Molecular Computing
Vorlesung Bioinformatik und Informatik
Sommersemester 2010

Thomas Hinze

LS Bioinformatik
Friedrich-Schiller-Universität Jena

thomas.hinze@uni-jena.de
http://users.minet.uni-jena.de/∼hinze

3. Modelle und Programmierspra-chen für molekulare Computer

Biologie + Informatik = Bioinformatik?

Eine (subjektive?) Beobachtung

Aus heutiger Sicht zeichnet sich ab:

Bioinformatik: **Informatik ist größtenteils Dienstleister für die Biologie**.

Computing in vitro/vivo: **Biologie ist größtenteils Dienstleister für die Informatik**.

$\Longrightarrow$ Bisher haben wir uns Experimentalaufbauten biomolekularer Hardware angesehen.

$\Longrightarrow$ Wir wollen sie jetzt mit Mitteln der (theoretischen) Informatik untersuchen und ausloten.

$\Longrightarrow$ Hierzu dienen mathematische Computingmodelle.

Was heißt und was kann „Berechnung“?

Mathematische Formalisierung des Berechenbarkeitsbegriffs

Wir haben eine **intuitive Vorstellung** davon, was eine Berechnung ist: **Algorithmisch gesteuerte Umwandlung von Eingabedaten in Ausgabedaten** bzw. **Erzeugung von Ausgabedaten**. **Aber:**

- Wo liegen die Grenzen der Berechenbarkeit?
- Welche Aufgaben der Informatik gelten als nicht berechenbar?
- Wie weit nähert sich Rechentechnik den Grenzen der Berechenbarkeit an?
- Können neuartige Computingkonzepte näher an diese Grenze vorstoßen?
- An welchen Ressourcenverbrauch sind Berechnungen gekoppelt?
- Wie sehen minimale Rechenmaschinen aus, die nachweislich frei programmierbar sind?
- Können molekulare Computer frei programmierbar sein? Falls ja, wie programmiert man sie effektiv?
- Wie kann man etablierte Algorithmen in vitro nachnutzen?

Was heißt und was kann „Berechnung“?

Mathematische Formalisierung des Berechenbarkeitsbegriffs

Wir haben eine **intuitive Vorstellung** davon, was eine Berechnung ist: **Algorithmisch gesteuerte Umwandlung von Eingabedaten in Ausgabedaten** bzw. **Erzeugung von Ausgabedaten**. **Aber:**

- Wo liegen die Grenzen der Berechenbarkeit?
- Welche Aufgaben der Informatik gelten als nicht berechenbar?
- Wie weit nähert sich Rechentechnik den Grenzen der Berechenbarkeit an?
- Können neuartige Computingkonzepte näher an diese Grenze vorstoßen?
- An welchen Ressourcenverbrauch sind Berechnungen gekoppelt?
- Wie sehen minimale Rechenmaschinen aus, die nachweislich frei programmierbar sind?
- Können molekulare Computer frei programmierbar sein? Falls ja, wie programmiert man sie effektiv?
- Wie kann man etablierte Algorithmen in vitro nachnutzen?

Was heißt und was kann „Berechnung“?

Mathematische Formalisierung des Berechenbarkeitsbegriffs

Wir haben eine **intuitive Vorstellung** davon, was eine Berechnung ist: **Algorithmisch gesteuerte Umwandlung von Eingabedaten in Ausgabedaten** bzw. **Erzeugung von Ausgabedaten**. **Aber:**

- Wo liegen die Grenzen der Berechenbarkeit?
- Welche Aufgaben der Informatik gelten als nicht berechenbar?
- Wie weit nähert sich Rechentechnik den Grenzen der Berechenbarkeit an?
- Können neuartige Computingkonzepte näher an diese Grenze vorstoßen?
- An welchen Ressourcenverbrauch sind Berechnungen gekoppelt?
- Wie sehen minimale Rechenmaschinen aus, die nachweislich frei programmierbar sind?
- Können molekulare Computer frei programmierbar sein? Falls ja, wie programmiert man sie effektiv?
- Wie kann man etablierte Algorithmen in vitro nachnutzen?

Was heißt und was kann „Berechnung“?

Mathematische Formalisierung des Berechenbarkeitsbegriffs

Wir haben eine **intuitive Vorstellung** davon, was eine Berechnung ist: **Algorithmisch gesteuerte Umwandlung von Eingabedaten in Ausgabedaten** bzw. **Erzeugung von Ausgabedaten**. **Aber:**

- Wo liegen die Grenzen der Berechenbarkeit?
- Welche Aufgaben der Informatik gelten als nicht berechenbar?
- Wie weit nähert sich Rechentechnik den Grenzen der Berechenbarkeit an?
- Können neuartige Computingkonzepte näher an diese Grenze vorstoßen?
- An welchen Ressourcenverbrauch sind Berechnungen gekoppelt?
- Wie sehen minimale Rechenmaschinen aus, die nachweislich frei programmierbar sind?
- Können molekulare Computer frei programmierbar sein? Falls ja, wie programmiert man sie effektiv?
- Wie kann man etablierte Algorithmen in vitro nachnutzen?

Wofür sind theoretische Zuarbeiten nützlich?

- Biologie/Biochemie stellt uns vielfältige Prozesse zur Verfügung, mit denen Biomoleküle geschaffen, verändert, separiert oder analysiert werden können.
- Daraus ergibt sich eine Vielzahl potentieller Prozessfolgen, die als Algorithmus bzw. Berechnung dienen können.
- Nur ein Bruchteil davon ist sinnvoll und im Sinne programmierbarer Biohardware ausbaufähig.
- Vorauswahl erfolgversprechender Prozessfolgen durch theoretische Untersuchungen/Vorarbeiten.
- Konzeptionelle Studien zur Algorithmierung, Programmierung und Kodierung erfordern mathematische Modelle, die dann schrittweise bis zur laborpraktischen Implementierung verfeinert werden.

Modellbildung – Schlüssel zur Beschreibung

Von Fallbeispielen zum Modell

Molekulares Reaktionssystem

- Beobachtung, Messung, statistische Auswertung
- gedankliche Isolierung, Abstraktion $\longrightarrow$

Mathematisches Modell

Reaktionsgleichungssystem, ggf.
+ Informationen über räumliche/zeitliche Stoffverteilung
+ submolekulare Raumstrukturen

- Nachzeichnen des Systemverhaltens (Simulation)
- Vorhersagen über erwartetes Systemverhalten unter angenommenen Bedingungen treffen (Validation/Verifikation)
- Was-wäre-wenn-Szenarien durchspielen
- Verstehen, wie/warum System funktioniert/entstanden ist (Abstraktion)
- Einordnen/Klassifizieren des Systems (Eigenschaften finden)

Modellbildung – Schlüssel zur Beschreibung

Von Fallbeispielen zum Modell

Molekulares Reaktionssystem

- Beobachtung, Messung, statistische Auswertung
- gedankliche Isolierung, Abstraktion $\longrightarrow$

Mathematisches Modell

Reaktionsgleichungssystem, ggf.
$+$ Informationen über räumliche/zeitliche Stoffverteilung
$+$ submolekulare Raumstrukturen

- Nachzeichnen des Systemverhaltens (Simulation)
- Vorhersagen über erwartetes Systemverhalten unter angenommenen Bedingungen treffen (Validation/Verifikation)
- Was-wäre-wenn-Szenarien durchspielen
- Verstehen, wie/warum System funktioniert/entstanden ist (Abstraktion)
- Einordnen/Klassifizieren des Systems (Eigenschaften finden)

Modellbildung – Schlüssel zur Beschreibung

Von Fallbeispielen zum Modell

Molekulares Reaktionssystem

- Beobachtung, Messung, statistische Auswertung
- gedankliche Isolierung, Abstraktion $\longrightarrow$

Mathematisches Modell

> Reaktionsgleichungssystem, ggf.
> $+$ Informationen über räumliche/zeitliche Stoffverteilung
> $+$ submolekulare Raumstrukturen

- Nachzeichnen des Systemverhaltens (Simulation)
- Vorhersagen über erwartetes Systemverhalten unter angenommenen Bedingungen treffen (Validation/Verifikation)
- Was-wäre-wenn-Szenarien durchspielen
- Verstehen, wie/warum System funktioniert/entstanden ist (Abstraktion)
- Einordnen/Klassifizieren des Systems (Eigenschaften finden)

Theoriebildung – Schlüssel zum Verständnis

Von Modellen zur Theorie

Wissenschaftliche Grundlagenforschung zielt auf:

- Aufdecken von Gemeinsamkeiten/Ähnlichkeiten zwischen Modellen
- Ableiten tieferliegender allgemeiner Gesetzmäßigkeiten und Zusammenhänge
- Fundierung genereller Aussagen, Begriffsbildung
- Zusammenführung mehrerer, zuvor getrennter Wissensgebiete
- Erschließung neuartiger Anwendungsfelder

$\Longrightarrow$ Das exakteste Beschreibungs- und Analysewerkzeug, auf das wir derzeit zurückgreifen können, ist die **Mathematik**.

Modelle und Programmiersprachen molek. Computer

Bindeglied zwischen universellen Berechnungsmodellen

- Turingmaschine
- Chomsky-Grammatiken (Typ 0)
- Registermaschine
- GOTO-Programme
- WHILE-Programme
- μ-rekursive Funktionen
- ungetypter λ-Kalkül
- Petri-Netze, ...

und abstrahierten Prozessfolgen *in vitro*, **beschrieben durch**

- Regelgesteuerte Termersetzung
  (DNA/RNA: dargestellt durch Zeichenketten oder Graphen)
- Abfolgen von Mengenoperationen
  (Reagenzglas: Menge von DNA/RNA-Molekülen)

$\Longrightarrow$ Top-Down-Algorithmenkonstruktion

Eigenschaften von Modellen des DNA-Computing (I)

im Hinblick auf Modellklassifikation

restriktiv

- Ein Modell des DNA-Computing ist *restriktiv*, wenn die Eingangsoperanden jeder Operation nach deren Ausführung nicht mehr zur Verfügung stehen.

multimengenbasiert

- Ein Modell des DNA-Computing ist *multimengenbasiert*, wenn im Modell abgebildete und verarbeitete Daten durch Multimengen beschrieben werden und die Operationen auf diesen Daten als Multimengenoperationen definiert sind.

Eigenschaften von Modellen des DNA-Computing (II)

im Hinblick auf Modellklassifikation

deterministisch

- Ein Modell des DNA-Computing ist *deterministisch*, wenn sich bei Ausführung jedes in diesem Modell beschreibbaren Algorithmus jede Operation einer Operationsfolge einschließlich ihrer Eingangsoperanden nur aus den zuvor abgearbeiteten Operationen dieses Algorithmus und seinen Eingangsdaten ergibt.

nichtdeterministisch

- Ein Modell des DNA-Computing ist *nichtdeterministisch*, wenn es mindestens einen in diesem Modell beschreibbaren Algorithmus gibt, bei dem mindestens eine Operation einer Operationsfolge einschließlich ihrer Eingangsoperanden nicht nur von den zuvor abgearbeiteten Operationen dieses Algorithmus und seinen Eingangsdaten abhängt.

Eigenschaften von Modellen des DNA-Computing (III)

Eine durch ein DNA-Computing-Modell definierte Programmiersprache unterliegt dem

imperativen Paradigma,

- wenn die Kontrollstruktur der in diesem Modell beschreibbaren Algorithmen durch eine Abfolge von Steueranweisungen und Wertzuweisungen vorgegeben ist.

regelbasierten Paradigma,

- wenn sich die Ausführung der damit beschreibbaren Algorithmen aus einer Abfolge von Termersetzungen basierend auf matchenden Ersetzungsregeln und ggf. zusätzlichen Constraints ergibt.

funktionalen Paradigma,

- wenn jeder in diesem Modell beschreibbare Algorithmus ausschließlich durch eine Menge von Funktionsdefinitionen gegeben ist und die Berechnung von Funktionswerten die Prinzipien Funktionsabstraktion und -applikation nutzt.

logischen Paradigma,

- wenn ein Algorithmus in Form einer Problemspezifikation durch Axiome beschrieben wird und die Abarbeitung des Algorithmus der Berechnung einer logischen Folgerung aus diesen Axiomen gemäß einer gegebenen Anfrage entspricht.

