UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Universitat Rovira i Virgili
Facultat de Lletres
Departament de Filologies Romàniques

Finite Models of Splicing and their Complexity

PhD Dissertation

Presented by
Remco LOOS

Supervised by
Victor MITRANA
Research Group on Mathematical Linguistics
Universitat Rovira i Virgili
and
Faculty of Mathematics and Computer Science
University of Bucharest

Tarragona, 2007

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Acknowledgements

I would like to open this thesis with a few words of thanks to those who helped me throughout the period in which this research was conducted, either by contributing in some way to said research or by helping me to maintain my sanity in the process.

In a roughly chronological order, thanks to Carlos Martín-Vide for creating the circumstances which made this thesis possible and for his support throughout my PhD. For my basis in formal language theory, I am indebted to the many excellent teachers in the PhD School in Formal Languages and Applications, as well as to the venerable Hopcroft and Ullman. Also thanks to Gabriela Martín for stimulating scientific and extra-scientific discussions, and for showing that mathematicians are just like normal people. I am very grateful to my supervisor, Victor Mitrana, for his constant willingness to contribute to my research, be it in the form of collaboration, feedback or advice, and for his general guidance in the world of science.

I gratefully acknowledge the financial support provided by PFI fellowship BES-2004-6316 of the Spanish Ministry of Education and Science, which in addition funded my research stays in Rochester, Frankfurt and Milan.

Big thanks are due to Mitsunori Ogihara, for hosting me during my very instructive stay at the University of Rochester, and for his important contributions and tireless dedication to our joint work. At the University of Frankfurt, I owe gratitude to Detlef Wotschke for hosting me and for the many inspiring discussions, and to Andreas Malcher for his great help in pointing me to the relevant literature and the endless stream of useful ideas and suggestions. Also thanks to Paola Bonizzoni for her hospitality and helpfulness during my stay at Milan Bicocca University.

Thanks to my friends and colleagues at the GRLMC (Mihai Ionescu, Szilard Fazekas, and a long etcetera), for the useful discussions in the scientific realm, and most of all for making these years in Tarragona thoroughly enjoyable. I also thank my other friends for their moral support and friendship. Special words of thanks for Miranda Lubbers for her constant interest for my work, encouragement and reassurance, and to Yvonne de Boer, for leading the way of heroic courage in this world made up of second-hand lenses and parents in car boots.

Finally, I must mention the inspiration provided by my wife Raquel. Probably every scientist harbours doubts at whether his work will have any real relevance and impact beyond the few people working in his field. In this sense, she has been priceless by revealing the potential of this work as a potent cure against insomnia.

Tarragona, September 2007
Remco Loos

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Table of Contents

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

1 Introduction

Over the last two decades, a tight collaboration has emerged between computer scientists, biochemists and molecular biologists. Computational methods have been essential in recent advances in molecular biology, such as the sequencing of the human genome. On the other hand, working with biochemists and biologists has changed computer scientists' perceptions about the nature of computing. In fact, many biological phenomena can be viewed as computational processes: storing genetic information in DNA molecules, protein synthesis, transport of molecules between cells, even evolution.

This realization has spurred research into an area known as DNA computing (also biomolecular computing or more generally natural computing). This is a multidisciplinary area, which has two major lines of research. The first line is of experimental research, using biomolecular operations to perform actual computations. The second is a theoretical line aiming at finding formal models, algorithms and paradigms, both to describe biological processes and to find new modes of computation, based on or inspired by these processes.

The work in this thesis belongs to the second line, and studies a computational model called *splicing system*. Splicing is the formal model of the cutting and recombination of DNA molecules under the influence of restriction enzymes. This process works as follows: Two DNA molecules are cut at specific subsequences and the first part of one molecule is connected to the second part of the other molecule, and vice versa. This process can be formalized as an operation on strings, described by a so-called *splicing rule* of the form

$$u_1\#u_2\$v_1\#v_2.$$

Here, u_1u_2 and v_1v_2 are the subsequences in question and the cuts are located between u_1 and u_2 and v_1 and v_2.

These rules are the basis of a computational model (language generating device) called *splicing system* or also *H system* after its inventor, Tom Head. A splicing system generates a language by starting with some initial set of strings. Then it applies a set of splicing rules and adds the newly created strings to the set. Iterating this process, we obtain a new language.

This thesis presents original work in the field of splicing systems, which, as the title already indicates, can be roughly divided into two parts: 'Finite models of splicing' on the one hand and 'their complexity' on the other.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Before presenting this work, we first introduce the basic notions and necessary definitions of formal language theory, DNA computing and splicing (Chapter 2).

In the first part, which covers Chapters 3 through 5, we propose and study new definitions of finite splicing systems. To situate the contribution of this part, it is helpful to recall the situation of the splicing field at the beginning of the work of this thesis. Simplifying slightly, it was known that the biologically more realistic basic, finite splicing systems have very little computational power (they generate only regular languages), whereas to achieve the same power as a Turing machine either an infinite set of rules or some (typically biologically unrealistic) additional control feature is needed. Many such control features have been proposed and studied. Here, we take a different approach: Starting from the original cutting and recombination operation, we study alternatives to the traditional splicing definition, rooted in the biochemical working of the operation.

In Chapter 3, we introduce an alternative definition of splicing systems, where we allow for the possibility of replacing strings instead of only adding (which is possible in biochemistry). We show that in this case, basic finite systems are computationally complete. We also introduce the notion of *delay* which allows to connect these new systems with the traditional ones. We extend our approach to other existing definitions which share the property of not maintaining all strings, generalizing them under the notion of non-preserving splicing. Chapter 4 casts a closer look at one of these definitions, namely time-varying H systems, a distributed variant of splicing systems. Since it is known that this non-preserving language definition by itself is already powerful enough for computational completeness (thus making the distributed architecture superfluous), we introduce and study weaker definitions in this framework. Finally, in Chapter 5, we consider the possibility (again suggested by biochemical reality) of simultaneously applying more than one splicing rule to a string. We present several formalizations of this process and study their computational power.

In the second part, we study the complexity of splicing systems, considering both computational complexity (quantifying the resources needed to perform computations) and descriptional complexity (dealing with the conciseness of description of formal objects). Strangely enough, though these complexity issues are very natural questions to ask about computational models, these questions had not been addressed before. This means that to investigate the complexity of splicing systems, first a framework for studying these issues had to be introduced, and appropriate complexity measures had to be defined. Using these measures, splicing systems are compared in terms of complexity with known models like finite automata or Turing machines.

In Chapter 6, we introduce a notion of time complexity for splicing systems in terms of the number of 'rounds' of rule applications that are needed to generate the word. Extending this notion to numbers by regarding the maximal complexity of a word of length n, we can define complexity classes. These classes are characterized in terms of well-known Turing machine-based complexity classes. Chapter 7 extends this approach to space complexity, defining space complexity as the size of the minimal la-

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

beled production tree of a word with respect to a given splicing system. Also here, complexity classes are defined and characterized. In Chapter 8 we explore the possibility of using splicing systems as problem solvers. For this we introduce the notion of an accepting splicing system. We show that this variant can be used to efficiently solve NP-complete problems. The descriptional complexity of splicing systems is addressed in Chapter 9 we compare the conciseness of descriptions of regular languages by finite automata and both generating and accepting finite splicing systems. Finally, in Chapter 10 we draw some general conclusions and present suggestions for further research.

Most of the work presented has been published. Sections 3.1 to 3.3 are based on an article in Theoretical Computer Science [38]. Sections 3.5 and 3.6 describe work published in the International Journal of Computer Mathematics [43]. Chapter 4 presents results published in the Journal of Universal Computer Science [39] and Chapter 5 work presented at the International Meeting on DNA Computing in Memphis [44].

The work in Chapter 6, except section 6.5, is based on an article in Theoretical Computer Science [46] (an earlier version was presented at Developments in Language Theory 2007 [45]). Chapter 7, as well as section 6.5, present work submitted for journal publication [47]. Chapter 8 describes results presented at Parallel Problem Solving from Nature [42]. Finally, Chapter 9 is based on an article submitted for journal publication [41], an earlier version of which was presented at Descriptional Complexity of Formal Systems 2007 [40].

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

2 Prerequisites

2.1 Formal Language Prerequisites

2.1.1 Basic Notions

A *string* or equivalently a *word* is a finite sequence of symbols. The number of symbols or *length* of a word w is denoted by $|w|$. The word of zero length is called the *empty word* and denoted by λ.

An *alphabet* is a finite and nonempty set of symbols. A word consisting only of symbols from an alphabet Σ is called a word over Σ. A *(formal) language* is a set of words over some alphabet. The set of all strings over Σ is denoted by Σ^* and the set of all non-empty strings by Σ^+. For a word w, the minimal alphabet W such that $w \in W^*$ is written as $alph(w)$.

A *relation* R over a set S is a set of pairs (a, b) such that $a, b \in S$. Instead of $(a, b) \in R$ we will also write aRb. A relation R over S is *reflexive* if aRa for all $a \in S$, *transitive* if aRb and bRc imply aRc and *symmetric* if aRb implies bRa. For a set $\mathcal{P}$ of such properties, the $\mathcal{P}$-closure of a relation R is the smallest set R' such that $R \subseteq R'$ and R' possesses the properties in $\mathcal{P}$.

2.1.2 Grammars and the Chomsky Hierarchy

A *(Chomsky or generative) grammar* is a quadruple $G = (T, N, S, P)$, where T is the alphabet of *terminals*, N is the alphabet of *non-terminals* disjoint from T, and $S \in N$ is the *start symbol*. The set P of *productions* or *rules* is a subset of $(T \cup N)^* \times (T \cup N)^*$. If $(u, v) \in P$, we also write $u \to v$.

For such a grammar G we define a derivation relation $\Rightarrow$ as follows: for words $x, y \in (T \cup N)^*$ we write $x \Rightarrow y$ iff $x = x_1 u x_2$ and $y = x_1 v x_2$, for $x_1, x_2 \in (T \cup N)^*$ and $u \to v \in P$.

Let $\Rightarrow^*$ denote the reflexive and transitive closure of this relation. Then the language generated by G is defined as $L(G) := \{w : w \in T^* \mid S \Rightarrow^* w\}$.

Chomsky grammars can be classified according to restrictions on the form of their rule. Specifically, a grammar $G = (T, N, S, P)$ is called

- *regular* iff all rules in P are of the form $A \to xB$ for $A \in N$, $B \in N \cup \{\lambda\}$ and $x \in T$,
- *linear* iff all rules in P are of the form $A \to xBy$ for $A \in N$, $B \in N \cup \{\lambda\}$, and $x, y \in T \cup \{\lambda\}$,

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- *context-free* iff all rules in P are of the form $A \to v$ for $A \in N$, and $v \in (T \cup N)^*$,
- *context-sensitive* iff all rules in P are of the form $xAy \to xvy$ for $A \in N$, $v \in (T \cup N)^+$ and $x, y \in (T \cup N)^*$.

The classes of languages generated by these types of grammars are called regular (REG), linear (LIN), context-free (CF), and context sensitive (CS) languages respectively. Further FIN denotes the class of finite languages, while generative grammars without restrictions generate the class of recursively enumerable (RE) languages.

The following theorem states the relations between these classes.

Theorem 2.1.1. $FIN \subset REG \subset LIN \subset CF \subset CS \subset RE$.

This hierarchy is known as the *Chomsky hierarchy* and is the most widely used reference scale for the computational power of formal systems.

A *normal form* for a type of grammars is a restriction on the format of the productions which does not reduce its generative power. For instance, the requirement that all rules of a context-free grammar are of the form $A \to a\alpha$, with $a \in T, \alpha \in N^*$ is known as the *Greibach normal form*.

2.1.3 Automata and Machines

Whereas grammars generate a language, automata define a language by *accepting* exactly those words which are part of the language. The simplest kind of automaton is the *deterministic finite automaton* (or *DFA*).

A deterministic finite automaton is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$, where Q is the set of states, Σ the input alphabet, $q_0 \in Q$ the initial state state and $F \subseteq Q$ is the set of final states. The transition function δ is a mapping $Q \times \Sigma \mapsto Q$. We can extend the transition function to words by defining $\delta^* : Q \times \Sigma^* \mapsto Q$ such that $\delta^*(q, \lambda) := q$ and for each $a \in \Sigma$, $\delta^*(q, wa) := \delta(\delta^*(q, w)), a)$. Then the language accepted by M is defined as $L(M) = \{w \mid \delta^*(q, w) \in F\}$.

A *nondeterministic finite automaton* (or *NFA*) $M = (Q, \Sigma, \delta, q_0, F)$ is defined just like a DFA, with the only difference that the transition function is a mapping $Q \times \Sigma \mapsto 2^Q$. Extending δ to $\delta^* : Q \times \Sigma^* \mapsto 2^Q$ as before, the language accepted by M is defined as $L(M) = \{w \mid \delta^*(q, w) \cap F \neq \emptyset\}$.

Let $\mathcal{L}(DFA)$ and $\mathcal{L}(NFA)$ denote the class of languages accepted by DFAs and NFAs respectively. Then the following holds.

Theorem 2.1.2. $REG = \mathcal{L}(DFA) = \mathcal{L}(NFA)$.

Thus both classes of automata accept exactly the same languages, and we will simply refer to them as finite automata (FAs) if the distinction is not important.

When equipping a finite automaton with a push-down memory, we obtain a *push-down automaton* (or *PDA*). This is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$, where Q, Σ, q_0, and F are as for finite automata. Γ is the stack alphabet, and γ_0 is the bottom-of-stack

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

symbol. The transition function δ is defined as a mapping $Q \times \Sigma \times \Gamma \mapsto Q \times \Gamma^*$. A *configuration* of M is a triple (q, w, γ), where $q \in Q$, $w \in \Sigma^*$ and $\gamma \in \Gamma^*$. We say $(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$ if $(p, \beta) \in \delta(q, a, Z)$. Let $\vdash^*$ denote the transitive and reflexive closure of this relation. Then the language accepted by M can be defined equivalently in one of the following ways.

- Acceptance by final state: $L(M) = \{w \mid (q_0, w, \gamma_0) \vdash^* (q, \lambda, Z)$ for some $q \in F, Z \in \Gamma^*\}$.
- Acceptance by empty stack: $L(M) = \{w \mid (q_0, w, \gamma_0) \vdash^* (q, \lambda, \lambda)$ for some $q \in Q\}$.

Let $\mathcal{L}(PDA)$ denote the class of languages accepted by PDAs.

Theorem 2.1.3. $CFL = \mathcal{L}(PDA)$.

A *Turing machine* is a tuple $M = (Q, \Sigma, \Delta, q_0, F, \delta, B)$, with Q the set of states, Σ and Δ respectively the input and tape alphabet, q_0 the initial state, F the set of final states, B the blank symbol and $\delta : Q \times \Delta \to 2^{Q \times \Delta \times \{L,R\}}$ the transition function. A *configuration* or *instantaneous description* of a Turing machine is a string $\alpha = \alpha_1 q \alpha_2$, with $\alpha_1, \alpha_2 \in \Delta^*, q \in Q$. We write $\alpha \vdash \alpha'$ if configuration $\alpha \vdash \alpha'$ can be reached from $\alpha \vdash \alpha'$ by one move of M. Let $\vdash^*$ be the transitive and reflexive closure of $\vdash$. Then the language accepted by M is defined as $L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha_1 q \alpha_2$ for some $q \in F\}$.

Let $\mathcal{L}(TM)$ denote the class of languages accepted by Turing machines.

Theorem 2.1.4. $RE = \mathcal{L}(TM)$.

A Turing machine may not always halt when the input word is not in the language. When we require the Turing machine to halt on all inputs, it accepts a smaller class of languages, the recursive languages (REC).

Theorem 2.1.5. $CS \subset REC \subset RE$.

2.1.4 Complexity

So far, we have classified sets by their structural complexity, that is the complexity of the device needed to generate or recognize them. There are other classifications possible, of which we will consider two in this thesis.

Computational complexity measures the amount of resources needed to recognize or generate a language on a (typically universal) computational device. The most studied resources are space and time, though many other reasonable measures are possible. They are usually defined in terms of the time and space-bounded off-line Turing machines. An off-line Turing machine is presented its input on an additional read-only input tape, which allows to consider only the space needed for the actual computation, without counting the space needed to store the input. For instance, for a function

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$f(n) : \mathbb{N} \mapsto \mathbb{N}$ the class DSPACE[$f(n)$] is the class of languages which can be accepted by a deterministic off-line Turing machine M which for any word $w \in L(M)$ uses at most $f(|w|)$ tape cells. NSPACE[$f(n)$] is the corresponding class for nondeterministic Turing machines. DTIME[$f(n)$] and NTIME[$f(n)$] are defined similarly, but counting the number of moves of the Turing machine. Important classes we will consider in this thesis are NP = $\bigcup_{i\geq 1}$ NTIME[n^i], PSPACE = $\bigcup_{i\geq 1}$ DSPACE[n^i] and NL = $\bigcup_{c\geq 1}$ NSPACE[$c(\log n)$]. A *reduction* of a language L' to another language L is a recursive mapping g such that for all strings x, $x \in L'$ if and only if $g(x) \in L$. A language L is *complete* for a class C with respect to logarithmic-space reductions if $L \in C$ and every language in C is reducible to L by a mapping computable in deterministic logarithmic space. Finally, a function $f(n)$ is *space-constructible* (*time-constructible*) if there is some Turing machine M that is $f(n)$-space(respectively $f(n)$-time) bounded and for any given n, there exists an input of length n for which M actually uses space (time) $f(n)$.

Descriptional complexity is concerned with the size of the description of formal objects, typically languages. For instance, a given regular language can be described by a deterministic finite automaton or by a nondeterministic finite automaton. However, the *size* of this description depends on the formalism used for describing them. One main question is how the size of description varies when the object is described by different descriptional systems. This variation can be described in terms of upper and lower bounds in the increase of the size of the description when passing from one formal system to another. In this context, we say a function $f(n) : \mathbb{N} \mapsto \mathbb{N}$ is an *upper bound* for the increase in size when changing from a descriptional system D_1 to another system D_2 if every description $M \in D_1$ of size n has an equivalent description $M' \in D_2$ of size at most $f(n)$. A function $g(n) : \mathbb{N} \mapsto \mathbb{N}$ is a *lower bound* for the increase in size when passing from D_1 to D_2 if there is an infinite sequence $(L_i, i \in \mathbb{N})$ of pairwise distinct languages L_i such that there is a description $M \in D_1$ of size n and every description $M' \in D_2$ has a size of at least $g(n)$. For instance, it is known that changing from NFAs to DFAs for describing the family of regular languages, both the upper and lower bound are 2^n ([19]).

It is possible that there exist no recursive function limiting the increase in the size of the description between two formal systems. This is known as a *non-recursive trade-off*. Hartmanis [29] provides an elegant proof technique for showing such non-recursive trade-offs, relating the size of description with decidability questions.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

2.2 Computing with DNA

The area of DNA computing was initiated by Leonard Adleman with his groundbreaking experiment in 1994 [1]. Since then, computer scientists and molecular biologists have worked on the ingredients of a possible molecular computer, both in terms of biochemical techniques and computational models. Adleman solved a small instance of a NP-complete problem using purely biochemical means. Before we start discussing DNA computations, we briefly introduce the basic properties of DNA molecules and some available techniques for manipulating DNA molecules.

2.2.1 DNA: Properties and Techniques

DNA molecules are polymers that are built from simple monomers called nucleotides. Each nucleotide consists of three basic components: sugar, phosphate, and base. There are four possible bases, denoted by A, C, G, and T. Since nucleotides differ only in their bases, we identify nucleotides with their bases. In this way, there are only four types of nucleotides, also denoted by A, C, G, and T. Nucleotides can form single stranded DNA molecules: two consecutive nucleotides bind through a strong (covalent) bond. The basic feature of the four bases A, C, G, T is the pairwise affinity, A with T, and C with G: we say that bases A and T are complementary, and so are C and G. This complementarity (called the Watson-Crick complementarity) underlies the formation of double stranded DNA molecules from the single stranded ones. Two single stranded DNA molecules can bind elementwise through their bases forming weak (hydrogen) bonds. These bonds can form only between complementary bases, hence between A and T, and between C and G. Since each nucleotide can be identified by one of the four letters from the alphabet N=A, C, G, T, a single stranded DNA molecule can be denoted by a string over N. To denote double stranded DNA molecules we use double strings. Double stranded DNA molecules can have a single stranded part at the end, which is called a *sticky end*.

One can separate the two strands of a double stranded DNA molecule by heating the solution containing the molecule. Since the hydrogen bonds between the two strands are much weaker than the bonds between the consecutive nucleotides within a single strand, such a separation will not break the single strands. By cooling down the solution, the separated strands will fuse together forming the original double strand. The fusing process is called *annealing* or *hybridization*, and the separation process is called *melting* or *denaturation*.

It is possible to amplify a specific DNA molecule using *polymerase chain reaction* (PCR). It is based on the activity of the enzyme called DNA polymerase. This enzyme turns a single stranded DNA molecule into a double stranded one by simply adding to each nucleotide in the single strand its complementary nucleotide in the other strand. The enzyme needs a short stretch of nucleotides at one end of the single stranded molecule, called a primer. The amplification takes place by repeated cycles of denaturing, priming and extension. This amplification is so effective that it can be used to

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

detect molecules. After repeated PCR steps one can assume that a molecule picked randomly is of the amplified type.

Gel electrophoresis can be used to separate DNA molecules by length. DNA molecules have a negative charge proportional to their length, so they travel at the same speed in an electric field. However, if there is some kind of resistance, like in a gel, smaller molecules will travel faster that bigger ones. Applying an electric field to a DNA sample placed in a gel is a precise method to separate DNA molecules by length. Afterwards, molecules of a specific length can be reused by simply cutting them out of the gel and dissolving the sample.

Finally, it is possible to filter out molecules containing a certain sequence of bases, by attaching the Watson-Crick complement of this sequence to a solid surface or to glass beads which can be sieved out easily. Denaturing the sample and pouring to over this 'filter' will remove these molecules from the sample.

This description of the processes is highly schematic and simplified. More precise descriptions can be found in [65] or [2].

2.2.2 Adleman's Experiment

The molecular algorithm used by Adleman is widely documented, for instance in [65]. Adleman's experiment solves an instance of the Hamiltonian path problem (HPP), which is the question if, given a graph and a final and initial vertex, there exists a path from the initial vertex to the final vertex which visits all other vertices exactly once.

The molecular algorithm was the following (given a graph with n vertices):

1. Randomly generate paths in the graph
2. Remove paths which do not begin with the initial vertex or do not end with the final vertex
3. Remove paths which are not of length n
4. For each vertex, remove all paths not containing this vertex.

The answer to HPP is "yes" if and only if a path remains.

The algorithm was implemented as follows. Each vertex was encoded in a single stranded sequence and, given such an encoding of vertices a and b, the edge $a \to b$ is encoded by a single stranded sequence which consists of the Watson-Crick complement of the second half of the encoding of a, followed by the complement of the first half of b. In this way, the coding of edge $a \to b$ can anneal to the coding of vertex a, and then the coding of vertex b can anneal to the resulting sticky end. If we have enough copies of all edges and vertices, we can assume that all paths (at least of length n) are formed when these are brought together in a test tube. Now, step 2 to 4 can be executed quite straightforwardly using the techniques of the previous section. Step 2

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

by PCR, step 3 by gel electrophoresis and step 4 by filtering. Our description of the experiment is very concise, for more details the reader is referred to e.g. [65].

HPP is an NP-complete problem. Looking at the algorithm above, we see that the number of biochemical operations is linear in the number of vertices. Since these operations theoretically represent a constant time, we have a linear time algorithm for an NP-complete problem. In fact, Lipton [37] showed that the same method can be used for other NP-complete problems, like the satisfiability problem (SAT). Despite its great promise of solving currently intractable problems, there are still many obstacles to the actual molecular implementation of non-trivial computations. Of course, biochemical operations are slow and labour-intensive. But even if this would be improved in the future, there remain systematic problems when we want to upscale this technique. First, if we have a larger graph, the codes for vertices and edges will have to be longer: To still have a reliable computation, we would need unrealistic amounts of DNA. Moreover, long single stranded DNA molecules are fragile and unreliable. Finally, procedures like PCR are not completely flawless, so we would need some way to cope with possible errors.

2.2.3 Other Experiments and New Directions

Adleman's algorithm was based on the Watson-Crick complementarity. In many subsequent experiments the basis of splicing, DNA recombination induced by restriction enzymes, was used.

The first significant experimental result after the work of Adleman was obtained by Ouyang et al. [55]. They solved an instance of the maximal clique problem, also a well known NP-complete problem. A clique is defined as a set of vertices in which every vertex is connected to all other vertices by an edge. The maximal clique problem is: Given a graph, how many vertices are in the largest subgraph that is a clique? Their solution works as follows.

For a graph of n nodes, subsets of vertices (hence possible cliques) can be represented by binary strings of length n, with 1 denoting the presence of a vertex and 0 denoting its absence. Each bit of the number is encoded by a sequence of bases, with the code of 0 longer than the code for 1. Moreover every encoding of value 1 contains a splicing site for a restriction enzyme (a different one for every position). The idea is to start with all possible cliques, and then use restriction enzymes to digest (i.e. cut) representations of subsets that cannot be cliques. More specifically, we have the following algorithm.

1. Generate the encodings of all binary strings of length n (the data pool)

2. For every edge $i \to j$ not in the graph, separate the data pool into two parts. In one part, cut all molecules containing the encoding of 1 at position i, in the other part, cut all molecules containing the encoding of 1 at position j. Combine the parts.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

3. Using PCR, amplify only the strings that have not been cut in step 2.

4. Apply gel electrophoresis to find the largest clique.

Because the encoding of 1 is shorter than the encoding of 0, the largest clique can be identified by gel electrophoresis; the shortest remaining molecule corresponds to the largest clique. Moreover, the size of the clique follows directly from the length of this molecule.

In Benenson et al. [4] a molecular implementation of a finite automaton is reported. The automaton performs the computation fully autonomously. The implementation is based on the properties of the restriction enzyme $FokI$. Figure 2.1 shows the recognition and cutting site of the enzyme (with N denoting any base).

$$\begin{array}{l} NNNGGATGNNNNNNNNN\ NNNN\vert NNN \\ NNNCCTACNNNNNNNNN\ NNNN\ NNN\vert \end{array}$$

Figure 2.1: The restriction site of enzyme $FokI$

The interesting thing about $FokI$ is that it cuts DNA strands at some distance (9 base pairs) from the recognition site. This property can be used to 'consume' the input at the same time as the state is changed. This works as follows: Consider an encoding of input symbols. We start with an input molecule containing the encoding of the input string of the automaton, preceded by a $FokI$ recognition site. This site is placed is such a way that when $FokI$ is added, the first input symbol is cut off, leaving a sticky end which allows to identify both the state and the symbol just cut off. This is possible because the encoding of a symbol is 6 base pairs long, so with the sticky end of 4 base pairs left by $FokI$, there are in principle 3 possibilities to cut it. Each possible cut leaves a different sticky end which we can identify with a state, as well as allowing us to recognize the symbol. Now, the transitions take place by having transition molecules in the mixture. Let us say that the state is q_0 and the input symbol a. If the automaton contains the transition $(q_0, a) \to q_1$, the corresponding transition molecule contains a sticky end corresponding to the Watson-Crick complement of the sticky end encoding the pair (q_0, a), as well as a splicing site for $FokI$ placed in such a way that the next cut leaves the encoding of the next symbol and state q_1. Thus by repeated cycles of cutting and ligation the automaton is simulated. When the input is fully read, an output molecule allows to read the final state. For more technical details we refer to [4].

