Biochemisch inspirierte Berechnungen

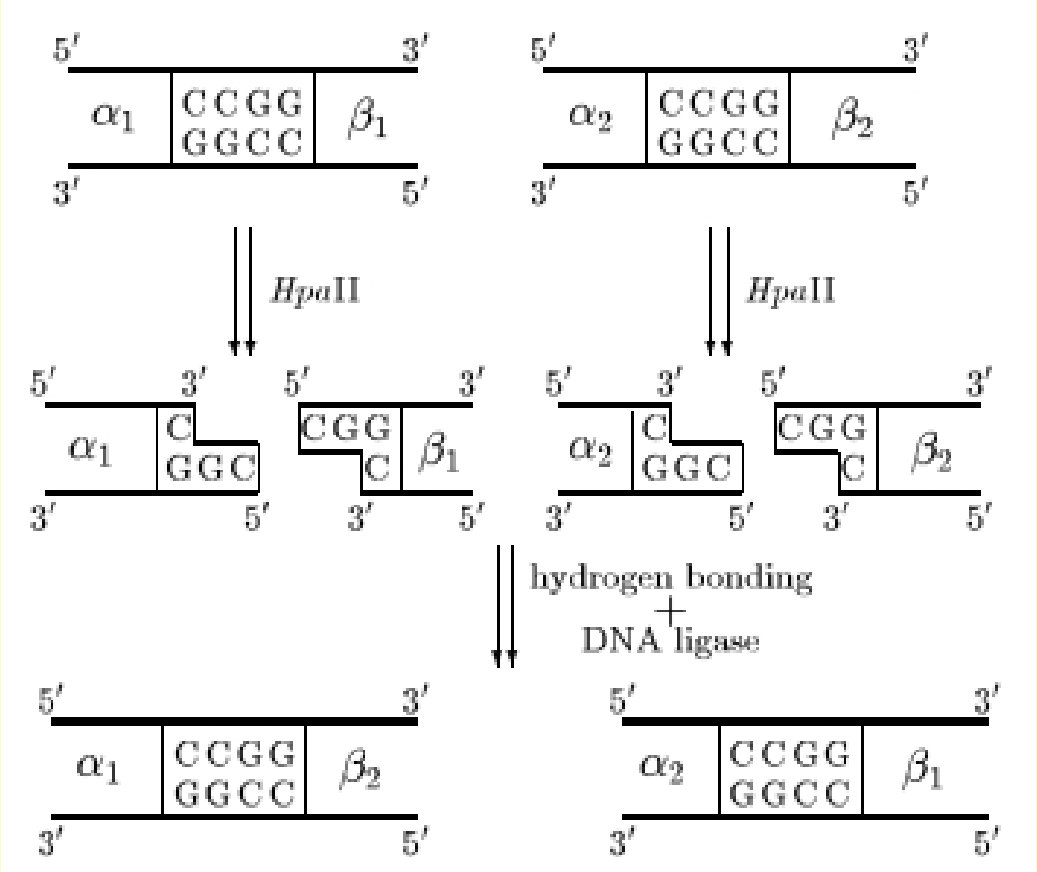
DNA Computing: Splicing Systeme
folgt weitgehend der Darstellung in:
Păun, Rozenberg, Salomaa: DNA Computing

Peter Leupold

Vorlesung Wintersemester 2009/2010

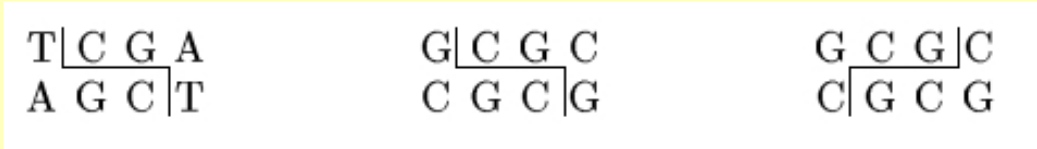
Biochemische Grundlage

Wir erinnern uns an das abschliessende Beispiel zur Kombination von Enzymen (einem Restriktionsenzym und einer Ligase):



Ein weiteres Beispiel

Die Restriktionsenzyme *Tac*1, *Sci*N1 und *Hha*1 haben folgende *Erkennungssequenzen* (das sind die Sequenzen, an denen sie sich anlagern um zu schneiden)



Die Erkennungssequenzen bzw. Schnitte sind also verschieden, die erzeugten *sticky ends* jedoch in den ersten beiden Fällen gleich.

