Special course in Computer Science: Molecular Computing

Lecture 8: DNA Computing by Splicing and by Insertion-Deletion

Vladimir Rogojin
Department of IT, Åbo Akademi
http://combio.abo.fi/teaching/special

Fall 2013

Theories inspired from biological processes

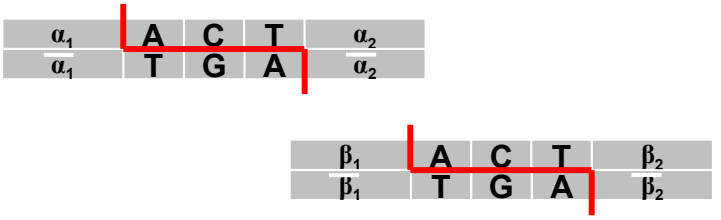
- Formal Language Theory has been positively influenced by the introduction of new models of computation, inspired by biological processes
- This trend started with the introduction of a mathematical formalism for biological phenomena of DNA recombinant processes proposed by Tom Head in 1987
- He has introduced the formalism of splicing systems to describe the generative power of some recombinant processes

Molecular computing and formal language theory

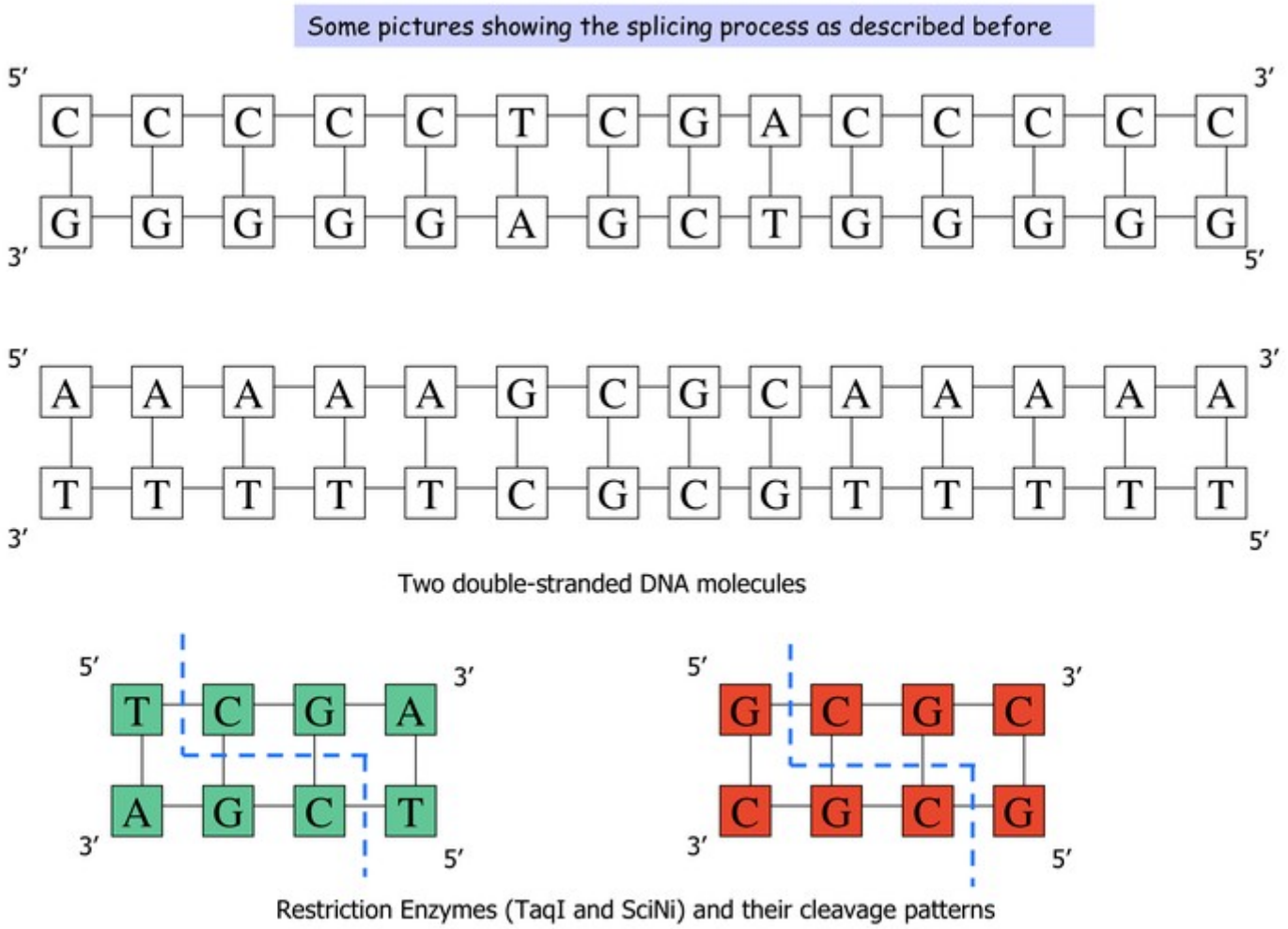
- Splicing systems is one of the growing areas in the field of Molecular Computing as well as in the field of Formal Language Theory
- Splicing systems – “dry” molecular computing, i.e., it is concerned with the development of theoretical computational models, starting from a biological mechanism