While this implementation is very elegant, it is also systematically limited. The number of base pairs for encoding both symbols and states is limited, so there seems to be no possibility to implement an automaton more complicated than the one used in [4], which has two states and 2 input symbols.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

An implementation of a pushdown automaton has been proposed [10], based on the same strategy but another restriction enzyme, *PrsI*. The enzyme has two cutting sites at either side of the recognition site, as shown in Figure 2.2.

$$\begin{array}{l} NNNNN|NNNNNNNGAACNNNNNTACNNNNNNN|NNNNN \\ |NNNNN\ NNNNNNNCTTGNNNNNATGNNNNNNN\ NNNNN| \end{array}$$

Figure 2.2: The restriction site of enzyme *PrsI*

Using this enzyme, on one side the input symbols and states are handled as before, on the other side the stack can be dealt with. An experimental realization using circular DNA molecules is in course. Obviously, the same strong restrictions on the number of symbols, state and stack symbols apply in this case.

Finally, it is interesting to note that both *FokI* and *PrsI* are recently discovered enzymes. This shows that advances in biochemistry can lead quite directly to new possibilities in molecular computing.

Tom Head, with various collaborators, has introduced another way to conceive molecular computations. In contrary to the algorithms presented above, his approach does not involve chemical processes performing (part of) the computation independently. Instead, a pool of (initially identical) molecules is considered as a *fluid memory*. The molecules contain a set of locations which can be "written" by some biochemical technique. The amount of molecules in such a fluid memory is so vast that we can make the following working assumptions, stated in [26].

1. The memory molecules are uniformly distributed through the memory.
2. The memory can be partitioned into separate portions that may be assumed identical.
3. Separate portions of fluid memory can be reunited into a single unit.

These three properties allow computations to be carried out. Independently of the technology used for writing, each writing step is done in parallel, in unit time, on every memory molecule in the body of the fluid into which the writing is being done. Each writing operation writes at the same location on every memory molecule in the body of fluid into which the writing is being done. After the completion of the writing phase of a computation, there typically follows a phase in which the molecules of the memory are read to yield the result of the computation. Examples of aqueous computation successfully carried out in a lab are reported in [25], [26] and [28]. These all used splicing techniques for writing. For instance, in [26] the satisfiability problem (SAT) for disjunctive clauses is solved. Here a circular DNA molecule with six 'memory

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

locations' is used. A memory location consists of a splicing site for an enzyme. This site can be deactivated, which can be interpreted as changing a bit of information from 1 to 0. Now, SAT can be solved by following an algorithm reminiscent of that of Ouyang et al. [55] discussed above, using writing in memory rather than cutting.

While these and other experiments introduced new techniques, they still suffer from the problems mentioned in Section 2.2.2, making upscaling these methods practically impossible. In fact, most researchers acknowledge that it is unlikely that a 'molecular computer' can rival the digital computer in a foreseeable future. Recent experiments have gone in different, but not less interesting, directions. Self-assembly based on Watson-Crick complementarity has been used to form complex nanostructures with DNA tiles [67]. Benenson and his collaborators [3, 11] are exploring the possibility to construct simple DNA computers for medical use. These so-called molecular automata test for the values of medical indicators, and release a medicine only if it is appropriate given these indicators. An in-vitro version was reported in [3], and recently also a version working in living cells was presented [66].

2.3 Splicing

In this section, we make the step from molecular operations to formal models. Splicing as a formal operation on strings was introduced by Tom Head in 1987 [22], so it actually predates Adleman's work. However, only since this experiment the field of splicing has attracted the attention of a larger research community and most of the work in this area dates from the last decade. The splicing operation is the basis of computing models which are called *Splicing systems* or *H systems*. In this section we introduce the basic notions and definitions of splicing, present the main results about basic H systems and review some variants of splicing systems which will be used in the rest of this work. More details can be found in [65], a fairly exhaustive overview of all theoretical models of DNA computing, or in [9], which is more recent, but not as extensive.

2.3.1 From Biochemistry to Formal Languages

In this section, we show how biochemical operations on DNA using restriction enzymes can be regarded as operations on strings, which can be used to define models of computation. The example of the working of restriction enzymes is from [9].

The splicing operation is a formal model of recombination of DNA molecules induced by restriction enzymes. Restriction enzymes cut DNA strands at specific sequences of base pairs.

Suppose we have the following DNA molecules

$$CGCGCTCGACGCGC \qquad ATATAGCGCTATAT$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$GCGCGAGCTGCGCG \qquad TATATCGCGATATA$

And the two restriction enzymes *TaqI* and *SciNI*, with the following patterns.

$$T\vert CG\ A \qquad\qquad G\vert CG\ C$$
$$A\ GC\vert T \qquad\qquad C\ GC\vert G$$

When we combine all these, the following four fragments will be produced.

$CGCGCT \quad CGACGCGC \quad ATATAG \quad CGCTATAT$
$GCGCGAGC \quad TGCGCG \quad TATATCGC \quad GATATA$

Since the sticky ends are the same, new molecules can be produced by recombining the parts of the two molecules.

$CGCGCTCGCTATAT \qquad ATATAGCGACGCGC$
$GCGCGAGCGATATA \qquad TATATCGCTGCGCG$

Now, regarding this biochemical operation as an operation on strings we make a series of abstractions. First, because of Watson-Crick complementarity, we can consider single stranded sequences, since each single stranded sequence uniquely defines a double stranded one. Moreover, instead of the alphabet of bases A, C, T and G, we work with strings over an arbitrary alphabet. Finally, we disregard the sticky ends. Indeed, looking at the example above we see that after recombination no trace is left of the sticky ends. For the result of the splicing operation it is irrelevant which were the sticky ends, it suffices to know that the operation is possible.

Thus we can regard the splicing operation as passing from two strings $w_1u_1u_2w_1'$ and $w_2u_3u_4w_2'$ to two new strings $w_1u_1u_4w_2'$ and $w_2u_3u_2w_1'$. This can be represented by a *splicing rule* of the type $r = u_1\#u_2\$u_3\#u_4$ where u_1,u_2,u_3,u_4 are strings over some alphabet V and \$ and # are special symbols not in V. We say that r is a splicing rule over V. An application of r, denoted by $\vdash_r$, is defined as follows.

Definition 2.3.1. *For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ and $x, y, w, z \in V^*$, we define*

$$\begin{aligned}(x, y) \vdash_r (w, z) \textit{ iff } & x = x_1u_1u_2x_2,\\ & y = y_1u_3u_4y_2,\\ & z = x_1u_1u_4y_2,\\ & w = y_1u_3u_2x_2,\\ & \textit{for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

In [65] this type of splicing is called 2-splicing. In the literature also another definition is considered, referred to as 1-splicing in [65]. In fact, many results on H systems are obtained using this definition.

Definition 2.3.2. *1-splicing: For a splicing rule $r = u_1\#u_2\$u_3\#u_4$ and $x, y, w, z \in V^*$*

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$$(x, y) \vdash_r^1 z \text{ iff } x = x_1u_1u_2x_2,\ y = y_1u_3u_4y_2,\ z = x_1u_1u_4y_2,\ \text{for some } x_1, x_2, y_1, y_2 \in V^*.$$

As in [65], we prefer to use the first definition to define computational systems. In this way, the models are closer to biochemical reality and also make more sense on a formal level. So, in what follows we always consider 2-splicing, unless specifically stated otherwise. If we write $(x, y) \vdash w$, this should be read as $(x, y) \vdash (w, z)$ or $(x, y) \vdash (z, w)$ for some z. In this respect, it is important to observe that, even if it is known that 2-splicing is slightly weaker than 1-splicing [79], all main theorems as well as the results presented here are true irrespective of which of the two definitions is used.

Now, to pass from an operation on strings to a computing device, first we extend the splicing operation to languages. Let R be a set of splicing rules over V. Then if we consider some language $L \subseteq V^*$, we can apply the splicing rules of R to it in the following way.

Definition 2.3.3. *For a set R of splicing rules and a language L, both over V, we define* $\sigma_R(L) = \{w, w' \in V^* \mid (w_1, w_2) \vdash_r (w, w') \text{ for some } w_1, w_2 \in L \text{ and some rule } r \in R\}$

$\sigma_R(L)$ contains the result of applying the splicing rules once. But in a test tube restriction enzymes will not stop working after one operation, they will act iteratively. Similarly, we define iterated splicing as follows.

Definition 2.3.4. *For a set R of splicing rules and a language L we define*

$$\sigma_R^0(L) = L,$$
$$\sigma_R^{i+1}(L) = \sigma^i(L) \cup \sigma(\sigma^i(L)),$$
$$\sigma_R^*(L) = \bigcup_{i \geq 0} \sigma^i(L).$$

So $\sigma_R^*(L)$ is the closure of L under splicing with respect to R.

We now define a *splicing system* or *H system*.

Definition 2.3.5. *A splicing system is a triple $\Gamma = (V, A, R)$, where V is an alphabet, $A \subseteq V^*$ the initial language and $R \subseteq V^*\#V^*\$V^*\#V^*$ is a set of splicing rules. The language generated by γ is, by definition,*

$$L(\gamma) = \sigma_R^*(A).$$

Note that both A and R are possibly infinite sets, of which we can consider the position in the Chomsky hierarchy or some other hierarchy. So for two families of languages FL_1 and FL_2, we can consider H systems with $A \in FL_1$ and $R \in FL_2$. By

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$H(FL_1, FL_2)$ we will denote the family of languages generated by such systems. So, formally

$$H(FL_1, FL_2) = \{\sigma_R^*(A) \mid A \in FL_1 \text{ and } R \in FL_2\}.$$

Also, in analogy with Chomsky grammars, it is quite natural to consider splicing systems which have a terminal alphabet. These are called *extended H systems*.

Definition 2.3.6. *An extended splicing system is a construct $\gamma = (V, T, A, R)$, with alphabet V, terminal alphabet $T \subseteq V$, $R \subseteq V^*\#V^*\$V^*\#V^*$ and $A \subseteq V^*$. The language generated by γ is, by definition,*

$$L(\gamma) = \sigma_R^*(A) \cap T^*.$$

Finally, it should also be mentioned that the way of representing splicing rules presented above, though the most usual, is not the only way used in the literature. In addition to this definition, known as Păun-splicing, also the so-called Head and Pixton notations are used, which are marginally different, also in power, at least in finite non-extended systems (see [6]).

In the next section, we consider the computational power of splicing systems.

2.3.2 The Power of Basic Splicing Systems

In this section we present the two basic results about splicing systems. The first, the so-called regularity preserving lemma, was first proved by Culik and Harju [12] as a consequence of a more general result, and later by a direct argument by Dennis Pixton [70]. It states that H systems with a regular initial language and finite rules generate only regular languages. For later reference, we here briefly sketch Pixton's proof, for more details the reader is referred to [70] or [65].

Lemma 2.3.7. $H(REG, FIN) \subseteq REG$

Proof. Let $H = (V, A, R)$ be a splicing system and $M = (K, V, s_0, F, \delta)$ be the finite automaton accepting initial language $A \subseteq V^*$. Now we construct automaton $M' = (K \cup K', V, s_0, F, \delta')$ which accepts $\sigma_R^*(A)$. Assume $R = \{r_1, r_2, ..., r_n\}$ and that $r_i = u_{i,1}\#u_{i,2}\$u_{i,3}\#u_{i,4}$, $1 \leq i \leq n$.

Moreover, let $u_{i,1}u_{i,4} = a_{i,1}a_{i,2}...a_{i,t_i}$ with $a_{i,j} \in V$, $1 \leq j \leq t_i$. Note that this proof uses 1-splicing, but extends directly to 2-splicing, since all 2-splicing can be represented by a symmetric 1-splicing scheme (i.e. one where for each rule $u_1\#u_2\$u_3\#u_4$, the rule $u_3\#u_4\$u_1\#u_2$ is also in the scheme). K' will contain the new states $q_{i,1}, q_{i,2} \ldots q_{i,t_i}, q_{i,t_i=1}$ for all $1 \leq i \leq n$. Transition function δ' contains all transitions of δ to which we add

$$\delta'(q_{i,j}, a_{i,j}) = \{q_{i,j+1}\}, 1 \leq j \leq t_i, 1 \leq i \leq n.$$

Moreover, for each splicing rule $r_i = u_{i,1}\#u_{i,2}\$u_{i,3}\#u_{i,4}$ in R we do the following. If for some state $s \in K \cup K'$ there exists a state $s \in K \cup K'$ and two strings $x_1, x_2 \in V^*$ such

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

that $s \in \delta'(s_0, x_1)$, $s_1 \in \delta'(s, u_{i,1}u_{i,2})$ and $\delta'(s_1, x_2) \cap F \neq \emptyset$, we put

$$\delta'(s, \lambda) = \{q_{i,1}\}.$$

In addition, if for some state $s' \in K \cup K'$ there exists a state $s_1 \in K \cup K'$ and two strings $y_1, y_2 \in V^*$ such that $s_1 \in \delta'(s_0, y_1)$, $s' \in \delta'(s_1, u_{i,3}u_{i,4})$ and $\delta'(s', y_2) \cap F \neq \emptyset$, we put

$$\delta'(q_{i,t_i+1}, \lambda) = \{s'\}.$$

The idea behind this is that if both $x_1u_1u_2x_2$ and $y_1u_3u_4y_2$ are in A, we add the necessary transitions to ensure that $x_1u_1u_4y_2$ is in $L(M')$. In fact, since the new transitions can give rise to new possible splicings, we have to iterate this step. But since the state set is finite, we know the procedure will finish. At this point, we have our automaton M'. For the complete formal proof that $L(M') = \sigma_R^*(A)$ we refer to [70]. □

We should mention here that the family of languages generated by non-extended splicing systems with finite rules is strictly included in the set of regular languages, with languages like $(aa)^*$ or $a^*ba^*ba^*$ being examples of regular languages which are not finite splicing languages.

The next important result, the basic universality lemma, shows that if we allow for regular rules, extended H systems with finite initial language are computationally complete. This was shown in [63]. The proof introduces a technique which has become widely used in splicing proofs, the rotate-and-simulate procedure. Since we will use variants of this technique in several places, we consider the proof in some detail.

Lemma 2.3.8. $EH(FIN, REG) \supseteq RE$

Proof. We construct a H system to simulate an unrestricted grammar $G = (N, T, S, P)$. The idea behind the rotate-and-simulate procedure is that it is easy to devise a splicing rule to simulate a rule application of G at the extremities of a string. For instance the rule $u \to v$ can be applied on some string wu by the splicing rule $\lambda\#u\$Z\#v$ in the presence of string Zv. However, rules of G can be applied at any place of the sentential form. To simulate this, we will rotate the string such that all symbols can appear at the extremities.

Formally, let $U = N \cup T \cup \{B\}$, where B is a new symbol not in $N \cup T$. We construct the extended H system $\gamma = (V, T, A, R)$ with

$$V = N \cup T \cup \{X, X', B, Y, Z\} \cup \{Y_\alpha \mid \alpha \in U\},$$

R contains the following rules

Simulate:	(1)	$Xw\#uY\$Z\#vY$, for $u \to v \in P, w \in U^*$
Rotate:	(2)	$Xw\#\alpha Y\$Z\#Y_\alpha$,for $\alpha \in U, w \in U^*$
	(3)	$X'\alpha\#Z\$X\#wY_\alpha$for $\alpha \in U, w \in U^*$
	(4)	$X'w\#Y_\alpha\$Z\#Y$, for $\alpha \in U, w \in U^*$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$$\begin{array}{rl} & (5)\ \ X\#Z\$X'\#wY, \quad \text{for } w \in U^* \\ \text{Terminate:} & (6)\ \ \lambda\#ZY\$XB\#wY, \text{for } w \in U^* \\ & (7)\ \ \lambda\#Y\$XZ\#\lambda. \end{array}$$

and initial language

$$A = \{XBSY, ZY, XZ\} \cup \{ZvY \mid u \to v \in P\} \cup \{ZY_\alpha, X'\alpha Z \mid \alpha \in U\}.$$

The symbol B is a marker for the beginning of the sentential form. The derivations starts from the string $XBSY$, so from the axiom of G, marked by B and end markers X and Y. For every string of the form XwY, $w \in U^*$, we can either simulate a rule or rotate the sentential form. The simulation of the rules is handled by the rules of (1), in the way we described above. For the rotation part, consider a string $Xw\alpha Y$, with $\alpha \in U, w \in U^*$. If we apply a rule of (2), we get XwY_α. Applying next a rule of type (3) leads to $X'\alpha wY_\alpha$. The rules of (4) and (5) lead us to $X'\alpha wY$ and finally $X\alpha wY$, which has the same form as the string we started with. The necessary auxiliary strings are in A and never lead to terminal strings. Thus we have rotated the symbol α from the end to the beginning of the string. Now we can choose to either simulate a rule or rotate again.

Finally the rules of (6) and (7) ensure that only terminal strings with B in the right position will have the markers removed, yielding terminal strings in the language. The complete proof can be found in [63] or [65]. □

So while a finite rule set restricts the power of H systems to the regular languages, a regular set of rules gives all RE languages. However, in terms of practical DNA computers or biochemical reality, infinite sets of rules are quite meaningless. Also formally, an infinite device is quite unattractive. We would like to have a finite computing device, but one which still has sufficient computational power. This search has led to a considerable number of variants of splicing systems, some of which we discuss in the next section.

2.3.3 Variants of Splicing Systems

The search for computationally complete H systems with a finite description led to several new approaches to splicing systems, which can be divided into two main groups. On the one hand, many models were introduced in which the derivations are controlled by some additional feature, often inspired by models from the regulated rewriting area (see [14]). On the other hand, many authors enhanced the power of splicing systems by embedding the splicing operation in some structure of distributed computing. We cannot describe all these models here, but to give a rough idea of the work in this area, we will list and in some cases informally describe several variants of splicing systems. For additional models, as well as formal definitions and proofs for the ones we mention, we refer to [65] and its references.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

In the family of systems with additional control features, we can mention *H systems with permitting contexts*, *H systems with forbidding contexts* and *ordered H systems*. In addition to the computational motivation, the restriction can also be viewed as a formal version of a biochemical process. Then permitting contexts correspond to catalysts, forbidding contexts to inhibitors and ordered rules can be seen as modelling differences in reactivity. Considering the proof of Lemma 2.3.8, it is not very hard to see that the same restrictions imposed there by regular languages can also be expressed in terms of contexts or orderings. Other systems are *programmed H systems*, and H systems with *targets* or *double splicing* (rules are applied in predefined couples, cf. matrix grammars). Also having splicing systems operate on *multisets* rather than sets gives rise to complete systems.

Among the distributed variants we can name *splicing grammar systems*, which are grammar systems where the context-free rewriting rules are replaced by splicing rules and *communicating distributed H systems*, which have several components where the strings are spliced according to the associated rules. Each component i is equipped with a *selector* or *filter* which is a subset V_i of the alphabet. After splicing, the result is redistributed among all components in such a way that a component i receives a string if it belongs to V_i^*. This process is repeated and the language is defined as the set of all terminal string appearing in the designated output component.

2.3.4 Time-varying H Systems

In this section, we present in more detail a variant of H systems called *time-varying H systems*, which we will refer to later in this thesis. They were first introduced in [62]. The biochemical inspiration behind these systems is that environment conditions can change the working of enzymes. This can be modelled by different sets of splicing rules, active at different times. If the environment changes periodically, then the set of active enzymes (in our case, rules) will also change periodically.

Definition 2.3.9. *A* time-varying distributed H system*(of degree n) is a construct:*

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$.

At each moment $k = n \cdot j + i$, for $j \geq 0$, $1 \leq i \leq n$, only the component R_i is used for splicing the currently available strings. Specifically, we define

$$L_1 = A,$$
$$L_{k+1} = \sigma_{h_i}(L_k), \text{ for } i \equiv k(mod\ n), k \geq 0, 1 \leq i \leq n, h_i = (V, R_i).$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k\geq 1} L_k) \cap T^*.$$

Note that in this type of systems from step k to the next step, $k + 1$, one passes only the result of splicing the strings in L_k according to the rules in R_i. The strings in L_k that cannot enter a splicing rule are removed.

Already in [62] it was shown that time-varying H systems are computationally complete. Subsequently, in a series of papers including [61],[58] and [50], the degree (i.e. the number of components) needed to obtain computational completeness for time-varying H systems has been lowered progressively. Finally, in [49, 51], it was shown that time-varying distributed H systems of degree 1 can generate all RE languages. So in fact, no time variation is needed to obtain this result. In other words, the way of derivation in which only the result of the splicing operations is passed to the next step is *by itself* powerful enough to obtain completeness.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

3 An Alternative Definition of the Language Generated by a Splicing System

3.1 Introduction

In this chapter, we propose a new definition of the language generated by a splicing system, motivated by both biochemical and mathematical considerations. The main feature of the new definition is that by applying a splicing rule, we not only create new strings, but also allow for the removal of the strings entering the rule. This behaviour seems to correspond better to biochemical reality and is in fact used as a tool in several experimental DNA computations. We show that using this new definition, finite extended H systems can generate all recursively enumerable languages. Even a weaker version of these H systems, defined using the new notion of delay, is shown to be strictly more powerful than H systems defined in the traditional way.

As we saw, the splicing operation is a formal model of recombination of DNA molecules induced by restriction enzymes. In many cases these DNA recombination operations are *reflexive*, that is, the original molecules will be among the result of the operation. However, this is not necessarily always the case. In some reactions, all original molecules disappear and only newly created molecules remain. This is for instance the case when two restriction enzymes with so-called *compatible cohesive ends* are used. These are restriction enzymes which cut at different sites, but leave identical sticky ends. When fragments created by such enzymes are recombined, the resulting strands can not be cut by either enzyme. Reactions of this type are widely used, both in molecular computing and in molecular biology. Head [24] reports the experimental verification of the behaviour described above.

The traditional definition of the language generated by a splicing systems we saw in Section 2.3.1 can be seen as a reflexive definition in the following sense. A splicing system is said to be *reflexive* if for every splicing rule $u_1\#u_2\$u_3\#u_4$ the set of rules also contains $u_1\#u_2\$u_1\#u_2$ and $u_3\#u_4\$u_3\#u_4$. This means applying the splicing rules will also yield the original strings.

However, in the usual definition of the evolution of a splicing and the language it generates, reflexivity is already assumed. By definition, after a step of rule application we obtain a set of new strings and in addition maintain all original strings. All strings present are preserved and are part of the generated language of the system. This is

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

true regardless of whether the system is reflexive. We propose a new definition, where reflexivity is induced by the set of splicing rules and we also allow for possible non-reflexive behaviour like in biochemical recombinations. To avoid confusion with other work dealing with reflexive splicing systems meant as traditionally defined H systems with a reflexive set of splicing rules (for instance [7] and [27]) we will call our systems *non-reflexively evolving*.

In a way, the splicing language as it is traditionally defined contains all strings that occur at some time in the computation. However, as we have seen, in biochemical splicing it is usual that molecules disappear in the course of the reaction. Furthermore, if we look at a typical scenario of a molecular computation, they contain a detection phase in which the presence of a given molecule is checked. In these computations, enough time is left for the reaction to run to completion, after which the presence of solutions is checked. In this way, none of the possible intermediate molecules is part of the detected solutions. This suggests a different definition of the language generated by a splicing system: the language contains all strings that can be effectively detected. In fact, the possibility of certain molecules to disappear in the course of the computation is employed as a technique in some theoretical and experimental molecular algorithms, for instance [69] and [28], based on the irreflexivity and irreversibility of some chemical operations.

For formal systems, this leads to a language definition in which all strings that disappear in the course of the computation are not considered to be part of the language. This new view also affects the derivations in the systems. In the usual definition of splicing systems, all strings generated at some point can enter a splicing rule at all moments after that point. If we allow strings entering a rule to disappear in the course of the computation, this also means that after this point those strings cannot enter splicing rules. So, to formalize the considerations above, we do not only need a new definition of the language, but also a new definition of the possible derivations in a splicing system.

The existence of non-reflexive biochemical splicing operations was already observed by Tom Head [24]. He introduced the notions of *adult splicing language* and *limit language*. The adult splicing language contains all molecules that cannot be cut by any restriction enzyme. The limit language contains all strings present 'at equilibrium'. Goode and Pixton [20] proposed a more precise definition of this concept. Their definition tries to model the dynamic behaviour of a biochemical splicing system. We stay closer to the formal definitions of splicing, trying to find a definition which is a generalization of the usual definition, in the sense that if the underlying set of splicing rules is reflexive, the definitions will coincide. This also points to a more mathematical motivation to consider a new definition; in the formal model, no a priori restrictions are imposed on the set of splicing rules. However, the language definition imposes a reflexive behaviour on the splicing systems. We feel that, in addition to the biochemical considerations outlined above, that dropping this feature is also more consistent with the in principle unrestricted nature of the formal splicing operation.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

It should be noted here that there already exist definitions of splicing systems in which strings disappear in the course of computation. For instance, the H systems studied by Harju and Margenstern [21] discard all strings not used at a step. This definition corresponds in fact to time-varying H systems of degree 1, known to be computationally complete (see [51]), modulo a terminal alphabet. Also, Verlan and Margenstern [78] consider splicing membrane systems with one membrane, where different possibilities of the result of sending a string out of the membrane are studied. After formulating our definition, we will discuss the similarities and differences between these systems and the ones defined here.

In the next section, we propose a new definition of a language generated by a splicing system, which formalizes these considerations. We then show that with this definition, finite extended H systems are computationally complete. Afterwards, we consider a weakened version of non-reflexively evolving splicing systems, introducing the concept of *delay*. H systems with delay are shown to be strictly more powerful than normal H systems.

3.2 A Non-reflexively Evolving Splicing Language

The definition of the language $L(\Gamma)$ generated by a splicing system Γ given in the previous section is a reflexive definition. For any application of a splicing rule $(x, y) \vdash (z, w)$, all four strings will be in the splicing language. So whether the splicing systems is reflexive or not, all strings are preserved. Since we do not impose reflexivity on the splicing system and we know non-reflexive splicing operations are possible, we will look to define the splicing language in such a way that the language evolves in a reflexive way if and only if the H system is reflexive.

First, we need a different definition of a splicing step, which we denote by τ.

Definition 3.2.1. *For a set of splicing rules R over V, and a language L over V we define*

$$\begin{aligned}\tau_R(L) =&\{w, w' \in V^* \mid (w_1, w_2) \vdash_r (w, w') \text{ for some } w_1, w_2 \in L \text{ and some} \\ &\text{rule } r \in R\} \cup \{w \in L \mid \text{there is no } x \in L \text{ and no rule } r \in R \text{ such} \\ &\text{that } (w, x) \vdash_r (y, z) \text{ or } (x, w) \vdash_r (y, z) \text{ for some } y, z \in V^*\}\end{aligned}$$

The difference with $\sigma_R(L)$ is that $\tau_R(L)$ not only contains all strings created by applying splicing rules, but also all strings that did not enter any splicing rule. In a way, $\tau_R(A)$ describes the contents of the test tube after splicing. This new definition is necessary because in our definition, splicing not only may create new strings but can also cause strings to disappear from the language, hence the need to keep track of all strings.