Eigenschaften von Modellen des DNA-Computing (IV)

im Hinblick auf Modellklassifikation

Multiple-Instruction-fähig

- Ein Modell des DNA-Computing ist *Multiple-Instruction-fähig*, wenn damit beschreibbare Algorithmen zeitgleich unterschiedliche Operationen im Sinne einer parallelen oder verteilten Berechnung ausführen können.

Multiple-Data-fähig

- Ein Modell des DNA-Computing ist *Multiple-Data-fähig*, wenn darin eine Operation definiert ist, die zeitgleich auf unterschiedliche DNA-basierte Daten zugreifen und diese im Sinne einer datenparallelen Berechnung verarbeiten kann.

Modelle und Programmiersprachen molek. Computer

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

Modelle und Programmiersprachen molek. Computer

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

Filtering-Modell nach Adleman, Lipton, Amos (I)

Steckbrief

- Erstveröffentlichung: Lipton 1995
- mehrere Erweiterungen (Adleman, Amos)
- Abstraktion von DNA-Strängen durch Bitketten, DNA-Operationen durch Mengenoperationen
- DNA-Library: alle Bitketten einer wählbaren Länge
- eng mit Brute-Force-Ansatz des Adleman-Experimentes verbunden („Adleman-Lipton-Paradigma“)
- keine Operationen zur Modifikation von Bitketten, aber:
- Separation nach Subsequenz in verschiedene Reagenzgläser (vgl. Auswahl-Transfer-Module von Mikroflussreaktoren!)
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- platzbeschränkt universell (Berechnungsstärke kontextsensitiver Chomsky-Grammatiken)
- Nachweis konstruktiv durch Simulation von WHILE-Programmen mit Platzbeschränkung

Filtering-Modell nach Adleman, Lipton, Amos (I)

Steckbrief

- Erstveröffentlichung: Lipton 1995
- mehrere Erweiterungen (Adleman, Amos)
- Abstraktion von DNA-Strängen durch Bitketten, DNA-Operationen durch Mengenoperationen
- DNA-Library: alle Bitketten einer wählbaren Länge
- eng mit Brute-Force-Ansatz des Adleman-Experimentes verbunden („Adleman-Lipton-Paradigma“)
- keine Operationen zur Modifikation von Bitketten, aber:
- Separation nach Subsequenz in verschiedene Reagenzgläser (vgl. Auswahl-Transfer-Module von Mikroflussreaktoren!)
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- platzbeschränkt universell (Berechnungsstärke kontextsensitiver Chomsky-Grammatiken)
- Nachweis konstruktiv durch Simulation von WHILE-Programmen mit Platzbeschränkung

Filtering-Modell nach Adleman, Lipton, Amos (I)

Steckbrief

- Erstveröffentlichung: Lipton 1995
- mehrere Erweiterungen (Adleman, Amos)
- Abstraktion von DNA-Strängen durch Bitketten, DNA-Operationen durch Mengenoperationen
- DNA-Library: alle Bitketten einer wählbaren Länge
- eng mit Brute-Force-Ansatz des Adleman-Experimentes verbunden („Adleman-Lipton-Paradigma“)
- keine Operationen zur Modifikation von Bitketten, aber:
- Separation nach Subsequenz in verschiedene Reagenzgläser (vgl. Auswahl-Transfer-Module von Mikroflussreaktoren!)
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- platzbeschränkt universell (Berechnungsstärke kontextsensitiver Chomsky-Grammatiken)
- Nachweis konstruktiv durch Simulation von WHILE-Programmen mit Platzbeschränkung

Modellbeschreibung

Filtering-Modell nach Adleman, Lipton, Amos (II)

DNA-basierte Daten

- DNA-Stränge durch Bitketten frei wählbarer, aber endlicher Länge $n \in \mathbb{N} \setminus \{0\}$ dargestellt.
- Bitkette: endliches nichtleeres Wort über dem Alphabet $\{0, 1\}$.
- Die Bitpositionen i innerhalb einer Bitkette $x_1 x_2 \ldots x_n$ links beginnend durchnummeriert.
- Keine Zuordnung zwischen den Bitwerten 0, 1 und entsprechenden DNA-Sequenzen in Primärveröffentlichungen
- Bei Laborimplementierung ist die Darstellung jedes Bits x_i durch eine korrespondierende DNA-Sequenz abhängig vom Bitwert (0 oder 1) *und* von der Position i des Bits innerhalb der Bitkette.
- Reagenzglasinhalte als Reagenzglas (test tube) bezeichnet.
- Jedes Reagenzglas t: endliche Menge von Bitketten und somit endliche formale Sprache. Es gilt: $t \subseteq \bigcup_{i=1}^{n} \{0, 1\}^i \in \mathit{FIN}$

Modellbeschreibung

Filtering-Modell nach Adleman, Lipton, Amos (III)

Modelloperationen

Seien t und u Reagenzgläser, $i, n \in \mathbb{N} \setminus \{0\}$ Parameter, $x_i \in \{0,1\}$ und $a \in \{0,1\}$ Bits sowie `true` und `false` boolesche Konstanten.

Initial set	t := $\{0,1\}^n$	erzeugt alle 2^n Bitketten der Länge n und weist sie dem Reagenzglas t zu. Für $n = 0$ ist die Operation nicht explizit spezifiziert. Sinnvoll ist in diesem Fall die Bereitstellung eines leeren Reagenzglases $t = \emptyset$.
Extract	u := `Extract`$(t, x_i = a)$	`Extract`$(t, x_i = a) :=$ $t \cap \left(\{0,1\}^{i-1} \otimes \{a\} \otimes \{0,1\}^*\right)$ dupliziert genau die Bitketten aus t in u, bei denen das i-te Bit den Wert a hat. In manchen Veröffentlichungen wird diese Operation auch als *Separate* bezeichnet.

Modellbeschreibung

Filtering-Modell nach Adleman, Lipton, Amos (IV)

Modelloperationen

Seien t, u, w Reagenzgläser, $i, n \in \mathbb{N} \setminus \{0\}$ Parameter, $x_i \in \{0, 1\}$ und $a \in \{0, 1\}$ Bits sowie `true` und `false` boolesche Konstanten.

Merge	w `:=` $t \cup u$	bildet die Vereinigung von t und u und weist das Ergebnis dem Reagenzglas w zu. Der Spezialfall w `:=` $t \cup t$ kann vereinfachend beschrieben werden durch w `:=` t.
Detect	`Detect(`t`)`	`Detect(`t`)` $= \begin{cases} \texttt{true} \text{ falls } t \neq \emptyset \\ \texttt{false} \text{ falls } t = \emptyset \end{cases}$ gibt den booleschen Wert `true` zurück, wenn sich mindestens eine Bitkette in t befindet, und sie gibt `false` zurück, wenn t leer ist.

Klasse der beschriebenen Programme

Filtering-Modell nach Adleman, Lipton, Amos (V)

Seien t, u und w Reagenzgläser, $i, j, n \in \mathbb{N}$ Parameter, $x_i \in \{0,1\}$ eine boolesche Variable und $a \in \{0,1\}$ eine Konstante. Alle Reagenzgläser sind initial leer. Induktive Definition:

(F1) Jedes Konstrukt t `:=` $\{0,1\}^n$ ist ein Programm des Filtering-Modells.

(F2) Jedes Konstrukt u `:= Extract(`$t, x_i = a$`)` ist ein Programm des Filtering-Modells.

(F3) Jedes Konstrukt w `:=` $t \cup u$ und w `:=` t ist ein Programm des Filtering-Modells.

(F4) Sind P und Q Programme des Filtering-Modells, dann auch P`;`Q .

(F5) Ist P ein Programm des Filtering-Modells, dann auch
`if (Detect(`t`) = true) then` P `end` und
`if (Detect(`t`) = false) then` P `end` .

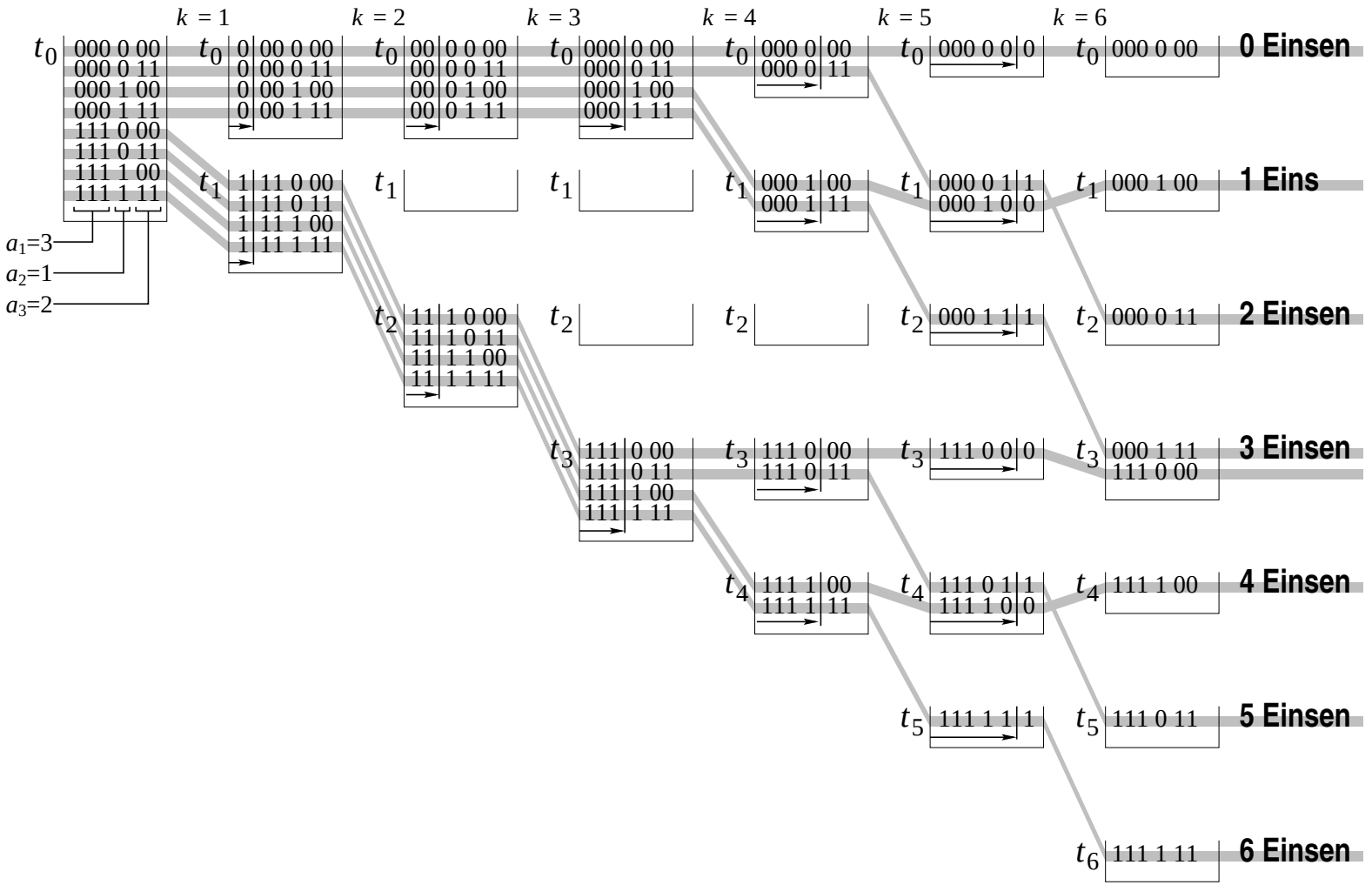
(F6) Ist P ein Programm des Filtering-Modells, dann auch
`for` i `:=` j `to` n `step 1 do` P `end` und
`for` i `:=` j `to` n `step -1 do` P `end` .

(F7) Ist P ein Programm des Filtering-Modells, dann auch
`while (Detect(`t`) = true) do` P `end` und
`while (Detect(`t`) = false) do` P `end` .