Biology of splicing

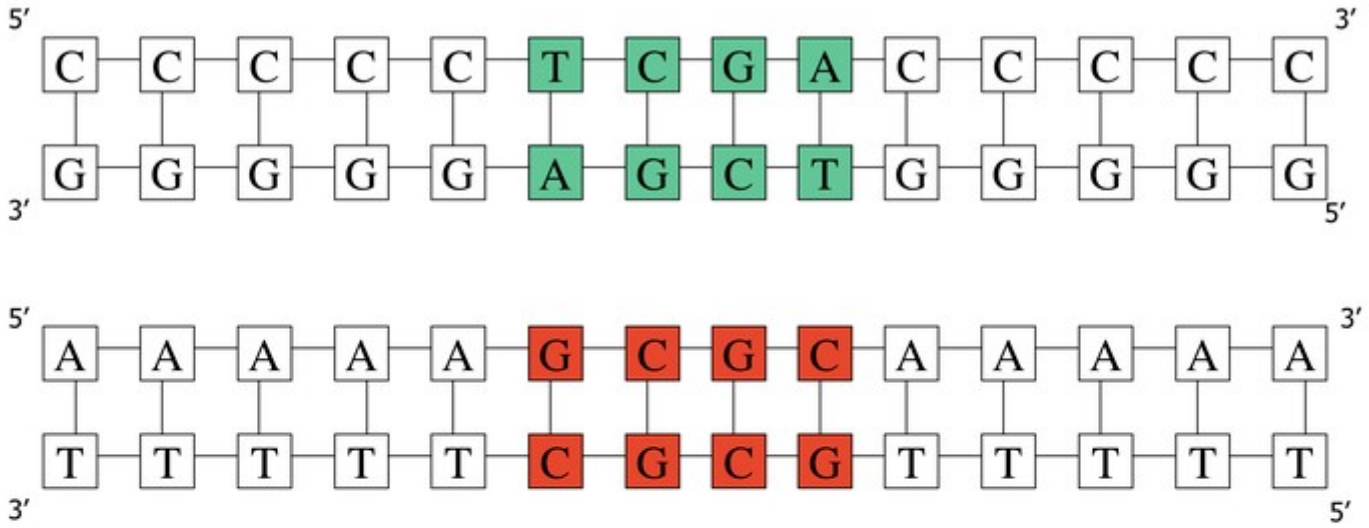
- Splicing: sequence of cut by restriction enzymes, recombination and paste by ligase operations
- Restriction enzimes recognize sites and perform cut
- Recombination
- Ligation