Now we can define iterated non-reflexively evolving splicing as follows.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Definition 3.2.2. *Given a splicing system* $\Gamma = (V, A, R)$*, we define*

$$\tau_R^0(A) = A,$$
$$\tau_R^{i+1}(A) = \tau_R(\tau_R^i(A)), i \geq 0.$$

As before, we omit the subscript when the splicing scheme is clear. To define the language generated by such a splicing system, we need to consider three classes of strings that are involved in derivations of the language. Any string w in $\tau^i(A)$ for some i belongs to one of the following classes:

- *Transient* strings: there exists k such that for all $n \geq k$, $w \notin \tau^n(A)$.
- *Eventually stable* (or *stable*) strings: there exists k such that for all $n \geq k$, $w \in \tau^n(A)$.
- *Recurrent* strings: for every k there exists $n \geq k$, such that $w \in \tau^n(A)$.

In addition, we use the term *inert* to denote strings that can never enter any splicing rule. It should be noted that inert is not synonymous with eventually stable. A string can be eventually stable and not inert by disappearing and being recreated at every step.

We observe that every eventually stable string is also a recurrent string. Transient strings will not be part of the language, for the reasons outlined above. Then we have the choice to consider only the eventually stable strings, or all recurrent strings. We will see below that in the general case, it is undecidable if a given string is recurrent. Thus, if we choose to include all recurrent strings in the generated language, membership can be undecidable. On the other hand, if we want our language to be the formal counterpart of the set of molecules that are present at the end of the computation and assume with [20] that at least one copy of recurrent molecules is present at all times, then they should be part of the language. We feel that this definition is best justified, so we will define the non-reflexively evolving splicing language, denoted by $\tau^\infty(A)$, as:

$$\tau^\infty(A) = \bigcap_{k=0}^{\infty} \{w \mid \exists n \geq k \text{ such that } w \in \tau^n(A)\}.$$

If we consider only the eventually stable strings, we will denote the language as $\tau_{st}^\infty(A)$ and define it as:

$$\tau_{st}^\infty(A) = \bigcup_{k=0}^{\infty} \{w \mid \forall n \geq k,\ w \in \tau^n(A)\}.$$

Observe that $\tau_{st}^\infty(A) \subseteq \tau^\infty(A)$. It is easily verified that for a reflexive splicing scheme, both definitions yield exactly the same language as the usual definition.

The following example shows how this way of defining the splicing language adds a dynamic aspect to the derivations of a splicing systems.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Example 3.2.3. *Let us consider the following NRE H system.*

$\Gamma = (\{a, b, c, d, e\}, \{ac, cd, ca^n b, be, ce\}, R)$ *for some* $n \geq 1$
with R defined as
$b\#e\$b\#e$, $c\#e\$c\#e$, *(1)*
$b\#e\$ca\#z$, $c\#e\$b\#z$, $z \in \{a, b\}$, *(2)*
$a\#c\$c\#b$, *(3)*
$a\#c\$c\#d$. *(4)*

By the rules of (2) we go from $ca^n b$ to $ba^{n-1}b$ to $ca^{n-1}b$ etc, and eventually to cb. The reflexive rules of (1) ensure that the strings be and ce needed for the application of (2) are always present. Rule (4) converts ac and cd into ad and cc, causing the strings ac and cd to disappear. This means that by the time cb is produced, rule (3) cannot be applied, since all ac are already lost. In the traditional definition of the splicing language, rule (3) will still be applied. So the non-reflexively evolving language $\tau^\infty(A)$ is $\{be, ce, cb, ad, cc, cae\}$, whereas the normal splicing language $\sigma^\infty(A)$ would in addition contain ab, created by rule (3), as well as all original and intermediate strings.

Finally, we will define *non-reflexively evolving* or in short *NRE H systems*. These are defined just as the H systems of section 2.3.1, but using the non-reflexively evolving definition of the splicing language. So for a NRE H system $\Gamma = (V, A, R)$, the generated language is $L(\Gamma) = \tau^\infty(A)$. Note that a given H system can be interpreted as a non-reflexively evolving or a usual H system. In what follows, we only consider NRE splicing systems, unless explicitly mentioned otherwise.

At this point, it may be instructive to compare our systems with other systems which can eliminate strings in the derivation. First of all, time-varying H systems of degree 1 as defined in section 2.3.4, as well as the systems considered in [21] keep only the newly created strings and eliminate all present strings at every step. So essentially in these systems (definition from [21]):

$$\sigma_R^{i+1}(A) = \sigma_R(\sigma_R^i(A)).$$

Compared to NRE H systems, these systems have an extra elimination feature, eliminating all strings not entering a rule. This extra feature is in fact a powerful tool, that is essential in the completeness proof of [51]. There is no obvious way to simulate this behaviour using NRE H systems. Moreover, the language is defined differently, as the union of all strings of all steps.

Then, there is some similarity between our definition and a type of splicing membrane systems considered by Verlan and Margenstern ([78]), specifically of the type 2b. These splicing membrane systems have only one membrane where strings evolve according to splicing rules enhanced with target indicators. When the target is *here*, the result of applying the rule remains inside the membrane, if it is *out* it will be sent out of the membrane. The language generated is the union of all strings sent out of the membrane. The reader is referred to [78] for details. In the mentioned type 2b, if a

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

string is sent out, it will disappear from the membrane unless some other rule generates it. This is slightly reminiscent of what happens in our definition.

However, the systems are in fact very different. First of all, the way the words are collected to yield the language is completely different. In the membrane system, the disappearing strings make up the language, in our case they are excluded from it. Moreover, the membrane systems allow the free and independent use of *out*-rules and *here*-rules, whereas in NRE H systems strings only disappear when used to produce other strings. These features of this type of membrane systems make it relatively easy to simulate time-varying H systems of degree 1 (thus showing universality), but the differences pointed out here make the techniques and results inapplicable to our systems. While the difference in language definition could be circumvented by using a terminal alphabet in the same way as we will see in Theorem 3.3.4, the crucial difference is that in our systems we do not have at our disposal the possibility to eliminate strings without further consequences.

In the following section we examine the computational power of NRE H systems.

3.3 Computational Power

We expect the NRE splicing systems to be more powerful than normal splicing systems, because of the strict separation between successive derivation steps. As an example, consider the following NRE H system.

Example 3.3.1.

$H = (V, A, R)$, where
$V = \{a, b, X, Y, Y', Z, Z'\}$,
$A = \{aY, Zb, XaY', Z'bX\}$,
$R = \{a\#Y\$X\#aY',\ a\#Y'\$X\#Y,$ (1)
$Z\#b\$Z'b\#X,\ Z'\#b\$Z\#X,$ (2)
$a\#Y\$Z\#b,$ (3)
$\alpha\#\lambda\$\alpha\#\lambda,\ \forall\alpha \in \{XaY', Z'bX\}\ \}.$ (4)

The rules of (1) rewrite a string of the form a^nY for some $n \geq 1$ into $a^{n+1}Y$, using axiom XaY' and with as intermediate strings $a^{n+1}Y'$ and XY. Similarly, the rules of (2) add a b to strings of the form Zb^n. The reflexive rules of (4) ensure that the necessary axioms are always present. Since in both cases one symbol is added in two steps, the strings a^nY and Zb^n grow in a synchronized way. These strings are not part of the language, because they will always be rewritten. If, however, we apply rule (3) we obtain a string of the form a^nb^n which is stable and will be in the language, along with part of the axioms and some additional stable strings created in the process. Thus the system generates a language of the form $\{a^nb^n \mid n \geq 1\} \cup L$, where each string of L contains some symbol X, Y, Y', Z, Z'. Such a language is not regular.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Before addressing directly the computational power of NRE splicing systems, we first investigate their relation to time-varying H systems.

Lemma 3.3.2. *Let $\Gamma = (V, A, R)$ be a NRE H system, and T a terminal alphabet. We define the time-varying H system $D = (V, T, A, R)$ of degree 1. Then $L(\Gamma) \cap T^* = L(D)$ if:*

1. *Strings in T^* cannot enter any rule $r \in R$.*
2. *All strings generated in some step k either enter a splicing rule in step $k+1$ or can never enter a splicing rule.*

Proof. First observe that $L_0 = \tau^0(A) = A$. Now by definition, L_1 contains all strings resulting from an application of some splicing rule in R. Also by definition, $\tau^1(A)$ contains all these strings as well as all strings that do not enter a splicing rule. So, $L_1 \subseteq \tau^1(A)$ and by induction, $L_i \subseteq \tau^i(A)$. Moreover, by condition 1, $\tau^{i-1}(A) \cap T^* \subseteq \tau^i(A) \cap T^*$. Together, this gives $(\bigcup_{k=0}^{i} L_k) \cap T^* \subseteq \tau^i(A) \cap T^*$ and with i going to infinity, $L(D) \subseteq L(\Gamma) \cap T^*$.

For the converse inclusion, consider some string $w \in L_i - \tau^i(A)$. By conditions 1 and 2, any string in $L_i - \tau^i(A)$ will never contribute to any new string in $\tau^{i+1}(A)$. Thus, any string in $\tau^{i+1}(A) - \tau^i(A)$ is also in L_{i+1}. If w is in T^*, there is some $j \leq i$ such that $w \in L_j$ and $w \in L(D)$. If $w \notin T^*$, it will not be in $L(\Gamma) \cap T^*$ nor in $L(D)$. □

This result suggests that it may be interesting to define an extended NRE H system. The definition is the same as for a normal extended H system, but with the non-reflexively evolving definition of the language.

Definition 3.3.3. *An* extended non-reflexively evolving (NRE) H system *is a construct*

$$\Gamma = (V, T, A, R),$$

where V is the alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and R is a finite set of splicing rules. The language generated by Γ is defined as

$$L(\Gamma) = \tau^\infty(A) \cap T^*.$$

Theorem 3.3.4. *Extended non-reflexively evolving H systems generate all recursively enumerable languages.*

Proof. In [49] it is shown that time-varying H systems of degree 1 are computationally complete. It suffices to show that the construction in [49] satisfies the two conditions stated in Lemma 3.3.2. Instead of proving this, we prefer to give a direct construction for NRE H systems. On the one hand for the sake of completeness, on the other hand because we feel this is more instructive about the way these systems work, since they differ in a significant way from time-varying H systems.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Given a recursively enumerable language M on an alphabet Σ, we can consider the words of the language as natural numbers expressed in basis $|\Sigma|$, obtaining a recursively enumerable set of natural numbers $\mathcal{N}(M)$. For such a set there exists a total recursive function f such that $\mathcal{N}(M) = \{f(n) \mid n \in \mathbb{N}-\{0\}\}$. Moreover there exists a Turing machine halting on each input that for each n converts an initial configuration q_i01^n into a final configuration $q_f01^{f(n)}$. From this Turing machine, it is straightforward to construct a Turing machine T that converts q_i01^n into $q_f0\langle f(n)\rangle_\Sigma$, where $\langle f(n)\rangle_\Sigma$ denotes the expression of $f(n)$ in basis $|\Sigma|$ over alphabet Σ. Now, to generate a language rather than a single string, we will simulate a deterministic Turing machine T' which computes $01^{n+1}q_f0\langle f(n)\rangle_\Sigma$ from q_i01^n. Then the splicing system will split this into a generated word $\langle f(n)\rangle_\Sigma$ and a new starting configuration q_i01^{n+1}.

Let T' be such a Turing machine, with input alphabet $\{0,1\}$, tape alphabet Δ, state set Q, initial state q_i, final state q_f, blank symbol B and transition function δ, where $(\mathbf{a}, x, y, \mathbf{b}, D) \in \delta$ means that reading symbol x in state $\mathbf{a}$, the head can move in direction D, changing the state to $\mathbf{b}$ and rewriting x by y.

Now we define the extended NRE H system $\Gamma = (V, \Sigma, A, R)$, where

$$
\begin{aligned}
&V = \Delta \cup V',\\
&V' = \{X, Y, Y', t_{q_f}, t_{r_1}, t_{r_2}, t_{l_1}, t_{l_2}, R, L, P, Z, \rightarrow, \leftarrow\} \cup Q,\\
&A = \{Xq_i01Y, Z\mathbf{a}0Y, Ly\mathbf{b}Y, Ly\mathbf{b}zR,\\
&\quad Ly\mathbf{b}zyR, RP \rightarrow t_{q_f},\\
&\quad t_{r_1}p \rightarrow t_{r_2}, Z \leftarrow p, t_{l_1} \leftarrow qt_{l_2}, Y', \leftarrow Y, Xq_i0Z\}\\
&\quad \text{for all } \mathbf{a} \in Q - \{q_f\}, \mathbf{b} \in Q, y, z \in \Delta, p \in \Sigma \text{ and } q \in \Sigma \cup \{1\}.
\end{aligned}
$$

We assume without loss of generality that Δ and V' are disjoint.

R contains the following splicing rules. In what follows we will denote by $\mathbf{a}$ any state in $Q - \{q_f\}$, by $\mathbf{b}$ any state in Q. Moreover, $s \in \Delta \cup \{X\}$, $e \in \Delta \cup \{Y\}$ and $z \in \Delta$.

Simulate moves

Right end of the tape

(1.1) : $st\#\mathbf{a}Y\$Z\#\mathbf{a}BY$ $\qquad t \in \Delta$

Right move

(2.1) : $s\#\mathbf{a}xze\$L\#y\mathbf{b}zR$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, R) \in \delta$
(2.1′) : $s\#\mathbf{a}xY\$L\#y\mathbf{b}Y$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, R) \in \delta$
(2.2) : $sy\mathbf{b}z\#R\$L\mathbf{a}xz\#e$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, R) \in \delta$

Left move

(3.1) : $s\#z\mathbf{a}xpe\$L\#\mathbf{b}zyR$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, L) \in \delta, p \in \Delta$
(3.1′) : $s\#z\mathbf{a}xY\$L\#\mathbf{b}zyR$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, L) \in \delta$
(3.2) : $s\mathbf{b}zy\#R\$Lz\mathbf{a}x\#e$ $\qquad$ if $(\mathbf{a}, x, y, \mathbf{b}, L) \in \delta$

Retrieve result and resume computation

Right signal

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

(4.1) : $1\#q_f0p\$R\#P \to t_{q_f}$ $p \in \Sigma$
(4.2) : $1P \to \#t_{q_f}\$Rq_f0\#\lambda$
(4.3) : $y\# \to p\$t_{r_1}\#p \to t_{r_2}$ $y \in \Sigma \cup \{P\}, p \in \Sigma$
(4.4) : $yp \to \#t_{r_2}\$t_{r_1} \to p\#\lambda$ $y \in \Sigma \cup \{P\}, p \in \Sigma$

Left signal

(5.1) : $y\#q \to Y\$Z\# \leftarrow q$ $y \in \Sigma \cup \{P\}, q \in \Sigma \cup \{1\}$
(5.2) : $q \leftarrow \#p\$t_{l_1} \leftarrow q\#t_{l_2}$ $p \in \Sigma \cup \{Y\}, q \in \Sigma \cup \{1\}$
(5.3) : $\lambda\#q \leftarrow t_{l_2}\$t_{l_1}\# \leftarrow qp$ $p \in \Sigma \cup \{Y\}, q \in \Sigma \cup \{1\}$

Result

(6.1) : $P \leftarrow \#p\$\lambda\#Y'$ $p \in \Sigma$
(6.2) : $1\#P \leftarrow Y'\$\lambda\# \leftarrow Y$
(6.3) : $X0 \leftarrow \#1\$Xq_i0\#Z$

Axioms reproduction

(7.1) : $\alpha\#\lambda\$\alpha\#\lambda$ $\alpha \in A - \{Xq_i01Y\}$

The simulation

We will use w, w_1 and w_2 to denote strings from V^*. In our simulation, if we have the string XwY, this means the current configuration of the T' is w. Assume the current word is $Xw_1sz\mathbf{a}xpw_2Y$. Then, if $(\mathbf{a}, x, y, \mathbf{b}, L) \in \delta$, the word corresponding to the next configuration is $Xw_1s\mathbf{b}zypw_2Y$. We get this result by applying rules of types 3.1 (or 3.1′) and 3.2. The rule of type 3.1 gives $Xw_1s\mathbf{b}zyR$ and $Lz\mathbf{a}xpw_2Y$. These strings enter rule 3.2 in the next step, yielding the representation of the new configuration and $LzaxR$, which is an axiom and will enter a rule of type 7.1 in the next step.

For a right move, i.e., $(\mathbf{a}, x, y, \mathbf{b}, R) \in \delta$, rules of types 2.1(2.1′) and 2.2 will take us from $Xw_1\mathbf{a}xpw_2Y$ to $Xw_1y\mathbf{b}pw_2Y$, through $Xw_1y\mathbf{b}pR$ and $L\mathbf{a}xpw_2Y$. Moreover, at the right end of the tape we may have to use a rule of type 1.1 first. Thus the simulation of a move of M involves the following strings: $Xw_1\mathbf{b}w_2Y$ (1.1, 2.1′, 2.2, 3.2), $Xw_1s\mathbf{b}zyR$ and $Lz\mathbf{a}xpw_2Y$ (2.1), $Xw_1s\mathbf{b}zyR$ (3.1, 3.1′), $Lz\mathbf{a}xpw_2Y$ (3.1), $L\mathbf{a}xzR$ (2.2) and $Lz\mathbf{a}xR$ (3.2), which will enter a splicing rule in the next step, and $Z\mathbf{a}Y$ (1.1) and $L\mathbf{a}xY$ (2.1′), which will never enter any rule, nor be part of the language. Observe that every string representing a new configuration will enter a splicing rule. Since we compute a total recursive function, there is either a new move available or we are in a final state, in which case rule 4.1 applies.

The second set of rules transforms a word of the form $X01^{n+1}q_f0\langle f(n)\rangle_\Sigma Y$ into the result $\langle f(n)\rangle_\Sigma$ and the word corresponding to the new starting configuration $Xq_i01^{n+1}Y$. First rule 4.1 is applied, yielding $X01^{n+1}P \to t_{q_f}$ and $Rq_f0\langle f(n)\rangle_\Sigma Y$. Both enter rule 4.2 to give $X01^{n+1}P \to \langle f(n)\rangle_\Sigma$ and the inert string $Rq_f0t_{q_f}$. Using rules of types 4.3 and 4.4 the arrow moves through the word over Σ. The strings resulting from applying a rule of type 4.3 enter a rule of 4.4, which in turn yields a string that enters a rule of 4.3 and an inert string $t_{r_1} \to pt_{r_2}$. When the end of $\langle f(n)\rangle_\Sigma$ is reached, a rule of type 5.1

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

applies to $X01^{n+1}Pwp \to Y$, where we represent $\langle f(n)\rangle_\Sigma$ as wp, $w \in \Sigma^*$, $p \in \Sigma$, giving $Zp \to Y$, which cannot enter any rule, and $X01^{n+1}Pw \leftarrow p$. From this form, alternating rules of types 5.2 and 5.3 moves the arrow back to the left, creating in addition inert strings of the form $t_{l_1}p \leftarrow t_{l_2}$. When reaching the word $X01^{n+1}P \leftarrow \langle f(n)\rangle_\Sigma$, rule 6.1 applies to give $X01^{n+1}P \leftarrow Y'$ and $\langle f(n)\rangle_\Sigma$, which is an inert terminal string and will be part of the language. $X01^{n+1}P \leftarrow Y'$ is converted to $X01^{n+1} \leftarrow Y$ by rule 6.2, then to $X0 \leftarrow 1^{n+1}Y$ by the rules of 5.2 and 5.3 and finally to the string representing the new initial configuration $Xq_i01^{n+1}Y$ by 6.3. In the process, the inert strings $P \leftarrow Y'$ (6.2), $t_{l_1}1 \leftarrow t_{l_2}$ (5.3), and $X0 \leftarrow Z$ (6.3) are also created.
This covers all the cases and shows that Γ correctly simulates T'. □

Corollary 3.3.5. *For a NRE H system Γ and a string w the following properties are undecidable:*

- *w is eventually stable in Γ.*
- *w is recurrent in Γ.*

Proof. Note that in our construction we made sure that all strings over the terminal alphabet are inert, so that all strings in the language will be eventually stable. This means both properties follow directly from the undecidability of membership in recursively enumerable languages. □

3.4 Non-reflexively Evolving H Systems with Delay

One important feature of NRE H systems is the strict separation of derivation steps. The computational power of extended NRE H systems stems from this strict separation. This is, however, not very realistic from a biochemical point of view. Not all molecules are converted simultaneously and homogeneously throughout the solution. At the same time, the usual reflexive definition is equally unrealistic, since it disregards all dynamic aspects of chemical reactions and allows for recombinations of strings present at different moments. A way to capture these considerations is the notion of *delay*. The delay expresses the number of successive steps whose results can interact. The NRE H systems discussed in the previous section have delay 0, normal splicing systems have infinite delay. Formally, we define:

Definition 3.4.1. *A* Non-reflexively evolving H system of delay d *is a construct*

$$\Gamma = (V, A, R),$$

where V is the alphabet, $A \subseteq V^*$ is a finite set of axioms, and R is a finite set of splicing

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

rules. We define

$$\tau^{-i}(A) = \emptyset, i \geq 1,$$
$$\tau^{0}(A) = A,$$
$$\tau^{i+1}(A) = \tau(\tau^{i}(A) \cup (\tau^{i-1}(A) \cup ... \cup (\tau^{i-d)}(A))), i \geq 0.$$

The language generated by Γ *is*

$$L_d(\Gamma) = \tau^{\infty}(A),$$

where $\tau^{\infty}(A)$ *is defined as in Section 3.2.*

As an example, let us consider the following NRE H system.

Example 3.4.2.

$\Gamma = (V, A, R)$, where
$V = \{a, b, c\}$,
$A = \{aabb, bbaa, bbc, cbb\}$,
$R = \{aa\#bb\$bb\#aa,$ (1)
$aa\#aa\$bb\#bb,$ (2)
$aaa\#a\$bb\#c,$ (3)
$bb\#c\$c\#bb,$ (4)
$c\#c\$bb\#bb\}.$ (5)

Interpreted as an NRE H system of delay 0, rule (3) is never applied. Indeed, using (1) and (2), we go back and forth between $\{aabb, bbaa\}$ and $\{aaaa, bbbb\}$. With (4) and (5), we move between $\{bbc, cbb\}$ and $\{cc, bbbb\}$. Rule (3) is never applied because *aaaa* and *bbc* are never present at the same time. As a system of delay 1, rule (3) will be applied, and inert strings *aaac* and *bba* will be in the language.

It is a matter of biochemistry which delay should be considered for a given splicing system. But in any case NRE H systems with delay ≥ 1 raise a number of interesting research questions, some of which we address here. Specifically, we will investigate their computational power.

At first sight, one might think that systems with finite delay can simulate systems of delay 0 using techniques used in [51] and [76]. These consist in making the axioms go through a cycle to make sure that a given axiom is only present in a usable form at every nth step (for a cycle of length n). However, there does not seem to be a straightforward way to make this technique work here. As an example, suppose we have a system of delay 1 and the axioms go through a cycle of some $n \geq 4$ steps. Let us consider the evolution of one such axiom, referring to its n forms as axiom 1 to axiom n. At the first step we have axiom 1. At the second step we have axiom 2 and by the delay also axiom 1. At step 3, we have axiom 3 and axiom 2 formed from axiom 1. By the delay, axiom 2 will still be there at step 4, along with axiom 3 and axiom 4. In general, if the

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

creation of a string takes n rule applications, with delay d this can take at most $(d+1)n$ steps. At all steps between n and $(d+1)n$ the string will be present (since we can take the 'quick' or 'slow' path at every step). So, after $(d+1)n$ steps, all forms of the axioms are present, which means that from this point on, we cannot use the axioms to control which rules we apply.

Theorem 3.4.3. *For all $d \geq 0$, the family of languages generated by non-reflexively evolving H systems of delay d contains non-context-free languages.*

Proof. For $d = 0$, this follows directly from Theorem 3.3.4. For $d \geq 1$, we devise an NRE H system that simultaneously extends substrings of a's, b's and c's and finally generates strings in $a^+b^+c^+$ of the form $a^ib^jc^k$. For all systems of some finite delay d, the values of j and k are bounded by a term containing i, thus yielding a non-context-free language.

Specifically, consider the NRE H system $\Gamma = (V, A, R)$ where
$V = \{a, b, c, X, Y, Z\}$, $R = R_1 \cup R_2 \cup R_3$,
$A = \{aaX, BbbY, Ccc, ZaX, BbZ, CcZ\}$,
and
$R_1 = \{aa\#X\$Z\#aX, Bb\#Z\$B\#bb, Cc\#Z\$C\#cc\}$,
$R_2 = \{aa\#X\$C\#cc, a\#ac\$B\#bb, b\#bbY\$Bac\#c\}$,
$R_3 = \{\alpha\#\lambda\$\alpha\#\lambda \mid \alpha \in \{ZaX, BbZ, CcZ\}\}$.
It is easy to note that strings of the form a^iX, Bb^jY and Cc^k, $i, j, k \geq 2$ are expanded simultaneously by rules from R_1. The reflexive rules of R_3 ensure that the auxiliary strings ZaX, BbZ and CcZ are always present.

Claim.

(i) If $a^iX \in \tau^s(A)$, then $2 + \left[\frac{s}{d+1}\right] \leq i \leq s + 2$.
(ii) If $Bb^jY \in \tau^s(A)$, then $2 + \left[\frac{s}{d+1}\right] \leq j \leq s + 2$.
(iii) If $Cc^k \in \tau^s(A)$, then $2 + \left[\frac{s}{d+1}\right] \leq k \leq s + 2$.

Here $[x]$ denotes the integer part of the rational x, that is the largest integer smaller than or equal to x.

Proof of the claim. We give a reasoning based on induction on s for the first item only. Let $a^iX \in \tau^s(A)$; if $s = 0$, then $2 + \left[\frac{s}{d+1}\right] \leq i \leq s + 2$ is immediately satisfied. Let $a^iX \in \tau^{s+1}(A) = \tau(\tau^s(A) \cup \tau^{s-1}(A) \cup \cdots \cup \tau^{s-d}(A))$. It follows that $a^{i-1}X \in \tau^m(A)$ for some $s - d \leq m \leq s$ and $(a^{i-1}X, ZaX) \vdash (a^iX, ZX)$. By the induction hypothesis,

$$2 + \left[\frac{s-d}{d+1}\right] \leq 2 + \left[\frac{m}{d+1}\right] \leq i - 1 \leq m + 2 \leq s + 2$$

holds. Since $\left[\frac{s+1}{d+1}\right] \leq \left[\frac{s-d}{d+1}\right] + 1$, it follows that $2 + \left[\frac{s+1}{d+1}\right] \leq i \leq s + 3$ which concludes the proof of the claim.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

In order to get a string in $a^+b^+c^+$ we must first apply the rule $aa\#X\$C\#cc$ to the pair (a^iX, Cc^k) for some i, k. Thus we get the string a^ic^k. Some remarks are necessary at this point.

– $i, k \geq 2$ and by the above claim, $|i-k| \leq s - \left\lfloor \frac{s}{d+1} \right\rfloor$ holds for any $s \geq 0$.

– If the strings a^ic^k and a^tc^q coexist at some step, then they coexist together with a^ic^q and a^tc^k.