Algorithmus zur Lösung des Rucksackproblems

Filtering-Modell nach Adleman, Lipton, Amos (VI)

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



Grafik: T. Hinze, M. Sturm. Rechnen mit DNA. Oldenbourg, 2004

Algorithmus zur Lösung des Rucksackproblems

Filtering-Modell nach Adleman, Lipton, Amos (VII)

Erzeugen des initialen DNA-Pools in t_0

$t_* := \{0,1\}^{\left(\sum_{i=1}^{n} a_i\right)};$
$t_{*0} := t_*;$
$t_{*1} := t_*;$
for j := 1 to n step 1 do
  for $k := \left(\sum_{i=1}^{j-1} a_i\right) + 1$ to $\sum_{i=1}^{j} a_i$ step 1 do
    $t_{*0} :=$ Extract$(t_{*0}, x_k = 0);$
    $t_{*1} :=$ Extract$(t_{*1}, x_k = 1)$
  next k;
  $t_* := t_{*0} \cup t_{*1};$
  $t_{*0} := t_*;$
  $t_{*1} := t_*$
next j;
$t_0 := t_*;$

Algorithmus zur Lösung des Rucksackproblems

Filtering-Modell nach Adleman, Lipton, Amos (VIII)

Separationsprozess

```
for k := 1 to Σ_{i=1}^{n} a_i step 1 do
   for j := k to 1 step -1 do
      t_neue_Eins := Extract(t_{j-1}, x_k = 1);
      t_Puffer := t_j;
      t_{j-1} := Extract(t_{j-1}, x_k = 0);
      t_j := t_neue_Eins ∪ t_Puffer
   next j
next k;
if (Detect(t_b) = true) then /* Loesung ja */ end;
if (Detect(t_b) = false) then /* Loesung nein */ end
```

Zeitkomplexität

Jedes Gegenstandsgewicht a_i als Konstante aufgefasst, so dass $\sum_{i=1}^{n} a_i \in O(n)$.

$O(n^2)$ Modelloperationen (`Extract`-Befehle) $\rightarrow$ Design Mikroflussreaktor

n: maximale Bitkettenlänge, $O(2^n)$: Anzahl Bitketten

Nachweis der platzbeschränkten Universalität

Filtering-Modell nach Adleman, Lipton, Amos (IX)

Grundannahmen

- Transformation von WHILE-Programmen in Programme des Filtering-Modells (am leichtesten zu zeigen)
- Programme des Filtering-Modells auf Bitketten endlicher Länge beschränkt $\rightarrow$ Notwendigkeit einer Platzschranke (space bound) transformierbarer WHILE-Programme
- Jede in einem WHILE-Programm als Wert einer Variablen oder Konstanten vorkommende natürliche Zahl n kodiert durch das genau eine Bitkette enthaltende Reagenzglas $\{1^n\} = \{\underbrace{1 \ldots 1}_{n-\text{mal}}\}$, falls $n > 0$ und durch das leere Reagenzglas $\emptyset$, falls $n = 0$.

Nachweis der platzbeschränkten Universalität

Filtering-Modell nach Adleman, Lipton, Amos (X)

Sei R ein WHILE-Programm mit Platzbeschränkung. Die Transformation von R in ein Programm des Filtering-Modells erfolgt induktiv über den Aufbau von R.

(1) Jedes Konstrukt der Form `x_i := c` wird ersetzt durch:

```
t_i := {0,1}^c;
for j := 1 to c step 1 do
  t_i := Extract(t_i, x_j = 1)
next j
```

Das angegebene Programm stellt das Reagenzglas $t_i = \{1^c\}$ bereit.

(2) Jedes Konstrukt der Form `x_i := x_j` wird ersetzt durch:

```
t_i := t_j ∪ t_j
```

Das angegebene Programm stellt das Reagenzglas $t_i = t_j$ bereit.

Nachweis der platzbeschränkten Universalität

Filtering-Modell nach Adleman, Lipton, Amos (XI)

(3) Jedes Konstrukt der Form `x_i := x_j + 1` wird ersetzt durch:

```
for k := 1 to spacebound step 1 do
  u_k := Extract(t_j, x_k = 1)
next k;
for k := spacebound to 1 step -1 do
  if (Detect(u_k) = false) then
    t_i := {0,1}^k;
    for j := 1 to k step 1 do
      t_i := Extract(t_i, x_j = 1)
    next j
  end
next k
```

Das angegebene Programm stellt das Reagenzglas $t_i = \{1^{j+1}\}$ bereit.

Nachweis der platzbeschränkten Universalität

Filtering-Modell nach Adleman, Lipton, Amos (XII)

(4) Jedes Konstrukt der Form `x_i := x_j - 1` wird ersetzt durch:

```
for k := 1 to spacebound step 1 do
  u_k := Extract(t_j, x_k = 1);
  if (Detect(u_k) = true) then
    t_i := {0,1}^(k-1);
    for j := 1 to k - 1 step 1 do
      t_i := Extract(t_i, x_j = 1)
    next j
  end
next k
```

Das angegebene Programm stellt das Reagenzglas $t_i = \{1^{\dot{j-1}}\}$ bereit.

Nachweis der platzbeschränkten Universalität

Filtering-Modell nach Adleman, Lipton, Amos (XIII)

(5) Seien P und Q WHILE-Programme. Jedes WHILE-Programm `P;Q` wird ersetzt durch:

```
P;Q
```

(6) Sei P ein WHILE-Programm. Jedes WHILE-Programm `WHILE x_i <> 0 DO P END` wird ersetzt durch:

```
while (Detect(t_i) = true) do P end
```

Bemerkungen

- Induktive Vorschrift garantiert Transformation aller platzbeschränkten WHILE-Programme
- Effizienzverringerung bei Transformation inkrementierender und dekrementierender Anweisungen (Einzelanweisung umgewandelt in `for`-Schleife)
- unäre Kodierung der natürlichen Zahlen vereinfacht Transformation, verursacht jedoch höheren Speicherplatzbedarf (Kompromiss)

Modelle und Programmiersprachen molek. Computer

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

DNA-Pascal (I)

Steckbrief

- Erstveröffentlichung: Rooß, Wagner 1996
- Erweiterung des Filtering-Modells, enthalten in DNA-Pascal
- Intention: kompakte Beschreibung DNA-basierter Programme durch erweiterten Befehlsvorrat
- Zuordnung von Befehlsgruppen zu Komplexitätsklassen damit lösbarer Probleme
- Abstraktion von DNA-Strängen durch Bitketten und von DNA-Operationen durch Mengenoperationen beibehalten
- Operationen zur Bitkettenmodifikation, -verlängerung zugefügt.
- Dadurch Überwindung der Platzbeschränktheit, aber:
- Für manche dieser Operationen ($\cap$, Subset Test $\subseteq$) keine Laborimplementierung angegeben
- Hinzunahme der Ausgabeanweisungen `accept`, `reject`
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- universell (Nachweis konstruktiv durch uneingeschränkte Simulation von WHILE-Programmen)

DNA-Pascal (I)

Steckbrief

- Erstveröffentlichung: Rooß, Wagner 1996
- Erweiterung des Filtering-Modells, enthalten in DNA-Pascal
- Intention: kompakte Beschreibung DNA-basierter Programme durch erweiterten Befehlsvorrat
- Zuordnung von Befehlsgruppen zu Komplexitätsklassen damit lösbarer Probleme
- Abstraktion von DNA-Strängen durch Bitketten und von DNA-Operationen durch Mengenoperationen beibehalten
- Operationen zur Bitkettenmodifikation, -verlängerung zugefügt.
- Dadurch Überwindung der Platzbeschränktheit, aber:
- Für manche dieser Operationen ($\cap$, Subset Test $\subseteq$) keine Laborimplementierung angegeben
- Hinzunahme der Ausgabeanweisungen `accept`, `reject`
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- universell (Nachweis konstruktiv durch uneingeschränkte Simulation von WHILE-Programmen)

DNA-Pascal (I)

Steckbrief

- Erstveröffentlichung: Rooß, Wagner 1996
- Erweiterung des Filtering-Modells, enthalten in DNA-Pascal
- Intention: kompakte Beschreibung DNA-basierter Programme durch erweiterten Befehlsvorrat
- Zuordnung von Befehlsgruppen zu Komplexitätsklassen damit lösbarer Probleme
- Abstraktion von DNA-Strängen durch Bitketten und von DNA-Operationen durch Mengenoperationen beibehalten
- Operationen zur Bitkettenmodifikation, -verlängerung zugefügt.
- Dadurch Überwindung der Platzbeschränktheit, aber:
- Für manche dieser Operationen ($\cap$, Subset Test $\subseteq$) keine Laborimplementierung angegeben
- Hinzunahme der Ausgabeanweisungen `accept`, `reject`
- imperativ, nichtrestriktiv, nicht multimengenbasiert, deterministisch, Multiple-Data-fähig
- universell (Nachweis konstruktiv durch uneingeschränkte Simulation von WHILE-Programmen)

Modellbeschreibung

DNA-Pascal (II)

Modelloperationen

Seien T, T_1 und $T_2 \subseteq \{0,1\}^*$ Reagenzgläser (Bitkettenmengen), $m \in \mathbb{N} \setminus \{0\}$ und $n \in \mathbb{N}$ Parameter, x und z Wörter über $\{0,1\}$, $a \in \{0,1\}$ ein Bit sowie `true` und `false` boolesche Konstanten.

Union	$T := T_1 \cup T_2$		
Bit Extraction	$T := \mathrm{BX}(T_1, m, a)$	mit	$\mathrm{BX}(T_1, m, a) = T_1 \cap (\{0,1\}^{m-1} \otimes \{a\} \otimes \{0,1\}^*)$, $a \in \{0,1\}$, $m \geq 1$
Subword Extraction	$T := \mathrm{SX}(T_1, x)$	mit	$\mathrm{SX}(T_1, x) = T_1 \cap (\{0,1\}^* \otimes \{x\} \otimes \{0,1\}^*)$, $x \in \{0,1\}^*$
Initialization	$T := \mathrm{IN}(n)$	mit	$\mathrm{IN}(n) = \{0,1\}^n$, $n \geq 0$
Empty Word	$T := \{\varepsilon\}$		(Es gilt auch: $\mathrm{IN}(0) = \{\varepsilon\}$)
Right Adding	$T := T_1 \cdot a$	mit	$T_1 \cdot a = \{z \circ a \mid \forall z \in T_1\}$
Left Adding	$T := a \cdot T_1$	mit	$a \cdot T_1 = \{a \circ z \mid \forall z \in T_1\}$
Concatenation	$T := T_1 \cdot T_2$	mit	$T_1 \cdot T_2 = \{x \circ z \mid \forall x \in T_1 \; \forall z \in T_2\}$

Modellbeschreibung

DNA-Pascal (III)

Modelloperationen

Seien T, T_1 und $T_2 \subseteq \{0,1\}^*$ Reagenzgläser (Bitkettenmengen), $m \in \mathbb{N} \setminus \{0\}$ und $n \in \mathbb{N}$ Parameter, x und z Wörter über $\{0,1\}$, $a \in \{0,1\}$ ein Bit sowie `true` und `false` boolesche Konstanten.

