# Distributed Splicing of $\mathcal{RE}$ with 6 Test Tubes

Monika Sturm and Thomas Hinze

Dresden University of Technology, Germany
Department of Theoretical Computer Science
e-mail: {sturm,hinze}@tcs.inf.tu-dresden.de
www: http://wwwtcs.inf.tu-dresden.de/dnacomp

February 26, 2001

**Abstract**

This paper introduces a functional approach to distributed splicing systems for generation of recursive enumerable languages with 6 test tubes. The specification of this system serves both, the formal mathematical and the lab-experimental aspect. The implementation of the splicing system using a functional description of laboratory operations supports particularly the last-mentioned aspect. Advantages of this approach consist in large experimental practicability as well as in the independence of certain Chomsky type 0 grammar parameters.

## 1 Introduction

Fast solutions of combinatorial problems have both, large economical and theoretical importance. Starting from the consideration of different approaches to tackle NP-complete problems, several unconventional ideas for their solutions emerge. These ideas can be divided into four raw categories: neural, quantum, heuristic, and molecular computing, particularly using the data carrier DNA.

An interdisciplinary collaboration between computer scientists and molecular biologists developed a DNA based algorithm to solve the NP-complete knapsack problem with natural object weights and implemented it repetitively in the laboratory. In parallel to these experimental studies the laboratory-like DNA computing model DNA-Haskell[7] was developed based on biochemical processes observed in detail including some side effects that can occur indeed. DNA computing models feature by their computational completeness. DNA-Haskell also owns this property proved by simulation of selected conventional universal models for computation.

Some DNA computing models are characterized by a high abstraction level and by a clear formal model specification. Between these models and an according implementation in the laboratory a gap exists that has to be discussed. Simulating those DNA computing models using the laboratory-like DNA-Haskell could be a promising approach to fill this gap and to combine the advantages of different models. The minimization of the number of used test tubes necessary for the execution of DNA based algorithms acts as a significant criterion to optimize DNA computing models. A one pot architecture embodies the ideal case. The splicing model is closed to this ideal. The splicing operation forms the principal item of this model. It is possible to simulate the splicing operation in DNA-HASKELL. Splicing systems were established to reach universal computational power[4]. They require either an infinite set of axioms or an infinite set of splicing rules to generate recursively enumerable languages (class $\mathcal{RE}$). A further approach based on multisets leads to the necessary to determine the number of strand duplicates with high accuracy. The recent state of the art in molecular bioengineering can not meet these requirements completely. Therefore other extensions of splicing systems were sought for a practicable possibility to generate $\mathcal{RE}$. The introduction of distributed splicing systems with $n$ test tubes seems to be a successful way. A functional approach to distributed splicing of $\mathcal{RE}$ with 6 test tubes is proposed below. The arisen distributed splicing system, named TT6 for short, owns a simulation in DNA-HASKELL and performs a distribution of test tube contents after each splicing operation. Further the TT6 uses compact sets of filter patterns resulting in a comparable low exchange of DNA strands between test tubes supporting a lab-implementation.

## 2 On the Basic Operation in Splicing Systems

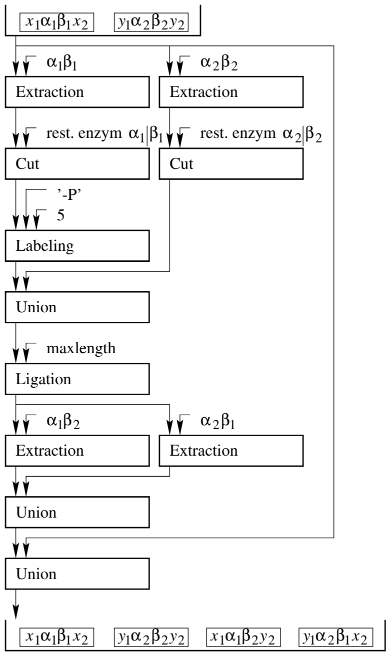
The splicing operation forms the core of all types of splicing systems and embodies an abstract formal emulation of DNA recombinant techniques cut with restriction enzymes (digestion) and ligation [4]. It is based on elements of mostly infinite sets that express DNA strands, further named words of formal languages. The description of the splicing operation on words of formal languages also leads to a generalization of the effect that is caused by digestion and ligation. The generalization suppresses certain DNA strands resp. words that can really additional occur during the ligation process as side effects. Here, we propose a sequence of DNA-HASKELL operations that simulate the splicing operation on linear data structures defined by a splicing rule in [6].

Consider an alphabet $\Sigma$, and two symbols \$ and # not in $\Sigma$. A splicing rule over $\Sigma$ is a string $r = \alpha_1\#\beta_1\$\alpha_2\#\beta_2$, where $\alpha_i, \beta_i \in \Sigma^*$, $1 \leq i \leq 2$.
For each such rule $r$ and strings $x, y, w, z \in \Sigma^*$ we define

$$(x, y) \vdash_r (z, w) \text{ if and only if } \quad \begin{array}{ll} x = x_1\alpha_1\beta_1x_2, & y = y_1\alpha_2\beta_2y_2, \\ z = x_1\alpha_1\beta_2y_2, & w = y_1\alpha_2\beta_1x_2. \end{array}$$

A wet splicing system and its experimental implementation was introduced in [9]. The results of this approach encourage the assumption that the splicing operation can be performed practically. This fact represents the first step to establish a universal DNA computer based on splicing.

