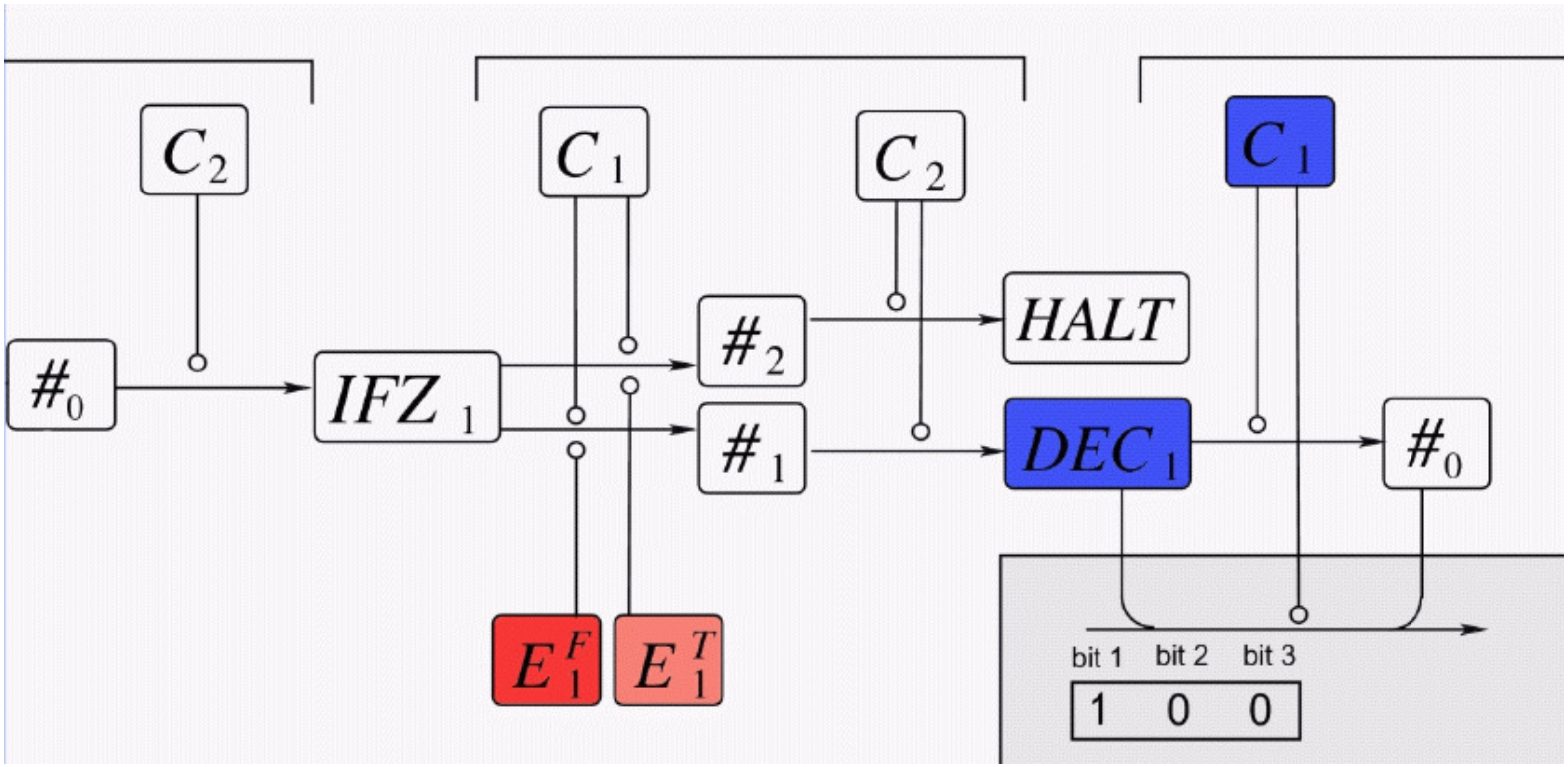
Chemische Programmsteuerung

einfaches Beispiel für sequentiellen Befehlsfluss:

$\#_0 : \text{IFZ } R_1 \ \#_2 \ \#_1$

$\#_1 : \text{DEC } R_1 \ \#_0$

$\#_2 : \text{HALT}$



T. Hinze, R. Fassler, T. Lenser, P. Dittrich. Register Machine Computations on Binary Numbers by Oscillating and Catalytic Chemical Reactions Modelled using Mass-Action Kinetics.