Right Cut	$T := T_1/$	mit $T_1/ = \{z/ \mid \forall z \in T_1\}$, $(z \circ a)/ = z$ für $a \in \{0,1\}$ und $\varepsilon/ = \varepsilon$
Left Cut	$T := \backslash T_1$	mit $\backslash T_1 = \{\backslash z \mid \forall z \in T_1\}$, $\backslash(a \circ z) = z$ für $a \in \{0,1\}$ und $\backslash\varepsilon = \varepsilon$
Intersection	$T := T_1 \cap T_2$	
Subset Test	$T_1 \subseteq T_2$	mit $(T_1 \subseteq T_2) = \begin{cases} \texttt{true} \text{ falls } T_1 \subseteq T_2 \\ \texttt{false} \text{ sonst} \end{cases}$
Emptiness Test	$T = \emptyset$	mit $(T = \emptyset) = \begin{cases} \texttt{true} \text{ falls } T = \emptyset \\ \texttt{false} \text{ sonst} \end{cases}$
Membership Test	$x \in T$	mit $(x \in T) = \begin{cases} \texttt{true} \text{ falls } x \in T \\ \texttt{false} \text{ sonst} \end{cases}$

Klasse der beschriebenen Programme

DNA-Pascal (IV)

Seien T, T_1 und T_2 Bitkettenmengen, $m \in \mathbb{N} \setminus \{0\}$ und $i, j, k, n \in \mathbb{N}$ Parameter, x und z Wörter über $\{0,1\}$, $a \in \{0,1\}$ ein Bit sowie `true` und `false` boolesche Konstanten. Alle Bitkettenmengen sind initial leer. Die Syntax der Klasse der DNA-Pascal-Programme wird durch die kontextfreie Chomsky-Grammatik $G = (V, \Sigma, P, S)$ definiert mit:

$V = \{S,\ \mathit{Ops},\ \mathit{Op},\ \mathit{Test}\}$

$\Sigma = \{$`begin`, `end`, `if`, `then`, `for` i `:=` j `to` k `by 1`, `for` i `:=` j `to` k `by -1`, `while`, `do`, `;` $T := T_1 \cup T_2$, $T := \mathrm{BX}(T_1, m, a)$, $T := \mathrm{SX}(T_1, x)$, $T := \mathrm{IN}(n)$, $T := \{\varepsilon\}$, $T := T_1 \cdot a$, $T := a \cdot T_1$, $T := T_1 \cdot T_2$, $T := T_1/$, $T := \backslash T_1$, $T := T_1 \cap T_2$, `accept`, `reject`, $(T_1 \subseteq T_2)$, $(T = \emptyset)$, $(x \in T)$, `=true`, `=false`$\}$

$P = \{(S,$ `begin` Ops `end`$)$, $(\mathit{Ops}, \mathit{Op})$, $(\mathit{Ops}, \mathit{Ops}$`;`$\mathit{Op})$, $(\mathit{Ops},$ `begin` Ops `end`$)$,
$(\mathit{Op}, T := T_1 \cup T_2)$, $(\mathit{Op}, T := \mathrm{BX}(T_1, m, a))$, $(\mathit{Op}, T := \mathrm{SX}(T_1, x))$,
$(\mathit{Op}, T := \mathrm{IN}(n))$, $(\mathit{Op}, T := \{\varepsilon\})$, $(\mathit{Op}, T := T_1 \cdot a)$, $(\mathit{Op}, T := a \cdot T_1)$,
$(\mathit{Op}, T := T_1 \cdot T_2)$, $(\mathit{Op}, T := T_1/)$, $(\mathit{Op}, T := \backslash T_1)$, $(\mathit{Op}, T := T_1 \cap T_2)$,
$(\mathit{Op},$ `accept`$)$, $(\mathit{Op},$ `reject`$)$, $(\mathit{Op},$ `if` Test `= true then begin` Ops `end`$)$,
$(\mathit{Op},$ `if` Test `= false then begin` Ops `end`$)$,
$(\mathit{Op},$ `for` i `:=` j `to` k `by 1 do begin` Ops `end`$)$,
$(\mathit{Op},$ `for` i `:=` j `to` k `by -1 do begin` Ops `end`$)$,
$(\mathit{Op},$ `while` Test `= true do begin` Ops `end`$)$,
$(\mathit{Op},$ `while` Test `= false do begin` Ops `end`$)$,
$(\mathit{Test}, (T_1 \subseteq T_2))$, $(\mathit{Test}, (T = \emptyset))$, $(\mathit{Test}, (x \in T))\}$

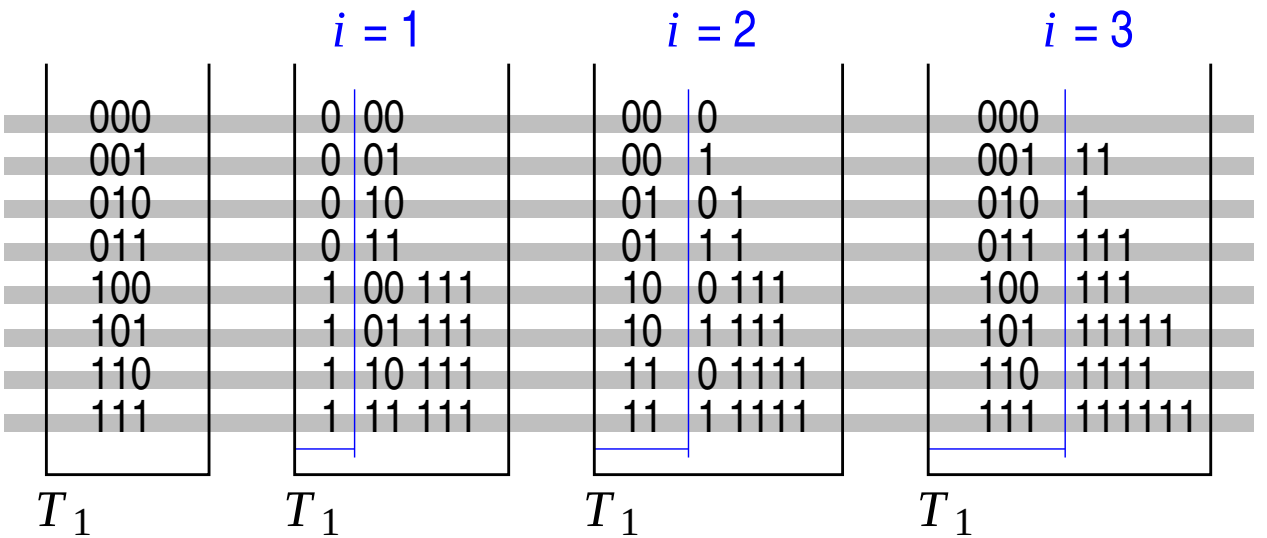
Algorithmus zur Lösung des Rucksackproblems

DNA-Pascal (V)

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

Generieren aller Rucksackgewichte

```
begin
  T1 := IN(n);
  for i := 1 to n by 1 do begin
    T2 := BX(T1, i, 1); T3 := BX(T1, i, 0); T6 := {ε};
    for j := 1 to ai by 1 do begin T6 := T6 · 1 end;
    T2 := T2 · T6; T1 := T2 ∪ T3
  end;
```



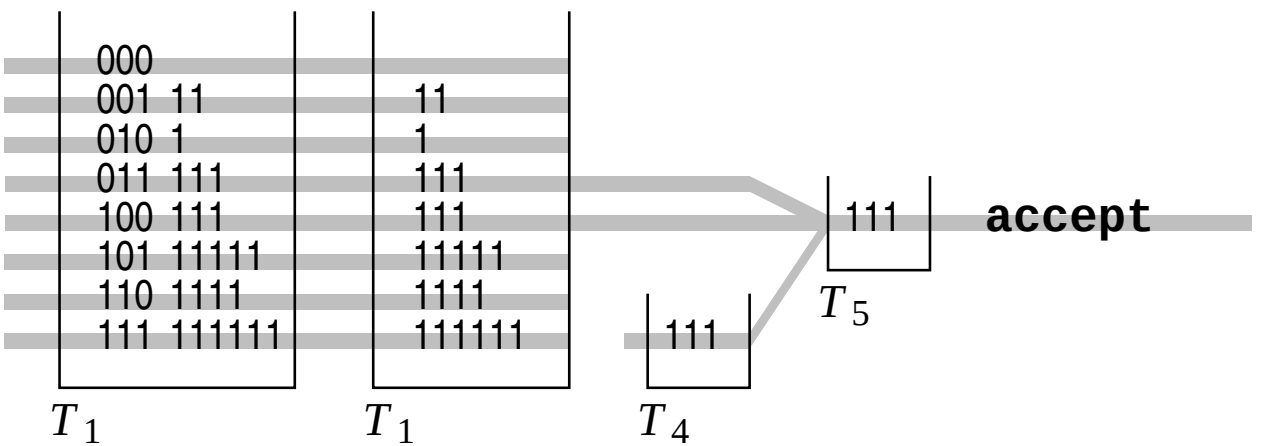
Algorithmus zur Lösung des Rucksackproblems

DNA-Pascal (VI)

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

Vergleich mit Referenzgewicht

```
  for i := 1 to n by 1 do beginT1 := \T1 end;
  T4 := {ε};
  for i := 1 to b by 1 do begin
    T4 := T4 · 1
  end;
  T5 := T1 ∩ T4;
  if (T5 = ∅) = false then begin accept end;
  if (T5 = ∅) = true then begin reject end
end
```



Universalitätsnachweis

DNA-Pascal (VII)

Grundannahmen

- Transformation von WHILE-Programmen in DNA-Pascal-Programme (am leichtesten zu zeigen)
- Bitketten können beliebig lang werden (Concatenation, Right Adding, Left Adding) und ermöglichen potentiell abzählbar unendlich großen Speicherraum (Bitkettenmengen sind reguläre Sprachen)
- Jede in einem WHILE-Programm als Wert einer Variablen oder Konstanten vorkommende natürliche Zahl n kodiert durch die genau eine Bitkette enthaltende Bitkettenmenge $\{1^n\} = \{\underbrace{1 \ldots 1}_{n-\text{mal}}\}$, falls $n > 0$ und durch $\{\varepsilon\}$, falls $n = 0$.

Universalitätsnachweis

DNA-Pascal (VIII)

Sei R ein WHILE-Programm. Die Transformation von R in ein DNA-Pascal-Programm erfolgt induktiv über den Aufbau von R.

(1) Jedes Konstrukt der Form `x_i := 0` wird ersetzt durch:
`begin` $T_i := \{\varepsilon\}$ `end`

(2) Jedes Konstrukt der Form `x_i := c` mit c $>$ 0 wird ersetzt durch:
`begin` $T_i := \mathrm{IN}(c)$; $T_i := \mathrm{SX}(T_i, 1^c)$ `end`

(3) Jedes Konstrukt der Form `x_i := x_j` wird ersetzt durch:
`begin` $T_i := T_j \cup T_j$ `end`

(4) Jedes Konstrukt der Form `x_i := x_j + 1` wird ersetzt durch:
`begin` $T_i := T_j \cup T_j$; $T_i := T_i \cdot 1$ `end`

(5) Jedes Konstrukt der Form `x_i := x_j - 1` wird ersetzt durch:
`begin` $T_i := T_j \cup T_j$; $T_i := T_i/$ `end`

(6) Seien P und Q WHILE-Programme. Jedes WHILE-Programm P`;`Q wird ersetzt durch:
`begin` P`;` Q `end`

(7) Sei P ein WHILE-Programm. Jedes WHILE-Programm `WHILE x_i <> 0 DO P END` wird ersetzt durch:
`begin while` $(\varepsilon \in T_i)$ `= false do begin` P `end end`

Modelle und Programmiersprachen molek. Computer

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

Insertion-Deletion-Systeme (I)

Steckbrief

- Erstveröffentlichung: Kari 1996 (+ Galiuschkov 81, Kari 91)
- Idee: aus linearen DNA-Strängen Abschnitte gezielt heraustrennen (*Deletion*) bzw. dort hinein einfügen (*Insertion*)
- Insertion und Deletion kontextabhängig machen (Subsequenzen an den Trennstellen bilden Kontext)
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Insertion/Deletion: wortmodifizierende Operationen
- Wortlängen nicht beschränkt und durch fortlaufendes Insertion beliebig zu vergrößern
- Laborimplementierung unter Verwendung von Restriktionsspaltung und Ligation (Daley et al. 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter Annahme minimaler Kontextlängen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Insertion-Deletion-Systeme (I)

Steckbrief

- Erstveröffentlichung: Kari 1996 (+ Galiuschkov 81, Kari 91)
- Idee: aus linearen DNA-Strängen Abschnitte gezielt heraustrennen (*Deletion*) bzw. dort hinein einfügen (*Insertion*)
- Insertion und Deletion kontextabhängig machen (Subsequenzen an den Trennstellen bilden Kontext)
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Insertion/Deletion: wortmodifizierende Operationen
- Wortlängen nicht beschränkt und durch fortlaufendes Insertion beliebig zu vergrößern
- Laborimplementierung unter Verwendung von Restriktionsspaltung und Ligation (Daley et al. 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter Annahme minimaler Kontextlängen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Insertion-Deletion-Systeme (I)

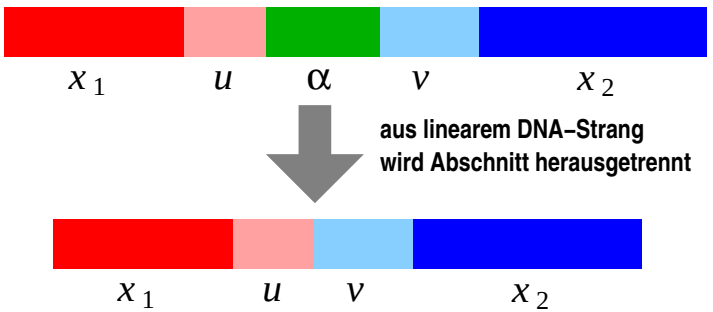
Steckbrief

- Erstveröffentlichung: Kari 1996 (+ Galiuschkov 81, Kari 91)
- Idee: aus linearen DNA-Strängen Abschnitte gezielt heraustrennen (*Deletion*) bzw. dort hinein einfügen (*Insertion*)
- Insertion und Deletion kontextabhängig machen (Subsequenzen an den Trennstellen bilden Kontext)
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Insertion/Deletion: wortmodifizierende Operationen
- Wortlängen nicht beschränkt und durch fortlaufendes Insertion beliebig zu vergrößern
- Laborimplementierung unter Verwendung von Restriktionsspaltung und Ligation (Daley et al. 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter Annahme minimaler Kontextlängen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Insertion- und Deletion-Operationen