– The earliest step in which a^ic^k is obtained is $max(i,k) - 1$.

– The latest step in which a^ic^k is effectively obtained is $min(i,k)d - 2$. Note that a^ic^k will be still available for splicing d steps more.

One needs two more steps in which rules from R_2 are applied. In the first step we apply the rule $a\#ac\$B\#bb$ to the pair (a^ic^k, Bb^jY) for some i, j, k satisfying the following conditions:

(i) $i, k \geq 2$ and $|i-k| \leq s - \left\lfloor \frac{s}{d+1} \right\rfloor$ for some $s \geq 0$.

(ii) $2 + \left\lfloor \frac{max(i,k)-1}{d+1} \right\rfloor \leq j \leq d(min(i,k)+1)$. This relation is obtained by substituting the earliest and latest step in which a^ic^k and Bb^jY can coexist into the above claim.

The above splicing step results in $(a^{i-1}b^jY, Bac^k)$. Finally, we apply the rule $b\#bbY\$Bac\#c$ to the pair $(a^{i-1}b^jY, Bac^q)$, where $a^{i-1}b^jY$ was obtained at the previous step. By this last splicing step we get $a^{i-1}b^{j-1}c^{q-1}$. Note that it is not obligatory that $q = k$.

Consequently, if $a^{i-1}b^{j-1}c^{k-1} \in L(\Gamma) \cap a^+b^+c^+$, then the following conditions are fulfilled:

(I) $i \geq 2$.

(II) $2 + \left\lfloor \frac{i-1}{d+1} \right\rfloor \leq j \leq d(i+1)$. This follows from the previous considerations.

(III) $2 + \left\lfloor \frac{i-3}{d+1} \right\rfloor \leq k \leq d(i+2) - 1$. This last relation follows from two facts. On the one hand, the word $a^{i-1}b^jY$ disappears after at most $d(i+2)$ steps, therefore the word Cc^k that contributes in getting $a^{i-1}b^{j-1}c^{k-1}$ must be obtained in the $(d(i+2)-3)$th step the latest. On the other hand, the same word Cc^k cannot be available earlier than the $(i-3)$th step.

The language $L(\Gamma) \cap a^+b^+c^+$ is not context-free as can be shown by applying the pumping lemma to the string $a^qb^qc^q$ with a sufficiently large q. Given the closure of context-free languages under intersection with regular sets, $L(\Gamma)$ is non-context-free as well. □

A logical next question is whether NRE H systems with delay include the regular languages. Here we show this for extended systems.

Theorem 3.4.4. *The family of languages generated by extended non-reflexively evolving H systems with delay properly includes the regular languages for any delay $d \geq 0$.*

Proof. Let $G = (V, T, P, S)$ be a right linear grammar. Now we can define the extended NRE H system $\Gamma = (V \cup \{X\}, T, A, R)$ that generates $L(G)$, where

$A = \{S\} \cup \{X\gamma \mid B \to \gamma \in P, B \in V\}$,
$R = \{\lambda\#B\$X\#\gamma \mid B \to \gamma \in P, B \in V\} \cup$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$\{X\#\gamma\$X\#\gamma \mid B \to \gamma \in P, B \in V\}$.

All strings without an X represent a sentential form of G. It is easily seen that Γ generates all and nothing but the sentential forms of G. Thus, all terminal strings of G are stable strings in Γ and $L(\Gamma) = L(G)$. Since the only strings that intervene in the rewriting of a string representing a sentential form are the axioms, which are present at every step, and the string itself, this is true for any delay $d \geq 0$. The strictness of the inclusion follows from Theorem 3.4.3. □

An interesting question raised by NRE H systems with delay is whether there exists some kind of hierarchy such that the set of languages generated by systems of some delay d strictly includes those generated by systems of delay $d + 1$. As a partial result, we have the following.

Theorem 3.4.5. *For any delay $d \geq 1$, the family of languages generated by finite extended non-reflexively evolving H systems with delay d includes those generated by such systems of delay $2d$.*

Proof. Given an NRE H system of delay $2d$ $\Gamma = (V, T, A, R)$ we construct an NRE H system of delay d $\Gamma' = (V', T, A', R')$, where

$V' = V \cup \{L_i, R_i, L'_i, R'_i \mid \text{for all } r_i \in R\}$,
$A' = A \cup \{L_iR_i, L'_iR'_i \mid \text{for all } r_i \in R\}$,
$R' = R \cup \{u\#v\$L_i\#R_i, y\#z\$L'_i\#R'_i \mid \text{for all } r_i = u\#v\$y\#z \in R\}$ (1)
$\quad \cup \{\lambda\#R_i\$L'_i\#\lambda, \lambda\#R'_i\$L_i\#\lambda \mid \text{for all } r_i \in R\}$. (2)

If in Γ some rule r_i can be applied to w_1w_2 and w_3w_4, resulting in w_1w_4 and w_3w_2, by the delay these strings will be available for $2d$ steps. In Γ' w_1w_4 and w_3w_2 are also generated, but in addition also w_1R_i, L_iw_2,$w_3R'_i$ and L'_iw_4 by the rules of (1). These strings will be available for d steps in Γ' and will generate w_1w_4 and w_3w_2 at the d next steps, using rule (2). By the delay d of Γ', the strings generated at the last of these steps, will survive for another d steps. So in total, the resulting strings w_1w_4 and w_3w_2 are available at all $2d$ steps after its creation, just as in Γ. It is easily seen that the additional rules have no other effect in the derivation and that they do not create any new terminal strings, so $L(\Gamma) = L(\Gamma')$. □

In fact, we can easily extend this technique to simulate all systems of delay $k \cdot d$ with a system of delay d, adding more intermediate steps (and ensuring that the final result of the rule can be obtained from all intermediate strings). We do not go into details here.

So, there exists some hierarchy in systems with delay. But it is interesting to observe that slowing down the derivation in the smallest possible way, that is adding just one step in the derivation process, we increase the delay by d. This points to the difficulty of simulating systems of delay $d + 1$ and in fact we conjecture incomparability results between systems with delay greater than d, but not equal to $k \cdot d$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Finally let us look at NRE H systems of infinite delay. We have already observed that traditional H systems have infinite delay. However, the two definitions do not coincide completely, that is, for a given H system its normal splicing language and its non-reflexive language when interpreted as a system of infinite delay are not necessarily the same. Nevertheless, we can state the following.

Theorem 3.4.6. *The family of languages generated by finite extended non-reflexively evolving H systems with infinite delay is included in the family of regular languages.*

Proof. First observe that there is little difference between the language of an NRE H system of infinite delay and the traditional splicing language of the same system. Because of the infinite delay, if a rule can be applied at some step $s \geq 1$, it can be applied in all subsequent steps. So all strings created at some point are in the language, just as in the traditional splicing language. Moreover, exactly the same strings are available for rule application in both cases. The only difference concerns strings in the initial language. If at some step a rule can be applied to a string in the initial language, it may disappear at this step. Now, because of the delay, it will disappear in all subsequent steps (unless it is created in some other way) and not be in the final non-reflexive language. So, the language generated by the system equals the traditional splicing language minus a subset of the initial language. Since both of these are regular and regular languages are closed under difference, this language is also regular. □

In the following table we summarize our results on the computational power of extended finite NRE H systems with delay.

Table 3.1: Extended NRE H systems with delay

Delay	Computational Power
0	RE
finite, ≥ 1	$\supset$ REG, contains non-CF
	delay $d \supseteq$ delay $k \cdot d, k \geq 1$
infinite	REG

One observation we can make concerns the surprising jump in power between systems with a finite but arbitrarily large delay, and those with infinite delay. The first can generate non-context-free languages whereas the second do not get beyond regular power.

3.5 Non-Preserving Splicing

In this section, we propose a more general notion of *non-preserving* splicing systems. Comparing the systems in this chapter with the H systems studied by Harju and Margenstern [21], which are basically time-varying H systems of degree 1, one sees they

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

share an important feature: not all strings are maintained from one splicing step to another. Here we generalize these systems as two types of non-preserving splicing, and introduce a third type. This type evolves as the non-reflexively evolving systems of the previous sections, but has the languages defined as is usual in time-varying systems and also basic systems: as the union of the strings in all steps. We investigate these systems using the notion of delay and extend some of the results obtained for non-reflexively evolving systems to the other types.

Harju and Margenstern [21] refer to the property of conserving all strings of all steps as *history preserving*, because every new "generation" contains all the information of the previous generations. They propose to drop the assumption that all strings are kept and introduce a new definition of iterated splicing, called the *non-preserving* iterated splicing operation. This operation, which we will call *strongly non-preserving* iterated splicing is defined as follows. For a splicing system $\Gamma = (V, A, R)$ we write:

$$\sigma^0_{snp}(A) = A, \qquad \sigma^{i+1}_{snp}(A) = \sigma(\sigma^i_{snp}(A)), \qquad \sigma^*_{snp}(A) = \bigcup_{i\geq 0} \sigma^i_{snp}(A).$$

The language generated by the splicing system $H = (V, A, R)$ with strongly non-preserving splicing is defined by $L_{snp}(H) = \sigma^*_{snp}(A)$. The language generated by an extended H system with strongly non-preserving splicing is defined analogously.

Similarly, the non-reflexively evolving splicing of the previous sections can be seen as *weakly non-preserving* splicing. So we may write, for a splicing system $\Gamma = (V, A, R)$:

$$\begin{aligned}\sigma_{wnp}(L) \quad = \quad & \sigma(L) \cup \{w \in L \mid \text{ there is no } x \in L \text{ and no rule } r \in R \text{ such that} \\ & (w, x) \vdash_r (y, z) \text{ or } (x, w) \vdash_r (y, z) \text{ for some } y, z \in V^*\}.\end{aligned}$$

Iterated weakly non-preserving splicing is then defined as follows.

$$\sigma^0_{wnp}(A) = A, \qquad \sigma^{i+1}_{wnp}(A) = \sigma_{wnp}(\sigma^i_{wnp}(A)), i \geq 0 \qquad \sigma^*_{wnp}(A) = \bigcup_{i\geq 0} \sigma^i_{wnp}(A).$$

We introduce here a new way to define the language generated by a H system based on the iterated weakly non-preserving splicing, namely the language generated by a splicing system $\Gamma = (V, A, R)$ with weakly non-preserving splicing which is $L_{wnp}(\Gamma) = \sigma^*_{wnp}(A)$, so as the union of all steps, as in the strongly non-preserving case.

Finally, the non-reflexively evolving language generated by $\Gamma = (V, A, R)$ is denoted by $L_{nr}(\Gamma)$.

3.6 Delay in H Systems With Non-Preserving Splicing

Also the completeness proof in [21] is an example of how the strict separation of splicing steps can be a powerful tool. It will be interesting to see how weakening this

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

separation by using the notion of delay affects the computational power of H systems with strongly/weakly non-preserving splicing.

Let $\Gamma = (V, A, R)$ be a usual splicing system and d be a nonnegative integer or ∞. We define

$$d\sigma_{snp}^{-k}(A) = \emptyset, k \geq 1, \qquad d\sigma_{snp}^{0}(A) = A,$$
$$d\sigma_{snp}^{i+1}(A) = \begin{cases} \sigma(d\sigma_{snp}^{i}(A) \cup d\sigma_{snp}^{i-1}(A) \cup ... \cup d\sigma_{snp}^{i-d}(A)), \text{ if d } \neq \infty \\ \sigma(d\sigma_{snp}^{i}(A) \cup d\sigma_{snp}^{i-1}(A) \cup ... \cup d\sigma_{snp}^{0}(A)), \text{ if } d = \infty \end{cases}$$
for all $i \geq 0$.

The language generated by Γ with strongly non-preserving splicing and delay d is

$$dL_{snp}(\Gamma) = \bigcup_{i \geq 0} d\sigma_{snp}^{i}(A).$$

For the same Γ and d as above we define

$$d\sigma_{wnp}^{-k}(A) = \emptyset, k \geq 1, \qquad d\sigma_{wnp}^{0}(A) = A,$$
$$d\sigma_{wnp}^{i+1}(A) = \begin{cases} \sigma_{wnp}(d\sigma_{wnp}^{i}(A) \cup d\sigma_{wnp}^{i-1}(A) \cup ... \cup d\sigma_{wnp}^{i-d}(A)), \text{ if d } \neq \infty \\ \sigma_{wnp}(d\sigma_{wnp}^{i}(A) \cup d\sigma_{wnp}^{i-1}(A) \cup ... \cup d\sigma_{wnp}^{0}(A)), \text{ if } d = \infty \end{cases}$$
for all $i \geq 0$.

The language generated by Γ with weakly non-preserving splicing and delay d is

$$dL_{wnp}(H) = \bigcup_{i \geq 0} d\sigma_{wnp}^{i}(A).$$

In a similar way one can define the language generated by a H system with non-reflexively evolving splicing and delay d as well as the language generated by the extended variants. We denote by $\mathcal{L}(dSNPH)$, $\mathcal{L}(dWNPH)$, $\mathcal{L}(dNRH)$ the families of languages generated by H systems with strongly non-preserving, weakly non-preserving, non-reflexively evolving splicing and delay $d \geq 0$, respectively and by $\mathcal{L}(dESNPH)$, $\mathcal{L}(dEWNPH)$, $\mathcal{L}(dENRH)$ the families of languages generated by the respective extended variants. $\mathcal{L}(H)$ and $\mathcal{L}(EH)$ denote the families generated by (non-extended and extended) basic splicing systems.

The following result establishes that all types of extended H systems with null delay are equivalent.

Theorem 3.6.1. *$\mathcal{L}(0ESNPH) = \mathcal{L}(0EWNPH) = \mathcal{L}(0ENRH) = RE$, where RE is the family of all recursively enumerable languages.*

Proof. The computational completeness of extended H systems with strongly non-preserving splicing was proved in [21] while the computational completeness of extended H systems with non-reflexively evolving splicing was shown in Theorem 3.3.4.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

The remaining statement can be proved by observing that the construction of 3.3.4 gives exactly the same terminal strings for weakly non-preserving systems. □

The following lemma is useful for showing a similar result for infinite delay.

Lemma 3.6.2. *For any splicing system $H = (V, A, R)$ the following equalities hold:*

$$L(H) = \infty L_{snp}(H) = \infty L_{wnp}(H) = \infty L_{nr}(H) \cup A.$$

Proof. Let $w \in L(H)$, hence $w \in \sigma^i(A)$ for some $i \geq 0$. We prove by induction on i that if $w \in \sigma^i(A)$, then $w \in \sigma^k_{snp}(A)$ for some $0 \leq k \leq i$. The assertion is immediately true for $i = 0$. Let $w \in \sigma^{i+1}(A)$, if $w \in \sigma^i(A)$, then $w \in \sigma^k_{snp}(A)$, for some $0 \leq k \leq i$, by the induction hypothesis. If w is an offspring obtained by splicing from a pair of strings $x, y \in \sigma^i(A)$, then w is an offspring obtained by splicing from a pair of strings $x \in \sigma^k_{snp}(A)$ and $y \in \sigma^j_{snp}(A)$ for some $0 \leq k, j \leq i$. Consequently, $w \in \sigma^t_{snp}(A)$ for some $0 \leq t \leq i + 1$. Therefore, $L(H) \subseteq \infty L_{snp}(H)$.

Let $w \in \infty L_{snp}(H)$, it follows that there exists $i \geq 0$ such that $w \in \infty\sigma^i_{snp}(A)$. We claim that $\infty\sigma^i_{snp}(A) \subseteq \infty\sigma^i_{wnp}(A)$ for any $i \geq 0$. The proof of this statement is based on induction on i. For $i = 0$ the assertion is trivially true. We assume that the assertion is true for any $i \leq t$ and prove it for $t + 1$. Let $w \in \infty\sigma^{t+1}_{snp}(A)$, it follows that either $(x, y) \vdash (w, w')$ or $(x, y) \vdash (w', w)$ for some w', $x \in \infty\sigma^j_{snp}(A)$, $y \in \infty\sigma^k_{snp}(A)$ with $0 \leq j, k \leq t$. By the induction hypothesis, $x \in \infty\sigma^j_{wnp}(A)$ and $y \in \infty\sigma^k_{wnp}(A)$ which implies that $w \in \infty\sigma^{t+1}_{wnp}(A)$. As a direct consequence of our assertion we conclude that $w \in \infty L_{snp}(H)$, so $\infty L_{snp}(H) \subseteq \infty L_{wnp}(H)$.

We take now $w \in \infty L_{wnp}(H)$, that is $w \in \infty\sigma^i_{wnp}(A)$ for some $i \geq 0$. We now claim that there exists $0 \leq t \leq i$ such that $w \in \sigma^t(A)$. Clearly, the assertion is true for $i = 0$. Assume inductively that the assertion is true for any $0 \leq i \leq s$. Assume that $w \in \infty\sigma^{s+1}_{wnp}(A)$; we distinguish two cases:

(i) Either $(x, y) \vdash (w, w')$ or $(x, y) \vdash (w', w)$ for some w', $x \in \infty\sigma^j_{wnp}(A)$, $y \in \infty\sigma^k_{wnp}(A)$ with $0 \leq j, k \leq s$. By the induction hypothesis there exist $0 \leq t_1 \leq j$ and $0 \leq t_2 \leq k$ such that $x \in \sigma^{t_1}(A)$ and $y \in \sigma^{t_2}(A)$. Consequently, $w \in \sigma^{s+1}(A)$.

(ii) $w \in \infty\sigma^k_{wnp}(A)$ for some $0 \leq k \leq s$ and there is no splicing rule r and string $x \in \infty\sigma^s_{wnp}(A) \cup \infty\sigma^{s-1}_{wnp}(A) \cup \cdots \cup A$ such that r can be applied to (w, x) or (x, w). By the induction hypothesis, there exists $0 \leq t \leq k$ such that $w \in \sigma^t(A)$. As a direct consequence of the second assertion we conclude that $w \in L(H)$. Thus $\infty L_{wnp}(H) \subseteq L(H)$.

Finally observe that there is little difference between the language generated by a H system with non-reflexively evolving splicing and infinite delay and the traditional splicing language of the same system. Because of the infinite delay, if a rule can be applied at some step $s \geq 1$ of the non-reflexively evolving splicing, it can be applied in all subsequent steps. So all strings created at some point are in the language generated with non-reflexively evolving splicing and infinite delay, just like in the traditional

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

splicing language. Moreover, exactly the same strings are available for rule application in both cases. The only difference concerns in strings in $\infty\sigma^0_{snp}(A)$, that is, in the initial language. In a non-reflexively evolving system a string in A may disappear if a rule can be applied to it and it will not be in the language. Since $L(H) \supseteq A$ it is clear from the above that $L(H) = \infty L_{nr}(H) \cup A$. □

From the above lemma, the strict inclusion of $\mathcal{L}(H)$ in the set of regular languages and the characterization of $\mathcal{L}(EH)$ in [64] we get:

Theorem 3.6.3.
1. $\mathcal{L}(H) = \mathcal{L}(\infty SNPH) = \mathcal{L}(\infty WNPH) \subset REG$*, where REG is the family of regular languages.*
2. $\mathcal{L}(\infty NRH) \subset REG$*, where REG is the family of regular languages.*
3. $\mathcal{L}(EH) = \mathcal{L}(\infty ESNPH) = \mathcal{L}(\infty EWNPH) = \mathcal{L}(\infty ENRH) = REG$.

Therefore, as far as the extended splicing systems are concerned, at one extreme ($d = 0$) there is the whole class of recursively enumerable languages while at the other ($d = \infty$) there is the whole class of regular languages. Hence, the large area between the two extremes might be refined by the nonnegative values of delay. Although we are not able to give a satisfactory characterization of the classes of languages lying in this area, we present two results which give a glimpse of this area, which according to these results seems quite interesting.

First we show that splicing systems with the types of splicing defined above and any finite delay can generate non-context-free languages.

Theorem 3.6.4. *For all* $d \geq 1$*, each of the families* $\mathcal{L}(dSNPH)$*,* $\mathcal{L}(dWNPH)$*,* $\mathcal{L}(dNRH)$ *contains non-context-free languages.*

Proof. For $\mathcal{L}(dNRH)$, this was proved in Theorem 3.4.3. For the other classes, observe that the strings containing a's, b's and c's always enter some rule with an axiom or with each other. This means that the construction gives exactly the same strings over $a^+b^+c^+$ for strongly non-preserving systems. Finally, since all strings in $a^+b^+c^+$ cannot be involved in further splicing, $dL_{wnp}(H) \cap a^+b^+c^+ = dL_{nr}(H) \cap a^+b^+c^+$. □

Also the construction of Theorem 3.4.4 is valid for all three classes. This gives the following result (the strictness of the inclusion follows from Theorem 3.6.4).

Theorem 3.6.5. *For any* $d \geq 1$*, the following strict inclusions hold:*

$$REG \subset (\mathcal{L}(dESNPH) \cap \mathcal{L}(dEWNPH) \cap \mathcal{L}(dENRH)).$$

We have seen that using the notion of delay in H systems with strongly/weakly non-preserving or non-reflexively evolving splicing, we get an interesting scale, where the extremes correspond to known and well-studied classes of H systems as well as the

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

extremes of the Chomsky hierarchy. Extended H systems of delay 0 have shown to be equivalent to the recursively enumerable languages and non-extended H system of infinite delay have shown to be equivalent to classic finite H systems, with the extended variant characterizing the regular languages.

Moreover this scale also connects the new types of systems with the traditional ones, it gives a new insight in the computational properties of traditional finite splicing systems. For both H systems with non-reflexively evolving and strongly/weakly non-preserving splicing we have a remarkable jump in power between systems with a finite but arbitrarily large delay, and those with infinite delay. The first can generate non-context-free languages whereas the second do not get beyond the regular limit. This indicates that it is just this infinite survival of strings that can be held responsible for the computational weakness of these systems.

As mentioned before, imposing this infinite survival can be considered too restrictive for both biochemical and mathematical considerations. This means that studying alternatives like the ones considered here can provide theoretical insights and can also make splicing more attractive as a model for molecular computing, given the fact that in this context we can find finite systems which are still computationally powerful.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

4 Time-varying H Systems Revisited

4.1 Introduction

In this chapter, we cast a new look on time-varying distributed H systems. In their original definition, where only new strings are passed to the next component, this language definition in itself is already enough to obtain computational completeness. Here, we consider two types of time-varying H systems with weaker language definitions, based on the usual definition of splicing systems: The next generation of strings consists of the union of all existing strings and the newly created strings. We show that if all strings, both old and new, are passed to the next component these systems are regular in power. If however, the new strings pass to the next component and the existing ones remain accessible to the current one, we prove that systems with 4 components are already computationally complete.

In Section 2.3.4 we saw that in a series of papers, including [61], [58] and [50], the degree (i.e. the number of different sets of splicing rules) of the time-varying H systems needed to obtain computational completeness has been decreased progressively. Finally, in [49], it was shown that time-varying distributed H systems of degree 1 can generate all recursively enumerable languages. Such systems are really no longer distributed nor time-varying, having only a single set of splicing rules.

This result can be explained by the way the language is defined in time-varying H systems: From one splicing step to the next, only the newly created strings are kept. The result in [49] shows that this way of defining the splicing language alone is sufficient to obtain computational completeness. Recently ([21], see also Chapter 3), this definition has also been studied in the context of basic finite splicing systems.

The fact that the computational power of the language definition alone makes the distributed architecture superfluous, suggests that it may be interesting to consider time-varying H systems with a weaker language definition. A candidate for such a definition is easy to find, since in basic splicing systems as well as in practically all splicing formalisms another definition is used. In this definition, the strings passed to the next step consist of the union of the strings present at this step and the new strings created by applying the splicing rules. This definition is considerably weaker from a computational point of view. In the case of basic extended splicing systems the definition which only conserves the new strings yields computationally complete systems, whereas defining the new generation by the union of old en new strings gives systems of only regular power.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

We introduce two new language definitions for time-varying H systems. In the first one, all strings, both new and already present, are passed to the next component. In the second one, the new strings are passed to the next component, whereas the existing strings remain accessible to the current set of rules. We know that both of these definitions are weaker than the original one, since in both cases systems of degree one are equivalent by definition to extended finite H systems, which generate exactly all regular languages.

After reviewing the basic definitions, we formally define the two new variants. We then prove that the first variant generates only regular languages for any degree (any number of components). For the second variant, we show that systems of degree of at least 4 are computationally complete.

4.2 New Definitions

Now we can formally define the systems presented informally in the introduction. In the first type, which we call time-varying H systems with *full transfer*, all strings, both new and already present, are passed to the next component.

Definition 4.2.1. *A* time-varying distributed H system (of degree n) with full transfer *is a construct:*

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$.
We define

$$L_0 = A,$$
$$L_k = L_{k-1} \cup \sigma_{h_i}(L_{k-1}), \text{ for } i \equiv k(mod\ n), k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k \geq 0} L_k) \cap T^*.$$

The definition of the second type is slightly more intricate. Here, the newly created strings are passed to the next component, and the strings already present before applying the rules, remain accessible to the current component. Formally, we define

Definition 4.2.2. *A* time-varying distributed H system (of degree n) with partial transfer *is a construct:*

$$D = (V, T, A, R_1, R_2, ..., R_n),$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

where V is an alphabet, $T \subseteq V$ is a terminal alphabet, $A \subseteq V^*$ is a finite set of axioms, and components R_i are finite sets of splicing rules over V, $1 \leq i \leq n$.
We define

$$L_{-k} = \emptyset,\ k \geq 1,$$
$$L_0 = A,$$
$$L_k = \sigma_{h_i}(\bigcup_{j \geq 0} L_{k-j \cdot n-1}), \text{ for } i \equiv k(mod\ n), k \geq 1, 1 \leq i \leq n, h_i = (V, R_i).$$

The language generated by D is, by definition:

$$L(D) = (\bigcup_{k \geq 0} L_k) \cap T^*.$$

The idea of distribution of strings to different components is reminiscent of communicating distributed H systems (see [65]), also known as test tube systems. But in these systems, components are full systems rather than sets of rules, and the contents of each component are redistributed to all components according to filters. This means our approach does not carry over naturally to these systems. Also the distinction between new and existing strings, which is an intrinsic part of time-varying H systems, has no natural expression there. This is still true for alternative types of communicating distributed H systems incorporating some aspects of time-varying systems ([16],[77].

In what follows, we use the abbreviations $FT\text{-}TVH_n$ and $PT\text{-}TVH_n$, $n \geq 1$ to denote the families of languages generated by time-varying H systems with full and partial transfer respectively, and of degree at most n. $FT\text{-}TVH_*$ and $PT\text{-}TVH_*$ correspond to the families of languages generated by such systems of any degree.

4.3 Computational Power

We start the investigation of the computational power of these systems with the following observation.

Theorem 4.3.1. $FT\text{-}TVH_1 = PT\text{-}TVH_1 = REG$.

Proof: It is easily verified that with only one component, these systems reduce to systems equal by definition to extended H systems with a finite set of rules and a finite initial language. The theorem follows from the the characterization of these systems in [64]. □

Theorem 4.3.2. $FT\text{-}TVH_* = REG$.

Proof: To show the theorem we will prove that any time-varying H system with full transfer $\Gamma = (V, T, A, R_1, ..., R_n)$ generates the same language as the extended finite H

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

system $H = (V, T, A, R)$, where

$$R = \bigcup_{i \geq 1}^{n} R_i.$$

As before, the theorem then follows from the characterization of extended finite H systems in [64].

