DNA SPLICING RULES

STAYING TRUE TO THE BIOLOGY

Elizabeth Goode   April 2015
HISTORY
HISTORY

• TOM HEAD introduced a mathematical model of the biological activity we call “DNA splicing.” [Tom Head, 1987]
HISTORY

• TOM HEAD introduced a mathematical model of the biological activity we call “DNA splicing.” [Tom Head, 1987]

• The model provided a way to study the generative power of double-stranded DNA (dsDNA) in the presence of enzymes that cut and paste in a site-specific manner – i.e., the computational power of the splicing operation.
• Over 5,000 articles and books have been published about DNA splicing systems since Head introduced the concept in 1987.
HISTORY

• Over 5,000 articles and books have been published about DNA splicing systems since Head introduced the concept in 1987.

• Algebra and graph theory have been used to attack the questions that arise in the area of DNA splicing theory.
HISTORY

• Over 5,000 articles and books have been published about DNA splicing systems since Head introduced the concept in 1987.

• Algebra and graph theory have been used to attack the questions that arise in the area of DNA splicing theory.

• Leonard Adleman examined the possibility of using DNA and its various biochemical operations to crack the RSA encryption code. DNA-computing was born. [Adleman 1994, 20-var SAT]
What is DNA?

Double-stranded DNA (dsDNA) is a double helix (twisted ladder) with a sugar-phosphate backbone. The “rungs” are made of nucleotides held together in a very specific orientation by hydrogen bonding.
What is DNA?
What is DNA?
What is DNA splicing?
What is DNA splicing?

• Splicing is the cutting and pasting of dsDNA.
• Restriction enzymes are proteins that can perform site-specific cutting of dsDNA.
• By site-specific cutting we mean the enzymes can recognize a specific sequence of nucleotides and perform a specific type of cut at that site.

Example: Bgl II
What is DNA splicing?

Example: Bgl II
What is DNA splicing?

Example: Bgl II
At 1.5Å resolution
$1.5 \times 10^{-7}$mm
Splicing as an Operation on dsDNA
Splicing as an Operation on dsDNA
Splicing as an Operation on dsDNA
Splicing as an Operation on dsDNA
Splicing as an Operation on dsDNA
Modeling the splicing of dsDNA
Modeling the splicing of dsDNA
Modeling the splicing of dsDNA

- The dsDNA that is initially in the test tube is represented as a set $I$. 
Modeling the splicing of dsDNA

• The dsDNA that is initially in the test tube is represented as a set $I$.

\[
I = \{S_1, S_2\}
\]

\[
S_1 = \alpha\text{GCCG}\gamma\text{CACCGGCC}\beta
\]

\[
S_2 = \gamma\text{CACCGTG}\delta
\]
Modeling the splicing of dsDNA

• The dsDNA that is initially in the test tube is represented as a set $I$.

\[ I = \{S_1, S_2\} \]

\[ S_1 = \alpha \text{GCCG}C\text{ACCGGC}\beta \]

\[ S_2 = \gamma \text{CAC}C\text{ACGTG}\delta \]
Modeling the splicing of dsDNA

• The dsDNA that is initially in the test tube is represented as a set I.
• The action of the restriction (cutting) enzymes is encoded by rules that belong to set R.

\[ I = \{S_1, S_2\} \]
\[ S_1 = \alpha GCGCGCACCGC \beta \]
\[ S_2 = \gamma CACCGACGTG \delta \]
Modeling the splicing of dsDNA

• The dsDNA that is initially in the test tube is represented as a set $I$.
• The action of the restriction (cutting) enzymes is encoded by rules that belong to set $R$.

$I = \{S_1, S_2\}$

$S_1 = \alpha \text{GCCG} \text{CACGGGC} \beta$

$S_2 = \gamma \text{CAC} \text{ACGTG} \delta$

$R = \{(\text{GCCN}, \text{NNN}, \text{NGGC}), (\text{CAC}, \text{NNN}, \text{GTG})\}$
Modeling the splicing of dsDNA

• The dsDNA that is initially in the test tube is represented as a set $I$.
• The action of the restriction (cutting) enzymes is encoded by rules that belong to set $R$.