Insertion-Deletion-Systeme (II)

Deletion



$x = x_1 u \alpha v x_2$

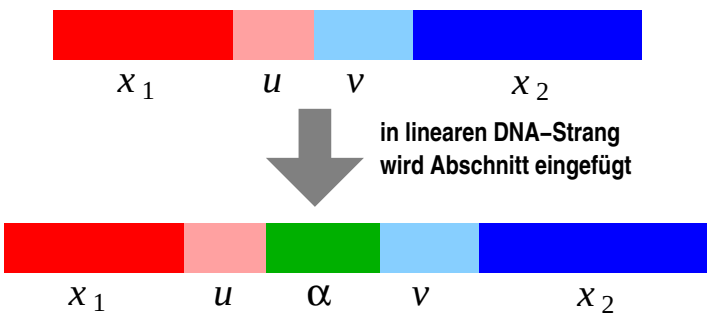
$y = x_1 u v x_2$

Ersetzungsregel $(u, \alpha/\varepsilon, v)$

u, v: Kontext

$x \vdash y$: Wort x in Wort y umgewandelt, α als „Abfall“ verworfen

Insertion



$x = x_1 u v x_2$

$y = x_1 u \alpha v x_2$

Ersetzungsregel $(u, \varepsilon/\alpha, v)$

u, v: Kontext

$x \vdash y$: Wort x in Wort y umgewandelt, α als vorhanden angesehen

Systemdefinition

Insertion-Deletion-Systeme (III)

Ein Insertion-Deletion-System γ ist ein Quadrupel $\gamma = (V, \Sigma, A, R)$ mit folgender Bedeutung der Komponenten:

- V bezeichnet ein beliebiges Alphabet,
- $\Sigma \subseteq V$ das Alphabet der Terminalsymbole und
- die endliche Sprache $A \subset V^*$ die Menge der Axiome.
- R ist eine endliche Menge von Tripeln der Form $(u, \alpha/\beta, v)$ mit $u, v \in V^*$ und $(\alpha, \beta) \in (V^+ \times \{\varepsilon\}) \cup (\{\varepsilon\} \times V^+)$. R beschreibt die Menge der Insertion-Deletion-Regeln.

Jede Regel $(u, \varepsilon/\beta, v) \in R$ wird als Insertion-Regel interpretiert, analog jede Regel $(u, \alpha/\varepsilon, v)$ als Deletion-Regel. In R ist die Menge der Insertion-Regeln und die Menge der Deletion-Regeln zusammengefasst.

Ableitungsschritt eines Insertion-Deletion-Systems

Insertion-Deletion-Systeme (IV)

Sei $\gamma = (V, \Sigma, A, R)$ ein Insertion-Deletion-System, $x, y \in V^*$.
Ein Ableitungsschritt (Regelanwendung, Produktion) von x nach y ist eine Relation $\vdash_\gamma \subset V^* \times V^*$, definiert durch: $x \vdash_\gamma y :=$

$$\{(x,y) \mid \begin{cases} x = x_1uvx_2 \wedge y = x_1u\beta vx_2 \wedge \exists x_1, x_2 \in V^*.((u, \varepsilon/\beta, v) \in R) & \text{(Ins.)} \\ x = x_1u\alpha vx_2 \wedge y = x_1uvx_2 \wedge \exists x_1, x_2 \in V^*.((u, \alpha/\varepsilon, v) \in R) & \text{(Del.)} \end{cases}\}$$

Bemerkungen

- $x \vdash_\gamma y$ drückt aus, dass x durch Anwendung genau einer Insertion-Deletion-Regel aus R zu y abgeleitet wird.
- Die Relation $\vdash_\gamma$ ist i.A. keine Funktion, d.h., zu einem x kann es mehrere y geben.
- Dies bedingt, dass Insertion-Deletion-Systeme i.A. nichtdeterministische Modelle des DNA-Computing sind.
- Die transitive Hülle von $\vdash_\gamma$ wird mit $\vdash_\gamma^+$, die reflexive transitive Hülle mit $\vdash_\gamma^*$ bezeichnet (Hintereinanderausführung von Ableitungsschritten).

Durch Insertion-Deletion-System beschriebene Sprache

Insertion-Deletion-Systeme (V)

Sei $\gamma = (V, \Sigma, A, R)$ ein Insertion-Deletion-System. Die durch γ beschriebene (generierte) Sprache $L(\gamma)$ ist definiert durch $L(\gamma) = \{w \in \Sigma^* \mid \exists x \in A \,.\, (x \vdash_\gamma^* w)\}$.

Bemerkungen

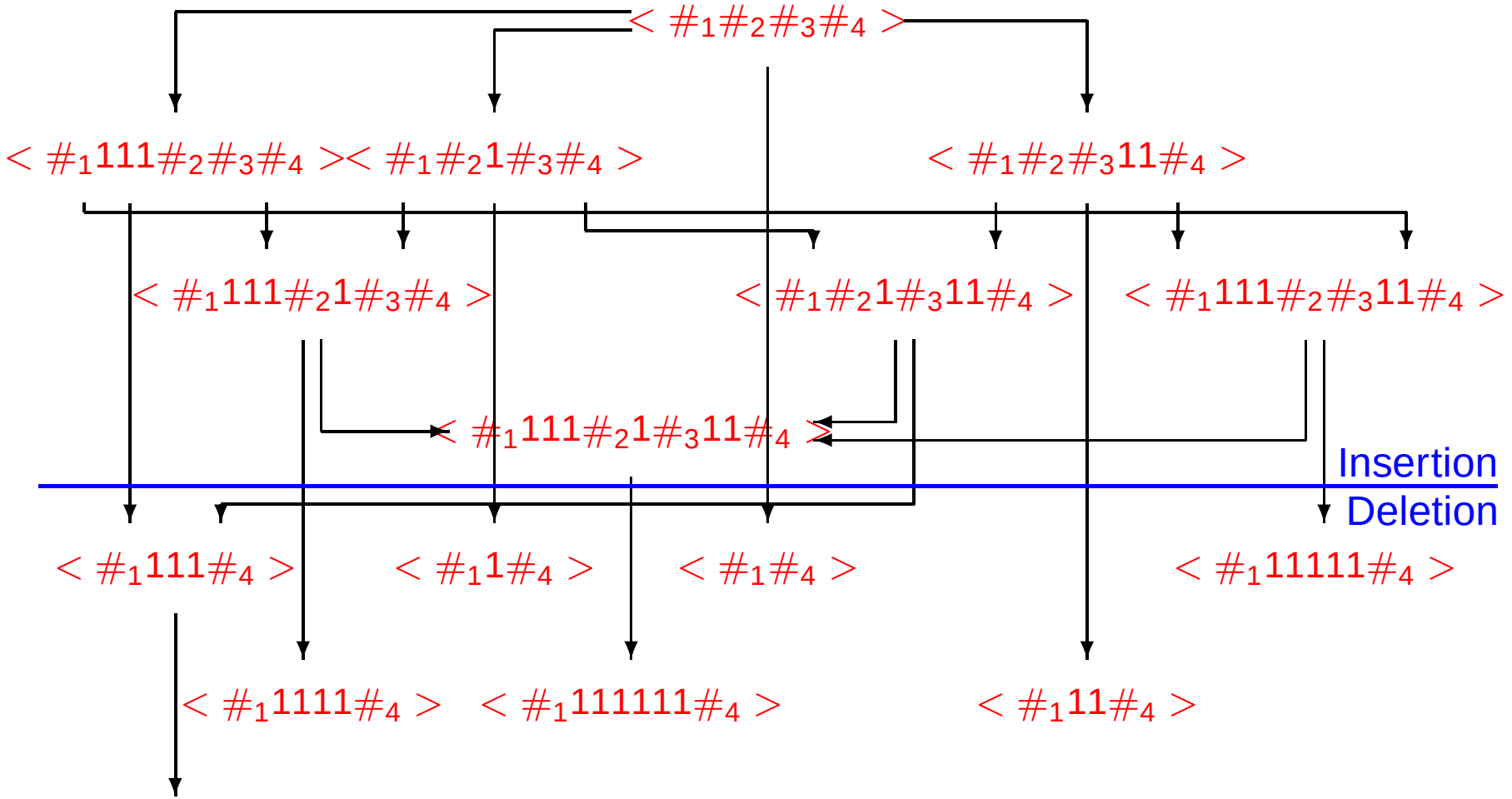
- Generierte Sprache enthält alle aus den Axiomen vom Insertion-Deletion-System hervorgebrachten Wörter, die nur aus Terminalsymbolen bestehen.
- Sprachgeneration geschieht durch Hintereinanderausführung aller matchenden Insertion- und Deletion-Regeln.
- In jedem Ableitungsschritt werden alle anwendbaren Regeln (parallel) in einem Gefäß (one pot) angwandt.

Algorithmus zur Lösung des Rucksackproblems

Insertion-Deletion-Systeme (VI)

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

Idee



Algorithmus zur Lösung des Rucksackproblems

Insertion-Deletion-Systeme (VII)

Notation des Algorithmus als Insertion-Deletion-System

$$
\begin{aligned}
\gamma &= (V, \Sigma, A, R) \\
V &= \{1, <, >, \#_1, \ldots, \#_{n+1}\} \\
\Sigma &= \{<, >\} \\
A &= \{< \#_1\#_2 \ldots \#_n\#_{n+1} >\} \\
R &= \{(\#_i, \varepsilon/1^{a_i}, \#_{i+1}) \mid \forall i = 1, \ldots, n\} \cup \\
&\quad \{(1, \#_i/\varepsilon, 1) \mid \forall i = 2, \ldots, n\} \cup \\
&\quad \{(1, \#_i/\varepsilon, \#_{i+1}) \mid \forall i = 2, \ldots, n\} \cup \\
&\quad \{(\#_i, \#_{i+1}/\varepsilon, 1) \mid \forall i = 1, \ldots, n-1\} \cup \\
&\quad \{(\#_j, \#_i/\varepsilon, \#_k) \mid \forall i = 2, \ldots, n \; \forall j = 1, \ldots, i-1 \\
&\qquad\qquad \forall k = i+1, \ldots, n+1\} \cup \\
&\quad \{(<, \#_1 1^b \#_{n+1}/\varepsilon, >)\}
\end{aligned}
$$

Es gilt: $L(\gamma) = \begin{cases} \{<>\} & \text{bei Lösung „ja“} \\ \emptyset & \text{bei Lösung „nein“} \end{cases}$

Wird nach $2n$ hintereinander ausgeführten Ableitungsschritten die Lösung „ja“ nicht erreicht, so kann sie auch später nicht mehr erreicht werden.