Unfortunately the ligation as enzymatic process can suppress expected DNA fragments and also produce unwanted DNA fragments (e.g. strand combinations with reverse inserted fragments or additional compositions like $x_1\alpha_1\alpha_2 y_1$ and $y_2\beta_2\beta_1 x_2$, iff the single stranded overhangs resulting from the digestion are half-sided antiparallel complementary to itself with dyadic symmetry). These side effects can influence the final result of iterated executions of splicing operations, often used to generate a formal language by applying splicing rules of a splicing system. The following figure 1 proposes an idea how to overcome this insufficiency. Let $x$ and $y$ be encoded by DNA double strands.




```
xy_pool :: Tube
α1β1 :: Dnastrand
α2β2 :: Dnastrand
α1β2 :: Dnastrand
α2β1 :: Dnastrand
maxlength :: Int

combine :: Tube -> Tube
combine t = lig
              (un
                (lab
                  (cut
                    (extr t α1β1)
                    'restriction enzym α1|β1')
                  '-P'
                  5)
                (cut
                  (extr t α2β2)
                  'restriction enzym α2|β2'))
              maxlength

splicingop :: Tube -> Tube
splicingop t = un
                 (un
                   (extr
                     (combine t)
                     α1β2)
                   (extr
                     (combine t)
                     α2β1))
                 t
```

application of the function:

```
splicingop xy_pool
```

Figure 1: splicing operation as a flowchart (left) and using the DNA-HASKELL syntax (right)

We are able to describe the splicing operation in a experimental convincing way. DNA computing should lead to a unconventional universal model for computations. Splicing systems based on the splicing operation represent an exact mathematical model according to this aim [8]. A practical execution of different splicing systems can contain the scenario from figure 2 as the central part. The challenge consists in finding a way how to adapt the real molecular biological processes in the laboratory to the formal definition of the splicing operation. The focus lies in the laboratory-like modelling of splicing systems to generate regular, context free, and recursive enumerable languages.

## 3 Splicing Systems for $\mathcal{RE}$ – Comparison and Classification

The sets $\mathcal{FIN}$, $\mathcal{REG}$, $\mathcal{CF}$, $\mathcal{CS}$, $\mathcal{RE}$ are used to denote the classes of finite, regular (Chomsky type 3), context free (2), context sensitive (1), and recursive enumerable (0) languages. These classes form the Chomsky hierarchy.

To describe and characterize these classes of languages, different formal systems like Chomsky grammars of a certain type were developed. Each of these language denotation systems is able to generate exactly those words the language is composed of. Language denotation systems represent in general models for computation: Since a language can be considered as a set (finite or infinite) consisting of its words as elements, generating a language by producing its words and testing whether or not a given string is a word of the described language can be assumed as computational process. From $\mathcal{FIN}$ to $\mathcal{RE}$ the computational power increases and descriptions of $\mathcal{RE}$ are said to be computational universal. Therefore, the generation of $\mathcal{RE}$ by splicing systems based on an appropriate language denotation system will be focussed.

Some extended splicing systems $EH(\mathcal{F}_1, \mathcal{F}_2)$, $\mathcal{F}_1$: set of axioms, $\mathcal{F}_2$: set of splicing rules, can produce $\mathcal{RE}$, see table 1 [4]:

| $\mathcal{F}_1/\mathcal{F}_2$ | $\mathcal{FIN}$ | $\mathcal{REG}$ | $\mathcal{CF}$ | $\mathcal{CS}$ | $\mathcal{RE}$ |
|---|---|---|---|---|---|
| $\mathcal{FIN}$ | $\mathcal{REG}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ |
| $\mathcal{REG}$ | $\mathcal{REG}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ |
| $\mathcal{CF}$ | $\mathcal{CF}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ |
| $\mathcal{CS}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ |
| $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ | $\mathcal{RE}$ |

Table 1: computational power of $EH$ systems

Weakest preconditions are required by $EH(\mathcal{FIN}, \mathcal{REG})$ and $EH(\mathcal{CS}, \mathcal{FIN})$. These systems need either a regular language to describe the splicing rules or a context sensitive language to describe the axioms. Both, $\mathcal{REG}$ and $\mathcal{CS}$, are infinite sets resulting in an infinite number of DNA strands and/or restriction enzymes for simulating the work of these splicing systems. To overcome this insufficiency, three different ideas leading to special types of extended splicing systems were pursued:

- introduction of multisets
- extension of the basic data structure "finite linear string"
- introduction of distributed splicing systems usually with more than one test tube and control mechanisms for the resulting test tube systems

Table 2 illustrates extended splicing systems able to generate the class of recursive enumerable languages together with a short characterization of some properties. Further considerations focus on classes $\mathcal{FIN}$, $\mathcal{REG}$, and $\mathcal{RE}$.