$I = \{S_1, S_2\}$
$S_1 = \alpha\text{GCCGCA}\text{CCGGC}\beta$
$S_2 = \gamma\text{CACCA}\text{CGTG}\delta$

$R = \{(\text{GCCN, NNN, NGGC}), (\text{CAC, NNN, GTG})\}$

T4 ligase pastes sticky end together
Modeling the Cutting using Rule Sets
Modeling the Cutting using Rule Sets

\[(p, s, q)\]
Modeling the Cutting using Rule Sets

\( \alpha \quad p \quad s' \quad s'' \quad q \quad \beta \)

\( (p, s, q) \)
Modeling the Cutting using Rule Sets

\[ (p, s, q) \]

\[ (u, s, v) \]
Modeling the Cutting using Rule Sets

\[(p, s, q)\]

\[(u, s, v)\]
Modeling the Cutting using Rule Sets

\((p, s, q) (u, s, v)\)
Modeling the Cutting using Rule Sets

\[(p, s, q) (u, s, v)\]
Modeling the Cutting using Rule Sets

\[(p, s, q) (u, s, v)\]
Modeling the Cutting using Rule Sets

\[(p, s, q) \ (u, s, v)\]
What is a Formal Language?
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.

Example:  $A = \{a, b\}$
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.

Example: \( A = \{a, b\} \)
\[ L_1 = aa + ab + ba + bb \]
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.

Example: \( A = \{a, b\} \)

\( L_1 = aa + ab + ba + bb \)

\( L_2 = ab^*a \)
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.

Example:  \( A = \{a, b\} \)
\( L_1 = aa + ab + ba + bb \)
\( L_2 = ab^*a \)
\( L_3 = a(bb)^*a \)
What is a Formal Language?

We can think of a formal language as a set of strings written over a finite alphabet. In general, two alphabet letters suffice.

Example: $A = \{a, b\}$
$L_1 = aa + ab + ba + bb$
$L_2 = ab^*a$
$L_3 = a(bb)^*a$
$L_4 = a(bb)^*a + ab + ba + bb$
Formal Language Characterization Schemas
Formal Language Characterization Schemas

The Chomsky Hierarchy of Languages classifies languages into families.

[Noam Chomsky 1956,1959]
Formal Language Characterization Schemas

The Chomsky Hierarchy of Languages classifies languages into families.

[Noam Chomsky 1956,1959]
Formal Language Characterization Schemas

The Chomsky Hierarchy of Languages classifies languages into families.

[Noam Chomsky 1956,1959]
Formal Language Characterization Schemas

The Chomsky Hierarchy of Languages classifies languages into families. [Noam Chomsky 1956,1959]
Formal Language Characterization Schemas

The Chomsky Hierarchy of Languages classifies languages into families.

[Noam Chomsky 1956, 1959]
The Chomsky Hierarchy of Languages classifies languages into families.

[Noam Chomsky 1956, 1959]
Computational Models for Formal Languages
Computational Models for Formal Languages

• The “Machine” Model
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

$L_2 = ab^*a$
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

\[ L_2 = ab^*a \]
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

\[ L_2 = ab^*a \]
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

\[ L_2 = ab^*a \]

- \[ aa \notin L_A \]
- \[ aba \in L_A \]
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

• Push-Down Automata and Context-Free Languages
  Deterministic $\subset$ Non-deterministic
Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages
  • Push-Down Automata and Context-Free Languages
    Deterministic $\subset$ Non-deterministic
  • Linear-Bounded Automata and Context-Sensitive Languages
    Turing machine with finite tape

Computational Models for Formal Languages

• The “Machine” Model
  • Finite Automata and Regular Languages

• Push-Down Automata and Context-Free Languages
  Deterministic $\subset$ Non-deterministic