Rekombination

Die folgenden drei Doppelstränge werden durch die Enzyme auf folgende Weise zerschitten:

5'-CCCCCTCGACCCCC-3'	$\longrightarrow$	5'-CCCCCT	CGACCCCC-3'
3'-GGGGGAGCTGGGGG-5'	$\longrightarrow$	3'-GGGGGAGC	TGGGGG-5'
5'-AAAAAGCGCAAAAA-3'	$\longrightarrow$	5'-AAAAAG	CGCAAAAA-3'
3'-TTTTTCGCGTTTTT-5'	$\longrightarrow$	3'-TTTTTCGC	GTTTTT-5'
5'-TTTTTGCGCTTTTT-3'	$\longrightarrow$	5'-TTTTTGCG	CTTTTT-3'
3'-AAAAACGCGAAAAA-5'	$\longrightarrow$	3'-AAAAAC	GCGAAAAA-5'

Rekombination

Bei Zugabe einer Ligase können sich nun neben den ursprünglichen Molekülen auch die folgenden beiden neuen Doppelstränge bilden:

5'-CCCCCTCGCAAAAA-3'
3'-GGGGGAGCGTTTTT-5'

5'-AAAAAGCGACCCCC-3'
3'-TTTTTCGCTGGGGG-5'

Da die Nahtstellen keine der Erkennungssequenzen neu bilden, ist der Vorgang nicht durch dieselben Enzyme umkehrbar.

Formalisierung

Die Darstellung von DNA-Sequenzen in der Form

5'-CCCCCTCGCAAAAA-3'
3'-GGGGGAGCGTTTTT-5'

ist äusserst unhandlich. Zudem ist sie aufgrund der Komplementarität der Stränge redundant.

Mit der Konvention, dass es wir stets in der Richtung $5'$ nach $3'$ lesen können wir die Moleküle also als Zeichenketten repräsentieren, ohne Information zu verlieren, im vorliegenden Fall als

CCCCCTCGCAAAAA

Damit betrachten wir ein wohlbekanntes Datenformat.

Formalisierung

Im Zeichenkettenformat lassen sich Rekombinationen wie im obigen Beispiel als Tripel

$$[u, w, v]$$

darstellen, wobei u und v den linken und rechten Kontext darstellen. w ist das erzeugte sticky end.

Für die drei Enzyme *Tac*1, *Sci*N1 und *Hha*1 haben wir die korrespondierenden Tripel

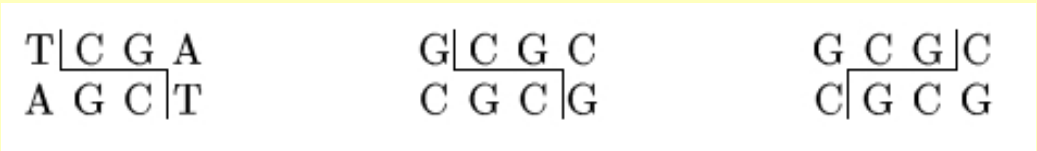
$$[T, CG, A], [G, CG, C], [G, CG, C]$$

Formalisierung

Da es sich bei den zerschnittenen DNA-Molekülen nicht mehr um vollständige Doppelstränge handelt, ist hier die Darstellung als Zeichenketten nicht mehr vollständig. Im vorliegenden Fall sind die beiden letzten Tripel identisch.

$$[T, CG, A], [G, CG, C], [G, CG, C]$$

Wir haben jedoch gesehen, dass die Schnitte genau invers erfolgen und die Teilstränge sich in diesem Fall nicht über Kreuz neu kombinieren können.