Biology of splicing, example

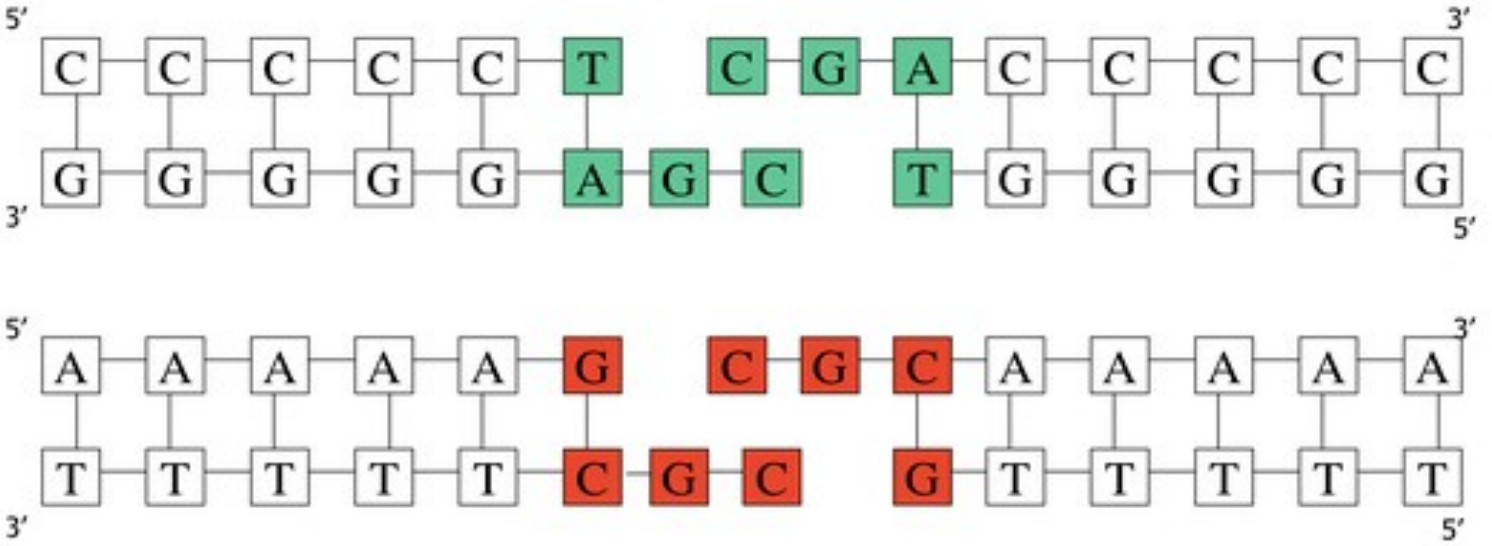


Biology of splicing, example



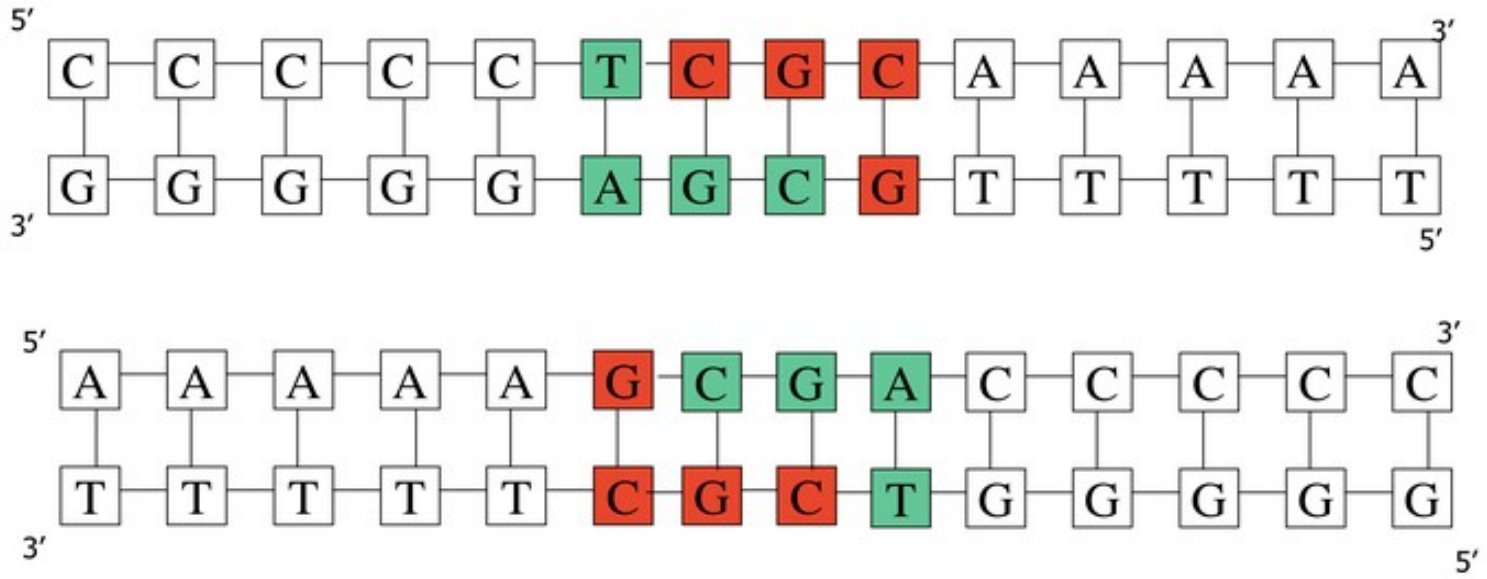
Restriction Enzymes recognize the patterns inside the molecules

Biology of splicing, example



The molecules are cut: four pieces are produced

Biology of splicing, example



Two new molecules may be produced by ligase

Splicing systems. Formal definition

- We abstract from 4-letter alphabet:
  - Consider a finite alphabet V
- A splicing rule over V is defined as string:
  - $r=u_1\#u_2\$u_3\#u_4$,
  - Where u_1, u_2, u_3 and u_4 are strings over V, and
  - # and $ are special symbols not in V
- Pairs (u1,u2) and (u3,u4) represent restriction sites for two enzymes, which cut DNA such that there will be produced sticky ends that match and the fragments could recombine

Splicing systems, formalizing DNA recombination

- Formalization:
  - Splicing rule $r=u_1\#u_2\$u_3\#u_4$,
- Pairs (u_1,u_2) and (u_3,u_4) represent restriction sites for two enzymes, which cut DNA such that there will be produced sticky ends that match and the fragments could recombine
- The first enzymes cuts molecules formalized as strings between u_1 and u_2