• Linear-Bounded Automata and Context-Sensitive Languages
  Turing machine with finite tape

• Universal Turing Machines and Recursively Enumerable Languages
  Turing machine with infinite tape
  [Turing 1936, Kleene 1936, Post 1946, Rice 1953, Minsky 1961]
Computational Models for Formal Languages

- The "Grammar" Model
Computational Models for Formal Languages

• The “Grammar” Model
  • Right (Left) Linear grammars and Regular Languages
Computational Models for Formal Languages

• The “Grammar” Model
  • Right (Left) Linear grammars and Regular Languages

\[ L_2 = ab^*a \]
Computational Models for Formal Languages

• The “Grammar” Model

  • Right (Left) Linear grammars and Regular Languages

\[ L_2 = ab^*a \]

S \rightarrow aA \quad A \rightarrow a \mid bB \quad B \rightarrow a \mid bB

\[ S \rightarrow aA \rightarrow aa \]

\[ S \rightarrow aA \rightarrow abB \rightarrow abbB \rightarrow abBsa \]
Computational Models for Formal Languages

• The “Grammar” Model

  • Right (Left) Linear Grammars and Regular Languages
    Rules have form $S \rightarrow aB \mid bD \mid \varepsilon$ [Chomsky 1956-59, Miller 1958]
Computational Models for Formal Languages

• The “Grammar” Model

  • Right (Left) Linear Grammars and Regular Languages
    Rules have form \( S \rightarrow aB | bD | \varepsilon \)  
    \[ S \rightarrow aB \quad S \rightarrow bD \quad S \rightarrow \varepsilon \]  
    [Chomsky 1956-59, Miller 1958]

  • Context-Free Grammars and Languages
    Rules can be two-sided  

  • Context-Sensitive Grammars and Languages
    Rules have form \( aAb \rightarrow aBb \)  
    [Kuroda 1964]

  • Semi-Thue (type 0) grammars and Recursively Enumerable Languages
    Undecideability of word problem for semi-Thue systems  
    \[ ? \quad u \rightarrow v \]  
    [Thue, 1914, Emil Post & A.A. Markov, 1942]
Computational Models for Formal Languages

• The “Splicing” Model
  • Gheorghe Paun proved that splicing is Turing complete – splicing systems not having finite initial sets can generate all languages that can be generated by the universal Turing machine. [1995]
Computational Models for Formal Languages

• The “Splicing” Model
  • Gheorghe Paun proved that splicing is Turing complete – splicing systems not having finite initial sets can generate all languages that can be generated by the universal Turing machine. [1995]

  • Pixton proved that all finitely generated splicing languages are regular. [1994]
Computational Models for Formal Languages

• The “Splicing” Model
  • Gheorghe Paun proved that splicing is Turing complete – splicing systems not having finite initial sets can generate all languages that can be generated by the universal Turing machine. [1995]
  • Pixton proved that all finitely generated splicing languages are regular. [1994]

  • There are assorted results “in-between” the classes of languages belonging to the regular set and the universal set of languages, primarily due to Gheorghe Paun.
Computational Models for Formal Languages

• The “Splicing” Model
Computational Models for Formal Languages

- The “Splicing” Model

- We denote a splicing schema $\sigma$ as a finite alphabet $A$ and a set of rules, $R$ that is a subset of a cross-product of copies of $A$. 

\text{Pauun} \; r \in R \quad r \in A^* \times \{\#\} \times A^*  
\text{Pixton/Goode} \; r \in R \quad r \in A^* \times A^* \times A^* \times \{\#\} \times A^*  

\text{Free monoid}  
\text{set of all words over } A
Computational Models for Formal Languages

• The “Splicing” Model

• We denote a splicing schema $\sigma$ as a finite alphabet $A$ and a set of rules, $R$ that is a subset of a cross-product of copies of $A$.

