Networks of Splicing Processors

José M. Sempere
Research Group on Computation Models and Formal Languages
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Networks of Splicing Processors

1. DNA Recombination and Splicing.
2. Splicing over strings (Type I and II).
3. H schemes. Iterative and non-iterative language classes.
4. Extended H Systems.
5. Extended H Systems with permitting context
6. Splicing processors
7. (Accepting) NSPs
8. (A)NSPs are computationally complete
9. Complexity issues

DNA Recombination and Splicing

```
5' - CCCCCTCGACCCCC - 3'
3' - GGGGGAGCTGGGGG - 5'

5' - AAAAAGCGCAAAAA - 3'
3' - TTTTTCGCGTTTTT - 5'

5' - TTTTTGCGCTTTTT - 3'
3' - AAAAACGCGAAAAA - 5'
```

double strands

TaqI	SciNI	HhaI
T C G A	G C G C	G C G C
A G C T	C G C G	C G C G

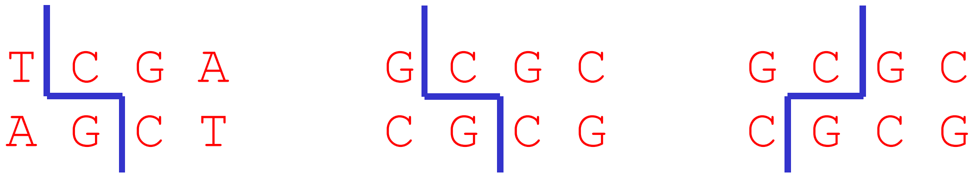
restriction enzimes (endonuclease)

```
5' - CCCCCT    CGACCCCC - 3'
3' - GGGGGAGC    TGGGGG - 5'

5' - AAAAAG    CGCAAAAA - 3'
3' - TTTTTCGC    GTTTTT - 5'

5' - TTTTTGCG    CTTTTT - 3'
3' - AAAAAC    GCGAAAAA - 5'
```

ligases

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

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

Splicing over strings (Types I and II)



TaqI SciNI HhaI

Patterns (T, CG, A) (C,CG,C) — Class I; (G,CG,G) — Class II

$$w_1 = w'_1\, u_1\, x_1\, v_1\, w''_1$$
$$w_2 = w'_2\, u_2\, x_2\, v_2\, w''_2$$

$$p_1 = (u_1\,,\, x_1\,,\, v_1)$$
$$p_2 = (u_2\,,\, x_2\,,\, v_2)$$

The splicing only occurs if p_1 and p_2 are of the same class and $x_1 = x_2$

$$z_1 = w'_1\, u_1\, x_1\, v_2\, w''_2$$
$$z_2 = w'_2\, u_2\, x_2\, v_1\, w''_1$$

Splicing over strings (Types I and II)

$w_1 = w'_1\ u_1\ x_1\ v_1\ w''_1$ $\quad p_1 = (u_1\ ,\ x_1\ ,\ v_1)$ $\quad z_1 = w'_1\ u_1\ x_1\ v_2\ w''_2$

$w_2 = w'_2\ u_2\ x_2\ v_2\ w''_2$ $\quad p_2 = (u_2\ ,\ x_2\ ,\ v_2)$ $\quad z_2 = w'_2\ u_2\ x_2\ v_1\ w''_1$

The patterns (p_1, p_2) can be denoted as $(u_1, u_2; u_3, u_4)$ or as the string $u_1\#u_2\$u_3u_4$.

Let r=$u_1\#u_2\$u_3u_4$ be an splicing rule, then we can define the following operations

<u>Type I splicing operation</u>

$(x,y) \vdash_r z$ sii $x=x_1u_1u_2x_2$,
$y=y_1u_3u_4y_2$,
$z=x_1u_1u_4y_2$,

<u>Type II splicing operation</u>

$(x,y) \models_r (z,w)$ sii $x=x_1u_1u_2x_2$,
$y=y_1u_3u_4y_2$,
$z=x_1u_1u_4y_2$,
$w= y_1u_3u_2x_2$

H schemes

$\sigma = (V, R)$ where

V an alphabet
$R \subseteq V^*\#V^*\$V^*\#V^*$ a set of splicing rules

If R belongs to the family of languages L then σ is of type L

$\forall L \subseteq V^*$

$$\sigma_1(L) = \{ z \in V^* : (x,y) \vdash_r z,\ x,y \in L,\ r \in R \}$$

$$\sigma_1(x,y) = \{ z \in V^* : (x,y) \vdash_r z,\ r \in R \}$$

$$\sigma_1(L) = \bigcup_{x,y \in L} \sigma_1(x,y)$$

Language classes denoted by the H schemes (the noniterative case)

$\sigma = (V, R)$

$S_1(L_1, L_2) = \{ \sigma_1(L) : L \in L_1, R \in L_2 \}$

L_1 **is closed under splicing of type** L_2 **if** $S_1(L_1, L_2) \subseteq L_1$