From the definition it is obvious that $L(\Gamma) \subseteq L(H)$. For the other direction, we show by induction that for all $i \geq 0$,

$$\sigma_h^i(A) \subseteq \bigcup_{k \geq 0} L_k,$$

where $h = (V, R)$ and L_k is defined for Γ as in Definition 4.2.1. For $i = 0$, $\sigma_h^0(A) = L_0 = A$. Now, assuming the assertion is true for i, we show it is true for $i + 1$.

Suppose that $x, y \in \sigma_h^i(A)$ and that $x, y \vdash_r w$ for some $r \in R$. By the induction hypothesis, $x, y \in \bigcup_{k \geq 0} L_k$ and, by the definition of H, $r \in R_j$ for some $1 \leq j \leq n$. This means that there exists an $s \geq i$ such that $s \equiv j (mod\ n)$. Then w will be in L_{s+1} and $w \in \bigcup_{k \geq 0} L_k$. □

Theorem 4.3.3. *PT-TVH*$_4$ = *RE*.

Proof: Consider a type-0 grammar $G = (N, T, S, P)$. We denote by $\alpha_1, ..., \alpha_{n-1}$ the symbols in $N \cup T$. Let $\alpha_n = F$ be a new symbol. Let $u_j \to v_j$ for $n + 1 \leq j \leq m$ denote the rules in P and assume we have $u_i = v_i = \alpha_i$ for $1 \leq i \leq n$.

We construct the time-varying H system with partial transfer
$\Gamma = (V, T, A, R_1, R_2, R_3, R_4)$, with

$$\begin{aligned}
&V = N \cup T \cup \{X, Y, Z, Z', Z_0, Z_0', F\} \cup \{X_i, Y_i \mid 0 \leq i \leq m\} \\
&\cup \{Z_j \mid 1 \leq j \leq m\}, \\
&A = \{XSFY, XZ', ZY, Z_0, Z_0'\} \cup \{X_jZ', ZY_j \mid 0 \leq j \leq m\} \\
&\cup \{X_jv_jZ_j' \mid 1 \leq j \leq m\}, \\
&R_1 = Q \cup \{\#Y\$\#Y, X_0\#\$X_0\#\} \cup \{\#Y_j\$\#Y_j \mid 1 \leq j \leq m\}, \\
&R_2 = Q \cup \{X\#\$X\#, \#Y_0\$\#Y_0\} \cup \{X_j\#\$X_j\# \mid 1 \leq j \leq m\}, \\
&R_3 = Q \cup \{\#Y_0\$Z\#Y, \#FY_0\$Z_0\#\} \cup \{\#u_jY\$Z\#Y_j \mid 1 \leq j \leq m\} \\
&\cup \{\#Y_j\$Z\#Y_{j-1} \mid 1 \leq j \leq m\}, \\
&R_4 = \{X_0\#\$X\#Z', X_0\#\$\#Z_0' \mid 1 \leq j \leq m\} \\
&\cup \{X\#\$\#Z_j, X_j\#\$X_{j-1}\#Z' \mid 1 \leq j \leq m\},
\end{aligned}$$

where

$$Q = \{Z\#\$Z\#, \#Z'\$\#Z', Z_0\#\$Z_0\#, \#Z_0'\$\#Z_0'\} \cup \{\#Z_j'\$\#Z_j' \mid 1 \leq j \leq m\}.$$

This system simulates G using the rotate-and-simulate technique first used in [63]. We simulate a rule application by a splicing at the right end of the string. To ensure that

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

all symbols in the string can be rewritten we circularly permutate the current sentential form. Here, as in [58], simulation and rotation are done in the same way: A suffix of the current string is removed and the corresponding string is added to the left. This is done by the rules in R_3 and R_4. The rules in R_1 and R_2 ensure these operations are applied correctly. We will prove the two inclusions $L(G) \subseteq L(\Gamma)$ and $L(G) \supseteq L(\Gamma)$.

1. $L(G) \subseteq L(\Gamma)$. Consider a string of the form XwY in component 1. This string encodes the current sentential form of G. Initially $w = SF$. This string is passed unchanged by the rule $\#Y\$\#Y$ to R_2 where it is again passed unchanged to R_3 using the rule $X\#\$X\#$. Note that all other axioms are passed to components 2, 3 and 4 by the rules in Q. In R_3, if $w = w'u_i$ for some $1 \leq i \leq m$ we can perform

$$(Xw'u_iY, ZY_i) \vdash (Xw'Y_i, Zu_iY).$$

The string $Xw'Y_i$ is passed to R_4 where we can apply

$$(X_iv_iZ_i', Xw'Y_i) \vdash (X_iv_iw'Y_i, XZ_i').$$

From R_1 this string is passed unchanged to R_2 and R_3 (by rules $\#Y_j\$\#Y_j$ and $X_j\#\$X_j\#$ respectively). In R_3 the subscript of Y is decreased by 1 using a rule $\#Y_j\$Z\#Y_{j-1}$. The resulting string $X_iv_iw'Y_{i-1}$ is passed to R_4 where by applying a rule $X_j\#\$X_{j-1}\#Z'$ the subscript of X is decreased by 1. Iterating this process, we get to a string of the form $X_0v_iw'Y_0$. This string passes through R_1 and R_2 unchanged. In R_3 Y_0 is replaced by Y and in R_4 we substitute X_0 by X. Thus we have passed from $Xw'u_iY$ to $Xv_iw'Y$. If $1 \leq i \leq n$ we have rotated one symbol, since $u_i = v_i = \alpha_i \in N \cup T \cup \{F\}$. If $n+1 \leq i \leq m$ we have simulated the application of the rule $u_i \to v_i$. Iterating this procedure, we can simulate any rule of G at any position. So, if in G $S \Rightarrow^* x_1x_2$, we can produce the string Xx_2Fx_1Y in Γ, and by circular permutation also $X_0x_1x_2FY_0$. In R_3 we can apply the rule $\#FY_0\$Z_0\#$ to obtain $X_0x_1x_2$: This string gets to R_4 where we remove the X_0 with the rule $X_0\#\$\#Z_0'$. If the resulting string x_1x_2 is in T^*, then $x_1x_2 \in L(\Gamma)$. So, $L(G) \subseteq L(\Gamma)$.

2. $L(G) \supseteq L(\Gamma)$. To see that Γ does not produce any strings not in $L(G)$, note that to continue the simulation of G, the strings should be passed to the next component. The rules of Γ are such that strings that remain in the current component do not interfere. If these strings encode valid simulations of G they remain unchanged and can resume the simulation at a later moment. All invalid simulations will be 'trapped' in one component and will not lead to strings in T^*.

Specifically, suppose we have performed in R_3

$$(Xw'u_iY, ZY_i) \vdash (Xw'Y_i, Zu_iY)$$

and in R_4

$$(X_jv_jZ_j', Xw'Y_i) \vdash (X_jv_jw'Y_i, XZ_j')$$

for some $1 \leq i, j \leq m$. As mentioned before, the axioms except $XSFY$ are available in

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

all components thanks to the rules in Q.

A string $X_j v_j w' Y_i$ with $i \geq 1$ can pass to R_2 by the rule $\#Y_i\$\#Y_i$. Then it passes to R_3 using $X_j\#\$X_j\#$ provided that $j \geq 1$. In R_3 we decrement the subscript of Y and in R_4 the subscript of X. These are the only operations possible in these components. This process is repeated until some subscript reaches zero.

Suppose that after R_4 we have a string of the form $X_0 w Y_k$ for $k \geq 1$. This string is passed to R_2 by the rules $X_0\#\$X_0\#$ and $\#Y_j\$\#Y_j$. Now, there is no rule in R_2 that can be applied to the string. So, it will remain in this component and never yield a terminal string. If after R_4 we have a string of the form $X_k w Y_0$ for $k \geq 1$, no rule in R_1 can be applied to it. Thus, this string will not derive a word in T^*. With subscript equal to zero, only strings of the form $X_0 w Y_0$ can pass through R_1 and R_2. Then in R_3 and R_4, X_0 and Y_0 are replaced by X and Y and a new simulation or rotation step can start.

So, the only strings that can continue the simulation are those where $i = j$ at the moment that we start to decrement the subscript. This means we have correctly simulated a rule in P or correctly rotated a symbol.

All strings that are produced as a by-product do not lead to terminal strings. All these strings contain the symbol Z or Z', so they are passed to all components by the rules in Q. But they do not interfere with the simulation process. As an example, consider a string $Z u_i Y$ produced in R_3. When it returns to R_3, we can replace $u_i Y$ by Y_i and then, also in R_3, decrease the subscript. When reaching Y_0, it can be removed or rewritten by Y. The strings ZY and ZY_i are axioms, and other strings of the form Zw or ZwY cannot enter in any splicing rule with strings of the form $Xw'Y$.

Finally, the symbol Y_0 can only be removed when it follows the symbol F, which guarantees that only the correct permutation yields a terminal string. Thus, the only terminal strings that are generated by Γ correspond to strings in $L(G)$. This concludes the proof that $L(G) \supseteq L(\Gamma)$. □

This last result gives rise to an interesting open question. If time-varying H systems with partial transfer of degree 4 generate all RE languages and those of degree 1 generate only regular languages, what is the power of systems of degree 2 and 3? We conjecture that systems of degree 2 can be shown to be universal, by using the technique of forcing the correct derivation to go through all components, as we did in Theorem 4.3.3. As an example, we give a very simple time-varying H system with partial transfer of degree 2 that generates a non-regular language.

$\Gamma = (V, T, A, R_1, R_2)$, with

$$
\begin{aligned}
V &= \{a, b, X, Y, Z\},\\
T &= \{a, b\},\\
A &= \{XabY, ZbY, XaZ\},\\
R_1 &= \{X\#a\$Xa\#Z, X\#a\$\#ZbY, \#bY\$XaZ\#\} \cup \{ZbY\#\$ZbY\#\},\\
R_2 &= \{b\#Y\$Z\#bY\}.
\end{aligned}
$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

The reader can verify that $L(\Gamma) = \{a^n b^n \mid n \geq 1\}$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

5 Multiple Splicing

5.1 Introduction

In this chapter we introduce a new type of splicing, multiple splicing, which differs from the usual definition in that several (not necessarily distinct) rules can be applied simultaneously to the same string. We consider restricted and unrestricted versions of multiple splicing. We define (k, p)-splicing, where k rules, different or not, out of p different rules, are applied to a string at the same time. We show that restricted and extended multiple splicing systems with (k, p)-splicing are computationally complete for $k = p = 2$ and $k \geq p \geq 3$, strictly more powerful than finite automata for $k = 2$ and $p \geq 3$ or $p = *$ and contain non-regular languages for $k = 2$ and $p = 1$. For the unrestricted case, a weaker and more realistic definition, we prove that no increase in the computational power is observed.

In the theory of splicing systems it is a standard assumption that more than one rule cannot act simultaneously on a string. If multiple rules are applicable to a string then those rules are given different copies of the string so as to avoid simultaneous application of multiple rules, and for such "independent" splicing to be possible, it is assumed that there is infinite supply of each string existing at each point of computation. That is, if two rules, say r_1 and r_2, can be applied to a string w we consider that r_1 is applied to one copy of w and r_2 to another copy, and after that splicing step, the language contains the resulting strings of both applications.

Here, we study the splicing systems in which the assumption that more than one rule can act on the same string at the same time is removed. Although it may look weird at the first glance, this "multiple-site" extension is natural, because two restriction enzyme can act on the same DNA strand so long as their restriction sites do not overlap (and, of course, the sites are far enough from each other so that there will be no interference), and the ensuing recombinations can occur at all of the restriction sites at the same time. The "multiple-site" we introduce here captures this idea. To be more specific, in the systems we introduce here, we allow more than one (not necessarily distinct) rules to apply to a given string. It should be noted that the rules are applied independently, depending only on the string itself, not on which other rules are applied to other parts of the string. This distinguishes our definition from other approaches, which consider simultaneous applications of splicing-like operations, like [52] further considered in [32].

We consider two possibilities for defining multiple splicing. In the first case, which we call *restricted* multiple splicing, we allow for exactly k rules, not necessarily dis-

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

tinct, chosen from a collection of rules of cardinality of p, to apply to a string simultaneously. We call this (k, p)-*splicing*. We study the computational power of this system for different values of k and p. Specifically, we show that extended (nonterminals are allowed) (k, p)-splicing systems (i) are *computationally complete* (that is, identical to the recursively enumerable) if either $k = p = 2$ and or $k \geq p \geq 3$, (ii) are strictly more powerful than finite automata if $k = 2$ and either $p \geq 3$ or there is no restriction on p (we will use $p = *$ to denote this unrestricted case), (iii) contain non-regular languages if $k = 2$ and $p = 1$.

In the case which we call *unrestricted multiple splicing*, there is no restriction on the number of cuts or on the number of rules involved in the splicing operation. This definition is closer to the actual behavior of DNA molecules under the influence of restriction enzymes; strings are cut at arbitrarily many sites, and the fragments are recombined in all possible ways. We show that this weaker definition is equivalent to the traditional definition as to their computational power.

5.2 Multiple Splicing

We now fix some terminology and notation for the discussion of multiple splicing. Let R be a set of splicing rules over some alphabet V. For a word $w \in V^*$, a set $X \subseteq R$, which we assume to be ordered, and an integer $k \geq 1$, we define

$$\begin{aligned} w_{cut_k}(X) \quad = \quad & \{w_1 x_1 \lhd_{i_1}^{p_1}, {}_{i_1}^{p_1}\rhd x_1' w_2 x_2 \lhd_{i_2}^{p_2}, {}_{i_2}^{p_2}\rhd x_2' w_3 x_3 \lhd_{i_3}^{p_3}, \ldots, {}_{i_k}^{p_k}\rhd x_k' w_{k+1} \mid \\ & w = w_1 x_1 x_1' w_2 x_2 x_2' \ldots x_k x_k' w_{k+1},\ p_j = \begin{cases} l, \text{ if } r_{i_j} = x_j \# x_j' \$ y_j \# y_j' \\ r, \text{ if } r_{i_j} = y_j \# y_j' \$ x_j \# x_j' \end{cases} \\ & r_{i_j} \in X, \text{ for all } 1 \leq j \leq k\}. \end{aligned}$$

Furthermore, $w_{cut}(X) = \bigcup_{k \geq 1} w_{cut_k}(X)$. The set X is omitted when $X = R$. We extend this definition by considering a set E of strings in place of w and define $E_{cut_k}(X) = \bigcup_{w \in E} w_{cut_k}(X)$. We define analogously E_{cut_k}, $E_{cut}(X)$, and E_{cut}.

For a set of strings E, a set of splicing rules X, and an integer k as above we define

$$\begin{aligned} E_{recomb_k}(X) \quad = \quad & \{z_1 z_2 \ldots z_{k+1} \mid z_1 \lhd_{i_1}^{p_1}, {}_{i_k}^{p_{2k}}\rhd z_{k+1} \in E_{cut_k}(X) \\ & {}_{i_j}^{p_{2j}}\rhd z_{j+1} \lhd_{i_{j+1}}^{p_{2j+1}} \in E_{cut_k}(X), 1 \leq j \leq k-1, \\ & p_{2j-1} \in \{l, r\},\ p_{2j} = \begin{cases} r, \text{ if } p_{2j-1} = l, \\ l, \text{ if } p_{2j-1} = r, \end{cases} 1 \leq j \leq k\} \end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

We define analogously $E_{recomb}(X)$, E_{recomb_k}, E_{recomb}. For instance,

$$\begin{aligned} E_{recomb}(X) \quad = \quad & \{z_1 z_2 \dots z_{k+1} \mid k \geq 1,\ z_1 \lhd_{i_1}^{p_1},\ {}_{i_k}^{p_{2k}}\rhd z_{k+1} \in E_{cut}(X) \\ & {}_{i_j}^{p_{2j}}\rhd z_{j+1} \lhd_{i_{j+1}}^{p_{2j+1}} \in E_{cut}(X), 1 \leq j \leq k-1 \\ & p_{2j-1} \in \{l, r\},\ p_{2j} = \begin{cases} r, \text{ if } p_{2j-1} = l, \\ l, \text{ if } p_{2j-1} = r, \end{cases} 1 \leq j \leq k\} \end{aligned}$$

Given a set of splicing rules R, a language L, and two positive integers k and p, the (k, p)-splicing language $\phi_R^*(L, k, p)$ is defined as follows.

$$\begin{aligned} & \phi_R^0(L, k, p) = L, \\ & \phi_R^{i+1}(L, k, p) = \phi_R^i(L, k, p) \cup \bigcup_{X \subseteq R, \|X\| = p} (\phi_R^i(L, k, p))_{recomb_k}(X),\ i \geq 0, \\ & \phi_R^*(L, k, p) = \bigcup_{i \geq 0} \phi_R^i(L, k, p). \end{aligned}$$

We omit the subscript R when the set of rules is clear from the context. Also, we write $\phi_R^*(L, k, *)$ when $p = \|R\|$, $\phi_R^*(L, *, p)$ when k is not fixed, and $\phi_R^*(L, *, *)$ when $p = \|R\|$ and k is not fixed. The (k, p)-splicing language generated by a splicing system $H = (V, A, R)$ is defined as $(k, p)L(H) = \phi^*(A, k, p)$. An extended H system $H = (V, T, A, R)$ generates the (k, p)-splicing language $(k, p)L(H) = \phi^*(A, k, p) \cap T^*$. The $(k, *)$-, $(*, p)$-, and $(*, *)$-splicing languages are defined analogously.

To close this section, we present an example to clarify our definition. Consider the splicing system $\Gamma = (V, A, R)$, with

$$\begin{aligned} V &= \{X, Y, a, b\}, \\ A &= \{XabY, YbaX\}, \\ R &= \{X\#a\$Y\#b, b\#Y\$a\#X\}. \end{aligned}$$

Now, $(2, 2)L(\Gamma) = (2, *)L(\Gamma) = A \cup \{XbaY, YabX\}$. However, $(2, 1)L(\Gamma) = A$, since we are forced to cut every string in two parts and we need both rules of R for doing this.

5.3 Restricted Multiple Splicing

In this section we investigate the effect of restricting the number of splicing sites in every string to a given constant. Clearly, for any splicing system H, extended or not, $(1, *)L(H) = (1, 1)L(H) = L(H)$ holds. We denote by $\mathcal{L}((k, p)H)$ the family of (k, p)-splicing languages generated by splicing systems. We write $\mathcal{L}((k, p)EH)$ for the (k, p)-splicing languages generated by the extended variants of splicing systems. The main result from [12, 70] can be stated as follows:

Theorem 5.3.1. *Both $\mathcal{L}((1, 1)EH)$ and $\mathcal{L}((1, *)EH)$ equal the class of regular lan-*

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

guages.

We now consider the $(2,2)$-splicing operation and prove the main result of this section, namely that extended splicing systems based on the $(2,2)$-splicing operation are computationally complete.

Theorem 5.3.2. *$\mathcal{L}((2,2)EH)$ equals the family of recursively enumerable languages.*

Proof. It suffices to prove that every recursively enumerable language is the $(2,2)$-splicing language generated by an extended splicing system. We will use the rotate-and-simulate technique introduced in [63]. The idea is that we simulate a type-0 grammar by simulating rule applications at one end of the string. To ensure that we simulate all possible rule applications, we circularly rotate the string so that the simulation can affect each part of the sentential form.

In our proof, we construct the splicing system in such a way that for any set of two rules, either no $(2,2)$-splicing is possible, or only $(2,2)$-splicing between exactly two strings takes place. This last process is illustrated in Figure 5.1. This allows us to control our derivations.

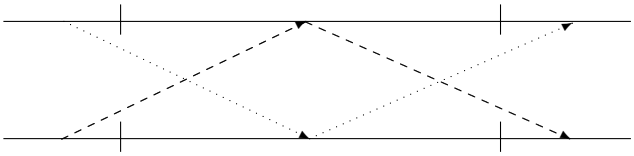

Figure 5.1: The 2, 2-splicing operation involving two strings

Specifically, let $G = (N, T, S, P)$ be a type-0 grammar. We construct an extended splicing system based on the $(2,2)$-splicing operation $\Gamma = (V, T, A, R)$ such that $L(G) = (2,2)L(\Gamma)$, where

$$V = N \cup T \cup \{X, Y, Z, B, W, Q\} \cup \{Z_a \mid a \in (N \cup T)\} \cup \{W_v \mid u \to v \in P\},$$
$$A = \{XBSY, QQ\} \cup \{XaZZ_aY \mid a \in N \cup T \cup \{B\}\} \cup$$
$$\{XvWW_vY \mid u \to v \in P\}.$$

We assume without loss of generality that none of the following belong to $N \cup T$: X, Y, Z, B, Q, W, Z_a for any $a \in N \cup T$, and W_v for any $u \to v \in P$. In the splicing system, the sentential form will be represented by a string $XwBvY, w, v \in (N \cup T)^*$. The symbol B marks the beginning of the sentential form, hence we start out with $XBSY$. The rotation and the rule simulation is done in a similar way to that from [58], namely the rightmost symbol is shifted to the beginning of the string (in case of rotation) or the part in the right-hand side of the string which can be rewritten by a rule in P is removed and the right-hand side of that rule is added to the beginning of the string (in case of rule simulation).

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

The set R contains the following rules:

Rotate:
$X\#\lambda\$Xa\#Z \qquad a \in N \cup T \cup \{B\}$
$\lambda\#aY\$Z_a\#Y \qquad a \in N \cup T \cup \{B\}$

Suppose we have a string $XwaY$ for some $a \in N \cup T \cup \{B\}$ and we want to shift a to the beginning of the string. In order to cut $XwaY$ at two sites we have to take a set of two rules $\{X\#\lambda\$Xb\#Z, \lambda\#aY\$Z_a\#Y\}$ for some $b \in N \cup T \cup \{B\}$. However, no recombination is possible if one chooses a set of two rules $\{X\#\lambda\$Xb\#Z, \lambda\#aY\$Z_a\#Y\}$ for two different symbols $a, b \in N \cup T \cup \{B\}$. We now consider the set of two rules $\{X\#\lambda\$Xa\#Z, \lambda\#aY\$Z_a\#Y\}$ for some $a \in N \cup T \cup \{B\}$ that can be applied to the strings $XwaY$ and $XaZZ_aY$ yielding $XawY$ and XZZ_aaY by a $(2,2)$-splicing.

Simulate:
$X\#\lambda\$Xv\#W \qquad u \to v \in P$
$\lambda\#uY\$W_v\#Y \qquad u \to v \in P$

Simulation goes in much the same way as rotation. For a string $XwuY$ and a production $u \to v \in P$, we have a $(2,2)$-splicing with the string $XvWW_vY$, giving XWW_vuY and the new string $XvwY$ encoding a circular permutation of the sentential form of G. As above, any choice of a set of two splicing rules $\{X\#\lambda\$Xv\#W, \lambda\#xY\$W_y\#Y\}$ for some rule $x \to y$, $y \neq v$, leads to no possible recombination.

Retrieve result:
$XB\#\lambda\$\lambda\#Q$
$\lambda\#Y\$Q\#\lambda$

Finally, the two rules above can be applied only when the string is of the form $XBwY$, that is in its unrotated form. Now these two rules can be applied to this string and QQ, yielding $XBQQY$ and w. If w is a terminal string, it will be in $L(\Gamma)$.

From the explanations above it should be clear that the $(2,2)$-splicing language generated by Γ contains all strings in $L(G)$. To see that it generates only those strings, we first notice that only the following sets of two rules lead to successful $(2,2)$-splicing: $\{X\#\lambda\$Xa\#Z, \lambda\#aY\$Z_a\#Y\}$, $a \in N \cup T \cup \{B\}$, $\{X\#\lambda\$Xv\#W, \lambda\#uY\$W_v\#Y\}$, $u \to v \in P$, $\{XB\#\lambda\$\lambda\#Q, \lambda\#Y\$Q\#\lambda\}$.

We now consider the residual strings of the form XZZ_aaY, $a \in N \cup T \cup \{B\}$. They can enter in a $(2,2)$-splicing with $XaZZ_aY$ only, yielding exactly the same two strings. The same goes for XWW_vuY and $XvWW_vY$. Finally, $XBQQY$ can enter in a $(2,2)$-splicing with some $XBwY$, also giving exactly the same strings. This shows that these strings do not interfere with the simulation of derivations in G and only strings in $L(G)$ are generated. □ □

Looking more closely at the construction, we see that it extends to splicing systems based on the (k, p)-splicing operation for any $k \geq p \geq 3$. Namely,

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Theorem 5.3.3. *$\mathcal{L}((k,p)EH)$ equals the class of recursively enumerable languages for all $k \geq p \geq 3$.*

Proof. The general idea is the same: we construct the splicing system in such a way that for any set of p rules, either no (k,p)-splicing is possible, or only (k,p)-splicing between exactly two strings takes place. Starting from the construction of Theorem 5.3.2, it suffices to take the set of axioms as

$$\begin{aligned} A &= \{XBSYX_1X_2\ldots X_{p-1}X_{p-2}^{k-p+1}, QQ\} \cup \{XaZZ_aYX_1X_2\ldots X_{p-1}X_{p-2}^{k-p+1} \mid \\ &\quad a \in N \cup T \cup \{B\}\} \cup \{XvWW_vYX_1X_2\ldots X_{p-1}X_{p-2}^{k-p+1} \mid u \to v \in P\}, \end{aligned}$$

where $X_1, X_2, \ldots, X_{p-2}$ are new symbols, and add the following set of splicing rules: $\{X_i\#\lambda\$X_i\#\lambda \mid 1 \leq i \leq p-2\}$. □ □

We do not know an exact characterization of the class of $(2,m)$-splicing languages, $m \geq 3$, generated by extended splicing systems but we can state that this class properly includes the class of regular languages.