• A splicing system is a splicing schema and an initial set of strings $(\sigma, I)$. 

Computational Models for Formal Languages

• The “Splicing” Model

• We denote a splicing schema \( \sigma \) as a finite alphabet \( A \) and a set of rules, \( R \) that is a subset of a cross-product of copies of \( A \).

• A splicing system is a splicing schema and an initial set of strings \( (\sigma, I) \).

• The language associated with a splicing system \( (\sigma, I) \) is the set of all well-formed strings that are present at any time during the iterated cutting and pasting.
Rules for Splicing
Rules for Splicing

Splicing languages that respect the biology
Rules have form \((c,x,d) (p,x,q)\) and handedness

[Head 1987]
Rules for Splicing

Splicing languages that respect the biology
Rules have form \((c,x,d) (p,x,q)\) and handedness

[Head 1987]
Rules for Splicing

Splicing languages that respect the biology
Rules have form \((c,x,d) (p,x,q)\) and handedness

Splicing languages with uniform rules are Strictly Locally Testable
Rules have form \((1,x,1) (1,w,1)\) where \(x,w \in A^P\) for some \(P > 0\)
Rules for Splicing

Splicing languages that respect the biology
Rules have form \((c,x,d) (p,x,q)\) and handedness

Splicing languages with uniform rules are Strictly Locally Testable
Rules have form \((1,x,1) (1,w,1)\) where \(x, w \in A^P\) for some \(P > 0\)

A constant in a language \(L\) is a substring \(x\) such that if \(cxd\) and \(pxq\) are words in \(L\), then \(cxq\) and \(pxd\) are also in \(L\).
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages

[Goode & Pixton 1999]
Rules for Splicing

Semi-Simple Splicing Languages  
Rule form is (a, 1; b, 1) with a, b in A

They are precisely the Strictly Locally Testable Languages  
[Goode & Pixton 1999]

Rule (a, 1; b, 1) cuts & pastes these strings:

| \(\mu\) | \(a\) | \(\eta\) |
| \(\gamma\) | \(b\) | \(\delta\) |
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is (a, 1; b, 1) with a,b in A
They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

Rule (a, 1; b, 1) cuts & pastes these strings:

| μ  | a  | η |
| γ  | b  | δ |

Producing these strings:

REGULAR
Rules for Splicing

Semi-Simple Splicing Languages  
Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)  
They are precisely the Strictly Locally Testable Languages  
[Goode & Pixton 1999]

Rule \((a, 1; b, 1)\) cuts & pastes these strings:

Producing this string:

\[
\begin{array}{ccc}
\mu & a & \eta \\
\gamma & b & \delta \\
\end{array}
\]

\[
\begin{array}{ccc}
\mu & a & \delta \\
\end{array}
\]
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages \([Goode & Pixton 1999]\)

\[
(a, 1) \sim (b, 1) \rightarrow (b, 1) \sim (a, 1)
\]

Rule \((a, 1; b, 1)\) cuts & pastes these strings:

\[
\begin{array}{ccc}
\mu & a & \eta \\
\gamma & b & \delta \\
\end{array}
\]

Producing this string:

\[
\begin{array}{ccc}
\mu & a & \delta \\
\gamma & b & \eta \\
\end{array}
\]

\(r = (b, 1; a, 1)\)
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages

[Goode & Pixton 1999]
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is (a, 1; b, 1) with a,b in A
They are precisely the Strictly Locally Testable Languages
They can be reflexive languages.

[Goode & Pixton 1999]
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is $\langle a, 1; b, 1 \rangle$ with $a,b$ in $A$

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

$A^k \subseteq$ Constants of $L$
Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

A reflexive rule has the form \((u, v; u, v)\).

Reflexive rule sets have the property that if \((u, v; u', v')\) is in the rule set, then so are \((u, v; u, v)\) and \((u', v'; u', v')\).
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq\) Constants of \(L\)

A reflexive rule has the form \((u, v; u, v)\).