Lemma For all the families of languages L_1, L_2, L'_1, L'_2 **such that** $L_1 \subseteq L'_1$ **and** $L_2 \subseteq L'_2$ **the inclusion** $S_1(L_1, L_2) \subseteq S_1(L'_1, L'_2)$ **holds.**

Language classes denoted by the H systems (the noniterative case)

L_1 \ L_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

$S_1(L_1, L_2)$

Language classes denoted by the H schemes (the iterative case)

$\sigma = (V, R) \qquad L \subseteq V^*$

$\sigma_1(L) = \{\ z \in V^* : (x,y) \vdash_r z,\ \ x,y \in L, r \in R\ \}$

$\sigma^0{}_1(L) = L$

$\sigma^{i+1}{}_1(L) = \sigma^i{}_1(L) \cup \sigma_1(\sigma^i{}_1(L)), \quad i \geq 0$

$$\sigma^*{}_1(L) = \bigcup_{i \geq 0} \sigma^i{}_1(L)$$

$H_1(\mathcal{L}_1,\mathcal{L}_2) = \{\ \sigma^*{}_1(L) : ,\ \ L \in \mathcal{L}_1, R \in \mathcal{L}_2\ \}$

Language classes denoted by the H systems
(the iterative case)

L_1 \ L_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

$H_1(L_1,L_2)$

Extended H Systems

$\sigma = (V, R)$ is an H scheme $\qquad\qquad$ $L \subseteq V^*$ is a language

$\gamma = (V, L, R)$ is a <u>H system</u>

$$L(\gamma) = \sigma^*_1(L)$$

$\gamma = (V, T, A, R)$ is an <u>extended H system</u>

- V is an alphabet
- $T \subseteq V$ is an alphabet of terminal symbols
- $A \subseteq V^*$ is a set of axioms
- $R \subseteq V^*\#V^*\$V^*\#V^*$ is a set of splicing rules

$$L(\gamma) = \sigma^*_1(A) \cap T^*$$

$$EH_1(L_1, L_2) = \{\ L(\gamma) : A \in L_1,\ R \in L_2\ \}$$

Language classes denoted by the extended H systems

L_1 \ L_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

$EH_1(L_1,L_2)$

Extended H systems with permitting contexts

γ = (V, T, A, R) is an extended H system

R is a finite set of 3-tuples in the form

$p = (r; C_1, C_2)$ $r = u_1\#u_2\$u_3\#u_4$

$C_1, C_2 \subseteq V^*$ (finite)

$(x,y) \models_p (z,w)$ iff $(x,y) \models_r (z,w)$

every element in C_1 appears in x

every element in C_2 appears in y

$$L(\gamma) = \sigma^*_2(A) \cap T^*$$

sets of permitting contexts C_1 and C_2

Lemma: $RE \subseteq EH_2(FIN,pFIN)$

set of axioms A

Splicing processors

Choudhary & Krithivasan, 2007

A splicing processor over *V* is a 8-tuple *(M,S,A,PI,FI,PO,FO,β)*,where:

M is a set of splicing rules with permitting context
S is a finite set of strings over V
A is a finite set of axioms over V
PI,FI ⊆ V are the input permitting/forbidding contexts of the processor
PO,FO ⊆ V are the output permitting/forbidding contexts of the processor
(with PI ∩ FI= ∅ and PO ∩ FO= ∅)
β ∈ {(1),(2)} defines the input/output filter

We can define the following predicates for the filters

$$rc_{(1)}(z, P, F) \equiv [P \subseteq alph(z)] \wedge [F \cap alph(z) = \emptyset]$$

$$rc_{(2)}(z, P, F) \equiv [alph(z) \cap P \neq \emptyset] \wedge [F \cap alph(z) = \emptyset]$$

Splicing processors

Manea, Martín-Vide & Mitrana, 2005

A splicing processor over *V* is a 6-tuple *(S,A,PI,FI,PO,FO)*,where:

S is a finite set of splicing rules over V
A is a finite set of auxiliary words over V
PI,FI $\subseteq$ V are the input permitting/forbidding contexts of the processor
PO,FO $\subseteq$ V are the output permitting/forbidding contexts of the processor
(with PI $\cap$ FI= $\emptyset$ and PO $\cap$ FO= $\emptyset$)

We can define the following predicates for the filters

$$rc_{(1)}(z,P,F) \equiv [P \subseteq alph(z)] \wedge [F \cap alph(z) = \emptyset]$$

$$rc_{(2)}(z,P,F) \equiv [alph(z) \cap P \neq \emptyset] \wedge [F \cap alph(z) = \emptyset]$$

Networks of Splicing Processors (NSPs)

Choudhary & Krithivasan, 2007

A NSP of size n is a tuple *$(V,N_1,N_2, \ldots, N_n,G)$*,where:

V is an alphabet
N_i is the *ith* splicing processor
G is an undirected graph without loops (the underlying topology of the network)

