Mächtigkeit von DNA-Rechnern

(Roos, Wagner: On the power of DNA-Computing, Information and Computation 131, 95–109 (1996)

In Anlehnung an Adlemans Experiment wollen wir eine Programmiersprache definieren, in der sich "DNA-Algorithmen" formalisieren lassen: DNA-Pascal

Wir werden dann die Klasse aller Probleme, die sich mit DNA-Pascal in Polynomialzeit lösen lassen, charakterisieren.

Definition von DNA-Pascal

- 2 Typen von Variablen:
  - Wortvariablen: enthalten einen String über {0,1}, werden mit Kleinbuchstaben bezeichnet.
  - Mengenvariablen: enthalten endliche Menge von Strings über {0,1}, werden mit Großbuchstaben bezeichnet ($T, T_1, T_2, \ldots$)
    Idee: Eine Mengenvariable T repräsentiert den Inhalt eines Reagenglases, gefüllt mit einer DNA-Lösung.
  - Programmierkonstrukte: wie in üblicher imperativer Sprache: if ... then ... else, while ... do, for ... do, ...

- **Operationen auf Wortvariablen**

  Lesen von Positionen in einem Wort,
  Setzen beliebiger Positionen auf
  0 bzw. 1
  ↪ Simulation von Integers möglich

- **Operationen auf Mengenvariablen**

  - UN (Union): $T := T_1 \cup T_2$
  - IN (Initialisierung) $T := Im(k)$
    wobei $k \in \mathbb{N}$.
    Danach hat T den Wert $\{0,1\}^k$,
    d.h. enthält alle (2^k viele) Bit-
    strings der Länge k.
  - BX (Bit-Extraktion): $T := BX(T_1, m, a)$
    wobei $m \in \mathbb{N}$, $a \in \{0,1\}$.
    Danach hat T den Wert
    $T_1 \cap \{0,1\}^{m-1} a \{0,1\}^*$
    d.h. aus T_1 werden alle Bitstrings
    mit einer a an der Stelle m
    herausgefiltert.
  - EM (Leerheitstest): $T = \emptyset$?
    liefert 1 (falls $T = \emptyset$) bzw.
    0 (falls $T \neq \emptyset$) zurück.

- **Kosten einer Operation $T := \ldots$**

  Kosten = maximale Länge eines
  Bitstrings in T nach Ausführung
  der Operation.
  Ist wichtig für IN-Operation

- Ein DNA-Pascal erhält seine
  Eingabe über evtl. mehrere ausge-
  zeichnete Wortvariablen.

Definition: **DNA(UN, IN, BX, EM)-P**
(im folgend kurz DNA-P) ist die
Klasse aller Probleme, die sich
mit einem DNA-Pascal Programm
wie oben beschrieben in Polynomial-
zeit lösen lassen.

Beachte: Eine Operation
$T := \mathrm{In}(k)$ kostet dabei k Zeitschritte.

Beispiel: (Liptom 94)

3-SAT $\in$ DNA-P

Eingabe: Eine 3-SAT Formel der Gestalt

$$H(x_1,\dots,x_m) = \bigwedge_{i=0}^{k} \left(x_{\sigma(i,1)}^{a(i,1)} \vee x_{\sigma(i,2)}^{a(i,2)} \vee x_{\sigma(i,3)}^{a(i,3)} \right)$$

wobei $\left.\begin{array}{l} \sigma(i,j) \in \{1,\dots,m\} \\ a(i,j) \in \{0,1\} \end{array}\right\}$ für $0 \le i \le k$, $1 \le j \le 3$

$x^0 := \neg x \quad x^1 := x$

Beachte: $x^a = 1 \Leftrightarrow x = a$

$H(x_1,\dots,x_m)$ sei irgendwie als Bitstring kodiert.

$T := \mathrm{In}(m)$; (Zeitbedarf $= m$)
for $i := 0$ to k do
$T_1 := BX(T, \sigma(i,1), a(i,1))$;
$T_2 := BX(T, \sigma(i,2), a(i,2))$;
$T_3 := BX(T, \sigma(i,3), a(i,3))$
$T := T_1 \cup T_2$; $T := T \cup T_3$
end
if $T = \emptyset$ then reject else accept

- Bemerkung: Im obigen Programm verwenden wir die Mengenvariable T mehrfach. Biologisch ist dies auf Grund der Polymerase-Ketten-Reaktion gerechtfertigt.
- Aus obigen Beispiel und der NP-Vollständigkeit von 3-SAT folgt sofort: NP $\subseteq$ DNA-P

Wie groß ist nun DNA-P?

Etwas Komplexitätstheorie

P = Klasse aller Probleme, die sich auf einer deterministischen Turing-Maschine in Polynomialzeit lösen lassen.