International Journal of Foundations of Computer Science **20(3)**:411-426, 2009

Chemische Programmsteuerung

Transformationsschema

Befehl	Chemische Reaktionen
$\#_i$: INC R_h $\#_j$	$\#_i + C_2 \xrightarrow{k_p} INC_h^j + C_2$ $INC_h^j + C_1 \xrightarrow{k_b} \#_j + C_1$
$\#_i$: DEC R_h $\#_j$	$\#_i + C_2 \xrightarrow{k_p} DEC_h^j + C_2$ $DEC_h^j + C_1 \xrightarrow{k_b} \#_j + C_1$
$\#_i$: IFZ R_h $\#_j$ $\#_q$	$\#_i + C_2 \xrightarrow{k_p} IFZ_h^{j,q} + C_2$ $IFZ_h^{j,q} + E_h^T + C_1 \xrightarrow{k_s} \#_j + E_h^T + C_1$ $IFZ_h^{j,q} + E_h^F + C_1 \xrightarrow{k_s} \#_q + E_h^F + C_1$
$\#_i$: HALT	$\#_i + C_2 \xrightarrow{k_p} HALT + C_2$

C_1, C_2: Spezien, die Taktsignale bereitstellen

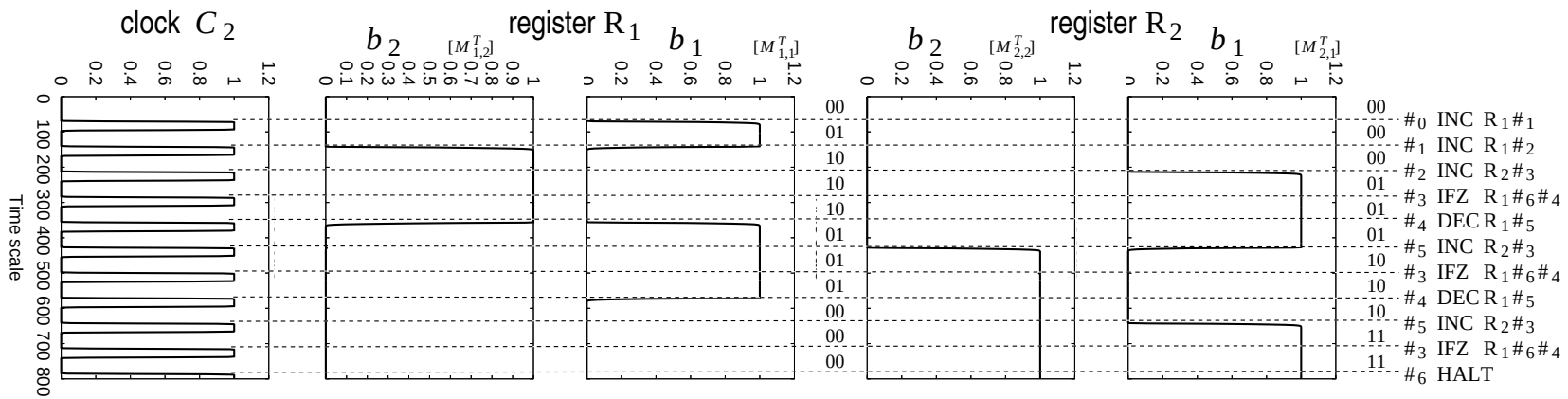
T. Hinze, R. Fassler, T. Lenser, P. Dittrich. Register Machine Computations on Binary Numbers by Oscillating and Catalytic Chemical Reactions Modelled using Mass-Action Kinetics.
International Journal of Foundations of Computer Science **20(3)**:411-426, 2009