Universalitätsnachweis

Insertion-Deletion-Systeme (VIII)

Simulation von Chomsky-Grammatiken

Sei $G = (V, \Sigma, P, S)$ eine Chomsky-Grammatik vom Typ 0, o.B.d.A. in Kuroda-Normalform (nur Regelstrukturen $A \to BC$, $A \to a$, $A \to \varepsilon$, $AB \to CD$ zulässig). G wird überführt in ein Insertion-Deletion-System γ. Seien $X, Y, U, Z \in V$, $x \in (V \cup \Sigma)^*$ mit $\mathrm{lgth}(x) \leq 2$. Das Insertion-Deletion-System $\gamma = (V \cup \Sigma \cup \{E, K_1, K_2\}, \Sigma, \{SEE\}, R)$ besitzt die Regelmenge:

$$
\begin{aligned}
R \;=\; & \{(X, \varepsilon / K_1 x, \alpha_1\alpha_2) \mid \forall (X, x) \in P.((X \in V) \wedge (x \in (V \cup \Sigma)^*) \wedge (\mathrm{lgth}(x) \leq 2)) \\
& \quad \forall \alpha_1, \alpha_2 \in V \cup \Sigma \cup \{E\}\} \cup \\
& \{(XY, \varepsilon / K_2 UZ, \alpha_1\alpha_2) \mid \forall (XY, UZ) \in P \;.\; (X, Y, U, Z \in V) \\
& \quad \forall \alpha_1, \alpha_2 \in V \cup \Sigma \cup \{E\}\} \cup \\
& \{(\varepsilon, XK_1 / \varepsilon, \varepsilon) \mid \forall X \in V\} \cup \{(\varepsilon, XYK_2 / \varepsilon, \varepsilon) \mid \forall X, Y \in V\} \cup \\
& \{(\varepsilon, EE / \varepsilon, \varepsilon)\}
\end{aligned}
$$

d.h. Insertion rechte Seite bei $A \to BC$, $A \to a$, $A \to \varepsilon$ und Marker K_1 für spätere Deletion linker Regelseite setzen

d.h. Insertion rechte Seite bei $AB \to CD$ und Marker K_2 für spätere Deletion linker Regelseite setzen

d.h. Deletion linke Regelseite und Marker

d.h. Deletion des künstlichen Kontexts

- K_1, K_2: Marker dafür, dass eine rechte Seite einer Grammatikregel in Wortform eingefügt wurde, aber ursprüngliche linke Seite noch daraus entfernt werden muss.
- EE liefert einen künstlichen Kontext, damit unmittelbar am Ende einer Wortform ersetzt werden kann.

Modelle und Programmiersprachen molek. Computer

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

Splicing-Systeme (EH-Systeme) (I)

Steckbrief

- Erstveröffentlichung: Head 1987 (+ Păun et al. 1996)
- Idee: kreuzweise Rekombination linearen DNA-Fragmente durch Schneiden und Religieren (Splicing-Operation)
- Schnittposition kontextabhängig machen
- Arbeitsprinzip labornäher als Insertion-Deletion-Systeme
- EH-System: Extended Head System (spezielle Form eines einfachen Splicing-Systems); viele weitere Formen entwickelt
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Splicing-Operation: wortmodifizierende Operation
- Wortlängen nicht beschränkt und durch fortlaufendes Splicing beliebig zu vergrößern
- Laborimplementierung einfacher Splicing-Systeme (Laun 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter verschiedenen Annahmen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Splicing-Systeme (EH-Systeme) (I)

Steckbrief

- Erstveröffentlichung: Head 1987 (+ Păun et al. 1996)
- Idee: kreuzweise Rekombination linearen DNA-Fragmente durch Schneiden und Religieren (Splicing-Operation)
- Schnittposition kontextabhängig machen
- Arbeitsprinzip labornäher als Insertion-Deletion-Systeme
- EH-System: Extended Head System (spezielle Form eines einfachen Splicing-Systems); viele weitere Formen entwickelt
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Splicing-Operation: wortmodifizierende Operation
- Wortlängen nicht beschränkt und durch fortlaufendes Splicing beliebig zu vergrößern
- Laborimplementierung einfacher Splicing-Systeme (Laun 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter verschiedenen Annahmen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Splicing-Systeme (EH-Systeme) (I)

Steckbrief

- Erstveröffentlichung: Head 1987 (+ Păun et al. 1996)
- Idee: kreuzweise Rekombination linearen DNA-Fragmente durch Schneiden und Religieren (Splicing-Operation)
- Schnittposition kontextabhängig machen
- Arbeitsprinzip labornäher als Insertion-Deletion-Systeme
- EH-System: Extended Head System (spezielle Form eines einfachen Splicing-Systems); viele weitere Formen entwickelt
- Abstraktion von DNA-Strängen durch Wörter formaler Sprachen
- Splicing-Operation: wortmodifizierende Operation
- Wortlängen nicht beschränkt und durch fortlaufendes Splicing beliebig zu vergrößern
- Laborimplementierung einfacher Splicing-Systeme (Laun 1999)
- regelbasiert, nichtrestriktiv, nicht multimengenbasiert, nichtdeterministisch, Multiple-Data-fähig
- Systeme, mit denen formale Sprachen generiert werden können
- universell unter verschiedenen Annahmen (Nachweis konstruktiv durch uneingeschränkte Simulation von Chomsky-Grammatiken)

Lineares DNA-Splicing *in vitro*

Splicing-Systeme (EH-Systeme) (II)

Wiederholung

- gleichzeitig Restriktionsspaltung mit **zwei** Restriktionsenzymen, die so aufeinander abgestimmt sind, dass die erzeugten sticky-Enden ligationskompatibel sind (darüber hinaus gleiche Reaktionsbedingungen für beide Enzyme vorausgesetzt)
- Ligation

Beispiel

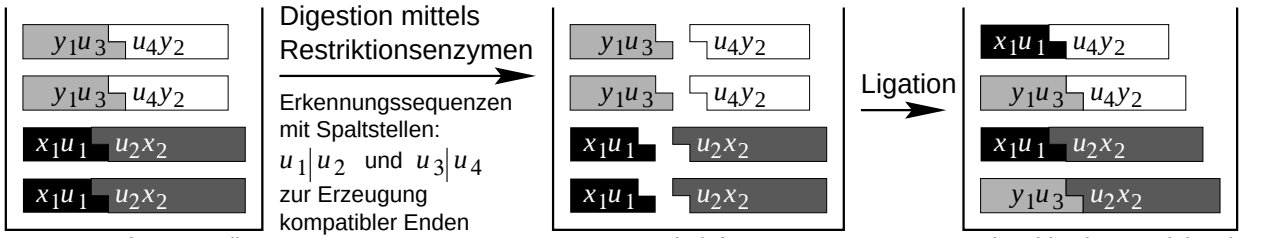
```
5′-CCCCCTCGACCCCC-3′        5′-AAAAAGCGCAAAAA-3′
3′-GGGGGAGCTGGGGG-5′  und   3′-TTTTTCGCGTTTTT-5′
```

Restriktionsenzyme: TaqI `T|CG A / A GC|T` und SciNI `G|CG C / C GC|G`

```
5′-CCCCCTCGCAAAAA-3′        5′-AAAAAGCGACCCCC-3′
3′-GGGGGAGCGTTTTT-5′  und   5′-TTTTTCGCTGGGGG-5′
```

Splicing-Operation

Splicing-Systeme (EH-Systeme) (III)



DNA-Doppelstränge, die die Wörter x und y kodieren

erzeugte Schnittfragmente Präfixe und Suffixe von x und y

Rekombinationsprodukt mit den neuen Wörtern z und w

Mathematische Beschreibung

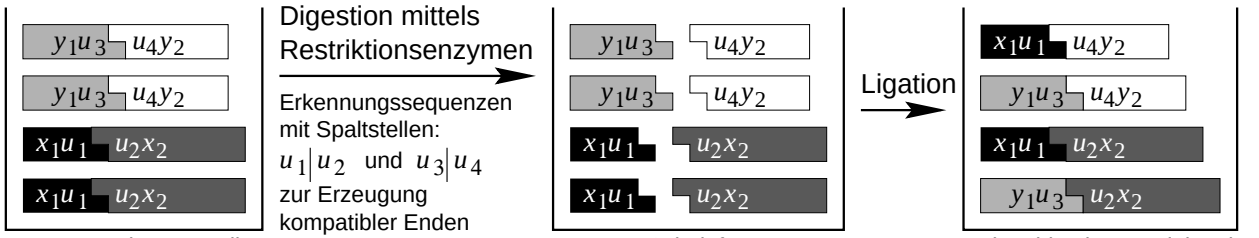
Sei V ein Alphabet ohne die Zeichen # und \$. Eine Splicing-Regel ist eine Zeichenkette $r = u_1\#u_2\$u_3\#u_4 \in V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$.

- Die Teilwörter u_1 und u_2 beschreiben Erkennungssequenz und Spaltstelle des ersten Restriktionsenzyms $(u_1|u_2)$. Analog $(u_3|u_4)$ für zweites Restriktionsenzym.
- Eine Regelmenge $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ enthält alle in einem System betrachteten Splicing-Regeln $r \in R$.
- Der Vorgang des Splicing wird beschrieben durch eine Relation $\vdash_\gamma \subseteq (V^* \times V^*) \times (V^* \times V^*)$, die die Wörter (x, y) in (z, w) umwandelt:

$$(x,y) \vdash_\gamma (z,w) = \{((x,y),(z,w)) \mid \exists x_1,x_2,y_1,y_2 \in V^* \; \exists r = u_1\#u_2\$u_3\#u_4 \in R \; \exists x = x_1u_1u_2x_2 \in V^* \; \exists y = y_1u_3u_4y_2 \in V^* \; . \; ((z = x_1u_1u_4y_2) \wedge (w = y_1u_3u_2x_2))\}$$

Splicing-Operation

Splicing-Systeme (EH-Systeme) (III)



DNA-Doppelstränge, die die Wörter x und y kodieren

erzeugte Schnittfragmente Präfixe und Suffixe von x und y

Rekombinationsprodukt mit den neuen Wörtern z und w

Mathematische Beschreibung

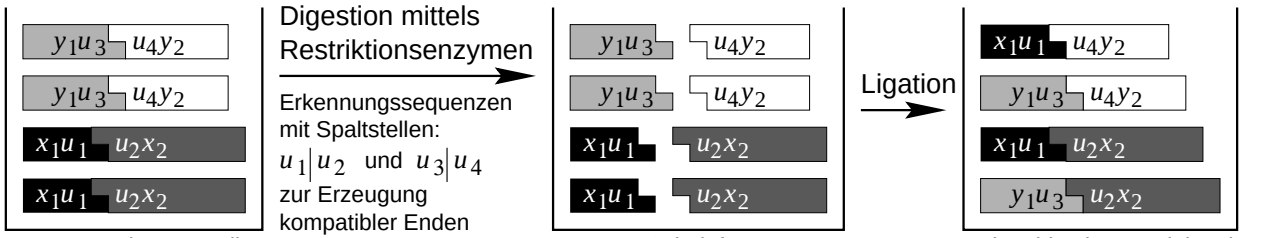
Sei V ein Alphabet ohne die Zeichen # und \$. Eine Splicing-Regel ist eine Zeichenkette $r = u_1\#u_2\$u_3\#u_4 \in V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$.

- Die Teilwörter u_1 und u_2 beschreiben Erkennungssequenz und Spaltstelle des ersten Restriktionsenzyms $(u_1|u_2)$. Analog $(u_3|u_4)$ für zweites Restriktionsenzym.
- Eine Regelmenge $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ enthält alle in einem System betrachteten Splicing-Regeln $r \in R$.
- Der Vorgang des Splicing wird beschrieben durch eine Relation $\vdash_\gamma \subseteq (V^* \times V^*) \times (V^* \times V^*)$, die die Wörter (x, y) in (z, w) umwandelt:

$$(x,y) \vdash_\gamma (z,w) = \{((x,y),(z,w)) \mid \exists x_1,x_2,y_1,y_2 \in V^* \; \exists r = u_1\#u_2\$u_3\#u_4 \in R \; \exists x = x_1u_1u_2x_2 \in V^* \; \exists y = y_1u_3u_4y_2 \in V^* \; . \; ((z = x_1u_1u_4y_2) \wedge (w = y_1u_3u_2x_2))\}$$

Splicing-Operation

Splicing-Systeme (EH-Systeme) (III)



DNA-Doppelstränge, die die Wörter x und y kodieren

erzeugte Schnittfragmente Präfixe und Suffixe von x und y

Rekombinationsprodukt mit den neuen Wörtern z und w

Mathematische Beschreibung

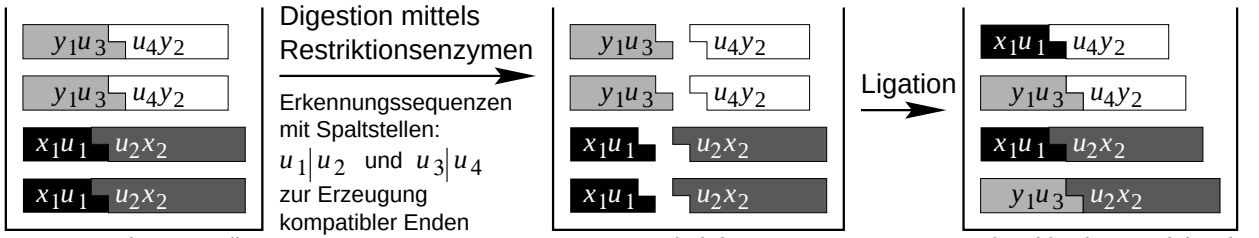
Sei V ein Alphabet ohne die Zeichen # und \$. Eine Splicing-Regel ist eine Zeichenkette $r = u_1\#u_2\$u_3\#u_4 \in V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$.