- The second enzymes cuts molecules formalized as strings between u_3 and u_4

Splicing operation on strings

- Formally:
  - String rewriting operation
  - Splicing rule: $r = u_1\#u_2\$u_3\#u_4$
  - Splicing transformation: $(x,y) \vdash_r (z,w)$, iff
    - $x=x_1u_1u_2x_2$, $y=y_1u_3u_4y_2$
    - $z=x_1u_1u_4y_2$, $w=y_1u_3u_2x_2$
    - For some x_1, x_2, y_1, y_2 words over V
  - 2-splicing:
    - We say that we splice x and y at the sites u_1u_2 and u_3u_4 respectively
    - Result consists of strings z and w

1-splicing

- 2-splicing:
  - We say that we splice x and y at the sites u_1u_2 and u_3u_4 respectively
  - Result consists of strings z and w
- 1-splicing:
  - In many contexts it is sufficient to consider a single resulting string, in particular the first one z
  - Here one rule $r = u_1\#u_2\$u_3\#u_4$ could be replaced with the pair of equivalent rules
    - $r = u_1\#u_2\$u_3\#u_4$
    - $r' = u_3\#u_4\$u_1\#u_2$
  - Rule r will produce z while rule r' will produce w

1- and 2-splicings

- 1-splicing: $(x,y) \vdash_r z$
- 2-splicing: $(x,y) \models_r (z,w)$
- It will be clear from the context which type of splicing we consider