| class | splicing system | reference | number of test tubes | number of axioms | number of splicing rules | basic data structure | required language denotation | supports multiple instructions |
|---|---|---|---|---|---|---|---|---|
| distributed | Multiple | [12] | 1 | $O(\|\mathcal{M}\|)$ | $O(\|\mathcal{D}\|^3)$ | linear | Elementary Formal System (EFS) $EFS = (\mathcal{D}, \Sigma, \mathcal{M})$, Post Normal System (PNS) $G = (\mathcal{V}, \Sigma, \mathcal{P}, \mathcal{A})$ | no |
| | TT system | [2] | $O(\|\Sigma_G\|)$ | $O(\|\mathcal{P}_G\| + \|\Sigma_G\|)$ | $O(\|\mathcal{P}_G\| + \|\Sigma_G\|)$ | linear | Chomsky type 0 grammar $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ | yes |
| | CDEH system | [10] [11] | 3 | $O(\|\mathcal{P}_G\| + \|\mathcal{V}_G\| + \|\Sigma_G\|)$ | $O(\|\mathcal{P}_G\| + \|\mathcal{V}_G\| + \|\Sigma_G\|)$ | linear | Chomsky type 0 grammar $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ | yes |
| | TT6 system | here | 6 | $O(\|\mathcal{P}_G\| + \|\Sigma_G\|)$ | $O(\|\mathcal{P}_G\| + \|\Sigma_G\|)$ | linear | Chomsky type 0 grammar $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ | yes |
| infinite set | $EH(\mathcal{FIN}, \mathcal{REG})$ | [10] | 1 | $O(\|\mathcal{P}_G\| + \|\mathcal{V}_G\| + \|\Sigma_G\|)$ | $\infty$ | linear | Chomsky type 0 grammar $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ | no |
| extended data structure | Circular | [13] | 1 | $O(\|\mathcal{P}\| + \|\mathcal{A}\|)$ | $O(\|\mathcal{P}\|)$ | circular | Post Normal System $G = (\mathcal{V}, \Sigma, \mathcal{P}, \mathcal{A})$ | no |
| multiset | $EH(m\mathcal{FIN}, \mathcal{FIN})$ | [1] | 1 | $O(\|\mathcal{P}_G\| + \|\mathcal{V}_G\| + \|\Sigma_G\|)$ | $O(\|\mathcal{P}_G\| \cdot (\|\mathcal{V}_G\| + \|\Sigma_G\|)^5)$ | linear | Chomsky type 0 grammar $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ | no |
| | | [5] | 1 | $O(\|\mathcal{V}\|^2\|\mathcal{Q}\| + \|\mathcal{Q}\|^2\|\mathcal{V}\|)$ | $O(\|\mathcal{V}\|^4\|\mathcal{Q}\|)$ | linear | deterministic Turing Machine $TM = (\mathcal{Q}, \mathcal{V}, \Sigma, \{L, R\}, \mathcal{F}, q_0, \varepsilon, \delta)$ | no |
| | | [3] | 1 | $O(\|\Sigma\| \cdot \|\mathcal{P}\|)$ | $O(\|\Sigma\| \cdot \|\mathcal{P}\|)$ | linear | Post Normal System $G = (\mathcal{V}, \Sigma, \mathcal{P}, \mathcal{A})$ | no |

Table 2: classification of splicing systems able to generate recursive enumerable languages

Table 2 shows that TT6 as a model for distributed splicing is characterized by a linear complexity of the number of axioms and splicing rules depending on the number of grammar rules and terminal symbols of the language. Further the TT6 uses 6 test tubes in any case working on a linear DNA data structure.

# 4 A Test Tube 6 Distributed Splicing System $\Gamma$ for Chomsky Type 0 Grammars

The definition of $\Gamma$ uses the components of Chomsky type 0 grammars that have to be given for the construction of a concrete $\Gamma$.

Let $G = (\mathcal{V}_G, \Sigma_G, \mathcal{P}_G, S_G)$ be a Chomsky type 0 grammar in Kuroda normal form with

| | | |
|---|---|---|
| $\mathcal{V}_G$: | finite set of nonterminal symbols | |
| $\Sigma_G$: | finite set of terminal symbols | $\Sigma_G = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}, \quad \mathcal{V}_G \cap \Sigma_G = \emptyset$ |
| $\mathcal{P}_G$: | finite set of productions | $\mathcal{P}_G \subseteq (\mathcal{V}_G \times (\mathcal{V}_G \otimes \mathcal{V}_G)) \cup (\mathcal{V}_G \times \Sigma_G) \cup (\mathcal{V}_G \times \{\varepsilon\}) \cup ((\mathcal{V}_G \otimes \mathcal{V}_G) \times (\mathcal{V}_G \otimes \mathcal{V}_G))$ |
| $S_G$: | start symbol | $S_G \in \mathcal{V}_G$ |

All words $\in \mathcal{L}(G)$ have to be strictly generated in a right associative way to apply the productions of the grammar to the intermediate derivation results $\in (\mathcal{V}_G \cup \Sigma_G)^* \setminus \Sigma_G^*$. That means, the rightmost part is always replaced by the next production. Otherwise the set of splicing rules and axioms has to be extended.