- Die Teilwörter u_1 und u_2 beschreiben Erkennungssequenz und Spaltstelle des ersten Restriktionsenzyms $(u_1|u_2)$. Analog $(u_3|u_4)$ für zweites Restriktionsenzym.
- Eine Regelmenge $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ enthält alle in einem System betrachteten Splicing-Regeln $r \in R$.
- Der Vorgang des Splicing wird beschrieben durch eine Relation $\vdash_\gamma \subseteq (V^* \times V^*) \times (V^* \times V^*)$, die die Wörter (x, y) in (z, w) umwandelt:

$$(x,y) \vdash_\gamma (z,w) = \{((x,y),(z,w)) \mid \exists x_1,x_2,y_1,y_2 \in V^* \; \exists r = u_1\#u_2\$u_3\#u_4 \in R \; \exists x = x_1u_1u_2x_2 \in V^* \; \exists y = y_1u_3u_4y_2 \in V^* \; . \; ((z = x_1u_1u_4y_2) \wedge (w = y_1u_3u_2x_2))\}$$

Splicing-Operation

Splicing-Systeme (EH-Systeme) (III)



DNA-Doppelstränge, die die Wörter x und y kodieren

erzeugte Schnittfragmente Präfixe und Suffixe von x und y

Rekombinationsprodukt mit den neuen Wörtern z und w

Mathematische Beschreibung

Sei V ein Alphabet ohne die Zeichen # und \$. Eine Splicing-Regel ist eine Zeichenkette $r = u_1\#u_2\$u_3\#u_4 \in V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$.

- Die Teilwörter u_1 und u_2 beschreiben Erkennungssequenz und Spaltstelle des ersten Restriktionsenzyms $(u_1|u_2)$. Analog $(u_3|u_4)$ für zweites Restriktionsenzym.
- Eine Regelmenge $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ enthält alle in einem System betrachteten Splicing-Regeln $r \in R$.
- Der Vorgang des Splicing wird beschrieben durch eine Relation $\vdash_\gamma \subseteq (V^* \times V^*) \times (V^* \times V^*)$, die die Wörter (x, y) in (z, w) umwandelt:

$$(x,y) \vdash_\gamma (z,w) = \{((x,y),(z,w)) \mid \exists x_1,x_2,y_1,y_2 \in V^* \; \exists r = u_1\#u_2\$u_3\#u_4 \in R \; \exists x = x_1u_1u_2x_2 \in V^* \; \exists y = y_1u_3u_4y_2 \in V^* \; . \; ((z = x_1u_1u_4y_2) \wedge (w = y_1u_3u_2x_2))\}$$

Systemdefinition (EH-System)

Splicing-Systeme (EH-Systeme) (IV)

Definition Extended Head-System

Ein EH-System γ ist ein Quadrupel $\gamma = (V, \Sigma, A, R)$ mit folgender Bedeutung der Komponenten: V bezeichnet das Alphabet von γ, $\Sigma \subseteq V$ das Alphabet der Terminalsymbole, die formale Sprache $A \subseteq V^*$ die Menge der Axiome und die formale Sprache $R \subseteq V^* \otimes \{\#\} \otimes V^* \otimes \{\$\} \otimes V^* \otimes \{\#\} \otimes V^*$ die Menge der Splicing-Regeln, wobei $\#, \$ \notin V$.

Bemerkungen

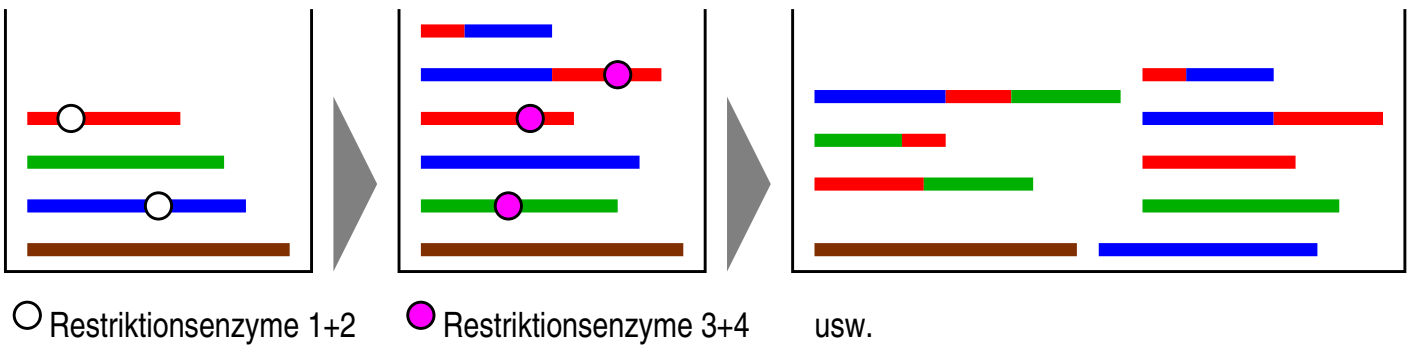
- Die Relation $\vdash_\gamma$ bildet einen Ableitungsschritt (Regelanwendung) des EH-Systems γ.
- Es wird unterstellt, dass die Axiome (Startwörter) in beliebig großer Kopienanzahl vorhanden sind.
- Regelanwendungen geschehen maximal parallel, d.h. alle matchenden Regeln werden genutzt.

Hintereinanderausführung von Splicing-Operationen

Splicing-Systeme (EH-Systeme) (V)

Arbeitsweise

- Bereitstellen eines initialen DNA-Pools (Axiome) durch formale Sprache A
- Daraus Generierung einer formalen Sprache $L(\gamma)$ durch fortlaufendes Splicing.
- Jede Splicing-Operation vergrößert die Anzahl unterschiedlicher Strangsequenzen im Pool (exponentielles Wachstum möglich)



$\Longrightarrow$ Mathematische Beschreibung dieses Vorganges:

Hintereinanderausführung von Splicing-Operationen

Splicing-Systeme (EH-Systeme) (VI)

Vergrößerung des DNA-Pools durch Anwendung einer Splicing-Regel

Seien $\gamma = (V, \Sigma, A, R)$ ein EH-System und $L' \in V^*$ eine formale Sprache. Die **Splicing-Funktion** $\sigma : V^* \to V^*$ ist definiert durch

$$\sigma(L') := \{z \in V^* \mid \exists x, y \in L' \; \exists r \in R \; . \; (((x, y) \vdash_\gamma (z, w)) \vee ((x, y) \vdash_\gamma (w, z)))\}$$

$\Longrightarrow$ $\sigma(L')$ umfasst alle Wörter, die zum DNA-Pool L' durch einmalige Anwendung aller matchenden Splicing-Regeln aus R neu hinzukommen.

$\Longrightarrow$ Schrittweise Entwicklung des DNA-Pools:

$$A \overset{\text{hinzu}}{\longrightarrow} \sigma(A) \overset{\text{hinzu}}{\longrightarrow} \sigma(\sigma(A)) \overset{\text{hinzu}}{\longrightarrow} \sigma(\sigma(\sigma(A))) \overset{\text{hinzu}}{\longrightarrow} \ldots$$

Iteriertes Splicing

Splicing-Systeme (EH-Systeme) (VII)

Mathematische Beschreibung durch Hüllenbildung

Seien $\gamma = (V, \Sigma, A, R)$ ein EH-System und $\sigma : V^* \to V^*$ die Splicing-Funktion. Das k-fach iterierte Splicing σ^k mit $k \in \mathbb{N}$ wird rekursiv definiert durch:

$$\sigma^0(A) = A$$
$$\sigma^{i+1}(A) = \sigma^i(A) \cup \sigma(\sigma^i(A)),\ i \in \mathbb{N}$$

Die **reflexive transitive Hülle** σ^* von σ ist definiert durch $\sigma^*(A) = \bigcup_{i \in \mathbb{N}} \sigma^i(A)$.

$\Longrightarrow$ Die reflexive transitive Hülle umfasst beliebige Anzahlen von Iterationsschritten: von gar keinem (pure Axiome) bis zu abzählbar unendlich vielen (Schrittzahl i geht gegen ∞).

$\Longrightarrow$ Es ist möglich, dass schon nach endlicher Schrittzahl nichts Neues mehr hinzukommt und somit eine im Endlichen abgeschlossene Hülle entsteht.

Durch EH-System generierte Sprache

Splicing-Systeme (EH-Systeme) (VIII)

Ausgabe eines EH-Systems

Sei $\gamma = (V, \Sigma, A, R)$ ein EH-System. Die durch γ beschriebene (generierte) Sprache $L(\gamma)$ ist definiert durch $L(\gamma) = \sigma^*(A) \cap \Sigma^*$.

Bemerkungen

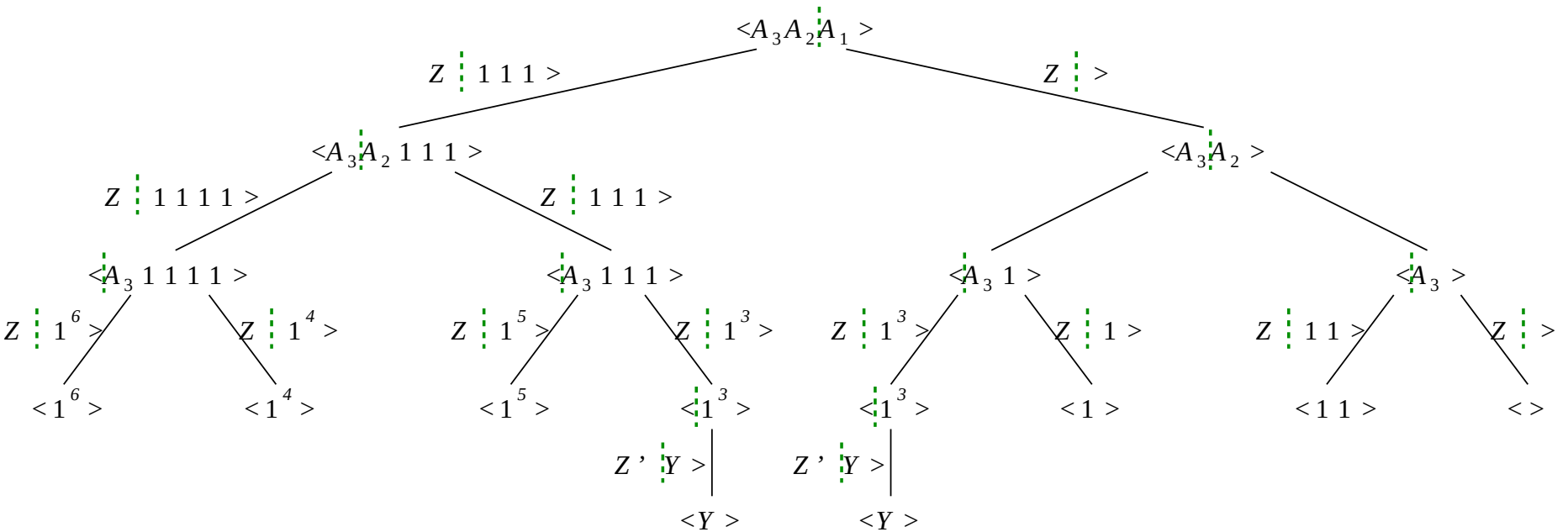
- EH-Systeme einschl. ihres dynamischen Verhaltens rein **algebraisch** beschreibbar
- Mit endlicher Axiomenmenge A und endlicher Regelmenge R beliebige reguläre Sprachen $L(\gamma)$ generierbar
- Für uneingeschränkte Turing-Universalität reguläre Regelmengen R (abz. unendlich groß) erforderlich
- Programmierung von EH-Systemen durch Vorgabe von A und R – effizient, aber mitunter schwer zu designen

Algorithmus zur Lösung des Rucksackproblems

Splicing-Systeme (EH-Systeme) (IX)

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

Idee



A_1, A_2, A_3: Platzhalter für Gegenstände, können durch ihr Gewicht ersetzt werden. Dabei schon eingebrachte Gewichte berücksichtigen.