Splicing-Regeln

Definition

Eine *Splicing-Regel* ist eine Zeichenkette

$$u_1 \# u_2 \$ u_3 \# u_4$$

wobei $u_1, u_2, u_3, u_4 \in \Sigma^*$.

- Das zentrale Sticky End der Tripel wie $[T, CG, A]$ lassen wir hierbei weg.
- Damit erledigt sich auch das Problem der inversen Schnitte.

Ableitung

Durch die Splicing-Regeln werden die beiden Ableitungsrelationen

- $\vdash$ (1-Splicing) und
- $\models$ (2-Splicing)

definiert.

Das 1-Splicing ignoriert zwar eines der beiden Produkte, kann aber durch symmetrische Regeln wie $u_3\#u_4\$u_1\#u_2$ für $u_1\#u_2\$u_3\#u_4$ dieselben Sprachen erzeugen, wie eine von Haus aus symmetrische Variante.

Splicing-Schema

Definition

Ein *Splicing-Schema* oder auch *H-Schema* (nach Tom Head) ist ein Paar $[\Sigma, R]$ eines Alphabets Σ und einer Menge $R \subseteq \Sigma^*\#\Sigma^*\$\Sigma^*\#\Sigma^*$ von Splicing-Regeln über diesem Alphabet.

Dabei läßt sich die Regelmenge R als Formale Sprache über dem Alphabet $\Sigma \cup \{\#, \$\}$ ansehen.

Bei der Ableitung durch eine beliebige Regel aus einem H-Schema $\sigma = [\Sigma, R]$ schreiben wir auch $\vdash_R$.

Splicing

Für ein H-Schema $\sigma = [\Sigma, R]$ und eine Formale Sprache $L \subseteq \Sigma^*$ definieren wir

$$\sigma_1(L) := \{w : w \in \Sigma^*, (u, v) \vdash_R w \text{ für } u, v \in \Sigma^*\}.$$

Für zwei Sprachfamilien F_1 und F_2 definieren wir

$$S_1(F_1, F_2) := \{\sigma_1(L) : L \in F_1, \sigma = [\Sigma, R], R \in F_2\}.$$

Das heisst, die erste Sprachfamilie stellt die Ausgangsmenge von Wörtern (auch: Axiome), die zweite Sprachfamilie charakterisiert die Komplexität der Menge der Splicingregeln.

Unser besonderes Interesse wird endlichen Mengen von Splicingregeln gelten. Da diese durch Enzyme implementiert werden müssten, scheint alles andere sehr unrealistisch.

Iteriertes Splicing

Bei einer Vielzahl von Molekülen in einem Reagenzglas werden sich nicht alle zeitgleich rekombinieren und nach einem solchen Schritt anhalten. Daher macht es mehr Sinn, Splicing in iterierter Form zu betrachten:

$$\sigma_1^0(L) := L,$$
$$\sigma_1^{i+1}(L) := \sigma_1(\sigma_1^i(L)),$$

und schliesslich

$$\sigma_1^*(L) := \bigcup_{i \geq 0} \sigma_1^i(L).$$

Entsprechend S_1 werden auch hier jeweils Sprachen bzw. Sprachfamilien miteinander rekombiniert:

$$H_1(F_1, F_2) := \{\sigma_1^*(L) : L \in F_1, \sigma = [\Sigma, R], R \in F_2\}.$$

Ein Beispiel

Wir betrachten das sehr einfache Splicing

$$H_1(\{a\#b\$a\#c\}, \{acab\}).$$

Die erzeugte Sprache ist

$$(ac)^+ab.$$

Radius

Um dem Begriff "einfach" eine genauere Bedeutung zu geben klassifizieren wir Splicing-Schemata nach der Länge der beteiligten Zeichenketten.

Definition

Der *Radius* einer Splicing-Regel $u_1\#u_2\$u_3\#u_4$ ist das Maximum von $\{|u_1|, |u_2|, |u_3|, |u_4|\}$. Der Radius eines Splicing-Schemas ist das Maximum der Radien seiner Regeln.