Theorem 5.3.4. *For all $m \geq 3$, $\mathcal{L}((2,m)EH)$ properly includes the class of regular languages.*

Proof. We first show that every regular language can be generated by an extended splicing system based on the $(2,m)$-splicing for some $m \geq 3$. Let $M = (Q, V, \delta, q_0, \{q_f\})$ be a nondeterministic finite automaton with just one final state such that no transition enters q_0 and no transition goes out from the final state q_f. We construct the extended splicing system $H = (U, V, A, R)$, where

$$\begin{aligned} U &= V \cup Q \cup \{X, ZZ\} \\ A &= \{qaq'XX \mid q' \in \delta(q,a), q \in Q, a \in V\} \cup \{ZZ\} \\ R &= \{a\#q\$q\#b \mid a,b \in V, q \in Q \setminus \{q_f\}\} \cup \{X\#X\$X\#X \mid q \in Q\} \cup \\ &\quad \{q_0\#a\$\lambda\#Z, a\#q_f\$Z\#\lambda \mid a \in V\}. \end{aligned}$$

If $\|R\| < m$, then one adds some "dummy" splicing rules to R that cannot be used in any splicing in order to complete the number of rules in R up to m. All paths in M of length one are axioms in the form $qaq'XX$. Inductively, we may assume that all paths in M of length t are generated by H with $(2,m)$-splicing and prove that all paths of length $t+1$ can be generated by H. Thus, from $qwq'XX$ and $q'aq''XX$ we get $q'waq''XX$. Note that this splicing step can be accomplished with two rules and the other rules do not influence the result. Therefore, all paths in M can be generated by H with $(2,m)$-splicing. Now the strings accepted by M are squeezed out by the rules in the set $\{q_0\#a\$\lambda\#Z, a\#q_f\$Z\#\lambda \mid a \in V\}$. Conversely, we stress that only strings of the form q_oxq_fXX can lead to strings in $(2,m)L(H)$. Furthermore, for every string of the form $qxq'XX$ generated by H with $(2,m)$-splicing, $q' \in \delta(q,x)$ holds.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

To show the strictness of the inclusion, consider the splicing system $H = (V, A, R)$ with

$$V = \{a, b, Z, X, Y, W\},$$
$$A = \{ZabZ, ZaabbZ, ZaXXY, WWZbZ\},$$
$$R = \{Z\#aa\$Za\#X, bb\#Z\$Z\#bZ, X\#Y\$W\#W\}.$$

It is rather plain that this non-extended system generates $\{Za^nb^nZ \mid n \geq 1\} \cup \{ZaXXY, WWZbZ\}$ by $(2, m)$-splicing. This is clearly a non-regular language. Obviously, the non-regular language $\{a^nb^n \mid n \geq 1\}$ can be generated by an extended splicing system based on the $(2, m)$-splicing, $m \geq 3$, as well as $(2, *)$-splicing. □ □

From the previous proof we can immediately infer that:

Corollary 5.3.5. *1. The class $\mathcal{L}((2, *)EH)$ properly includes the class of regular languages.*
2. For all $m \geq 3$, both $\mathcal{L}((2, *)H)$ and $\mathcal{L}((2, m)H)$ contains non-regular languages.

A natural question arises: What is the computational power of splicing system based on the $(2, 1)$-splicing? We do not know whether all regular languages can still be generated but we can state:

Theorem 5.3.6. *Both classes $\mathcal{L}((2, 1)H)$ and $\mathcal{L}((2, 1)EH)$ contain non-regular languages.*

Proof. We consider the splicing system

$$H = (\{a, b, Z\}, \{ZabaZ, ZaZaZ\}, \{Z\#a\$a\#Z\}).$$

One can easily check that $(2, 1)L(H) = \{Za^nba^nZ \mid n \geq 1\} \cup \{ZaZaZ, ZZZZ\}$ which is not regular. □ □

A more precise characterization of the computational power of the $(2, m)$-splicing, $m \neq 2$, as well as (k, p)-splicing with k, p not considered above remains to be settled.

5.4 Unrestricted Multiple Splicing

We now turn to the $(*, m)$-splicing for different values of m. A natural question to ask is what computational power unrestricted multiple splicing systems have. Here we provide a partial answer.

Theorem 5.4.1. *Let H be an arbitrary splicing system. Then $(*, *)L(H) = L(H)$ and for all $m \geq 1$, $(*, m)L(H) = L(H)$.*

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Proof. Let $H = (V, A, R)$ be a splicing system; it suffices to prove the equality $(*, 1)L(H) = L(H)$. The inclusion $L(H) \subseteq (*, 1)L(H)$ is immediate. On the other hand, if $E \subseteq L(H)$ and $X \subseteq R$ of cardinality one, then $E_{recomb}(X) \subseteq L(H)$. This is a consequence of the fact that every $(*, 1)$-splicing can be simulated by a sequence of $(1, 1)$-splicing. Therefore, one can prove by induction on i that $\phi^i(A, *, 1) \subseteq L(H)$. Consequently, $(*, 1)L(H) \subseteq L(H)$ and we are done. □ □

Corollary 5.4.2. *$\mathcal{L}((*, *)EH)$ as well as $\mathcal{L}((*, m)EH)$ for any $m \geq 1$ is equal to the class of regular languages.*

A rather interesting phenomenon we can observe in $(*, *)$-splicing is that in a single step we can form strings of arbitrary length. The way in which this can happen is reminiscent of the biochemical process of self-assembly. Suppose a string has two sticky ends, then it can be extended at either end. If the attached parts in turn have other sticky ends, this process can potentially continue indefinitely. This suggests another way of defining multiple splicing, namely as generating a language in a single step, rather than by iterating. We will call this non-iterated multiple splicing. Formally, given an extended splicing system $H = (V, T, A, R)$ the non-iterated multiple splicing defined by H is $A_{recomb} \cap T^*$.

Theorem 5.4.3. *Extended non-iterated multiple splicing systems generate exactly all regular languages.*

Proof. The construction from the proof of Theorem 5.3.4 can be easily adapted to show that extended finite non-iterated multiple splicing systems can generate all regular languages. We leave this simple task to the reader.

For the reverse direction, given a system $\Gamma = (V, A, R)$ with the rules of R labelled by $r_1, r_2, \ldots, r_n$, for some $n \geq 1$, consider all strings $w \in A$. The following procedure produces a finite automaton with one final state q_f only recognizing the $(*, *)$-splicing language generated by Γ:

Procedure *Construct_finite_automaton*
begin
for every string $w \in A$ do
  for every string $x\blacktriangleleft_i^p \in w_{cut}$, $1 \leq i \leq n$, $p \in \{l, r\}$ do
    construct a path labelled x from the initial state to the state (p, i);
  endfor;
  for every string ${}_i^p\blacktriangleright x\blacktriangleleft_j^{p'} \in w_{cut}$, $1 \leq i, j \leq n$, $p, p' \in \{l, r\}$ do
    construct a path labelled x from the state (p, i) to the state (p', j);
  endfor;
  for every string ${}_i^p\blacktriangleright x \in w_{cut}$, $1 \leq i \leq n$, $p \in \{l, r\}$ do
    construct a path labelled x from the state (p, i) to the final state q_f;
  endfor;
endfor;

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

for every pair of states $((s,i),(d,i))$, $1 \leq i \leq n$, add a λ-transition from (s,i) to (d,i)
and from (d,i) to (s,i).
endfor;
end.

It should be clear from the construction that this automaton accepts exactly those strings which are created by non-iterated multiple splicing. To obtain the language generated by the system, we take the union of these strings with the finite initial language. The result will also be regular. □ □

Finally, we consider that other variants of multiple splicing like $(\leq k, p)$-, $(\geq k, p)$-, (max, p)-splicing, $k, p \geq 1$, where every string is cut in at most k, at least k and a maximal number of segments, respectively, deserve to be investigated.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

6 Time Complexity for Splicing Systems

6.1 Introduction

This chapter proposes a notion of time complexity in splicing systems. The time complexity of a splicing system at length n is defined to be the smallest integer t such that all the words of the system having length n are produced within t rounds. For a function t from the set of natural numbers to itself, the class of languages with splicing system time complexity $t(n)$ is denoted by SplTime[$f(n)$]. We present fundamental properties of SplTime and explores its relation to classes based on standard computational models, both in terms of upper bounds and in terms of lower bounds. As to upper bounds, it is shown that for any function $t(n)$ SplTime[$t(n)$] is included in 1-NSPACE[$t(n)$]; i.e., the class of languages accepted by a $t(n)$-space-bounded nondeterministic Turing machine with one-way input head. Expanding on this result, it is shown that 1-NSPACE[$t(n)$] is characterized in terms of splicing systems: it is the class of languages accepted by a $t(n)$-space uniform family of extended splicing systems having production time $O(t(n))$ with the additional property that each finite automaton appearing in the family of splicing systems has at most a constant number of states.

As to lower bounds, it is shown that for all functions $t(n) \geq \log n$, all languages accepted by a pushdown automaton with maximal stack height $t(|x|)$ for a word x are in SplTime[$t(n)$]. From this result, it follows that the regular languages are in SplTime[$O(\log n)$] and that the context-free languages are in SplTime[$O(n)$]. It is also shown that all languages accepted by $t(n)$ space-bounded nondeterministic Turing machines are in SplTime[$O(t(n)^2)$]. Combined with the 1-NSPACE upper bound, this shows that the class of languages generated by polynomially time bounded extended regular splicing systems is exactly PSPACE.

The universality of extended H systems with a regular set of rules states that the model is equivalent to other standard abstract computation models, such as Turing machines and random access machines. Since these standard models are used to define computational complexity by introducing the concept of resources, one may wonder whether there exists a natural concept of computational resources in the extended splicing system and what complexity classes are defined in terms of the resource concept.

Surprisingly, although these questions sound quite natural, and indeed for other models of DNA computing similar questions have been addressed before [53, 54, 71],

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

this is the first time that the questions are addressed with respect to splicing systems. We introduce here a notion of computational complexity in the splicing model.

We can naturally view that a splicing system produces its language in rounds of rule applications, and a formal definition of a splicing system uses this view. Our proposal is to consider the minimum number of rounds that it takes for the system to produce the word as the complexity of the word with respect to the system. The complexity of the language produced by the system at length n is then defined to be the maximum of the time complexity of the word with respect to the system for all members of the language having length n. This time complexity concept is reminiscent of the derivational complexity of grammars [8, 18], where the complexity of a word with respect to a grammar is defined to be the smallest number of derivational steps for producing the word with respect to the grammar. Although the derivational complexity uses the number of operational steps as a measure, it is fundamentally different from our notion of time complexity because splicing is applied to two words and the two input words for splicing can be produced asynchronously in preceding steps.

Formally speaking, the time complexity of a splicing system is given as follows. Let Γ be a splicing system over an alphabet Σ such that regular languages define its pattern quadruples for splicing rules (we say that Γ has a *regular set of rules*). For a word $w \in \Sigma^*$, let $\mathrm{SplTime}_\Gamma(w)$ denote the smallest number of rounds in which Γ produces w. For w not produced by Γ, let this quantity be 0. For a function t from the set of natural numbers to itself, we say that Γ has time complexity $t(n)$ if for all $w \in \Sigma^*$, $\mathrm{SplTime}_\Gamma(w) \leq t(|w|)$. We define $\mathrm{SplTime}[t(n)]$ to be the set of all languages produced by some extended splicing system with a finite initial language and with a regular set of rules with time complexity $t(n)$, and then, for a class of functions $\mathcal{F}$, we define $\mathrm{SplTime}[\mathcal{F}] = \cup_{t\in\mathcal{F}}\mathrm{SplTime}[t(n)]$.

In this chapter we explore properties of the proposed notion of time complexity. Because the first universality result was obtained for extended splicing systems with a finite initial language and a regular set of rules, they can be considered to be the "standard" universal splicing system. Indeed, many well-studied variants having universal power, like those mentioned in Section 2.3.3 can be straightforwardly simulated with these systems, with at most a constant slowdown[1]. We will thus take the extended splicing systems with a finite initial language and a regular set of rules as the reference model for universal splicing systems, and define our complexity notions in terms of these systems.

Before introducing our complexity definitions, we state a few conventions. We consider the set R of splicing rules is regular. The definition of such a regular set R can be given as follows.

Definition 6.1.1. *A set of rules R is regular if there exist some $m \geq 1$ and m quadruples of regular languages (A_i, B_i, C_i, D_i), $1 \leq i \leq m$, such that $R = \cup_{1\leq i\leq m}\{a_i\#b_i\$c_i\#d_i \mid a_i \in A_i, b_i \in B_i, c_i \in C_i, d_i \in D_i\}$.*

[1] Such straightforward constant-slow-down simulation results do not appear to hold for the alternative systems introduced in Chapter 3.

Since each regular language appearing in R can be represented by a finite automaton, R can be actually viewed as a finite collection of finite automaton quadruples. In our proofs, we will take this view. Also, any such quadruple $r = (A, B, C, D)$ appearing in R can be extended as follows. From A we construct a finite automaton A' accepting $\Sigma^* L(A)$. We perform the same modification to C to obtain C'. For B and D, we modify them so that they accept $L(B)\Sigma^*$ and $L(D)\Sigma^*$, respectively, obtaining B' and D'. These changes simplify the action of splicing rules by allowing to divide an input word to a splicing operation into two parts, not four parts. Specifically, we can assume that the modified quadruple (A', B', C', D') operates on two words $u = u_1u_2$ and $v = v_1v_2$ such that $u_1 \in L(A)$, $u_2 \in L(B)$, $u_3 \in L(C)$, and $u_4 \in L(D)$ and produces u_1v_2 and v_1u_2. We assume that each quadruple in R has undergone such changes. Clearly, the language produced with the modified rules is identical to L.

Note that if (A, B, C, D) is a quadruple in R, then adding (C, D, A, B) to R does not change the language L, because the new quadruple simply swaps the order of two words appearing on each side in the specification of a splicing operation. We thus assume that the rule set R has the property that if (A, B, C, D) appears in R then so does (C, D, A, B). Then, given a quadruple $r = (A, B, C, D)$ in R and given an application of r, $(u, v) \vdash_r (x, y)$, we designate x as the "primary result" and y as the "secondary result" of r, and write

$$r(u, v) \to x$$

to mean that x is the primary result of an legitimate application of r to pair (u, v).

6.2 Time Complexity for Splicing Systems

Let $\Gamma = (V, \Sigma, I, R)$ be an extended splicing system. For each $w \in V^*$, define

$$\mathrm{SplTime}_\Gamma(w) = \begin{cases} \min\{i \mid w \in \sigma_R^i(I)\} & \text{if } w \in \sigma_R^*(I) \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathbb{N}$ denote the set of all natural numbers.

Definition 6.2.1. *Let $T(n)$ be a monotonically nondecreasing function from $\mathbb{N}$ to itself. Then we define* $\mathrm{SplTime}[T(n)]$ *to be the set of all languages L for which there exists an extended splicing system with regular rules $\Gamma = (V, \Sigma, I, R)$ such that for all $w \in L$, it holds that* $\mathrm{SplTime}_\Gamma(w) \leq T(|w|)$.

Definition 6.2.2. *For a class C of functions from $\mathbb{N}$ to itself, define*

$$\mathrm{SplTime}[C] = \cup_{T(n)\in C}\mathrm{SplTime}[T(n)].$$

A simple observation here is that for any extended splicing system $\Gamma = (V, \Sigma, I, R)$, at any step i, the length of the longest word in $\sigma_\Gamma^i(I)$ is at most twice that in $\sigma_\Gamma^{i-1}(I)$. This implies that the length is at most 2^i times the longest word in I. Thus, we have:

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Proposition 6.2.3. SplTime[$o(\log n)$] *contains no infinite languages.*

The following proposition is trivial.

Proposition 6.2.4. *For all monotonically nondecreasing functions $T_1(n)$ and $T_2(n)$ such that for all $n \geq 0$, $T_1(n) \leq T_2(n)$, we have*

$$\text{SplTime}[T_1(n)] \subseteq \text{SplTime}[T_2(n)].$$

Due to Proposition 6.2.3 a time complexity function $T(n)$ is meaningful for extended splicing systems if $T(n) \in \Omega(\log n)$. Thus, the smallest splicing time complexity class is SplTime[$O(\log n)$]. Here we show some fundamental results about this class.

First, it is not hard to see that the regular language a^+ belongs to SplTime[$O(\log n)$] via the following unextended system $\Gamma = (V, I, R)$:

$$V = \{a\},\ I = \{a\},\ \text{and}\ R = \{\ a\#\lambda\$\lambda\#a\ \}.$$

This splicing system generates $\{\lambda, a, aa\}$ in the first step, $\{\lambda, a, aa, a^3, a^4\}$ in the second and in general $\sigma^i_\Gamma(I) = \{a^x \mid 0 \leq x \leq 2^i\}$.

Actually, it is not very difficult to show that every regular language belongs to this class.

Theorem 6.2.5. REG $\subseteq$ SplTime[$O(\log n)$].

Proof. Let L be an arbitrary regular language. If L is finite, an unextended system whose initial language is L and whose rule set is empty produces L in no rounds, and so $L \in$ SplTime[$O(\log n)$].

Suppose that L is an infinite regular language. We will construct an extended finite splicing system $\Gamma = (V, \Sigma, I, R)$ witnessing that $L \in$ SplTime[$O(\log n)$]. Let $M = (Q, \Sigma, \delta, q_0, F)$ be a non-deterministic finite automaton accepting L, where Q is the set of states, Σ the input alphabet, q_0 the initial state, F the set of final states, and δ the transition function. We assume without loss of generality that M has no λ-transitions. We construct Γ as follows:

- $V = \Sigma \cup Q \cup \{Z\}$, where Z is a new symbol not in $\Sigma \cup Q$.
- $I = \{Z\} \cup \{q_i a q_j \mid q_i, q_j \in Q, a \in \Sigma, \delta(q_i, a) = q_j\}$.
- R consists of the following rules:
  - $a\#q\$q\#b$ for all $q \in Q, a, b \in \Sigma$,
  - $q_0\#\lambda\$\lambda\#Z$,
  - $\lambda\#q_f\$Z\#\lambda$ for all $q_f \in F$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

The initial language I contains all the words of the form $q_i a q_j$ such that M makes a transition from q_i to q_j on a. Thus, I is the set of all valid paths of length 1. The rules of the form $a\#q\$q\#b$ connect two paths sharing the same state in the middle. The last two rules eliminate the initial state appearing at the beginning and the final state appearing at the end. The strings of the form qq cannot give any string over Σ, except λ when $q_0 = q_f$ for some $q_f \in F$, in which case we want to have λ in our language.

Production of a word w in L can be in a divide-and-conquer fashion: split w into halves, separately produce them with the corresponding states at each end, and connect them. Thus, the time that it takes to produce a word having length n is $\lceil \log(n+1) \rceil + 2$ (the additive term of 2 is for eliminating the initial and accept states after producing a word of the form $q_0 w q_f$ such that $q_f \in F$). Thus, $L \in \text{SplTime}[O(\log n)]$. This proves the theorem. □

Note that the set of rules in the above construction is finite. The class of languages generated by extended splicing systems with a finite set of rules is known to be equal to the set of all regular languages ([64], see also [65]).

Corollary 6.2.6. *Let $\mathcal{F}$ be an arbitrary class of monotonically nondecreasing functions from $\mathbb{N}$ to itself such that $\mathcal{F} \supseteq O(\log n)$. If the set of rules is restricted to be finite, then* $\text{REG} = \text{SplTime}[O(\log n)] = \text{SplTime}[\mathcal{F}]$.

Theorem 6.2.7. *Let $T(n)$ be an arbitrary monotonically nondecreasing function such that $T(n) \in \Omega(\log n)$. Then the class* $\text{SplTime}[O(T(n))]$ *is closed under concatenation, star-operation, and union.*

Proof. Let L_1 and L_2 be languages in $\text{SplTime}[O(T(n))]$. For each $i \in \{1, 2\}$, suppose that $L_i \in \text{SplTime}[T(n)]$ is witnessed by a regular extended splicing system $H_i = (V_i, \Sigma_i, I_i, R_i)$; i.e., there is a constant c_i such that for all $w \in L_i$ it holds that $\text{SplTime}_{H_i}(w) \leq c_i T(|w|)$. Without loss of generality, assume that there is no common nonterminal in H_1 and H_2, that is, $(V_1 - \Sigma_1) \cap (V_2 - \Sigma_2) = \emptyset$. To prove the theorem it suffices to show that $L_1 \cup L_2$, $L_1 L_2$, and $(L_1)^*$ each belong to $\text{SplTime}[O(T(n))]$.

To show that $L_1 \cup L_2 \in \text{SplTime}[O(T(n))]$, for each $i \in \{1, 2\}$, do the following:

- Construct V_i' by adding to V_i three new nonterminals α_i, β_i, and γ_i. Here the new nonterminals for H_1 are different from those for H_2.
- For each finite automaton quadruple $Q = (\phi_1, \phi_2, \phi_3, \phi_4)$ in R_i, create a new quadruple $(\phi_1', \phi_2', \phi_3', \phi_4')$ such that
  - ϕ_1' is a finite automaton accepting $\{\alpha_i u w \mid u \in V_i^* \text{ and } w \in L(\phi_1)\}$,
  - ϕ_2' is a finite automaton accepting $\{w u \beta_i \mid u \in V_i^* \text{ and } w \in L(\phi_2)\}$,
  - ϕ_3' is a finite automaton accepting $\{\alpha_i u w \mid u \in V_i^* \text{ and } w \in L(\phi_3)\}$, and
  - ϕ_4' is a finite automaton accepting $\{w u \beta_i \mid u \in V_i^* \text{ and } w \in L(\phi_4)\}$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Let R_i' be the set of rules obtained as the collection of all the quadruples thus produced.

- Set I_i' to $\{\gamma_i\} \cup \{\alpha_i w \beta_i \mid w \in I_i\}$.

For each $i \in \{1,2\}$, the rules R_i' act as R_i, except that every word produced by the rules is the one produced by H_i with the corresponding sequence of splicing operations with an α_i at the beginning and a β_i at the end. Thus, for each $i \in \{1,2\}$, for each $w \in V_i^*$, and for each integer r, H_i produces w at round r if and only if $(V_i', \Sigma_i, I_i', R_i')$ produces $\alpha_i w \beta_i$ at round r. For each $i \in \{1,2\}$, construct R_i'' by adding two rules to R_i':

$$\alpha_i \Sigma^* \# \beta_i \$ \gamma_i \# \lambda \text{ and } \alpha_i \# \lambda \$ \lambda \# \gamma_i.$$

The former has the role of eliminating β_i from a word of the form $\alpha_i u \beta_i$ such that $u \in \Sigma^*$, while the latter has the role of dropping from any word beginning with an α_i the first symbol α_i. Then, for each $i \in \{1,2\}$, $(V_i', \Sigma, I_i', R_i'')$ produces L_i with the property that every word produced by H_i is produced by the new system with exactly two additional rounds. Now define

$$\Gamma = (V_1' \cup V_2', \Sigma, I_1' \cup I_2', R_1'' \cup R_2'').$$

Then $L(\Gamma) = L_1 \cup L_2$, and for all $w \in L_1 \cup L_2$,

$$\begin{aligned} \text{SplTime}_\Gamma(w) &\leq \min\{\text{SplTime}_{H_1}(w), \text{SplTime}_{H_2}(w)\} + 2 \\ &\leq \min\{c_1, c_2\} T(|w|) + 2. \end{aligned}$$

Thus, $L_1 \cup L_2 \in \text{SplTime}[O(T(n))]$.

To show that $L_1 L_2 \in \text{SplTime}[O(T(n))]$, construct Γ' from Γ in the above by replacing the rules $\alpha_i \Sigma^* \# \beta_i \$ \gamma_i \# \lambda$ and $\alpha_i \# \lambda \$ \lambda \# \gamma_i$ by three rules:

$$\alpha_1 \Sigma^* \# \beta_1 \$ \alpha_2 \# (V_2)^* \beta_2,\ \alpha_1 \Sigma^* \# \beta_2 \$ \gamma_1 \# \lambda, \text{ and } \alpha_1 \# \lambda \$ \lambda \# \gamma_1.$$

The first has the role of splicing any two words of the form $\alpha_1 u \beta_1$ and $\alpha_2 v \beta_2$ such that $u, v \in \Sigma^*$ to produce $\alpha_1 u v \beta_2$, the second has the role of eliminating β_2 from a word of the form $\alpha_1 u \beta_2$ such that $u \in \Sigma^*$, and the last has the role of dropping α_1 at the beginning of any word. (N.B. The nonterminal γ_2 will never be used.) Then, $L(\Gamma') = L_1 L_2$. For each word w of the form $w_1 w_2$ such that $w_1 \in L_1$ and $w_2 \in L_2$,

$$\begin{aligned} &\text{SplTime}_{\Gamma'}(w) \\ &\quad = \max\{\text{SplTime}_{H_1'}(w_1), \text{SplTime}_{H_2'}(w_2))\} + 3 \\ &\quad \leq \max\{c_1 T(|w_1|), c_2 T(|w_2|)\} + 3. \end{aligned}$$

Since $T(n)$ is monotonically nondecreasing, the last term is at most

$$\max\{c_1, c_2\} T(|w|) + 3.$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Thus, $L_1 L_2 \in \text{SplTime}[O(T(n))]$.

To show that $(L_1)^* \in \text{SplTime}[O(T(n))]$, construct from $(V_1', \Sigma_1, I_1', R_1'')$ a new extended splicing system Γ'' by adding one rule:

$$\alpha_1 V^* \# \beta_1 \$ \alpha_1 \# V^* \beta_1.$$

This has the effect of splicing any two words of the form $\alpha_1 u \beta_1$ and $\alpha_1 v \beta_1$ such that $u, v \in \Sigma^*$ to produce $\alpha_1 uv\beta_1$. It is not hard to see that $L(\Gamma'') = (L_1)^*$. Let w be an arbitrary word in $(L_1)^*$ that can be decomposed into $w_1 \cdots w_k$ such that $w_1, \ldots, w_k$ are in L_1. Then,

$$\text{SplTime}_{\Gamma''}(w) \leq \max\{\text{SplTime}_{H_1}(w_i) \mid 1 \leq i \leq k\} + \lceil \log k \rceil + 2.$$

For each i, $1 \leq i \leq k$, $\text{SplTime}_{H_1}(w_i) \leq c_1 T(|w_i|)$ and $|w_i| \leq |w|$. Also, we have $k \leq |w|$. Thus,

$$\text{SplTime}_{\Gamma''}(w) \leq c_1 T(|w|) + \lceil \log n \rceil + 2.$$

Since $T(n) \in \Omega(\log n)$, the right-hand side of the inequality is $O(T(n))$. Thus, we have $(L_1)^* \in \text{SplTime}[O(T(n))]$. □

We note here that it is unknown whether $\text{SplTime}[O(T(n))]$ is closed under intersection or under complementation.

6.3 Splicing Systems versus One-way Nondeterministic Space

In this section we consider an upper bound of splicing time complexity classes. The difficulty here is that, although the extended splicing system is universal, there does not appear to exist any immediate connection between the running time of a Turing machine and the number of production rounds required by the splicing system that produces the language recognized by the Turing machine.

6.3.1 Straightforward Upper Bounds

A straightforward method for checking the membership of a word w in a language L in $\text{SplTime}[T(n)]$ would be to simulate the splicing system for at most $T(|w|)$ rounds while keeping the collection of the words that have been produced and then check whether w appears in the final collection. Though correct, the space needed for this algorithm can increase rapidly. Let H be an extended splicing system. Let a be the number of finite automaton quadruples of H, let k be the length of the longest word in the initial word set, and let d be the cardinality of the initial word set. Suppose that this straightforward algorithm is applied to H. For each $i \geq 0$, let s_i be the cardinality of the collection immediately after the ith round and let ℓ_i be the length of the longest

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

word in the collection s_i. We have $s_0 = d$ and $\ell_i = k$. Since any pair of words can be spliced with respect to any one of the automaton quadruples at any position on either component of the pair, we have for all $i \geq 1$,

$$s_i \leq s_{i-1} + 2(s_{i-1})^2 a(\ell_{i-1} + 1)^2$$

and $\ell_i \leq 2\ell_{i-1}$. Thus, we have $\ell_i \leq k2^i$ and $s_i \leq ab^{2^i}$ for some b. This gives rise to a doubly-exponential (!) space upper bound:

Proposition 6.3.1. *For all monotonically nondecreasing functions $T(n)$,*

$$\text{SplTime}[T(n)] \subseteq \cup_{c>0}\text{DSPACE}[c^{2^{T(n)}}].$$

This upper bound is, not surprisingly, very naive. By guessing the "components" of the splicing operations that are conducted to produce a word w, we can reduce the upper bound to a nondeterministic exponential time.