Reflexive rule sets have the property that if \((u, v; u', v')\) is in the rule set, then so are \((u, v; u, v)\) and \((u', v'; u', v')\).

The sites \(uv\) and \(u'v'\) are constants in \(L\).
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq \text{Constants of } L\)
Rules for Splicing

Semi-Simple Splicing Languages  
Rule form is \( (a, 1; b, 1) \) with \( a, b \) in \( A \)

They are precisely the Strictly Locally Testable Languages  
[Goode & Pixton 1999]

They can be reflexive languages.

\( A^k \subseteq \text{Constants of } L \)

If \( uav \) and \( sbt \) are words in \( L \) and the rule \( (a, 1; b, 1) \) is in \( R \),
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq \text{Constants of } L\)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).
Rules for Splicing

Semi-Simple Splicing Languages  
Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages  
[Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq \text{Constants of } L\)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).

One rule models the action of 2 enzymes.
Rules for Splicing

Semi-Simple Splicing Languages

Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq \text{Constants of } L\)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).

One rule models the action of 2 enzymes.

The biology also allows the formation also of word \(sbt\).
Rules for Splicing

Semi-Simple Splicing Languages
Rule form is \((a, 1; b, 1)\) with \(a, b\) in \(A\)

They are precisely the Strictly Locally Testable Languages [Goode & Pixton 1999]

They can be reflexive languages.

\(A^k \subseteq \text{Constants of } L\)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).

One rule models the action of 2 enzymes.

The biology also allows the formation also of word \(sbt\).

In order to model this we turn to new types of rule sets.
Consider a rule of the form $(a, 1; a, 1)$ with $a$ in $A$. 

Rules for Splicing
Consider a rule of the form \((a, 1; a, 1)\) with \(a\) in \(A\). This type of rule is reflexive.
Rules for Splicing

Consider a rule of the form \((a, 1; a, 1)\) with \(a\) in \(A\). (Simple rule)
This type of rule is reflexive.

Now consider a rule of the form \((a, 1; b, 1)\) with \(a\) in \(A\). (Semi-simple rule)
Rules for Splicing

Consider a rule of the form \((a, 1; a, 1)\) with \(a\) in \(A\). 
This type of rule is reflexive.  

(Simple rule)

Now consider a rule of the form \((a, 1; b, 1)\) with \(a\) in \(A\). 

(Semi-simple rule)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).
Consider a rule of the form \((a, 1; a, 1)\) with \(a\) in \(A\). (Simple rule)
This type of rule is reflexive.

Now consider a rule of the form \((a, 1; b, 1)\) with \(a\) in \(A\). (Semi-simple rule)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).

The biology allows the formation also of word \(sbt\).
So we include a rule of the form \((b, 1; a, 1)\).
Rules for Splicing

Consider a rule of the form \((a, 1; a, 1)\) with \(a\) in \(A\). (Simple rule)
This type of rule is reflexive.

Now consider a rule of the form \((a, 1; b, 1)\) with \(a\) in \(A\). (Semi-simple rule)

If \(uav\) and \(sbt\) are words in \(L\) and the rule \((a, 1; b, 1)\) is in \(R\), then splicing can occur, producing the string \(uat\).

The biology allows the formation also of word \(sbt\).
So we include a rule of the form \((b, 1; a, 1)\).
We call this the “symmetric twin” of \((a, 1; b, 1)\). Symmetry respects the biology.
Rules for Splicing

The form of rules can reveal or obscure the biological activity.
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Ex: Head’s original notation was intended to show the “sticky ends” or “crossing” of the restriction enzyme cutting sites. \((u, x, v) (p, x, q)\)
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Ex: Head’s original notation was intended to show the “sticky ends” or “crossing” of the restriction enzyme cutting sites. \((u, x, v) (p, x, q)\)

Ex: A Pixton-type rule of the form \((u, v ; u’, v’)\) does not necessarily respect the biological notion of the crossing of the restriction site.
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Ex: Head’s original notation was intended to show the “sticky ends” or “crossing” of the restriction enzyme cutting sites.