H scheme

- Extending splicing operations to languages
- H-scheme:
  - δ=(V,R),
    - Where V is an alphabet
    - R is a subset of splicing rules in $V^*\#V^*\$V^*\#V^*$
  - Note: in this formulation R is a language. In this way we can consider its complexity with respect to language hierarchies. For instance, with respect to Chomsky hierarchy from the previous lecture

1-step splicing operations

- Let $\delta=(V,R)$ be a splicing scheme,
- Let L be a language over V
- We define $\delta_1(L)=\{z|(x,y) \vdash_r z$, for some x, y from L and r from R$\}$
- For two families of languages (sets of languages) FL_1 and FL_2 we denote
  - $S_1(FL_1,FL_2)=\{\delta_1(L)|$L is a language from FL_1 and $\delta=(V,R)$ with a language R from $FL_2\}$

Iterating splicing operations

- 1-step splicing operation:
  - $\delta_1(L)=\{z|(x,y) \vdash_r z$, for some x, y from L and r from R$\}$
- Multistep splicing:
  - $\delta^0{}_1(L) = L$
  - $\delta^{i+1}{}_1(L) = \delta^i{}_1(L) \cup \delta_1(\delta^i{}_1(L))$, $i \geq 0$
  - $\delta^*{}_1(L) = \cup_{i \geq 0} \delta^i{}_1(L)$
  - $\delta^*{}_1(L)$ – is the closure of L under the splicing with the respect to δ

Iterating splicing operations

- For two families of languages FL_1 and FL_2
- We define:
  - $H_1(FL_1,FL_2) = \{\delta^*_1(L) | L$ is a language from FL_1 and $\delta=(V,R)$ with a language R from $FL_2\}$
  - Closure of FL_1,FL_2 under the splicing with respect to $S_1(FL_1,FL_2)$
- The same definitions could be considered for 2-splicing. Then we will get:
  - $S_2(FL_1,FL_2)$ and $H_2(FL_1,FL_2)$

Chomsky hierarchy

- Finite languages, included in
- Regular languages, included in
- Linear languages, included in
- Context-free languages, included in
- Context-sensitive languages, included in
- Recursive enumerable languages:
  - Include any other language family
  - Can express any algorithm
  - Can be computed by Universal Turing Machines

Chomsky language families

- (FIN) Finite languages –
  - finite set of words
- (REG) Regular languages
  - defined by regular expressions,
  - recognized by finite automata
- (LIN) Linear languages –
  - defined by linear grammar,
  - recognized by pushdown automata
- (CF) Context-free languages
  - defined by context-free grammar, recognized by push-down automata
- (CS) Context-sensitive languages
  - defined by type-1 grammar,
  - recognized by linear bounded automation
- (RE) Recursive enumerable languages –
  - defined by unrestricted grammars,
  - recognized by Turing machines

Computational power of splicing operations

- We will examine the size of families $S_1(FL_1,FL_2)$ and $H_1(FL_1,FL_2)$ for FL_1 and FL_2 being one of the families:
  - FIN, REG, LIN, CF, CS, RE

Power of single-step splicing

The size of families $S_1(FL_1, FL_2)$

$FL_1 \backslash FL_2$	FIN	REG	LIN	CF	CS	RE
FIN	FIN	FIN	FIN	FIN	FIN	FIN
REG	REG	REG	REG, LIN	REG, CF	REG, RE	REG, RE
LIN	LIN, CF	LIN, CF	RE	RE	RE	RE
CF	CF	CF	RE	RE	RE	RE
CS	RE	RE	RE	RE	RE	RE
RE	RE	RE	RE	RE	RE	RE

- Intersection of row FL_1 and column FL_2 contains either $S_1(FL_1,FL_2)$, or FL_3 and FL_4 with FL_3 being proper subset of $S_1(FL_1,FL_2)$ and $S_1(FL_1,FL_2)$ being a proper subset of FL_4