Theorem 6.3.2. *For all monotonically nondecreasing functions $T(n)$,*

$$\text{SplTime}[T(n)] \subseteq \cup_{c>0}\text{NTIME}[c^{T(n)}].$$

Proof. Let $L \in \text{SplTime}[T(n)]$ be witnessed by an extended splicing system $\Gamma = (V, \Sigma, I, R)$. Let d be the length of the longest word in I. For all natural numbers $i \geq 0$, and for all $w \in V^*$ such that $\text{SplTime}_\Gamma(w) \leq i$, $|w| \leq d2^i$. Also, for all positive integers i and for all $w \in V^*$, $\text{SplTime}_\Gamma(w) \leq i$ if and only if either

- $w \in \text{SplTime}_\Gamma(w) \leq i - 1$ or
- there exist $x, y, z \in V^*$ and a rule $r \in R$ such that $\text{SplTime}_\Gamma(u) \leq i-1$, $\text{SplTime}_\Gamma(v) \leq i - 1$, and $(u, v) \vdash_r (z, w)$ (recall that we use $(u, v) \vdash_r (z, w)$ and $(u, v) \vdash_r (w, z)$ interchangeably).

Consider the following nondeterministic algorithm Q that takes as input an integer $i \geq 0$ and a word $w \in V^*$ and tests whether w is produced by Γ within i rounds.

Step 1 If $i = 0$, return 1 if $w \in I$ and 0 otherwise.

Step 2 Nondeterministically select $u, v \in V^*$ having length at most $d2^{i-1}$, $z \in V^*$ having length at most $d2^i$, and a finite automaton quadruple $r \in R$.

Step 3 Test whether $(u, v) \vdash_r (z, w)$ by exhaustively examining all possible positions for aligning the finite automata on u and v. If $(u, v) \vdash_r (z, w)$ doesn't hold, return 0.

Step 4 Make two recursive calls, $Q(i - 1, u)$ and $Q(i - 1, v)$. Both return with 1 as the value, return 1; otherwise, return 0.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

It is not difficult to see that this nondeterministic algorithm works correctly. The total number of recursive calls to Q on input (i, w) is at most $2 + 2^2 + \cdots + 2^i < 2^{i+1}$; the running time for the algorithm excluding the time spent on recursive calls is bounded by polynomial in $d2^i$ on input (i, w). Thus, the total running time is $O(c^i)$ for some constant $c > 0$. Now, to test whether $w \in L$, we have only to execute $Q(T(|w|), w)$. This implies, that $L \in \text{NTIME}[c^{T(n)}]$. This proves the theorem. □

From the above theorem, we immediately have the following corollary.

Corollary 6.3.3. $\text{SplTime}[O(\log n)] \subseteq \text{NP}$.

6.3.2 Bounding the Complexity of Splicing Systems in terms of One-way Nondeterministic Space

The idea of nondeterministic verification shown in the above can be further explored to tighten the upper bound. For a function $T(n)$ from $\mathbb{N}$ to $\mathbb{N}$, 1-NSPACE$[T(n)]$ is the set of all languages accepted by a $T(n)$ space-bounded nondeterministic Turing machine with one-way input tape [30]. We stipulate that, in the one-way nondeterministic space-bounded Turing machine model, since the input head moves from left to right only, the usable amount of space must be communicated to the machine prior to computation. For a $T(n)$ space-bounded machine in this model, this communication is accomplished by assuming that on input of length n on each work tape a blank word of length $T(n)$ is written flanked by end markers with the initial position of the head being at the symbol immediately to the right of the left end marker and that the head never goes beyond the end markers.

In addition, as is standard in the space-bounded computation under the Turing model, we assume that each space-bound $T(n)$ is fully-space constructible in the following sense: There exists a deterministic Turing machine M such that on each input x, M on x uses exactly the first $T(|x|)$ of the work-tape cells. Note that so long as $T(n)$ is monotonically nondecreasing and in $\Omega(\log n)$, the machine M can be modified so that its input head is one-way does not move backwards: Maintain in binary the number m of input symbols that have been scanned so far; while the end of input is yet to be reached, move the input head by one cell, increment the counter, and then simulate the action of M to scan exactly $T(m)$ cells. Unfortunately, when $T(n)$ is in $o(n)$, the fully space-constructibility with one-way input head does not seem strong enough to eliminate the requirement that the allowable space of size $T(n)$ be marked at the beginning of computation in the one-way space-bounded computation for the following reason: Suppose that the programs of a one-way nondeterministic $T(n)$-space-bounded machine R and of a one-way deterministic machine M that fully space-constructs $T(n)$ are concurrently executed on an input of length n. For each m, $0 \leq m \leq n$, let $s_R(m)$ be the number of work-tape cells that have been scanned by R when the first m symbols of the input have been scanned, and let $s_M(m)$ be the number for M. Since the space of size $T(n)$ is being constructed, a natural method for ensuring that R will use no more

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

than $T(n)$ space would be to require that $s_R(m) \leq s_M(m)$ for all m, but this seems a too stringent assumption to make for functions $T(n)$ in $o(n)$. When $T(n)$ is in $\Omega(n)$, the one-way input-head restriction is superfluous because there is enough space in the work-tape to hold a full copy of the input.

Among the many 1-NSPACE classes of particular interest to us is 1-NL, which is $\cup_{c>0}$1-NSPACE$[c(\log n)]$. Hartmanis and Mahaney [30] show that the reachability problem of topologically sorted directed graph is complete for 1-NL under the logarithmic space-bounded many-one reductions.

We show an improved upper-bound of 1-NL for SplTime.

Theorem 6.3.4. SplTime$[O(\log n)] \subseteq$ 1-NL.

This theorem is straightforwardly derived from the following more general statement.

Theorem 6.3.5. *For all monotonically increasing and fully space-constructible functions $f(n) \geq \log n$, it holds that* SplTime$[f(n)] \subseteq$ 1-NSPACE$[f(n)]$.

Proof. Let $f(n) \geq \log n$. Let L be a language in SplTime$[f(n)]$. Let $\Gamma = (V, \Sigma, I, R)$ be an extended splicing system that witnesses $L \in$ SplTime$[f(n)]$.

As before, we regard the set R as a finite collection of finite automaton quadruples, and we assume that all automata appearing in the rules are extended such that they cover the entire words, as discussed in Section 6.1.

Let n be an arbitrary natural number. The process in which Γ produces a word in at most $f(n)$ rounds can be described as a node-labeled, full binary (each non-leaf having two children) tree of height at most $f(n)$ with the following labeling rules:

- Each leaf is labeled with a word in I.
- Each non-leaf is labeled with a word $w \in V^*$, a rule r, and two natural numbers i and j.

Each non-leaf represents a splicing operation as follows: Let g be a non-leaf with labels w, r, i, j. Let $r = (A, B, C, D)$. Let u be the word label of the left child and let v be the word label of the right child. Then u is of the form u_1u_2 and v is of the form v_1v_2 such that

- $w = u_1v_2$, $|u_1| = i$, $|v_1| = j$,
- $u_1 \in L(A)$, $u_2 \in L(B)$, $v_1 \in L(C)$, and $v_2 \in L(D)$.

Note that given a valid production tree the word label of each non-leaf can be computed from the labels of its proper descendants. The output of the production tree is the word label of the root.

Using the above notion of production trees, the membership test of any word $x \in \Sigma^n$ in L can be done by testing whether there is a production tree of height at most $f(n)$ whose output is x. Our goal is to design an $O(f(n))$ space-bounded one-way nondeterministic algorithm for this task. This is achieved as follows:

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- We will design a (fixed) scheme for encoding, as a word over a finite alphabet Δ, the tree structure, the leaf labels (not necessarily members of I), and the rule and the splicing positions at each non-leaf.

- We will design an $f(n)$ space-bounded one-way deterministic algorithm for testing, given a word $y \in \Delta^*$, whether y is in a valid format and if so, in the purported production tree specified by y, whether the word assigned to each leaf is in the initial language I, and whether the splicing rule specified at each non-leaf can be successfully applied to the word labels of the children.

- We will design an $f(n)$ space-bounded one-way deterministic algorithm for testing, given an input x and a word $y \in \Delta^*$ that passes the aforementioned test, whether x is the output of the tree encoded by y.

By combining these three, we will construct an $f(n)$ space-bounded one-way nondeterministic algorithm for L.

We need some notation. For a node u in a production tree, $E(u)$ denotes the encoding of the subtree rooted at u and $W(u)$ denotes the word produced at u. The attributes $E(u)$ and $W(u)$ are given so that the letters of $W(u)$ appear in $E(u)$ in order; that is,

(*) if $W(u) = a_1 \cdots a_m$ for some $a_1, \ldots, a_m \in V$, then there exist positions $k_1, \ldots, k_m$, $1 \leq k_1 < \cdots < k_m \leq |E(u)|$, such that for all h, $1 \leq h \leq m$, a_h is the symbol of $E(u)$ at position k_m.

Also, for a non-root u, $W_L(u)$ and $W_R(u)$ respectively denote the first and the second segments of $W(u)$ when $W(u)$ is spliced at its parent node.

We introduce a number of new nonterminals. First, we treat each $r \in R$ as a nonterminal, introduce a special nonterminal r_0, and then set $R_0 = R \cup \{r_0\}$. Next, we introduce five nonterminals $|^{(0)}$, $|^{(1)}$, $|^{(2)}$, $|^{(3)}$, and $|^{(4)}$, that act as delimiters. Finally, we introduce three symbols L, R, and $\perp$ for encoding each node by the downward path from the root to the node. Here L and R respectively represent the left branch and the right branch, and $\perp$ is a delimiter attached at the very end to indicate the termination of the path. For example, the grandchild of the root that is the left child of the right child of the root is represented by the path $RL\perp$. For a node u, $\pi(u)$ denote the downward path from the root to u encoded this way. The alphabet Δ is now defined to be:

$$V \cup R_0 \cup \{|^{(0)}, \ldots, |^{(4)}\} \cup \{L, R, \perp\}.$$

The encoding of a production tree over Δ is constructed recursively as follows.

- Let u be a leaf with w as the word label. Then

$$E(u) = |^{(0)}\pi(u)r_0w|^{(1)}\pi(u)|^{(2)}\pi(u)|^{(3)}\pi(u)|^{(4)}\pi(u)$$

  and $W(u) = w$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- Let u be a non-leaf with r as the rule label. Let y and z be respectively its left and right children. Then

$$E(u) = |^{(0)}\pi(u)re'_1|^{(2)}\pi(u)e'_2|^{(4)}\pi(u),$$

where e'_1 is constructed from $E(y)$ by inserting $|^{(1)}\pi(u)$ and e'_2 is constructed from $E(z)$ by inserting $|^{(3)}\pi(u)$. These insertions are subject to the following rules:

  - For every descendant v of y, $|^{(1)}\pi(u)$ appears either (after $|^{(0)}\pi(v)$ and before $|^{(1)}\pi(v)$) or (after $|^{(2)}\pi(v)$ and before $|^{(3)}\pi(v)$).
  - Similarly, for each descendant v of z, $|^{(2)}\pi(u)$ appears either (after $|^{(0)}\pi(v)$ and before $|^{(1)}\pi(v)$) or (after $|^{(2)}\pi(v)$ and before $|^{(3)}\pi(v)$).

The words $W(u)$, $W_L(y)$, $W_R(y)$, $W_L(z)$, and $W_R(z)$ are determined as follows:

  - Let $W(y)$ be of the form $\xi|^{(1)}\pi(u)\theta$. Then $W_L(y) = \xi$ and $W_R(y) = |^{(1)}\pi(u)\theta$.
  - Let $W(z)$ be of the form $\alpha|^{(2)}\pi(u)\beta$. Then $W_R(z) = \alpha$ and $W_R(z) = |^{(2)}\pi(u)\beta$.
  - The position labels i and j of u are respectively $|W_L(y)|$ and $|W_L(z)|$.
  - $W(u) = W_L(y)W_R(z)$.

Note that the property (*) in the above holds for u.

We identify a set of syntactic properties that the encoding of a production tree must satisfy. A production tree is a full binary tree, and so, a part of an infinite full binary tree. We will thus speak of a generic node u below.

1. Each occurrence of $|^{(0)}, \ldots, |^{(4)}$ must be followed by a path encoding; i.e., a word in $\{L, R\}^*\perp$.
2. Each occurrence of $|^{(0)}\pi(u)$ for any node u must be followed by an occurrence of a member of R_0.
3. For any node u, the occurrence of $\pi(u)$ must be preceded by one of $|^{(0)}, \ldots, |^{(4)}$.
4. For any node u, if $\pi(u)$ appears at all, then it has to appear exactly five times, as $|^{(0)}\pi(u), \ldots, |^{(4)}\pi(u)$ in this order.
5. For any non-root u, if $\pi(u)$ appears at all, then for each ancestor v of u, $\pi(v)$ appears in the following way:
   - $|^{(0)}\pi(v)$ appears to the left of $|^{(0)}\pi(u)$.
   - $|^{(4)}\pi(v)$ appears to the right of $|^{(4)}\pi(u)$.
   - $|^{(1)}\pi(v)$ appears either to the left of $|^{(1)}\pi(u)$ or to the right of $|^{(3)}\pi(u)$.
   - $|^{(3)}\pi(v)$ appears either to the left of $|^{(1)}\pi(u)$ or to the right of $|^{(3)}\pi(u)$.
   - If u is in the left subtree of v, then $|^{(2)}\pi(v)$ appears to the right of $|^{(3)}\pi(u)$.
   - If u is in the right subtree of v, then $|^{(2)}\pi(v)$ appears to the left of $|^{(1)}\pi(u)$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

6. If $|^{(0)}\pi(u)$ is followed by an r_0, then for no descendant v of u, $|^{(0)}\pi(v), \ldots, |^{(4)}\pi(v)$ appear.

7. If $|^{(0)}\pi(u)$ is followed by an $r \in R$, then for each child v of u, $|^{(0)}\pi(v), \ldots, |^{(4)}\pi(v)$ appear.

Suppose that $w \in \Delta^*$ satisfies all of the above conditions. Suppose w is scanned from left to right. We think of encountering $|^{(0)}\pi(u)$ as entering the node u and encountering $|^{(4)}\pi(u)$ as exiting u. With this view, w is thought of as specifying the depth-first traversal of the tree represented by w. Also, if we assign to each node u appearing in w the interval $J(u)$ of integers ranging between the beginning position in w of $|^{(1)}\pi(u)$ and the ending position in w of $|^{(3)}\pi(u)$, then these intervals become a presentation of an interval tree, that is, given any two intervals $J(u)$ and $J(v)$, one of the following holds: $J(u) \subseteq J(v)$, $J(u) \supseteq J(v)$, and $J(u) \cap J(v) = \emptyset$.

It is not difficult to see that whether a word $w \in \Delta^*$ is written according to the syntactic rules in the above and in which the farthest node from the root has distance at most $f(n)$ can be tested deterministically in space $O(f(n))$ by scanning w from left to right, assuming that $f(n)$ is already given.

Let w be a word in Δ^* that satisfies all the syntactic conditions in the above. Satisfying those conditions does not necessarily guarantee that w is a valid encoding of a production tree, because the splicing rules specified at non-leaves may not be applicable and because the words specified at leaves may not be in the initial language I. For these reasons, we introduce the following semantic conditions that the valid encoding of a production tree must satisfy.

1. Suppose that $|^{(0)}\pi(u)r_0$ appears in w. If $|^{(i)}\pi(v)$ is removed from w for all i, $0 \leq i \leq 4$, and for each node $v \neq u$, then between $|^{(0)}\pi(u)$ and $|^{(4)}\pi(u)$ remains a word of the form

$$|^{(0)}\pi(u)r_0h|^{(1)}\pi(u)|^{(2)}\pi(u)|^{(3)}\pi(u)|^{(4)}\pi(u)$$

   such that $h \in I$. If this condition is satisfied for u, then $W(u)$, the word represented by u, is equal to h. If this is not satisfied for u, then w is not valid and thus $W(u)$ is undefined.

2. Suppose that $|^{(0)}\pi(u)r$ with $r \in R$ appears in w and for all descendants v of u, $W(v)$ is defined. Let $r = (A, B, C, D)$ such that A, B, C, and D are finite automata. Let y and z be the left child and the right child of u, respectively. Let $W_L(y)$ be the prefix of $W(y)$ that appears before $|^{(2)}\pi(u)$ and let $W_R(y)$ the remainder of $W(y)$. Let $W_L(z)$ be the prefix of $W(z)$ that appears before $|^{(2)}\pi(u)$ and let $W_R(z)$ the remainder of $W(z)$. Then, it must be the case that $W_L(y) \in L(A)$, $W_R(y) \in L(B)$, $W_L(z) \in L(C)$, and $W_R(z) \in L(D)$. If these membership conditions are met, then $W(u) = W_L(y)W_R(z)$. Otherwise, w is semantically incorrect and thus $W(u)$ is undefined.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

It is not hard to see that if the above semantic conditions are met for all nodes u such that $\pi(u)$ appears in w, then w encodes a production tree.

Note that for any node u such that $\pi(u)$ appears in w and if $W(u)$ is defined with respect to w, then $W(u)$ is equal to the word constructed from w by eliminating from w:

(i) every symbol either before $|^{(0)}\pi(u)$ or after $|^{(4)}\pi(u)$,

(ii) every symbol appearing between $|^{(1)}\pi(v)$ and $|^{(3)}\pi(v)$ for each node v, and then

(iii) every symbol not belonging to V.

This means that, for each node u such that $W(u)$ is defined with respect to w, the letters of $W(u)$ can be computed from left to right while scanning w from left to right, in the following manner:

- Output only symbols in V.
- If for some node u, $|^{(1)}\pi(v)$ has been encountered, then suspend the output process until $|^{(3)}\pi(v)$ has been encountered.

Since membership in a regular language can be tested by simply scanning the input from left to right and only nodes appearing in a downward path are considered simultaneously, the semantic test in the above can be done deterministically in space $O(f(n))$ by scanning w from left to right. Finally, the W-value of the root of the tree encoded by w can be computed scanning w from left to right. To check whether the production tree produces x can be tested by comparing the letters of W of x letter by letter.

Thus, by concurrently running the three tests while nondeterministically producing a word over Δ, the membership of x in L can be tested in space $O(f(n))$. Note that the length of w can be bounded by (the maximum number of leaves) $\times$ (the length of the longest word in I) + (the maximum number of nodes) $\times(5(2 + f(n)) + 1)$. This quantity is bounded by $c^{f(n)}$ for some constant $c > 0$. Then, by simple counting, the nondeterministic test can be forced made to halt regardless of the nondeterministic choices that are made during the computation.

This proves the theorem. □

6.3.3 Characterizing One-way Nondeterministic Space by Splicing Systems

Theorem 6.3.4 immediately raises the question of whether the inclusion $\mathrm{SplTime}[O(\log n)] \subseteq$ 1-NL is an equality. We show that this is unlikely—even allowing the use of larger splicing systems for longer words does not enable logarithmic time-bounded splicing systems to produce anything beyond 1-NL. A family of boolean circuits $\mathcal{F} = \{F_n\}_{n\geq 0}$ is said to be *logarithmic-space uniform* [72] if the function $1^n \mapsto F_n$ is computable by a logarithmic space-bounded Turing machine. We introduce a concept of uniform families of splicing systems.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Let $\Gamma = (V, \Sigma, I, R)$ be a splicing system. We consider a binary encoding of Γ similar to those given for Turing machines (see, e.g., [31]) as follows:

- The size of V is specified as $1^{\|[\|V]}$.
- The size of Σ is specified as $1^{\|\Sigma\|}$.
- The symbols in V are numbered from 1 to $\|[\|V]$ and for each i, $1 \leq i \leq \|[\|V]$, the i-th member of V is encoded as 1^i.
- Each word in I is encoded by concatenating the encodings of the characters in I with a 0 in between, and the set I is encoded by concatenating the encodings of the words in I with a 00 in between.
- A finite automaton $A = (V, Q, \delta, q_0, q_f)$ is encoded in the following manner:
  - The size of Q is specified as $1^{\|[\|Q]}$.
  - The states in Q are numbered from 1 to $\|[\|Q]$ and for each i, $1 \leq i \leq \|[\|V]$, the i-th member of Q is specified as 1^i.
  - Let m_0 be the number assigned to q_0. Then q_0 is specified as 1^{m_0}.
  - Let m_1 be the number assigned to q_f. Then q_f is specified as 1^{m_1}.
  - The transition function δ is encoded as an enumeration of all permissible transitions in δ. We introduce some conventions to save space: (i) $1^{\|[\|V]+1}$ represents "any symbol." (ii) For each i, $1 \leq i \leq \|[\|V]$, $1^{\|[\|V]+1+i}$ represents "any symbol but the i-th one." (iii) Any transition not specified in the enumeration takes the automaton to reject.
  - A transition from the i-th state to the j-th state upon the k-th symbol/conventional-symbol is encoded as $1^i 0 1^j 0 1^k$. The transitions are concatenated with a 00 in between.
  - The encoding of A is the concatenation of the encodings of the components of A with a 000.
- Each quadruple of automata is encoded by concatenating the encodings of the four automata with a 0000 in between.
- The encoding of Γ is the concatenation of the encodings of the components of Γ with a 00000.

Definition 6.3.6. *Let $f(n) \in \Omega(\log n)$ be a function from $\mathbb{N}$ to itself. We say that a family of extended splicing systems, $\mathcal{G} = \{\Gamma_n\}_{n \geq 0}$, is* $f(n)$-space uniform *if the function that maps for each $n \geq 0$ from 1^n to the encoding of Γ_n is computable in deterministic $f(n)$ space.*

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Definition 6.3.7. *We say that a family of extended splicing systems, $\mathcal{G} = \{\Gamma_n\}_{n\geq 0}$ accepts a language L if the splicing systems in $\mathcal{G}$ have the same terminal alphabet Σ such that $L \subseteq \Sigma^*$ and for all $n \geq 0$, it holds that $L^{=n}$, the length-n portion of L, is equal to that of $L(\Gamma_n)$.*

Now we characterize 1-NSPACE using uniform families of splicing systems.

Theorem 6.3.8. *Let $f(n) \in \Omega(\log n)$ be a monotonically nondecreasing fully space-constructible function. A language L is in* 1-NSPACE$[f(n)]$ *if and only if there is an $f(n)$-space uniform family $\mathcal{G} = \{\Gamma_n\}_{n\geq 0}$ of splicing systems that accepts L with the following properties:*

1. *There exists a constant c such that for all $n \geq 0$, each automaton appearing in the rule set of Γ_n has at most c states.*

2. *There exists a constant d such that for all $n \geq 0$ and for all $w \in L^{=n}$, there is a production tree of Γ_n to produce w of height not more than $df(n)$.*

Proof. To prove the "if"-part, let $f(n) \geq \log n$ and suppose that L is accepted by an $f(n)$-space uniform family $\mathcal{G} = \{\Gamma_n\}_{n\geq 0}$ of splicing systems satisfying the two conditions in the statement of the theorem with the constants c and d. For each $n \geq 0$, let $\Gamma_n = (V_n, \Sigma, I_n, R_n)$. Let D be a machine that produces the encoding of Γ_n from 1^n in space $f(n)$. We will construct a one-way nondeterministic algorithm that uses $f(n)$ space for deciding membership in L. We can assume that, given x as input, $f(|x|)$ cells of each work tape of our machine are marked prior to the computation.

Let n be fixed and let x be a word in Σ^n whose membership in L we are testing. For now, assume further that an encoding of Γ_n is given on a separate tape with two-way read head that our machine can look up any time it deems necessary without violating the restriction on the work space size.

Our machine uses the nondeterministic one-way algorithm presented in the proof of Theorem 6.3.5, where the depth of each path is bounded by $df(n)$. The machine must use a work-tape alphabet of fixed size, so the elements in R_n and V_n have to be represented using words over the fixed work-tape alphabet. Note that an $f(n)$ space-bounded machine is $C^{f(n)}$ time-bounded for some constant C. This means that $O(f(n))$ binary bits are sufficient for indexing a position on the encoding of Γ_n, for indexing an automaton appearing in R_n, and for specifying a symbol in V_n. Thus, our machine runs the algorithm using binary indices to represent the elements of V_n and R_n. Since the nondeterministically guessed encoding of a production tree is read one-way, $O(f(n))$ work space is sufficient in producing a symbol in the encoding.

On scanning a symbol in the encoding, our machine may have to simulate at most $df(n)$ finite automata concurrently. According to our assumption, each finite automaton has at most c states, so the states of the finite automata that are being simulated can be memorized using $O(f(n))$ symbols. To identify the transition to be made, our machine has only to scan the encoding of Γ_n to find a match.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

To check whether a word appearing at a leaf node is in I_n, our machine guesses, when it is about to start scanning a word at a leaf (which must follow an occurrence of the fixed empty automaton symbol r_0), which word in I_n is about to be seen and then checks whether this guessed word is equal to the word appearing in the production tree. If unsuccessful, it rejects.

Call this modified algorithm $\mathcal{A}$. The algorithm $\mathcal{A}$ correctly works in one-way nondeterministic $O(f(n))$ space so long as the encoding of Γ_n is readily available. Since it is not, the encoding of Γ_n has to be computed. However, the machine has only $f(n)$ space on each work-tape, so the encoding of Γ_n has to be dynamically computed. Unfortunately, $f(n)$ may not be invertible and the input is one-way, so it is not *a priori* method for learning the value of n. Let $\alpha \geq 2$ be the smallest integer such that $\alpha^{f(n)} - 1 \geq n$. The quantity α is well defined since $f(n) \in \Omega(\log n)$. Our machine guesses n using an alphabet of size α. Given a space of $f(n)$ tape cells on one tape, it attempts to produce, using the α-adic expression naturally constructed over the size-α alphabet, our machine can cycle through the numbers in the interval $[0, \alpha^{f(n)} - 1]$. By the choice of α, the maximum number in the range is greater than or equal to n. For each number m thus produced, our machine simulates D, the machine for producing the encoding of Γ_m, on demand; that is, when $\mathcal{A}$ needs to see the letter of the encoding of Γ_n at a particular position, say k, our machine simulates D from the beginning until the kth output letter is produced. Such an on-demand simulation is possible if $m \leq n$. If a simulation needs to require more than the given space on any work tape, our machine rejects.

With this mechanism our machine executes the algorithm in the above, while counting the number of characters in the input. At the end of simulation, it checks whether the count is equal to m, and if the count is different, then it rejects regardless of the outcome of the simulation. It accepts only if the simulation accepts and the length was correctly guessed. This completes the proof of the "if"-part.

To prove the "only if"-part, let L be accepted by a one-way nondeterministic $f(n)$ space-bounded machine M. Let Σ be the input alphabet of M. We assume that a special symbol $\dashv$ not in Σ is appended at the end of the input of M so that M knows the end of the input. We also assume that M accepts only after seeing a $\dashv$. Let Q be the set of states of M. Let q_A be a unique accept state of M. For each symbol $a \in \Sigma$, introduce a new symbol $\hat{a}$. Let $\hat{\Sigma}$ be the collection of all newly introduced symbols and let $\Delta = \Sigma \cup \{\dashv\} \cup \hat{\Sigma}$.