Let $\Gamma = (\mathcal{V}, T_1, T_2, T_3, T_4, T_5, T_6)$ be a Test Tube Distributed Extended Head System of degree 6 (TT6 for short) based on $G$ with

| | | |
|---|---|---|
| $\mathcal{V}$: | finite set of alphabet symbols | $\mathcal{V} = \mathcal{V}_G \cup \Sigma_G \cup \{B, \alpha, \beta, X, X', Y, Y', Y'_\alpha, Y'_\beta, Z, Z', Z''\}$ |
| $T_i$: | test tube $i$, $i = 1, \ldots, 6$ with | $T_i = (\mathcal{A}_i, \mathcal{R}_i, \mathcal{F}_i)$ |
| $\mathcal{A}_i$: | finite set of axioms | $\mathcal{A}_i \subset \mathcal{V}^*$ |
| $\mathcal{R}_i$: | finite set of splicing rules | $\mathcal{R}_i \subset \mathcal{V}^* \otimes \{\#\} \otimes \mathcal{V}^* \otimes \{\$\} \otimes \mathcal{V}^* \otimes \{\#\} \otimes \mathcal{V}^*$ |
| $\mathcal{F}_i$: | finite set of filter patterns | $\mathcal{F}_i \subset (\mathcal{V} \cup \{@\})^*$, with an ambiguous letter @, $\{@\}^* \equiv \{@\}$, that stands for any finite word $\in \mathcal{V}^*$ |
| $\varepsilon$: | empty word | |

Each test tube $T_i$ is called a *component* of $\Gamma$. Any $\mathcal{A}_i$ can be represented as finite languages over $\mathcal{V}$, and any $\mathcal{R}_i$ can be represented as finite languages over $\mathcal{V} \cup \{\#\} \cup \{\$\}$. # and $ are auxiliary symbols not in $\mathcal{V}$. For simplicity, the operation $^+$ on an arbitrary set $\mathcal{U}$ is defined as $\mathcal{U}^+ := \mathcal{U}^* \otimes \mathcal{U}$. It excludes $\varepsilon$ from the result of the $^*$-operation. The components of $\Gamma$ are defined as follows:

$$
\begin{aligned}
T_1 &= (\mathcal{A}_1, \mathcal{R}_1, \mathcal{F}_1)\\
&\mathcal{A}_1 = \{XBS_GY\} \cup \{ZvY' \mid \exists u \in (\mathcal{V}_G \cup \Sigma_G)^+.(u,v) \in \mathcal{P}_G\}\\
&\mathcal{R}_1 = \{r_{11}\}\\
&\qquad r_{11} = \varepsilon\#uY\$Z\#vY';\ (u,v) \in \mathcal{P}_G\\
&\mathcal{F}_1 = \{X@Y\}
\end{aligned}
$$

$$
\begin{aligned}
T_2 &= (\mathcal{A}_2, \mathcal{R}_2, \mathcal{F}_2)\\
&\mathcal{A}_2 = \{Z\beta\alpha^i\beta Y', X\sigma_i Z \mid i = 1,\ldots,n\} \cup \{ZY'_\alpha, ZY'_\beta, X'Z, ZY\}\\
&\mathcal{R}_2 = \{r_{21}, r_{22}, r_{23}, r_{24}, r_{25}, r_{26}\}\\
&\qquad r_{21} = \varepsilon\#\sigma_i Y'\$Z\#\beta\alpha^i\beta Y';\ i = 1,\ldots,n\\
&\qquad r_{22} = \varepsilon\#\beta Y'\$Z\#Y'_\beta\\
&\qquad r_{23} = \varepsilon\#\alpha Y'\$Z\#Y'_\alpha\\
&\qquad r_{24} = X\#\varepsilon\$X'\#Z\\
&\qquad r_{25} = X\beta\alpha^i\beta\#\varepsilon\$X\sigma_i\#Z;\ i = 1,\ldots,n\\
&\qquad r_{26} = \varepsilon\#Y'\$Z\#Y\\
&\mathcal{F}_2 = \{X@Y', X@Y'_\alpha, X@Y'_\beta, X\beta@, X\alpha@\}
\end{aligned}
$$

$$
\begin{aligned}
T_3 &= (\mathcal{A}_3, \mathcal{R}_3, \mathcal{F}_3)\\
&\mathcal{A}_3 = \{ZY', X\alpha Z\}\\
&\mathcal{R}_3 = \{r_{31}, r_{32}\}\\
&\qquad r_{31} = \varepsilon\#Y'_\alpha\$Z\#Y'\\
&\qquad r_{32} = X'\#\varepsilon\$X\alpha\#Z\\
&\mathcal{F}_3 = \{X'@Y'_\alpha\}
\end{aligned}
$$

$$
\begin{aligned}
T_4 &= (\mathcal{A}_4, \mathcal{R}_4, \mathcal{F}_4)\\
&\mathcal{A}_4 = \{ZY', X\beta Z\}\\
&\mathcal{R}_4 = \{r_{41}, r_{42}\}\\
&\qquad r_{41} = \varepsilon\#Y'_\beta\$Z\#Y'\\
&\qquad r_{42} = X'\#\varepsilon\$X\beta\#Z\\
&\mathcal{F}_4 = \{X'@Y'_\beta\}
\end{aligned}
$$