Satz

Wenn F eine Sprachfamilie ist, die unter λ-freien GSM-Abbildungen abgeschlossen ist, dann ist $S_1(F, FIN) = S_1(F, [1])$ wo $[1]$ die Menge aller Splicing-Schemata mit Radius eins ist.

Bemerkung: alle gängigen Sprachklassen (regulär, kontextfrei, kontextsensitiv, aufzählbar) sind unter λ-freien GSM-Abbildungen abgeschlossen.

Einfaches Splicing

Wir untersuchen nun, inwieweit sich mit endlich vielen Splicing-Regeln die Komplexität von Sprachen steigern lässt.

Satz

$H_1(REG, FIN) = REG.$

Andererseits:

Satz

Jede rekursiv aufzählbare Sprache $L \subseteq \Sigma^*$ lässt sich als $L = U \cap \Sigma^*$ darstellen wobei U eine Sprache aus $H_1(FIN, REG)$ ist.

Beweisskizzen an der Tafel...

Übersicht

Splicings mit Klassen der Chomsky-Hierarchie resultieren in den folgenden Komplexitäten:

	FIN	REG	LIN	CF	CS	RE
FIN	FIN, REG	FIN, RE	FIN, RE	FIN, RE	FIN, RE	FIN, RE
REG	REG	REG, RE	REG, RE	REG, RE	REG, RE	REG, RE
LIN	LIN, CF	LIN, RE	LIN, RE	LIN, RE	LIN, RE	LIN, RE
CF	CF	CF, RE	CF, RE	CF, RE	CF, RE	CF, RE
CS	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE	CS, RE
RE	RE	RE	RE	RE	RE	RE

Sind zwei Klassen angegeben, so wird die linke komplett erzeugt, die rechte stellt eine obere Schranke dar.

Fazit

- Mit einfachem Splicing gewinnen wir kaum an Komplexität, insbesondere nicht mit endlichen Regelmengen.
- Andererseits haben wir gesehen, dass mit einem erweiterten Alphabet relativ schnell alle rekursiv aufzählbaren Sprachen erzeugt werden können.
- Dies motiviert die Einführung *Erweiterter Splicing-Systeme*.

Erweitertes Splicing

Wie bei generativen Grammatiken unterscheiden wir zwischen Terminalen und Nichtterminalen. Das heisst, es steht eine Art Schmierpapier zur Verfügung.

Definition

Ein *erweitertes H-System* ist ein Quadrupel $[\Sigma, T, A, R]$ wobei Σ ein Alphabet, $T \subset \Sigma$, $A \subset \Sigma^*$ und R eine Menge von Splicingregeln über Σ ist.

Die erzeugte Sprache ist $L([\Sigma, T, A, R]) := \sigma_1^*(A) \cap T^*$ für das Splicing-Schema $\sigma = [\Sigma, R]$

Erweitertes Splicing

Hier können wir die regulaẗren Sprachen mit endlich vielen Regeln aus endlich vielen Axiomen erzeugen.

Satz

$REG \subset EH_1(FIN, FIN)$.

Beweis an der Tafel...

Übersicht

Erweiterte Splicings mit Klassen der Chomsky-Hierarchie resultieren in den folgenden Komplexitäten:

	FIN	REG	LIN	CF	CS	RE
FIN	REG	RE	RE	RE	RE	RE
REG	REG	RE	RE	RE	RE	RE
LIN	LIN, CF	RE	RE	RE	RE	RE
CF	CF	RE	RE	RE	RE	RE
CS	RE	RE	RE	RE	RE	RE
RE	RE	RE	RE	RE	RE	RE

Auch hier ist der Gewinn an Berechnungskraft sehr beschränkt.

Sonstiges

Versuche, die Berechungskraft durch zusätzliche Kontrollmechnismen zu stärken:

- Forbidding and Enforcing (Kontextbedingungen)
- Target Languages
- Programmierte Systeme