Ex: A Pixton-type rule of the form \((u, v ; u', v')\) does not necessarily respect the biological notion of the crossing of the restriction site.

\[
\begin{array}{cccc}
\alpha & p & x & q \\
\end{array}
\]

\[
\begin{array}{cccc}
\alpha & s & x & t \\
\end{array}
\]
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Ex: Head’s original notation was intended to show the “sticky ends” or “crossing” of the restriction enzyme cutting sites.

Ex: A Pixton-type rule of the form \((u, v ; u', v')\) does not necessarily respect the biological notion of the crossing of the restriction site.

But it can: \((px, q ; sx, t)\) or \((p, xq ; s, xt)\)

\[
\begin{array}{cccc}
\alpha & p & x & q \\
\hline
\beta \\
\end{array}
\]

\[
\begin{array}{cccc}
\alpha & s & x & t \\
\hline
\beta \\
\end{array}
\]
Rules for Splicing

The form of rules can reveal or obscure the biological activity.
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Yuhani Yusof and E Goode decided to rewrite the rules in a form that makes the biological activity apparent.

Simple rules: \((a, 1; a, 1) \rightarrow (a, 1, 1: a, 1, 1)\) or \((1, 1, a: 1, 1, a)\) or \((1, a, 1: 1, a, 1)\)
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Yuhani Yusof and E Goode decided to rewrite the rules in a form that makes the biological activity apparent.

Simple rules: \((a, 1; a, 1) \rightarrow (a, 1, 1: a, 1, 1) \) or \((1, 1, a: 1, 1, a) \) or \((1, a, 1: 1, a, 1) \)

Semi-simple rules: \((a, 1: b, 1) \rightarrow (a, 1, 1: b, 1, 1) \)
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Yuhani Yusof and E Goode decided to rewrite the rules in a form that makes the biological activity apparent.

Simple rules: \((a, 1; a, 1) \rightarrow (a, 1, 1: a, 1, 1) \text{ or } (1, 1, a: 1, 1, a) \text{ or } (1, a, 1: 1, a, 1)\)

Semi-simple rules: \((a, 1; b, 1) \rightarrow (a, 1, 1: b, 1, 1)\)

Semi-null rules: \((u, 1; v, 1) \rightarrow (u, 1, 1: v, 1, 1) \text{ or } (u_1, w, 1: v_1, w, 1)\)
Rules for Splicing

The form of rules can reveal or obscure the biological activity.

Yuhani Yusof and E Goode decided to rewrite the rules in a form that makes the biological activity apparent.

Simple rules: \((a, 1; a, 1) \rightarrow (a, 1, 1: a, 1, 1)\) or \((1, 1, a: 1, 1, a)\) or \((1, a, 1: 1, a, 1)\)

Semi-simple rules: \((a, 1; b, 1) \rightarrow (a, 1, 1: b, 1, 1)\)

Semi-null rules: \((u, 1; v, 1) \rightarrow (u, 1, 1: v, 1, 1)\) or \((u_1, w, 1: v_1, w, 1)\)

Theorems include: The languages generated by simple semi-simple and semi-null systems are exactly those generated by the corresponding rule sets written in the Yusof-Goode rule notation.
Rules for Splicing

As the biology demands that symmetry and reflexivity be respected, the Yusof-Goode model adopted the constraint that the splicing system would always be built to include any rules required to ensure its reflexive and symmetric closure.
Rules for Splicing

As the biology demands that symmetry and reflexivity be respected, the Yusof-Goode model adopted the constraint that the splicing system would always be built to include any rules required to ensure its reflexive and symmetric closure.