Beispiel: Addition von Binärzahlen „2 + 1“

- Initialisierung der Register R_1 und R_2 mit Summanden
- $\mathrm{R}_2 := \mathrm{R}_2 + \mathrm{R}_1;\ \mathrm{R}_1 := 0$
- R_2 schrittweise erhöht, bis R_1 auf 0 abgebaut ist

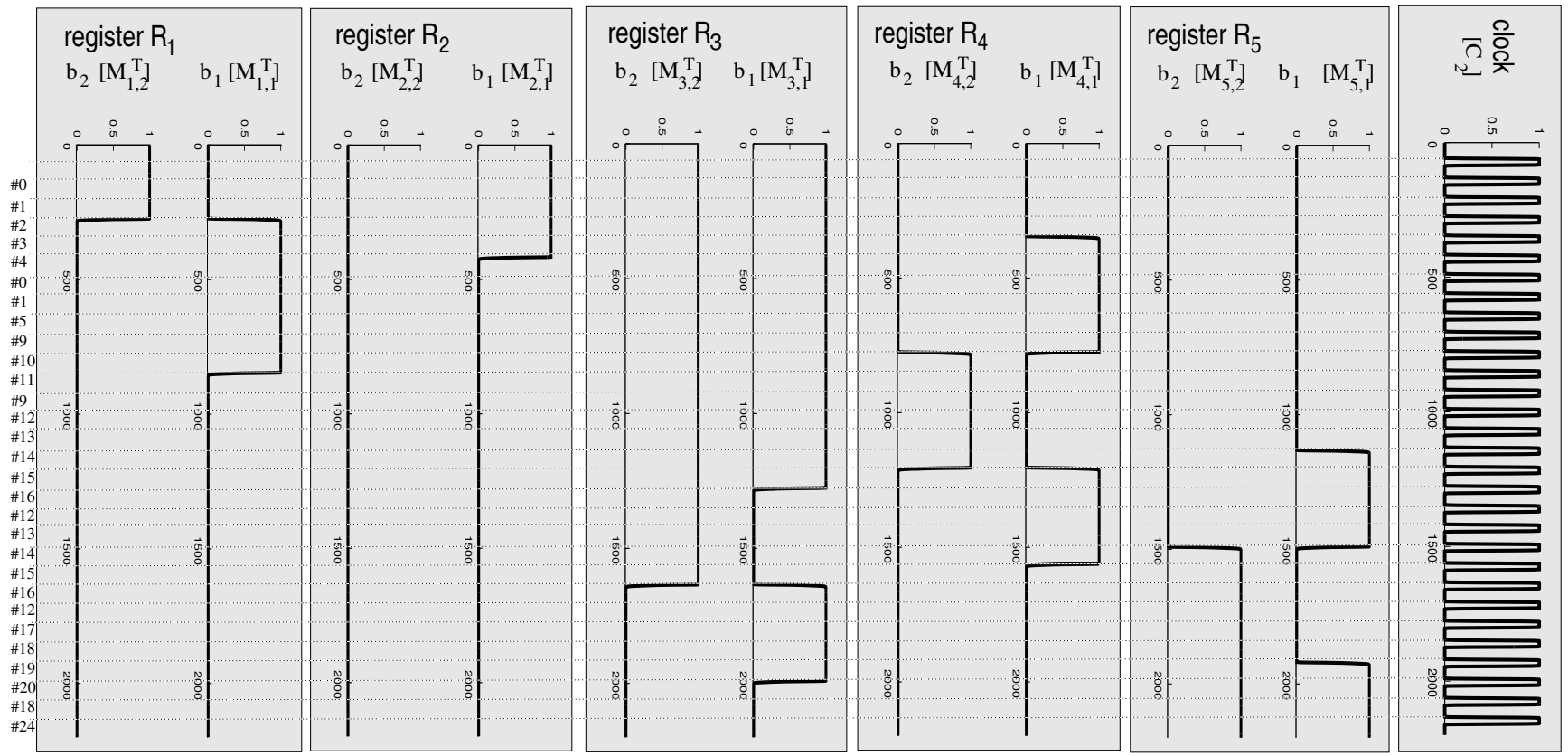
$M = (\{\mathrm{R}_1, \mathrm{R}_2\}, \{\#_0, \ldots, \#_6\}, P, \#_0)$ mit folgendem Programm P:

$\#_0$:	INC	R_1	$\#_1$		.....Initialisiere $\mathrm{R}_1 := 2$
$\#_1$:	INC	R_1	$\#_2$		
$\#_2$:	INC	R_2	$\#_3$		.....Initialisiere $\mathrm{R}_2 := 1$
$\#_3$:	IFZ	R_1	$\#_6$	$\#_4$	.....Additionsschleife
$\#_4$:	DEC	R_1	$\#_5$		
$\#_5$:	INC	R_2	$\#_3$		
$\#_6$:	HALT				



Beispiel: Maximum von drei Zahlen „$\max(2, 1, 3)$“

- $R_5 := \max(R_1, R_2, R_3)$
- Idee: $R_4 := \max(R_1, R_2)$; $R_5 := \max(R_4, R_3)$
- gesamtes Netzwerk: 142 Spezien und 223 Reaktionen



T. Hinze, R. Fassler, T. Lenser, N. Matsumaru, P. Dittrich. Event-Driven Metamorphoses of P Systems. *Lecture Notes in Computer Science* **5391**:231-245, 2009

Fazit

- Durch Emulation von Registermaschinen ist der konstruktive Nachweis erbracht, dass chemische Reaktionsnetzwerke die *Berechnungsstärke von Turingmaschinen* besitzen und somit *frei programmierbar* sind.

Fazit

- Durch Emulation von Registermaschinen ist der konstruktive Nachweis erbracht, dass chemische Reaktionsnetzwerke die *Berechnungsstärke von Turingmaschinen* besitzen und somit *frei programmierbar* sind.
- Ähnlich wie das *Eingabe-Ausgabe-Band* der Turingmaschine bei Bedarf verlängert wird, vergrößert man hier falls notwendig die *Register* um weitere Spezien und die verbindenden Reaktionen.

Fazit

- Durch Emulation von Registermaschinen ist der konstruktive Nachweis erbracht, dass chemische Reaktionsnetzwerke die *Berechnungsstärke von Turingmaschinen* besitzen und somit *frei programmierbar* sind.
- Ähnlich wie das *Eingabe-Ausgabe-Band* der Turingmaschine bei Bedarf verlängert wird, vergrößert man hier falls notwendig die *Register* um weitere Spezien und die verbindenden Reaktionen.
- Theoretisch könnte man alle imperativ notierten *Algorithmen* in ein chemisches Modell *abbilden*, aber das wäre aus praktischer Sicht nicht sinnvoll.

Fazit

- Durch Emulation von Registermaschinen ist der konstruktive Nachweis erbracht, dass chemische Reaktionsnetzwerke die *Berechnungsstärke von Turingmaschinen* besitzen und somit *frei programmierbar* sind.
- Ähnlich wie das *Eingabe-Ausgabe-Band* der Turingmaschine bei Bedarf verlängert wird, vergrößert man hier falls notwendig die *Register* um weitere Spezien und die verbindenden Reaktionen.
- Theoretisch könnte man alle imperativ notierten *Algorithmen* in ein chemisches Modell *abbilden*, aber das wäre aus praktischer Sicht nicht sinnvoll.
- Von der Möglichkeit massiver Parallelverarbeitung wurde hier nicht Gebrauch gemacht, aber sie lässt sich vorteilhaft einsetzen, wenn man die innere Struktur von Molekülen (z.B. DNA) geeignet nutzt, so dass Chomsky-Grammatiken nachgebildet werden.