- The configuration of the network consists of the strings at every processor (excluding the axioms for the splicing rule)
- The network evolves as in the Networks of Evolutionary Processors (NEPs) with *splicing* steps and *communication* steps
- There exists an output processor which collects the strings as the product of a computation sequence
- The network halts whenever no splicing operation can be carried out and no string can be communicated

Accepting Networks of Splicing Processors (ANSPs)

Manea, Martín-Vide & Mitrana, 2005

An ANSP is a 9-tuple $(V,U,<,>,G,N,\alpha,x_I,x_O)$,where:

V,U are the input and network alphabets
$<,> \in U \setminus V$ are special symbols
$G=(X_G,E_G)$ is an undirected graph without loops (the underlying topology of the network)
$N: X_G \rightarrow SP_U$ associates to each node in the graph a splicing processor over U
$\alpha: X_G \rightarrow \{(1),(2)\}$ defines the type of filter at every processor
$x_I,x_O \in X_G$ are the input and output processors

- The configuration of the network consists of the strings at every processor
- The network evolves as in the Networks of Evolutionary Processors (NEPs) with *splicing* steps and *communication* steps
- The input processor initially holds the string to be analyzed
- The network halts whenever: (1) a string enters into the output processor (accepting computation) or, (2) There exists two identical configurations obtained either in consecutive splicing steps or in consecutive communication steps (not an accepting computation)

(A)NSPs are computationally complete

Choudhary & Krithivasan, 2007

Theorem. Each recursively enumerable language can be generated by a complete NSP of size two where the splicing rules are of type regular.

⬇

Simulate a type 0 Chomsky grammar which works in the same way as the $EH_2(FIN,pFIN)$ system

(A)NSPs are computationally complete

Manea, Martín-Vide & Mitrana, 2005

Theorem. For any Turing machine M there exists an ANSP that accepts exactly the same language as M does.

Simulate the movements of a Turing machine with a number of processors that linearly depends on the size of the alphabet and states of the Turing machine

(A)NSPs are computationally complete

Manea, Martín-Vide & Mitrana, 2005

Theorem. For any ANSP Γ, accepting the language L, there exists a Turing machine M that accepts the same language L.

The nondeterministic Turing machine associates every state to a node in the ANSP. The splicing rules and evolution strings are nondeterministically chosen. Whenever the Turing machine enters into the state which is associated to the output node, then it halts and accepts the input word.

Complexity issues

Manea, Martín-Vide & Mitrana, 2007

Introducing time complexity measures

We consider an ANSP Γ with the input alphabet V that halts on every input. The time complexity of the computation

$$C_0(x), C_1(x), C_2(x), \ldots, C_m(x)$$

of Γ on x is denoted by **$Time_\Gamma(x)$** and equals m.

The time complexity of Γ is the partial function from N to N,

$$\mathbf{Time_\Gamma(n) = \max\{Time_\Gamma(x) \mid |x| = n\}.}$$

For a function f : N → N we define

$Time_{ANSP}(f(n)) = \{L \mid L = L(\Gamma)$ for an ANSP Γ with $Time_\Gamma(n) \leq f(n)$ for some $n \geq n_0\}$.

$$PTime_{ANSP} = \bigcup_{k \geq 0} Time_{ANSP}(n^k)$$

Manea, Martín-Vide & Mitrana, 2007

Complexity results

Proposition. **If $L \in NP$ then $L \in PTime_{ANSP}$.**

Proposition. **If $L \in PTime_{ANSP}$ then $L \in NP$.**

$$\Downarrow$$

$$PTime_{ANSP} = NP$$

Complexity issues

Manea, Martín-Vide & Mitrana, 2007

Introducing space complexity measures

The length complexity of the computation

$$C_0(x), C_1(x), C_2(x), \dots, C_m(x)$$

of Γ on x is denoted by **$Length_\Gamma(x)$** and equals to

$$\max\{ |w| : w \in C_i(x) : 1 \leq i \leq m\}.$$

The length complexity of Γ is the partial function from N to N,

$$\mathbf{Length_\Gamma(n) = max\{Length_\Gamma(x) \mid |x| = n\}.}$$

For a function f : N → N we define

$Length_{ANSP}(f(n)) = \{L \mid L = L(\Gamma)$ for an ANSP Γ with $Length_\Gamma(n) \leq f(n)$ for some $n \geq n_0\}$.

$$PLength_{ANSP} = \bigcup_{k \geq 0} Length_{ANSP}(n^k)$$

Manea, Martín-Vide & Mitrana, 2007

Complexity results

Proposition. **If $L \in$ PSPACE then $L \in \mathrm{PLength}_{ANSP}$.**

Proposition. **If $L \in \mathrm{PLength}_{ANSP}$ then $L \in$ PSPACE.**

$$\Downarrow$$

$$\mathrm{PLength}_{ANSP} = \mathrm{PSPACE}$$