$$
\begin{aligned}
T_5 &= (\mathcal{A}_5, \mathcal{R}_5, \mathcal{F}_5)\\
&\mathcal{A}_5 = \{Z'Z', Z''Z''\}\\
&\mathcal{R}_5 = \{r_{51}, r_{52}\}\\
&\qquad r_{51} = \varepsilon\#BY\$Z'Z'\#\varepsilon\\
&\qquad r_{52} = X\#\varepsilon\$\varepsilon\#Z''Z''\\
&\mathcal{F}_5 = \{@BY\}
\end{aligned}
$$

$$
\begin{aligned}
T_6 &= (\mathcal{A}_6, \mathcal{R}_6, \mathcal{F}_6)\\
&\mathcal{A}_6 = \emptyset\\
&\mathcal{R}_6 = \emptyset\\
&\mathcal{F}_6 = \Sigma_G^+
\end{aligned}
$$

The work of $\Gamma$ can be described as an iterated loop in each of the test tubes $T_i$, $i = 1,\ldots,5$. The iterated loop consists of the consecutive executed steps splicing, filtering, and distributing. The test tube $T_6$ has the function of a final tube and stores the resulting words $\in \mathcal{L}(G)$. The current contents of test tube $T_i$ is denoted as $\mathcal{L}_i$. Initially, $\mathcal{L}_i = \mathcal{A}_i$ with $i = 1, \ldots, 6$.

The iterated loop in each test tube $T_i$, $i = 1, \ldots, 5$ starts with the according sets of axioms $\mathcal{A}_i$. Subsequently, any executable splicing rule has been selected and applied. If there is no applicable splicing rule, no splicing will be performed and the set of strings in the relevant test tube remains unchanged. The application of a splicing operation in a test tube $T_i$ with the contents $\mathcal{L}_i \subset \mathcal{V}^*$ is defined by the function $\sigma$:

$$\sigma(\mathcal{L}_i) := \{z \in \mathcal{V}^* \mid (x, y) \vdash_r (z, w) \text{ or } (x, y) \vdash_r (w, z), \text{ for some } x, y \in \mathcal{L}_i, r \in \mathcal{R}_i\}$$

During the splicing step of the iterated loop the splicing operation is performed at most once in $T_i$. In consequence, the $k$-fold iterated splicing $\sigma^k(\mathcal{L}_i)$ is defined by

$$\begin{aligned} \sigma^0(\mathcal{L}_i) &= \mathcal{L}_i \\ \sigma^{k+1}(\mathcal{L}_i) &= \sigma^k(\mathcal{L}_i) \cup \sigma(\sigma^k(\mathcal{L}_i)) \text{ for } k \geq 0,\ 1 \leq i \leq 5 \end{aligned}$$

with

$$\sigma^*(\mathcal{L}_i) = \bigcup_{k \geq 0} \sigma^k(\mathcal{L}_i).$$

During one pass of the iterated loop the splicing operation is executed at most once in each test tube $T_1$ until $T_5$.

After splicing, the filtering step prepares copies of those strings that have to be distributed. Every test tube $T_i$, $i = 1, \ldots, 5$ provides separately exactly those strings that will be moved into other test tubes. To do so, $T_i$ evaluates all filter pattern $\mathcal{F}_j$, $j = 1, \ldots, 5$, $j \neq i$. Those strands that are transmitted into other tubes are removed from the producing tube $T_i$ if they do not match its own filter pattern $\mathcal{F}_i$. Each $\mathcal{F}_i$ describes those strings that are moved from other test tubes into $T_i$.

The subsequent distributing step exchanges the strings prepared by filtering between the test tubes.

The test tube $T_6$ receives copies of all strings produced by splicing in $T_1$ until $T_5$ during each iterated loop and collects those strings that describe words of the language $\mathcal{L}(G)$, implemented by its filter pattern $\mathcal{F}_6$. All other arriving strings are eliminated. $T_6$ does not practise a splicing operation itself, it evaluates the produced words from all other test tubes. Because of this behavior $T_6$ carries the meaning of a final tube (master tube).

The steps splicing, filtering, and distributing forming the iterated loop are executed consecutively. After distributing, the next round of splicing starts. All test tubes $T_1$ until $T_5$ perform iterated loops in parallel. The number of iterated loops is not limited by $\Gamma$. One pass of the iterated loop transforms stepwise the contents $(\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_6)$ into $(\mathcal{L}'_1, \mathcal{L}'_2, \ldots, \mathcal{L}'_6)$, denoted by the $\xrightarrow{1}_{TT6}$ operator

$$(\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_6) \xrightarrow{1}_{TT6} (\mathcal{L}'_1, \mathcal{L}'_2, \ldots, \mathcal{L}'_6)$$

with

$$\mathcal{L}'_i = \left( \bigcup_{\substack{j=1 \wedge \\ j \neq i}}^{6} \sigma^k(\mathcal{L}_j) \right) \cap \mathcal{F}_i \cup \left( \sigma^k(\mathcal{L}_i) \setminus \left( \bigcup_{\substack{j=1 \wedge \\ j \neq i}}^{6} \mathcal{F}_j \right) \right), \; k \in \{0,1\}, \; 1 \leq i \leq 6.$$

The contents of each $T_i$ is composed by the result of the own splicing minus those strings that move to other test tubes plus those strings that arrive from other test tubes.