Power of iterative splicing

The size of families $H_1(FL_1, FL_2)$

$FL_1 \backslash FL_2$	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

- Intersection of row FL_1 and column FL_2 contains either $H_1(FL_1,FL_2)$, or FL_3 and FL_4 with FL_3 being proper subset of $H_1(FL_1,FL_2)$ and $H_1(FL_1,FL_2)$ being a proper subset of FL_4

H systems

- Formal splicing-based computational device:
  - H system
- Extended H system:
  - Quadruple γ = (V, T, A, R), where
    - V is an alphabet
    - T – set of terminals, subset of V
    - A - the set of axioms over V
    - R – set of splicing rules, subset of $V^*\#V^*\$V^*\#V^*$
      - # and $ special symbols not in V

H systems

- Extended H system:
  - Quadruple γ = (V, T, A, R), where
    - V is an alphabet
    - T – set of terminals, subset of V
    - A - the set of axioms over V
    - R – set of splicing rules, subset of $V^*\#V^*\$V^*\#V^*$
      - # and $ special symbols not in V
  - Extended H system –
    - An underlying H scheme δ=(V,R) augmented with a given subset T of V and a set of axioms.
  - Non-extended H system:
    - An H system with T=V

Generating languages with H systems

- Let us have H system $\gamma = (V, T, A, R)$
- The language generated by γ we denote as $L(\gamma)$:
  - $L(\gamma) = \delta^*_1(A) \cap T^*$,
    - where δ is the underlying H scheme for γ
- Let FL_1 and FL_2 be two families of languages
- The family of languages $L(\gamma)$ generated by extended H system γ we denote by $EH_1(FL_1,FL_2)$
  - Where $\gamma = (V, T, A, R)$, with A belonging to FL_1 and R belonging to FL_2

Power of H systems

The generative power of extended H systems: $EH_1(FL_1, FL_2)$

$FL_1 \backslash FL_2$	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

- Intersection of row FL_1 and column FL_2 contains either $EH_1(FL_1,FL_2)$, or FL_3 and FL_4 with FL_3 being proper subset of $EH_1(FL_1,FL_2)$ and $EH_1(FL_1,FL_2)$ being a proper subset of FL_4
- Central relations for splicing-based DNA computing:
  - EH1(FIN,FIN) = REG – finite H system
  - EH1(FIN,REG) = RE – not practical, because of required infinite number of enzymes

Universal H systems

- From practical point of view we need finite splicing systems capable to generate class of RE languages
- It turns out that adding a little of control to splicing rules may lead to the jump from the family of REG languages directly to RE languages
- Universal H system:
  - An H system capable of simulating any other H system encoded through the set of axioms A

Types of universal H systems

- Context-sensitive splicing rules:
  - Permitting contexts –
    - each rule has associated to it some symbols, and the rule can be applied only if the input strings contain the respective symbols
  - Forbidding contexts -
    - Same as above, but the symbols should not be present in the input string
  - Target languages -
    - Local targets – a language is associated with each rule that accepts/rejects the result of the rule
    - Global targets – the same language is associated with all the rules and accepts/rejects the result of the rules

Types of universal H systems (cont.)

- Context-sensitive splicing rules:
  - Programmed H systems -
    - A next rule mapping is considered, which controls the sequence of rules used
  - Double splicing -
    - Programmed H systems restricted to two consecutive rule control
  - Multisets -
    - Counting the copies of strings present in any step of the computation

Universal computations with splicing

- Universal computing devices can be constructed
  - At least at theoretical level
- Universality results from here show that the computability can be “reconstructed” on the basis of an operation, splicing,
- That is much different from an usual operation of rewriting (Turning machines, Chomsky grammars, etc.)
- Significant result, because
  - At the level of DNA, nature mainly “computes” by means of splicing.