NP = Klasse aller Probleme, die sich auf einer nichtdeterministischen Turing-Maschine in Polynomialzeit lösen lassen.

- $P \subseteq NP$, ob $P = NP$ gilt, ist offen. Allgemeine Vermutung: $P \subsetneq NP$.
- Wahl von Turing-Maschinen als Maschinenmodell in der obigen Definition von P und NP ist nicht wesentlich.

  z.B. P = Klasse aller Probleme, die sich mit C++ in Polynomialzeit lösen lassen.

Orakel-Turing-Maschinen

Sei $A \subseteq \{0,1\}^*$ eine beliebige Menge (das Orakel). Eine Turing-Maschine mit Orakelzugriff auf A ist eine Turing-Maschine T mit einem zusätzlichen Band O (Orakelband).

- T darf auf das Band O nur schreiben.
- T hat 3 ausgezeichnete Zustände: $q_?, q_J, q_N$
- Geht die Maschine in den Zustand $q_?$ über, und befindet sich auf dem Orakelband O aktuell der String $w \in \{0,1\}^*$, so befindet sich T zum nächsten Zeitpunkt in Zustand q_J (falls $w \in A$) bzw. q_N (falls $w \notin A$). Außerdem ist der Inhalt des Orakelbands gelöscht (d.h. Inhalt = ε), alle anderen Bänder und Kopfpositionen sind unverändert.

Idee: T bekommt korrekte Antworten auf Fragen der Form "$w \in A$?" for free, egal wie schwer A selbst ist (A kann sogar unentscheidbar sein)

- P^A ist die Klasse aller deterministischen Probleme, die sich auf einer Turing-Maschine mit Orakelzugriff auf A in Polynomialzeit lösen lassen.

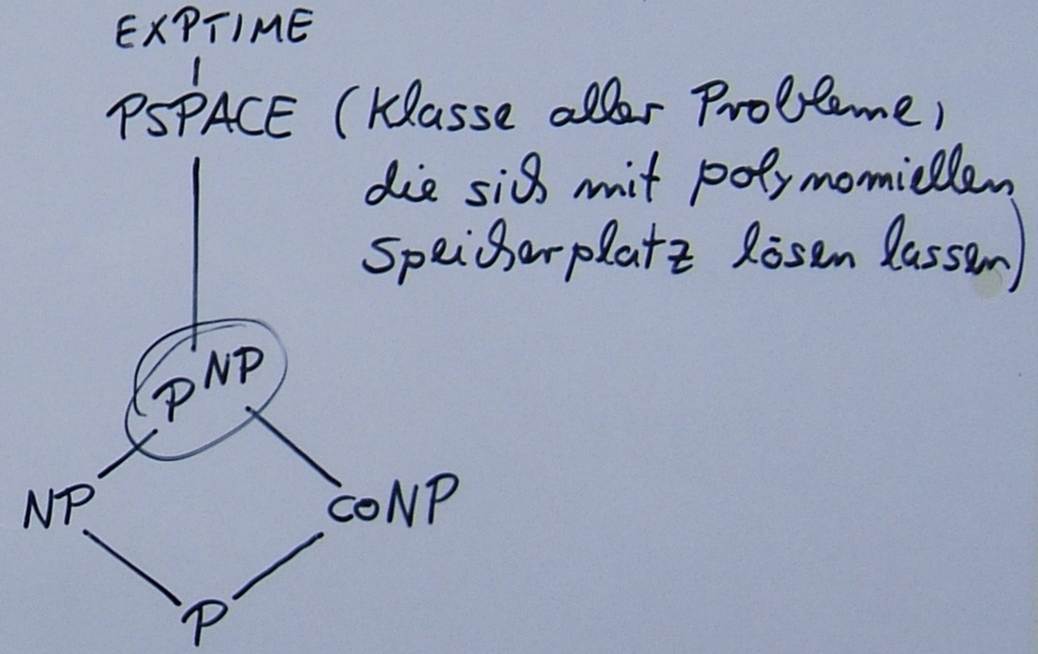
- Sei $\mathcal{C}$ eine Klasse von Sprachen (z.B. P, NP, PSPACE):

$$P^{\mathcal{C}} = \bigcup_{A \in \mathcal{C}} P^A$$

- Einige einfache Sachverhalte:
  - $P^{NP} = P^{coNP}$
  - $P^P = P$
  - $\mathcal{C} \subseteq P^{\mathcal{C}}$, insbesondere $NP \subseteq P^{NP}$
  - $coNP \subseteq P^{NP}$ (dies legt die Vermutung $NP \subsetneq P^{NP}$ nahe)

- Sei A ein NP-vollständiges Problem.
  $\Rightarrow P^A = P^{NP}$

Also z.B. $P^{3\text{-SAT}} = P^{NP}$



Satz: $DNA\text{-}P = P^{NP}$

$\supseteq$: Offensichtlich gilt:

$$P^{DNA\text{-}P} = DNA\text{-}P$$

$$\rightarrow P^{NP} = P^{3\text{-SAT}} \subseteq P^{DNA\text{-}P} = DNA\text{-}P$$

$\in$: Sei $L \in$ DNA-P und sei C ein DNA-Pascal Programm, welches L in Polynomialzeit $p(n)$ (für ein Polynom $p(n)$) entscheidet.