Therefore, the language $\mathcal{L}$ generated by $\Gamma$ is described by

$$\mathcal{L}(\Gamma) = \left\{ w \in \Sigma_G^+ \;\wedge\; w \in \mathcal{L}_6 \mid (\mathcal{A}_1, \ldots, \mathcal{A}_6) \xrightarrow{*}_{TT6} (\mathcal{L}_1, \ldots, \mathcal{L}_6) \right\},$$

where $\xrightarrow{*}_{TT6}$ is the reflexive and transitive closure of $\xrightarrow{1}_{TT6}$.

Each of the splicing test tubes assumes a special task in the whole behavior of $\Gamma$, listed in table 3.

| **test tube** | **task** |
|---|---|
| $T_1$ | • apply a production of the grammar $G$ |
| $T_2$ | • encode the rightmost completing terminal symbol $\sigma_j \in \Sigma_G$ into the sequence $\beta\alpha^j\beta$ if existing<br>• prepare the rightmost-leftmost rotation of the completing $\alpha$ or $\beta$<br>• decode the leftmost complete rotated $\beta\alpha^j\beta$ into the terminal symbol $\sigma_j$<br>• enable consecutive applications of productions without rotation in between |
| $T_3$ | • rightmost-leftmost rotation of one symbol $\alpha$ |
| $T_4$ | • rightmost-leftmost rotation of one symbol $\beta$ |
| $T_5$ | • extract a ready generated word of the language $\mathcal{L}(G)$ from the terminating auxiliary symbols $X$ and $BY$ |