Algorithmus zur Lösung des Rucksackproblems

Splicing-Systeme (EH-Systeme) (X)

Notation des Algorithmus als EH-System (Beispielinstanz)

$\gamma = (V, \Sigma, A, R)$

$V = \{Z, Z', 1, A_1, A_2, A_3\} \cup \Sigma$

$\Sigma = \{<, Y, >\}$

$A = \{<A_3A_2A_1>, Z>, Z1>, Z11>, Z111>, Z1111>, Z11111>, Z111111>, Z'Y>\}$

$R = \{$

$\#A_1 > \$Z\#111>$, $\#A_1 1 > \$Z\#1111>$, $\#A_1 11 > \$Z\#11111>$, $\#A_1 111 > \$Z\#111111>$, $\#A_1 1111 > \$Z\#1111111>$, $\#A_1 11111 > \$Z\#11111111>$, $\#A_1 111111 > \$Z\#111111111>$, $\#A_1 > \$Z\#>$, $\#A_1 1 > \$Z\#1>$, $\#A_1 11 > \$Z\#11>$, $\#A_1 111 > \$Z\#111>$, $\#A_1 1111 > \$Z\#1111>$, $\#A_1 11111 > \$Z\#11111>$, $\#A_1 111111 > \$Z\#111111>$,

$\#A_2 > \$Z\#1>$, $\#A_2 1 > \$Z\#11>$, $\#A_2 11 > \$Z\#111>$, $\#A_2 111 > \$Z\#1111>$, $\#A_2 1111 > \$Z\#11111>$, $\#A_2 11111 > \$Z\#111111>$, $\#A_2 111111 > \$Z\#1111111>$, $\#A_2 > \$Z\#>$, $\#A_2 1 > \$Z\#1>$, $\#A_2 11 > \$Z\#11>$, $\#A_2 111 > \$Z\#111>$, $\#A_2 1111 > \$Z\#1111>$, $\#A_2 11111 > \$Z\#11111>$, $\#A_2 111111 > \$Z\#111111>$,

$\#A_3 > \$Z\#11>$, $\#A_3 1 > \$Z\#111>$, $\#A_3 11 > \$Z\#1111>$, $\#A_3 111 > \$Z\#11111>$, $\#A_3 1111 > \$Z\#111111>$, $\#A_3 11111 > \$Z\#1111111>$, $\#A_3 111111 > \$Z\#11111111>$, $\#A_3 > \$Z\#>$, $\#A_3 1 > \$Z\#1>$, $\#A_3 11 > \$Z\#11>$, $\#A_3 111 > \$Z\#111>$, $\#A_3 1111 > \$Z\#1111>$, $\#A_3 11111 > \$Z\#11111>$, $\#A_3 111111 > \$Z\#111111>$,

$\#111 > \$Z'\#Y>\}$

Algorithmus zur Lösung des Rucksackproblems

Splicing-Systeme (EH-Systeme) (XI)

Notation des Algorithmus als EH-System (geg: $a_1, \ldots, a_n, b$)

$$
\begin{aligned}
\gamma &= (V, \Sigma, A, R) \\
V &= \{Z, Z', 1\} \cup \{A_i \mid \forall i = 1, \ldots, n\} \cup \Sigma \\
\Sigma &= \{<, Y, >\} \\
A &= \{<A_n A_{n-1} \ldots A_2 A_1>\} \cup \{Z 1^k > \mid \forall k = 0, \ldots, \sum_{j=1}^{n} a_j\} \cup \{Z' Y >\}
\end{aligned}
$$

$$
\begin{aligned}
R &= \{\#A_i 1^k > \$ Z \# 1^{k+a_i} > \mid \forall k = 0, \ldots, \sum_{j=1}^{n} a_j \ \forall i = 1, \ldots, n\} \ \cup \\
&\quad \{\#A_i 1^k > \$ Z \# 1^k > \mid \forall k = 0, \ldots, \sum_{j=1}^{n} a_j \ \forall i = 1, \ldots, n\} \ \cup \\
&\quad \{<\#1^b > \$ Z' \# Y >\}
\end{aligned}
$$

Es gilt: $L(\gamma) = \begin{cases} \{<Y>, <>\} & \text{bei Lösung „ja“} \\ \{<>\} & \text{bei Lösung „nein“} \end{cases}$

Wird Lösung „ja“ nicht innerhalb von $n + 1$ Zeitschritten erreicht, so kann sie auch später nicht mehr erreicht werden.

Universalitätsnachweis

Splicing-Systeme (EH-Systeme) (XII)

Idee einer Simulation von Chomsky-Grammatiken

Angenommen, Grammatik soll ableiten $pqCDrst \overset{CD \to EF}{\longrightarrow} pqEFrst$

- **Einbetten in** $XB \ldots Y$ (Simulationsanfang mit Startsymbol der Grammatik)

$$S \mapsto XBSY$$
$$\ldots$$
$$XBpqCDrstY$$

- **Rotieren**, so dass CDY ganz rechts steht. (B: Beginn-Marker)

$$XtBpqCDrsY$$
$$XstBpqCDrY$$
$$XrstBpqCDY$$

- **Ersetzen entsprechend Grammatik-Regel**

$$XrstBpqEFY$$

- **Dekodieren der Ausgabewörter**

$$XBpqEFrstY \mapsto pqEFrst$$

Universalitätsnachweis

Splicing-Systeme (EH-Systeme) (XIII)

Simulation von Chomsky-Grammatiken

Sei $G = (V', \Sigma, P, S)$ eine Chomsky-Grammatik vom Typ 0. G wird überführt in ein EH-System γ. Sei $U = V' \cup \Sigma \cup \{B\}$ mit $B \notin V' \cup \Sigma$. Das EH-System γ ist definiert durch:

$$
\begin{aligned}
\gamma &= (V, \Sigma, A, R) \\
V &= V' \cup \Sigma \cup \{X, X', B, Y, Z\} \cup \{Y_\alpha \mid \forall \alpha \in U\} \\
A &= \{XBSY, ZY, XZ\} \cup \{ZvY \mid \forall (u, v) \in P\} \cup \{ZY_\alpha, X'\alpha Z \mid \forall \alpha \in U\} \\
R &= \{Xw\#uY\$Z\#vY \mid \forall (u, v) \in P \; \forall w \in U^*\} \cup \\
&\quad \{Xw\#\alpha Y\$Z\#Y_\alpha \mid \forall \alpha \in U \; \forall w \in U^*\} \cup \\
&\quad \{X'\alpha\#Z\$X\#wY_\alpha \mid \forall \alpha \in U \; \forall w \in U^*\} \cup \\
&\quad \{X'w\#Y_\alpha\$Z\#Y \mid \forall \alpha \in U \; \forall w \in U^*\} \cup \\
&\quad \{X\#Z\$X'\#wY \mid \forall w \in U^*\} \cup \\
&\quad \{\#ZY\$XB\#wY \mid \forall w \in \Sigma^*\} \cup \\
&\quad \{\#Y\$XZ\#\}
\end{aligned}
$$

Es gilt: $L(G) = L(\gamma)$.

Abschließende Betrachtungen

- **Filtering-Modell nach Adleman, Lipton und Amos**
- **Modell Parallel Associative Memory (PAM)**
- **Sprache DNA-Pascal**
- **Modell Equality Checking (EC)**
- **Insertion-Deletion-Systeme**
- **Watson-Crick D0L-Systeme**
- **Splicing-Systeme (H-Systeme, EH-Systeme)**
- **Sticker-Systeme**
- **Modelle des Gene Assembly und DNA/RNA Self-Assembly**
- **P-Systeme**
- **Künstliche Chemien (Artificial Chemistries)**
- **Reaction-Diffusion Systems**

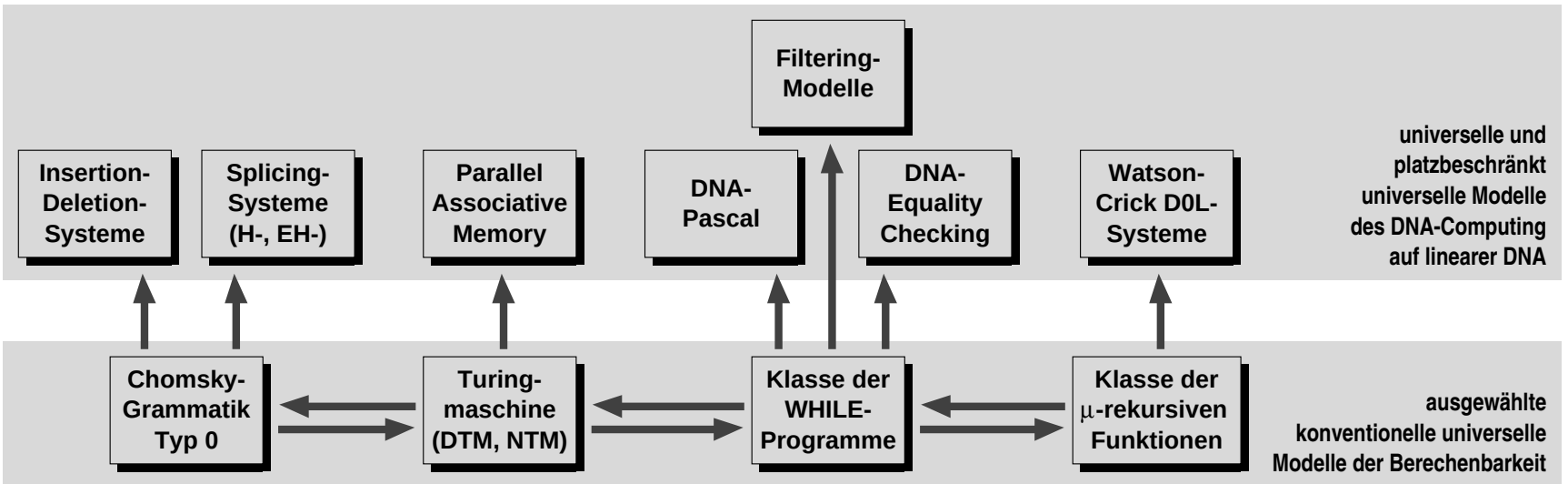
Modellvergleich

ausgewählte Modelle des DNA-Computing	universell	platzbeschränkt universell	restriktiv	multimengenbasiert	deterministisch	imperativ	regelbasiert	Multiple-Instruction-fähig	Multiple-Data-fähig	direkte Algorithmen-transformation von
Filtering-Modelle		■			■	■			■	WHILE
Parallel Associative Memory		■		■	■	■			■	TM
DNA-Pascal	■				■	■			■	WHILE
DNA Equality Checking	■				■	■			■	WHILE
Insertion-Deletion-Systeme	■						■	■	■	G
Watson-Crick D0L-Systeme	■		■		■				■	μ
Splicing-Systeme (H-, EH-)	■						■	■	■	G

WHILE: WHILE-Programm G: Chomsky-Grammatik vom Typ 0
TM: Turingmaschine μ: μ-rekursive Funktion, beschrieben im μ-Rekursionsschema

- imperative Modelle: Vorlage für Mikroflussreaktoren
- regelbasierte Modelle: Vorlage für Ein- oder Mehrtubesysteme
- funktionale Modelle (z.B. Watson-Crick D0L-Systeme): Ausgangspunkt für formale Verifikation (Korrektheitsbeweise)

Modelltransformationen



- Modelltransformationen sind zugleich Algorithmentransformationen.
- Sie können automatisiert durchgeführt werden.
- Effizienzverluste treten dabei auf,
  - wenn Nichtdeterminismus in Determinismus abzubilden ist oder
  - wenn spezifische bzw. kombinationsstarke Modelloperationen durch entsprechend schwächere ausgedrückt werden müssen

Zusammenfassung

- Modelle und Programmiersprachen für molekulare Computer sind ein Bindeglied zwischen abstrakten Berechnungsmodellen der Theoretischen Informatik und Implementierungen *in vitro*.
- Modelle sind immer idealisiert: Bestimmte Aspekte werden berücksichtigt, andere ignoriert.
- Modelle dienen der Algorithmenkonstruktion. Resultierende Algorithmen werden dann schrittweise bis zur Laborimplementierung verfeinert und optimiert.
- Wir haben einen Einblick in Programmierstrategien molekularer Computer auf algebraischer Ebene bekommen.
- Abstraktionen von Ligation und/oder Hybridisierung kommen in jedem universellen Modell des DNA/RNA-Computing vor.