- Seien $T_1, \ldots, T_k$ alle Mengenvariablen, die in C vorkommen.
- Wir definieren ein 5-stelliges Prädikat R_C wie folgt:

Seien $x, z \in \{0,1\}^*$, $a_1, \ldots, a_m \in \{0,1\}$, $t \in \mathbb{N}$, $t \leq p(|x|)$, $1 \leq i \leq k$

$(x, a_1 \ldots a_m, z, i, t) \in R_C$

$:\Leftrightarrow$

Wird das Programm C auf der Eingabe x gestartet, so gilt $z \in T_i$ zum Zeitpunkt t, falls wir den zeitlich j-ten EM-Test (etwa $T_j = \emptyset$?) durch "$a_j = 1$?" ersetzen und so bis zum Zeitpunkt t genau m viele EM-Tests

($\exists v_1 \ldots \forall v_1 \ldots$)

machen.

Behauptung: R_C gehört zur Klasse P, ist also in Polynomialzeit entscheidbar.

Beweis der Behauptung:

Führe für jede Mengenvariable T_j eine boolsche Variable $s_j \in \{0,1\}$ ein.

Idee: Zu jedem Zeitpunkt im Ablauf von C soll gelten:

$$z \in T_j \Leftrightarrow s_j = 1$$

Ersetze nun Mengenoperationen in C wie folgt:

$T_j := T_u \cup T_v \rightarrow s_j := (s_u \vee s_v)$

$T_j := \ln(k) \rightarrow$ if $|z| = k$ then $s_j := 1$ else $s_j := 0$

$T_j := BX(T_u, m, a) \rightarrow$ if $z[m] = a$ then $s_j := s_u$ else $s_j := 0$

- Sei C' das Programm, dass wir so erhalten.
- Mengenvariablen kommen in C' nur noch in EM-Tests ($T_j = \emptyset$?) vor.
- Wir lassen nun C' mit Eingabe x für t Schritte laufen.

  Den zeitlich j-ten EM-Test (~~etwa $T_j = \emptyset$?~~) ersetzen wir durch $a_j = 1$? $(1 \leq j \leq m)$

- Dann gilt:

$$(x, a_1 \cdots a_m, z, i, t) \in R_C$$

$$\Leftrightarrow$$

$$\underbrace{s_i = 1 \text{ nach } t \text{ Schritten.}}$$

kann in Polynomialzeit entschieden werden.

- Wir zeigen nun $L \in P^{NP} = P^{coNP}$
- Beachte: Da die Laufzeit des DNA-Pascal Programms C bei Eingabe x durch $\xrightarrow{\text{Polynom}} p(|x|)$ beschränkt ist, ist auch die Länge der Wörter in $T_1, \ldots, T_k$ zu jedem Zeitpunkt in Ablauf von C durch $p(|x|)$ beschränkt.
- Eine Turing-Maschine T mit Orakelzugriff auf eine Menge in coNP kann nun "$x \in L$" wie folgt entscheiden:

(1) T simuliert das DNA-Pascal Programm C auf der Eingabe x, wobei alle Mengenoperationen $T_j := \ldots$ komplett weggelassen werden.

(2) Angenommen T kommt während der Simulation zum Zeitpunkt t bei einem EM-Test "$T_j = \emptyset$?" an. Die Ergebnisse der bisherigen EM-Tests sind in einem Bitstring $a_1 a_2 \cdots a_m$ gespeichert:

$$a_i = \begin{cases} 1 & \text{falls der i-te EM-Test 1 geliefert hat.} \\ 0 & \text{sonst} \end{cases}$$

Der aktuelle EM-Test "$T_j = \emptyset$?" wird nun durch folgende Frage an ein coNP-Orakel ersetzt:

$$\neg\exists z \Big(\underbrace{|z| \le p(|x|) \wedge \underbrace{(x, a_1 \cdots a_m, z, j, t) \in R_C}_{\text{kann in Polynomialzeit entschieden werden}}}_{\in NP} \Big)$$

Wir hängen dann eine 1 (bzw. eine 0) an den String $a_1 a_2 \cdots a_m$ an, falls der Orakeltest "JA" (bzw. "NEIN") liefert $\square$