Table 3: tasks of the splicing test tubes

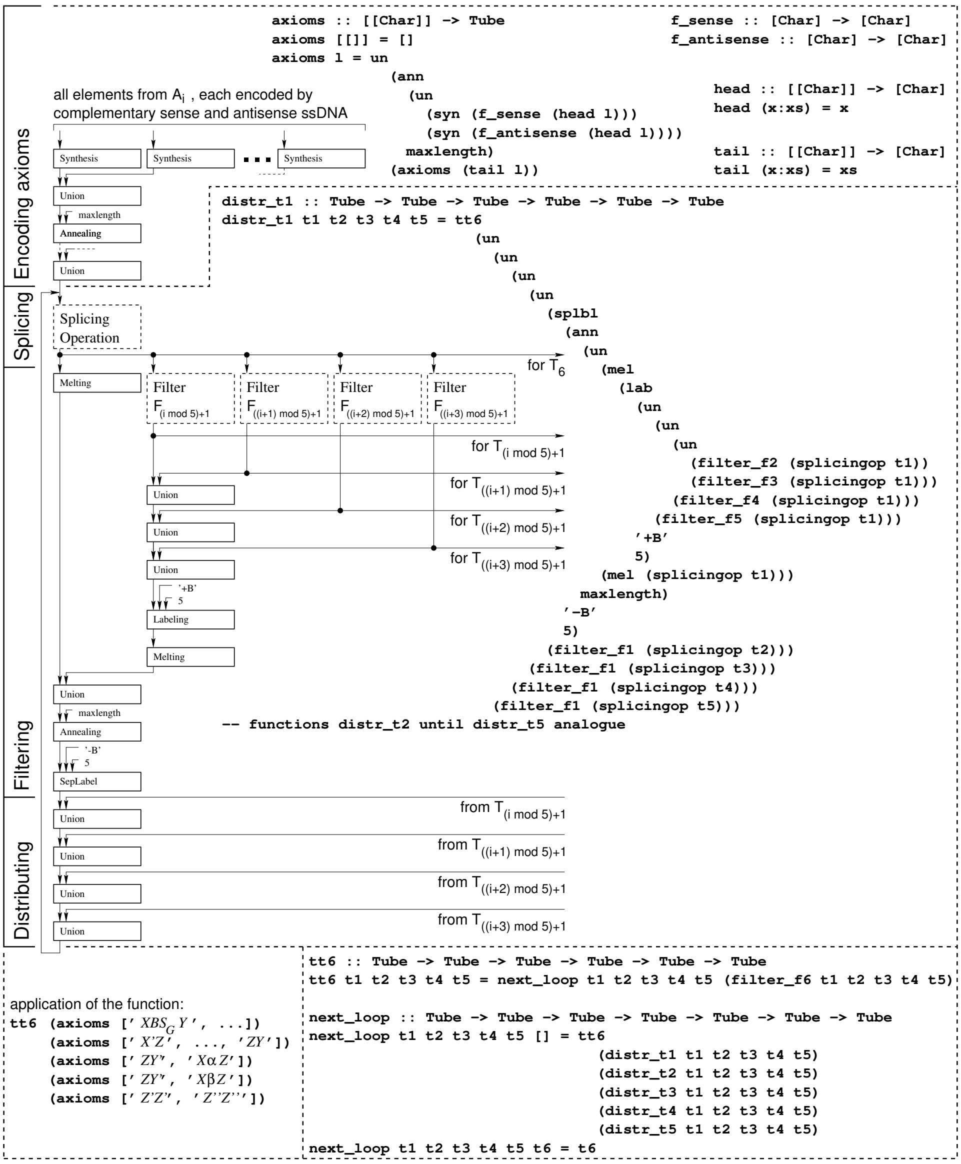



Figure 2: iterated loop for $T_i$, $i = 1, \ldots, 5$; flowchart and DNA-HASKELL syntax [7]

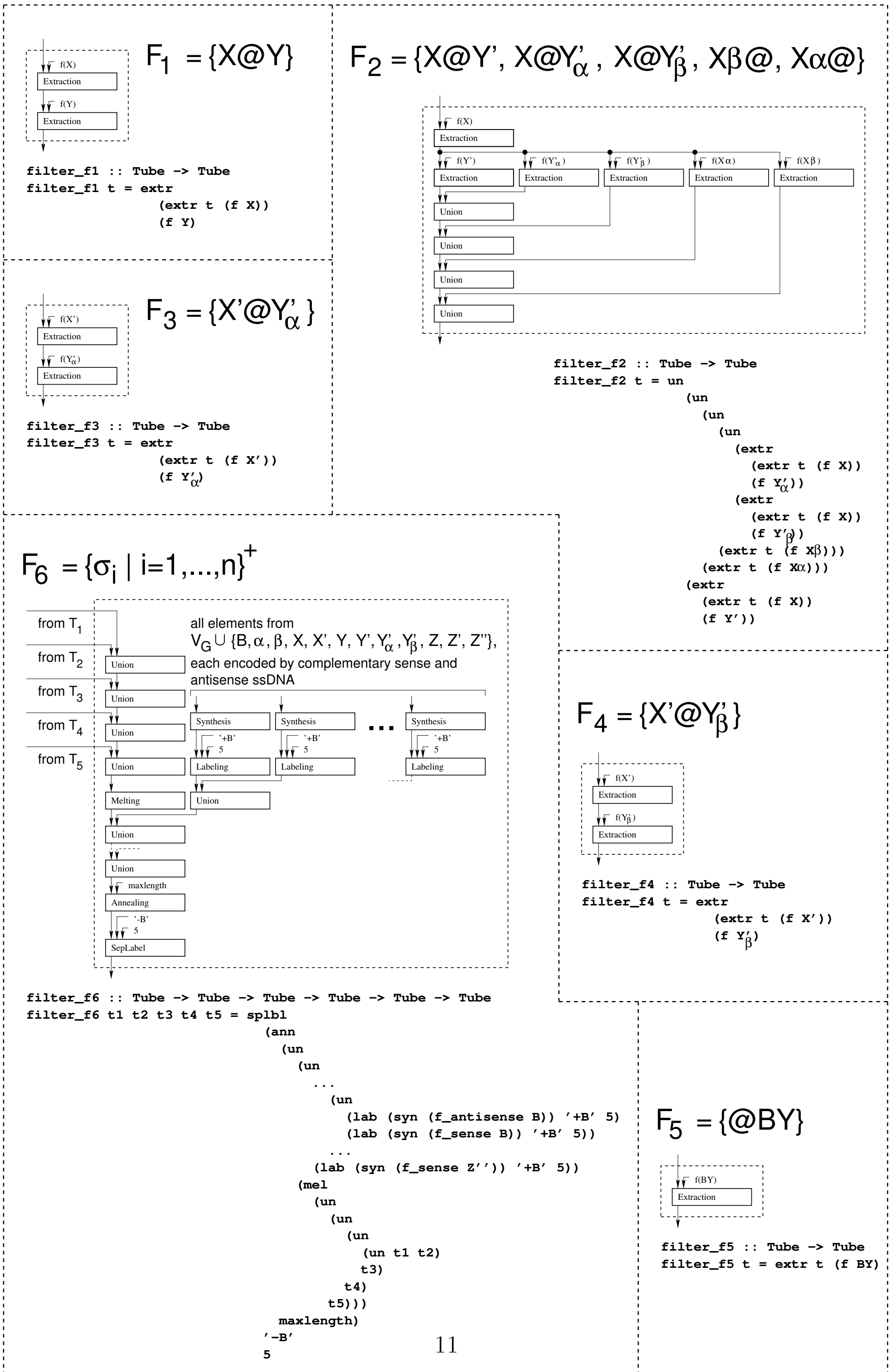



Figure 3: filter patterns $\mathcal{F}_1$ until $\mathcal{F}_6$ and implementation of filtering processes including collection of words $\in \mathcal{L}(G)$ in $T_6$; flowchart and DNA-HASKELL syntax