These notions become important in the investigation of the limit language associated with a splicing schema.
Wet Splicing of dsDNA

Lanes 1 and 7: Comparison DNA (Ladder)

Lane 2: Intact strands only

Lanes 3-6 are splicing events taken every 5 minutes + pole

INITIAL MOLECULES

Long:
- GCCGCCGCCGC
- Bgl I
- 1442 bp
- 1.6 kbp
- 1312 bp
- 2.1 kbp

Short:
- CACCCCGTGC
- Dra III
- 825 bp
- 0.98 kbp

ADULT MOLECULES
Planning

Experiment
### Tm Calculator Results

Results for these primers are as follows:

<table>
<thead>
<tr>
<th>Primer</th>
<th>Tm (°C)</th>
<th>Salt (mM)</th>
<th>Primer (pM)</th>
<th>%GC</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primer 1:</td>
<td>50.11</td>
<td>1.7</td>
<td>0.2</td>
<td>60</td>
<td>25</td>
</tr>
<tr>
<td>Primer 2:</td>
<td>47.05</td>
<td>1.7</td>
<td>0.2</td>
<td>60</td>
<td>25</td>
</tr>
</tbody>
</table>

Tm Delta = 2.05

---

### Tm Calculator Results

Results for these primers are as follows:

<table>
<thead>
<tr>
<th>Primer</th>
<th>Tm (°C)</th>
<th>Salt (mM)</th>
<th>Primer (pM)</th>
<th>%GC</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primer 1:</td>
<td>40.50</td>
<td>1.7</td>
<td>0.2</td>
<td>44</td>
<td>25</td>
</tr>
<tr>
<td>Primer 2:</td>
<td>45.99</td>
<td>1.7</td>
<td>0.2</td>
<td>52</td>
<td>25</td>
</tr>
</tbody>
</table>

Tm Delta = 5.79

---

Do another Tm Calculation
Planning Experiment

From Yuhani on 03/12/11

Dear Dr Liz,

These are actually my plan tomorrow:

Run on the double 8-well
1. 3 alpha-beta+3 gamma-delta+1 combination alpha-beta & gamma-delta of the PCR Rx#4 +LMW ladder 2. 1 alpha-beta, 1 gamma-delta, 1 combination of alpha-beta & gamma-delta of the Purification PCR Rx#3 (that I did last Thursday) - I'll take 5 ul each since it is suggested from Qiagen purification kit to run the purification product with 5ul purification lambda with 1 ul 6X loading dye, 1 combination of alpha-beta that you purified, 1 combination alpha-beta & gamma-delta (2nd) purified, 1

If all look goods, I will do the RE digestion and ligation after that and run for final results.

Just two question that I want to ensure.

1. If I did all the digestion and ligation sample-mean until the last sample of Lane 8 that stored in -70 degC, how long that I need to wait to run all the sample (Lane 1-Lane 8) on 3% gel.

2. The RE digestion and ligation sample should be done in 37 degC. So, I 'll put in the heater box at 37 degC and take out the sample from that with the time t =5, 10.....that I plan. Is it right?
Cutting out Stands 1 and 2 from E.Coli genome
Purification of Cut outs from E.Coli
PCR results from purified DNA for Initial Strand 1
Dynamics of the DNA splicing system:

Lanes 1 and 9 contain DNA Ladder

Limit language is apparent in Lanes 4, 5, 6, 7

Transient strings appear in Lanes 3 – 7
The “Splicing” Model

Splicing languages with uniform rules are Strictly Locally Testable.
Rules have form \((c,x,d) (p,x,q)\) and handedness \([\text{Head 1987}]\).

Semi-Simple Splicing Languages
Rule \(= (a, 1; b, 1)\) with \(a, b\) in \(A\)
Non-constant word length is bounded below; SLT; Reflexive \([\text{Goode \& Pixton 1999}]\).

Limit languages are precisely the regular languages. \([\text{Goode \& Pixton 2004}]\).

Yusof-Goode splicing generates simple, semi-simple and non-semi-simple languages. \([\text{Goode \& Yusof 2011}]\).
The end.

Thank you for your time.