Insertion-Deletion operations

- An ins-del system is a construct
  - $\gamma = (V, T, A, I, D)$, where
    - V is an alphabet
    - T – set of terminal symbols, subset of V
    - A – a finite language of axioms over V
    - I – finite set of triples (u, α, v) – insertion rules
    - D – finite set of triples (u, α, v) – deletion rules

String rewriting with ins-del systems

- Insertion:
  - $x \rightarrow_{ins} y$ iff
    - $x=x_1uvx_2$, $y=x_1uwvx_2$ for some ins. (u,w,v) from I and x_1 and x_2 words over V
- Deletion:
  - $x \rightarrow_{del} y$ iff
    - $x=x_1uwvx_2$, $y=x_1uvx_2$ for some del. (u,w,v) from D and x_1 and x_2 words over V

Generating languages with ins-del systems

- The reflexive and transitive closure of $\rightarrow_{ins}$ and of $\rightarrow_{del}$ we denote as $\rightarrow^*_{ins}$ and $\rightarrow^*_{del}$ respectively. When both insertion and deletion are used, we write $\rightarrow^*$.
- Let us have an ins-del system γ = (V, T, A, I, D)
  - The language generated by γ we define as L(γ):
    - L(γ)={w over T | z$\rightarrow^*$w, z an axiom from A}

Representing ins-del rules as rewriting rules

- Insertion:
  - (u,w,v)
  - $uv \rightarrow uwv$
- Deletion:
  - (u,w,v)
  - $uwv \rightarrow uv$

Complexity of ins-del systems

- The complexity of an ins-del system is evaluated in terms of the complexity of its rules
- The complexity of a rule:
  - Length of the context, and
  - Length of the inserted/deleted string
- Formally:
  - Let $\gamma = (V, T, A, I, D)$ be an ins-del system
  - The weight $(n, m; p, q)$ of γ is defined as follows:...

Complexity of ins-del systems

- Formally:
  - Let γ = (V, T, A, I, D) be an ins-del system
  - The weight (n, m; p, q) of γ is defined as follows:
    - n = max{|α| | (u, α, v) in I}
    - m = max{|u| | (u, α, v) in I or (v, α, u) in I}
    - p = max{|α| | (u, α, v) in D}
    - q = max{|u| | (u, α, v) in D or (v, α, u) in D}
  - The family of languages L(γ) generated by ins-del systems we denote as:
    - $INS^{m}_{n}DEL^{q}_{p}$ is the set of all L(γ), where
    - Γ is of weight (n', m'; p', q') with n'≤n, m'≤m, p'≤p, q'≤q

Complexity of ins-del systems

- For a family of languages $INS^{m}_{n}DEL^{q}_{p}$ we replace any of m, n, p, q with * in case when the respective parameter is unbound.
  - In this way, the family of all ins-del languages is $INS^{*}_{*}DEL^{*}_{*}$
- If n=0, then also we suppose that m=0, since insertion of an empty string does not change anything
- Equivalently, if p=0, then q=0

Ins-del systems, example

- Let $\gamma = (\{a, b\}, \{a, b\}, \{ab\}, \{(a, ab, b)\}, \theta)$
  - $L(\gamma) = \{a^n b^n \mid n \geq 1\}$
- Let $\gamma = (\{a, b\}, \{a, b\}, \{ab\}, \{(\lambda, ab, \lambda)\}, \theta)$
  - $L(\gamma)$ – Dyck language – empty word (the language of balanced parenthesis)

Universality results for $INS^{m}_{n}DEL^{q}_{p}$

- We have $INS^{m}_{n}DEL^{q}_{p}$ = RE for the following quadruples:
  - (1,2;1,1)
  - (1,2;2,0)
  - (2,1;2,0)
  - (1,1;1,2)
  - (2,1;1,1)
  - (2,0;3,0)
  - (3,0;2,0)
  - (1,1;2,0)
  - (1,1;1,1)

Concluding remarks

- Computing by splicing and by insertion-deletion provides convincing evidence that
- Computer science has much to learn from the biochemistry of DNA
- Hopefully this will prove to be useful for practical computer science
- This research is related to the question:
  - “what does it mean to compute in a natural way?”