Figure 2 shows the general iterated loop practised in $T_1$ until $T_5$. Each iterated loop in $T_i$, $i = 1, \ldots, 5$ has been initialized with the axioms from the according set $\mathcal{A}_i$ encoded by appropriate unique DNA double strands. The encoding is done by two one to one functions `f_sense` and `f_antisense` that produce complementary single stranded DNA strings from any $\in \mathcal{A}_i$ and from any $\in \mathcal{V}$. The body of each iterated loop is composed by steps splicing, filtering, and distributing. Each splicing step executes the splicing operation according to $\mathcal{R}_i$ at most once. The subsequent filtering step prepares in each $T_i$ copies of those DNA strands according to filter patterns $\mathcal{F}_j$, $j \neq i$ separately. All prepared DNA strands matching $\mathcal{F}_j$ are distributed to $T_j$. $\mathcal{F}_6$ extracts those DNA strands representing an arbitrary word $\in \mathcal{L}(G)$ and collects it in $T_6$. The iterated loops of TT6 terminate as soon as an arbitrary word $\in \mathcal{L}(G)$ exists in $T_6$. The case $\mathcal{L}(G) = \emptyset$ leads to a nonterminating process. This consequence coincides to a terminating property of programs with mutual recursion.

# 5 Conclusions

This paper implies a proposal to the discussion about distributed splicing systems. The objectives leading to the development of TT6 include the compliance with $\mathcal{RE}$ by a constant number of test tubes (6), by a nonextended DNA structure, and by an efficient derivation of complexity theoretical relevant system parameters directly from the grammar. Let $\Sigma_G$ be the set of terminal symbols and $\mathcal{P}_G$ the set of grammar productions, then TT6 requires $O(|\mathcal{P}_G| + |\Sigma_G|)$ axioms and splicing rules independent of the number of nonterminal grammar symbols.

TT6 is constructed with regard to a practicable implementation in the laboratory. The distribution of DNA strands between test tubes is organized in a way that minimizes the number of transferred DNA strands. Beyond only few strand duplicates are necessary to perform all filtering and distributing processes. The number of DNA double strands that have to be available initially is equal to the number of axioms. The operations defined in the specification of DNA-Haskell are based on observable processes in the laboratory. The experimental practicability of each single operation was shown.

# References


[1] E. Csuhaj-Varju, R. Freund, L. Kari, G. Păun. DNA computing based on splicing: universality results. *Proc. of First Annual Pacific Symp. on Biocomputing*, Hawaii, 1996 (L. Hunter, T. E. Klein, eds.), World Sci. Publ., Singapore, p. 179–190, 1996

[2] E. Csuhaj-Varj, L. Kari, G. Păun. Test tube distributed systems based on splicing. *Computers and AI*, vol. 15(2–3), p. 211-232, 1996

[3] C. Ferretti, G. Mauri, S. Kobayashi, T. Yokomori. On the universality of Post and splicing systems. *Proc. Second Intern. Coll. Universal Machines and Compu-*

*tations*, vol. II, p. 12–28, Metz, 1998, and *Theoretical Computer Sci.*, vol. 231(2), p. 157–170, 2000

[4] R. Freund, L. Kari, G. Păun. DNA computing based on splicing: the existence of universal computers. *Theory of Computing Systems*, vol. 32, p. 69–112, 1999

[5] P. Frisco, G. Mauri, C. Ferretti. Simulating Turing machines through extended mH systems, *Computing with Bio-Molecules. Theory and Experiments* (G. Păun, ed.), Springer-Verlag, Singapore, p. 221–238, 1998

[6] T. Head. Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of the Mathematical Biology*, vol. 49(6), p. 737–759, 1987

[7] T. Hinze, M. Sturm. A universal functional approach to DNA computing and its experimental practicability. *PreProceedings 6th DIMACS Workshop on DNA Based Computers*, University of Leiden, Leiden, The Netherlands, p. 257, 2000

[8] L. Kari. DNA computing: the arrival of biological mathematics. *The mathematical Intelligencer*, vol. 19(2), 1997

[9] E. Laun, K. J. Reddy. Wet splicing systems. *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, University of Pennsylvania, p. 115–126, 1997

[10] C. Martin-Vide, G. Păun. Cooperating distributed splicing systems. *J. Automata Languages and Combinatorics*, vol. 4(1), p. 3–16, 1999

[11] G. Păun, G. Rozenberg, A. Salomaa. DNA Computing. *New Computing Paradigms*, Springer-Verlag, 1998

[12] Y. Sakakibara. Splicing, tree splicing, and multiple splicing, *Workshop on Molecular Computing*, Chennai, India, (K. Krithivasan, R. Rama, eds.), p. 76–95, December 1998

[13] T. Yokomori, S. Kobayashi, C. Ferretti. On the power of circular splicing systems and DNA computability, *IEEE Intern. Conf. on Evolutionary Computing*, Indianapolis, p. 219–224, 1997

[14] C. Zandron, C. Ferretti, G. Mauri. A reduced distributed splicing system for $\mathcal{RE}$ languages. In G. Păun and A. Salomaa, editors, *Lecture Notes in Computer Science*, vol. 1218, p. 346–366. Springer Verlag, Berlin, Heidelberg, New York, 1997