Let n be fixed. We construct Γ_n as follows: Let S_n be the set of all configurations of M on an input of length M without the specification of the input head position and without the symbol scanned by the input head. Since M is $f(n)$ space-bounded, each element in S_n can be encoded using $O(f(n))$ characters. Let α_0 be the initial configuration in S_n and let Θ be the set of all accepting configurations in S_n.

Suppose M may nondeterministically transition from a configuration α to another β upon scanning an input symbol a. If the input head moves to the right when M makes the transition, we describe this as $(\alpha, a) \to \beta$; if the input head does not move when M

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

makes the transition, we describe this as $(\alpha, \hat{a}) \to \beta$.

Let @ and % be new symbols. We define

$$V_n = \Delta \cup S_n \cup \{@, \%\}$$

and Σ remains to be the terminals for Γ_n.

The initial language I_n consists of the following:

- the words @ and %; and
- $\alpha a \beta$ for all possible transitions.

The splicing rules are given as follows:

- For each $\alpha \in S_n$, $S_n \Sigma^+ \# \alpha \$ \alpha \# (\Sigma \cup \{\dashv\})^+ S_n$.
- For each $\alpha \in S_n$, and for $h \in \hat{\Sigma} \cup \{\dashv\}$, $S_n h \# \alpha \$ \alpha h \# S_n$.
- For each $\alpha \in S_n$, and for $h \in \Sigma$, $S_n \# \hat{h} \alpha \$ \alpha \# h S_n$.
- $\alpha_0 \Sigma^* \# \dashv \Theta \$ @ \# \lambda$.
- $\alpha_0 \# \Sigma^* \$ \lambda \# \&$.

Note that each finite automaton appearing in these rules has at most four states, so its deterministic version has at most 16 states. The constant c thus can be 16.

Note that Γ_n produces a word in Σ^* only from words of the form $\alpha_0 \Sigma^* \dashv \Theta$ by eliminating the α_0 at the beginning and the $\dashv \Theta$ at the end. These two eliminations are carried out by using the last two rules in the above. The other rules are for splicing two words having an element from S_n at the end, no element S_n in between, and at least another element in between. The rules of the first allow to combine transitions $(\alpha, \hat{a}) \to \beta$ and $(\beta, a) \to \gamma$, into an expression equivalent to $(\alpha, a) \to \gamma$. The rules of the second type allow to combine transitions $(\alpha, \hat{a}) \to \beta$ and $(\beta, \hat{a}) \to \gamma$, into an expression equivalent to $(\alpha, \hat{a}) \to \gamma$. The rules of the third type allow to combine transitions $(\alpha, u) \to \beta$ and $(\beta, v) \to \gamma$ such that $u, v \in \Sigma^+$, into an expression equivalent to $(\alpha, uv) \to \gamma$. The by-product of these rules is a word containing $S_0 S_0$. Such a word will never be spliced again. These observations allow us to conclude that if a word in Σ^n is produced by these splicing rules then it is in $L^{=n}$.

Now we show that every $x \in L^{=n}$ has a production tree of height at most $df(n)$ for some constant d. Let Π be any accepting computation of M on an input word x of length n. We can assume that this path Π does not contain repeated "full configuration." Here a "full configuration" is a "configuration" plus the head position. By this assumption, the length of Π is at most $h^{f(n)}$ for some fixed constant h. We write out Π as a sequence in which a simplified configuration in S_n and a symbol scanned by the head alternate so that a symbol a is represented by $\hat{a}$ if $a \in \Sigma$ and the input head does not move at that simplified configuration. Construct from this sequence a new

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

one in which an occurrence of an element in S_n excluding those at the end is repeated twice. By dividing this sequence into triples from the start produces the sequence, Ξ, of elements in I_n whose splicing with respect to the rules in R_n produces x. Since $|x| = n$, Ξ can be divided into n blocks corresponding to the letters of x so that within which block the Δ part forms the sequence either of the form $\hat{a} \cdots \hat{a}a$ for some $a \in \Sigma$ or of the form $\dashv \cdots \dashv$. In both cases, if the block has K triples then assembling all of them requires $\lceil \log(K + 1) \rceil$ splicing rounds. Once splicing within a block has been completed, assembling the characters in x requires $\lceil \log(n + 1) \rceil$ rounds. Thus, the total number of rounds for producing x is bounded by

$$\lceil \log(h^{f(n)} + 1) \rceil + \lceil \log(n + 1) \rceil + 2.$$

Since $f(n) \geq \log n$, this bound is $\Theta(f(n))$. Thus, there exists a desired constant d. This completes the "only if"-part of the proof.

This completes the proof of the theorem. □

The following corollary immediately follows from the above theorem.

Corollary 6.3.9. *A language L is in* 1-NL *if and only if there is a logarithmic-space uniform family $\mathcal{G} = \{\Gamma_n\}_{n \geq 0}$ of splicing systems that accepts L with the following properties:*

1. *There exists a constant c such that for all $n \geq 0$, each automaton appearing in the rule set of Γ_n has at most c states and has at most c transitions appearing in the encoding.*

2. *There exists a constant d such that for all $n \geq 0$ and for each $w \in L^{=n}$, there is a production tree of Γ_n to produce w of height not more than $d \log n$.*

6.4 Splicing Systems versus Pushdown Automata

Theorem 6.3.4 sheds light on the question we asked earlier: is CFL included in SplTime[$O(\log n)$]? Since the closure of 1-NL under the logarithmic-space Turing reducibility (see [35]) is NL, the closure of SplTime[$O(\log n)$] under that reducibility is included in NL. On the other hand, LOGCFL, the closure of CFL under the logarithmic-space many-one reducibility, is equal to SAC^1, the languages accepted by a logarithmic space uniform, polynomial-size, logarithmic-depth semi-unbounded-fan-in circuits [75]. The class SAC^1 is known to include NL but it is unknown whether the two classes are equal to each other. If CFL $\subseteq$ SplTime[$O(\log n)$], then we have that SAC^1 = NL. Because of this, it appears difficult to settle the question of whether CFL $\subseteq$ SplTime[$O(\log n)$]. We show that CFL $\subseteq$ SplTime[$O(n)$]. This inclusion follows from the following general result.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Theorem 6.4.1. *Let $f(n) \geq \log n$ be an arbitrary function. Let L be a language accepted by a pushdown automaton M with the property that for each member x of L there exists an accepting computation of M on x such that the height of the stack of M never exceeds $f(|x|)$. Then L belongs to* SplTime$[f(n)]$.

Proof. Let f, L, and M be as in the hypothesis of the theorem. Let Σ be the alphabet over which L is defined. We will first construct an extended splicing system $\Gamma = (V, \Sigma, I, R)$ such that $L = L(\Gamma)$. We will then show that this Γ has the desired property.

Let Q be the state set. Let q_0 and q_A be the initial state and the accept state of M, respectively. Let Θ be the stack alphabet. We assume that a special symbol $\dashv$ is appended to the input of M, that M must accept upon encountering a $\dashv$, and that M accepts with the empty stack. Without loss of generality, we decompose each move of M into the following two phases:

Phase 1 This phase is nondeterministic and may involve the head move. Depending on the input symbol its scanning M nondeterministically chooses the next state. It may move the input head to the right by one position.

Phase 2 This phase is deterministic and involves a stack operation. There are two disjoint subsets, Q_1 and Q_2, of Q. Depending on its state M executes one of the following:

(a) If the current state p is in Q_1, then M pushes a symbol determined by p and enters some state in $Q - (Q_1 \cup Q_2)$.

(b) If the current state p is in Q_2, then M pops a symbol from the stack and then does the following: if the stack doesn't return a symbol (because the stack is already empty), M halts without accepting; if the stack returns a symbol, say a, M enters a state in $Q - (Q_1 \cup Q_2)$, which is determined from p and a.

(c) If the current state p is not in $Q_1 \cup Q_2$, no stack operation is performed.

We will design a splicing system $\Gamma = (V, \Sigma, I, R)$ that simulates the computation of M. The design of Γ is partly similar to the design we used in the proof of Theorem 6.3.8. We first design the set of symbols B. The symbol $\dashv$ is considered to be a nonterminal. We introduce a new nonterminal $\hat{a}$ for each $a \in \Sigma$. Let $\hat{\Sigma} = \{\hat{a} \mid a \in \Sigma\}$. Each state in Q is considered to be a nonterminal. Furthermore, we introduce two nonterminals, $[p, p'q, q', a]_1$ and $[p, p'q, q', a]_2$, for each 5-tuple (p, p', q, q', a) such that

- $p, p', q, q' \in Q$, $a \in \Theta$,
- at state p, M may push an a into the stack and enter p', and
- at state q, M may pop from the stack and if the symbol is an a then may enter q'.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Let V_S denote the set of these nonterminals denoted by $[p, p'q, q', a]_1$ and $[p, p'q, q', a]_2$. Finally, we introduce nonterminals @ and %. These nonterminals and the elements of Σ, which are the terminals, comprise V; that is,

$$V = \Sigma \cup \hat{\Sigma} \cup Q \cup V_S \cup \{\dashv, @, \%\}.$$

The initial language I consists of the following:

- the words @ and %;
- for each combination of a symbol $a \in \Sigma$ and two states $p, q \in Q$ such that M in state p on symbol a may enter state q and move the head to the next position, the word paq;
- for each combination of a symbol $a \in \Sigma \in \{\dashv\}$ and two states $p, q \in Q$ such that M in state p on symbol a may enter state q without the head move, the word $p\hat{a}q$;
- for each state $q \in Q$, the words $@q$ and $q@$;
- for each 5-tuple (p, p', q, q', a) such that Ξ contains $(p, p', q, q', a)_1$ and $(p, p', q, q', a)_2$, the words pq' and $(p, p', q, q', a)_1(p, p', q, q', a)_2$.

The splicing rules R are given as follows:

1. For all $p \in Q$,
$$Q\Sigma^+\#p\$p\#(\Sigma^+ \cup \Sigma^*\{\dashv\})Q.$$
2. For each $p \in Q$, and for each $\hat{a} \in \hat{\Sigma}$,
$$Q\Sigma^*\#\hat{a}p\$p\#a(\Sigma^* \cup \Sigma^*(\hat{\Sigma} \cup \{\dashv\}))Q.$$
3. For each $p \in Q$, and for each $\hat{a} \in \hat{\Sigma}$,
$$Q\Sigma^*\hat{a}\#p\$p\hat{a}\#\Sigma^* \cup \Sigma^+(\hat{\Sigma} \cup \{\dashv\}))Q.$$
4. For each 5-tuple (p, p', q, q', a) such that $(p, p', q, q', a)_1$ and $(p, p', q, q', a)_2$ are in Ξ,
$$\begin{aligned}
&p'\Sigma(\Sigma^* \cup \Sigma^*\hat{\Sigma})\#q\$(p, p', q, q', a)_1\#(p, p', q, q', a)_2,\\
&p'\#\Sigma(\Sigma^* \cup \Sigma^*\hat{\Sigma})(p, p', q, q', a)_2\$(p, p', q, q', a)_1\#(p, p', q, q', a)_2,\\
&(p, p', q, q', a)_1\Sigma(\Sigma^* \cup \Sigma^*\hat{\Sigma})\#(p, p', q, q', a)_2\$p\#q',\\
&(p, p', q, q', a)_1\#\Sigma(\Sigma^* \cup \Sigma^*\hat{\Sigma})q'\$p\#q'.
\end{aligned}$$
5. $q_0\Sigma^*\# \dashv q_F\$@\#\lambda$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

6. $q_0\#\Sigma^*\$\lambda\#\&$.

This completes the design of Γ.

It is not very difficult to see that for all $w \in \Sigma^+ \cup \Sigma^* \cup \Sigma^*(\hat{\Sigma} \cup \{\dashv\})$ and for all $p, q \in Q$, the word pwq is produced by the splicing system if and only if M can arrive from p to q by scanning the word w with the following property on the stack: starting from any stack height h, the stack height never goes below h while scanning w, and at the end the height returns to h. In fact the first three sets of rules allow joining two such words pwq and $qw'r$ to produce $pww'r$. If w happens to end with a symbol $\hat{a}$ in $\hat{\Sigma}$ then w' must start with either an a or a $\hat{a}$ and the $\hat{a}$ at the end of w' will be eliminated. Also, in the case when $\hat{a}$ is the first symbol of w' and $|w'| > 1$ then the second symbol of w' must be in Σ. These joining operations produce words of the form pp and $p\hat{a}p$ as the by-product. The former type will never be spliced again unless $p = q_0 = q_F$ and $\lambda \in L$. When the latter type is spliced, the $\hat{a}p$-part will be replaced by something else to produce either a word of the form $p\hat{a}q$ where M indeed may enter from p to q upon a without moving the head or a word of the form paq where M indeed may enter from p to q upon a with a head move. The fourth set of rules is to insert a push operation at the beginning and a pop operation with respect to the same symbol at the end. The property about the maintenance of height is preserved. The remaining rules are for removing q_0 at the beginning and $\dashv q_F$ at the end. Thus, $L = L(\Gamma)$.

Now we show that there exists a constant δ such that for each word $x \in L$, $\mathrm{SplTime}_\Gamma(x) \leq \delta f(|x|)$. Let an integer $n \geq 0$ be fixed and let x be a member of $L^{=n}$. Select an arbitrary accepting computation path Π of M on input x such that during the execution of Π the stack height of M never exceeds $f(n)$. Let $h_{\max}$ be the maximum stack height that M achieves during the execution of Π. Then we have $h_{\max} \leq f(n)$. We think of Π as a sequence of pairs of the form (γ, α) such that γ is a configuration of M (consisting of a head position, a state, and stack contents) and α is an action M takes at γ. Here, by following our two-phase decomposition, α is either a move in Phase 1 or a move in Phase 2. Let $\Pi = [\pi_1, \ldots, \pi_m]$. Without loss of generality, we can assume that Π is minimal, in the sense that no element appears more than once. Since $f(n) \geq \log n$, from this assumption it follows that there exists a constant $C > 0$ depending only on M such that $m \leq C^{f(n)}$.

For an element $\pi = (\gamma, \alpha)$ in Π, we define the *height* of π to be the height of the stack of γ, and denote it by $\mathrm{height}(\pi)$. Note that this is the height before M takes the action α and the height of the stack immediately after α must coincide with the height of the entry immediately following π in the sequence Π. We use $\mathrm{height}'(\pi)$ to denote the height of the stack immediately after α. We say that a subsequence $\rho = [\pi_i, \ldots, \pi_j]$ of Π is a *tour* if ρ satisfies one of the following two conditions:

- $\pi_i, \ldots, \pi_j$ are all Phase 1 operations.
- $\mathrm{height}'(\pi_i) = \mathrm{height}(\pi) + 1 = \mathrm{height}(\pi_j) = \mathrm{height}'(\pi_j) + 1$, for all $k, i + 1 \leq k \leq j - 1$, $\mathrm{height}(\pi_k) \geq \mathrm{height}(\pi_i)$ and $\mathrm{height}'(\pi_k) \geq \mathrm{height}(\pi_i)$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Note that in the latter case, π_i is a push operation, π_j is a pop operation, the stack height after π_j is the same as the stack height before π_i, and the stack height is maintained larger than or equal to that before π_i during the execution of the operations in between.

We call a tour $[\pi_i, \ldots, \pi_j]$ a *single-round tour* if it is either a tour of the first type or either a tour of the second type with the additional property that the stack height is maintained larger than that before π_i during the execution of the operations between π_i and π_j. If a tour is not a single-round tour, then it is called a *multiple-round tour*.

For a tour $\rho = [\pi_i, \ldots, \pi_j]$, we define its *relative height*, denoted by $r(\rho)$, to be

$$\max\{\text{height}(\pi_k) - \text{height}(\pi_i) \mid i \leq k \leq j\},$$

that is, how much higher the stack gets after entering ρ. Note that each of the first type has relative height of 0.

A *multiple-round tour* can be expressed as the concatenation of at least two *single-round tours*. When decomposing a *multiple-round tour* into single-round tours, since each single Phase 1 operation is a tour, the decomposition may not be unique. To make the decomposition unique, we join each neighboring pair of single-round tours of relative height 0 and thereby use maximally long stretches of Phase 1 operations.

Note that each tour $\rho = [\pi_i, \ldots, \pi_j]$ corresponds to a word produced by Γ of the form pwq such that $p, q \in Q$ and $w \in \Sigma^*(\Sigma \cup \hat{\Sigma} \cup \{\dashv\}))$, where p is the state of π_i, q is the state that M enters by the action in π_j, and the word w consists of the partial input scanned by M during the partial computation ρ, but the last symbol of w is allowed to be either $\dashv$ or some $\hat{a} \in \hat{\Sigma}$ in the case when the last action in Phase 1 that occurs in ρ does not involve head movement. We denote this word pwq by $W(\rho)$ and define $t(\rho) = \text{SplTime}_\Gamma(W(\rho))$, the minimum time for Γ to produce $W(\rho)$. Also, for a tour ρ, the size of ρ, denoted by $s(\rho)$, is the number of Phase 1 operations in ρ.

We claim the following.

Claim. For all single-round tours ρ,

$$t(\rho) \leq 2\lceil \log s(\rho) \rceil + 6r(\rho).$$

We prove the claim by double induction on $r(\rho)$ and $s(\rho)$.

The base case for $r(\rho)$ is when $r(\rho) = 0$. In this case ρ consists only of Phase 1 operations, and thus, $s(\rho) = |\rho|$. This means that the action part of each element of ρ corresponds to a word in the initial language. The time that it takes for these initial words to assemble into $W(\rho)$ is $\lceil \log s(\rho) \rceil$, and thus, we have, $t(\rho) \leq \lceil \log s(\rho) \rceil$. Hence, the claim holds for the base case.

For the induction step, let $r(\rho) = r_0 \geq 1$ and assume that the claim holds for the values of $r(\rho)$ that are smaller than r_0. The shortest tour of relative-height r_0 has r_0 pairs of push and pop and, since these push and pop operations are in Phase 2, contains a Phase 1 operations between each neighboring pair of stack operations. Let ρ be one

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

of the shortest tours. Then we have

$$|\rho| = 4r_0 - 1 \text{ and } s(\rho) = 2r_0 - 1.$$

It takes four rounds for M to process a stack-operation pair and two rounds for appending the words corresponding two non-stack operations flanking the pair. The total number of rounds required to produce $W(\rho)$ is thus $2r_0 - 2 + 4r_0 < 6r_0$, and hence, the claim holds for this smallest single-round tour.

Next, for the induction step on $s(\rho)$, let $s_0 = s(\rho)$ be the size of ρ such that $s_0 > 2r_0 - 1$. Suppose that the claim holds for all values of $s(\rho)$ smaller than s_0. Let $\rho = [\pi_i, \ldots, \pi_j]$ be a single-round tour of relative height r_0 and of size s_0. Let $\rho' = [\pi_{i+1}, \ldots, \pi_{j-1}]$. Since a stack operation that is not at the end is necessarily followed by a non-stack operation and since π_i is a stack operation, π_{i+1} is not a stack operation. This means that ρ' is not a single-round tour. Suppose that ρ' is the concatenation of m single-round tours, $\xi_1, \ldots, \xi_m$, where $m \geq 2$. Since ρ is a single-round tour, each one of $\xi_1, \ldots, \xi_m$ has relative height at most $r_0 - 1$. For each i, $1 \leq i \leq m$, let $\ell_i = s(\xi_i)$. Clearly, $\ell_i + \cdots + \ell_m = s_0$.

Suppose $m = 2$. Both ξ_1 and ξ_2 have relative height at most $r_0 - 1$. By our induction hypothesis, we have

$$t(\xi_1) \leq 2 \log \ell_1 + 6(r_0 - 1) \text{ and } t(\xi_2) \leq 2 \log \ell_2 + 6(r_0 - 1).$$

We have $t(\rho) = \max\{t(\xi_1), t(\xi_2)\} + 5$. The reason is that the words $W(\xi_1)$ and $W(\xi_2)$ are spliced together in the round immediately after the round in which both two words are present for the first time and appending the push-pop pair represented by the two ends of ρ requires four rounds. We thus have

$$t(\rho) = 2 \max\{\log \ell_1, \log \ell_2\} + 6(r_0 - 1) + 5.$$

This is less than $2 \log s_0 + 6r_0$, and thus, the claim holds.

Next suppose that $m \geq 3$. Let c be the smallest i such that $\ell_1 + \cdots + \ell_c > s_0/2$. Such c clearly exists. Partition $\ell_1, \ldots, \ell_m$ into three groups:

$$(\ell_1, \ldots, \ell_{c-1}), (\ell_c), (\ell_{c+1}, \ldots, \ell_m).$$

Here the first group is empty if $\ell_1 > s_0/2$ and the second group is empty if $\ell_m \geq s_0/2$. Because of the definition of c, we have

$$\begin{aligned} \ell_1 + \cdots + \ell_{c-1} &\leq s_0/2, \text{ and} \\ \ell_{c+1} + \cdots + \ell_m &\leq s_0/2. \end{aligned}$$

Consider the production of $W(\rho)$ in which the word corresponding to each of these groups is produced first, three words are then connected, and then finally the push-pop pair at the end of ρ is processed. The time required for producing the word for the first

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

group is

> "the time required for producing the word for a hypothetical single-round tour of relative height $\leq r_0$ that is constructed from $[\ell_1, \ldots, \ell_{c-1}]$ by inserting a push operation at the beginning and a pop operation at the end"-4.

Here 4 that is substracted is the time required for processing the new stack operation pair. So, by our induction hypothesis, the time in question is at most

$$2\lceil \log(s_0/2) \rceil + 6r_0 - 4 = 2\lceil \log s_0 \rceil + 6(r_0 - 1).$$

The same holds for the third group. As to ℓ_c, by our induction hypothesis,

$$t(\ell_c) \leq 2\lceil \log s_0 \rceil + 6(r_0 - 1).$$

The additional number of rounds for joining the three words and processing the push-pop pair of ρ is $2 + 4 = 6$. Thus, we have

$$t(\rho) \leq 2\lceil \log s_0 \rceil + 6(r_0 - 1) + 6 = 2\lceil \log s_0 \rceil + 6r_0$$

as desired.

The claim has been proven. When Π is not a single-round tour, think of a hypothetical single-round tour $\hat{\Pi}$, which is constructed from Π by inserting one push operation at the beginning and one pop operation at the end. This increases the relative height by 1 but preserves the size. Then, we have

$$t(\hat{\Pi}) \leq 2 \log s(\Pi) + 6(f(n) + 1).$$

Removing the two state symbols at the end of $W(\hat{\Pi})$ requires two additional rounds, so we have

$$\mathrm{SplTime}_\Gamma(x) \leq 2 \log s(\Pi) + 6f(n) + 8.$$

By our supposition $s(\Pi) \leq |\Pi| \leq C^{f(n)}$. So, there exists a constant D such that

$$\mathrm{SplTime}_\Gamma(x) \leq Df(n).$$

This proves the theorem. □

Since a finite automaton can be viewed as a PDA without stack operations, the result of Theorem 6.2.5 that REG $\subseteq$ SplTime$[O(\log n)]$ follows from this theorem.

Also, the standard PDA algorithm for a context-free language uses the Greibach normal form and the stack height is bounded by a linear function of the input size. More precisely, one push operation is executed when a production rule of the form $A \to BC$ is performed and when alignment of C with the input is postponed until the alignment of B with the input has been completed (see, for example, [31]). Since each nonterminal produces a nonempty word, this property means that the number

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

of symbols in the stack does not exceed the length of the input. This observation immediately yields the following corollary.

Corollary 6.4.2. CFL $\subseteq$ SplTime[$O(n)$],

Can we strengthen the above upper bound of SplTime[$O(n)$] presented in Corollary 6.4.2 to SplTime[$O(\log n)$]? This is a hard question to answer. As mentioned earlier, the logarithmic-space reducibility closure of CFL is equal to SAC^1, the languages accepted by polynomial size-bounded, logarithmic depth-bounded, logarithmic-space uniform families of semi-unbounded-fan-in (OR gates have no limits on the number of input signals feeding into them while the number is two for AND gates) circuits. This class clearly solves the reachability problem, so $\mathrm{SAC}^1 \subseteq$ NL, but it is not known whether the converse holds. Theorem 6.3.4 shows that SplTime[$O(\log n)$]$\subseteq$ 1-NL. Since NL is closed under logarithmic-space reductions, we have that the closure of SplTime[$O(\log n)$] under logarithmic-space reductions is included in NL. Thus, the hypothesis CFL $\subseteq$ SplTime[$O(\log n)$] implies $\mathrm{SAC}^1 = \mathrm{NL}$.

Proposition 6.4.3. CFL $\not\subseteq$ SplTime[$O(\log n)$] *unless* $\mathrm{SAC}^1 = \mathrm{NL}$.

6.5 Splicing Systems versus Nondeterministic Space

Theorem 6.4.1 shows a lower bound of splicing time complexity classes with respect to stack-height-bounded pushdown automata. Here we prove a lower bound with respect to space-bounded nondeterministic Turing machines.

Theorem 6.5.1. *For all* $f(n) = \Omega(n)$,

$$\mathrm{NSPACE}[f(n)] \subseteq \mathrm{SplTime}[O(f(n)^2)].$$

Proof. Let $L \in \mathrm{NSPACE}[f(n)]$ and $f(n) = \Omega(n)$. Without loss of generality, we assume that there exists an $f(n)$ space-bounded one-tape nondeterministic Turing machine M that accepts L. Let Q, Σ, δ, q_0, and q_f be respectively the state set, the input alphabet, the work alphabet, the transition function, the initial state, and the final state of M. Without loss of generality, we may assume that the tape of M tape is one-way infinite and that the head of M does not make a stationary move.

We define some notation. Let $\Gamma' = Q \times \Gamma$ be the set of all state-symbol combinations of M. We will treat each member of Γ' as a symbol. Let $C = \Gamma^*\Gamma'\Gamma^*$. Then C represents the set of all configurations of M. A word $w = a_1 \cdots a_k(q, b)c_1 \cdots c_l$ in C such that $a_1, \ldots, a_k, c_1, \ldots, c_l \in \Gamma$ and $(q, b) \in \Gamma'$ encodes the configuration in which the following conditions hold:

- The word on the tape of M is $a_1 \cdots a_k b c_1 \ldots c_l B \cdots$.
- The head of M is on the $(k + 1)$-st cell.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- The state of M is q.

Note that for all words $w \in C$ and for all $t \geq 1$, w and wB^t encode the same configuration.

For each pair (u, v), $u, v \in C$, such that $|u| = |v|$ and M can reach from the configuration represented by u to the configuration represented by v in a single step, we write $u \vdash_M v$. For each pair of configurations (u, v), $u, v \in C$, $|u| = |v|$, and for each $t \geq 1$, we write $u \vdash_M^{\leq t} v$ if there exists a series of configurations $u_0, \ldots, u_s$ such that $0 \leq s \leq t$, $u = u_0$, $v = u_s$, and for each i, $1 \leq i \leq s$, $u_{i-1} \vdash_M u_i$; that is, v is reachable from u in at most t steps. Also, for $u, v \in C$, write $u \vdash_M^* v$ to mean $(\exists t \geq 1)[u \vdash_M^t v]$. For a word w, let w^r denote the reverse of w. For each $t \geq 1$, let

$$W_t = \{\%_L u@v^r\%_R \mid (u, v \in C) \wedge |u| = |v| \wedge u \vdash_M^{\leq n} v, n \leq t\},$$

where $@$, $\%_L$, and $\%_R$ are new symbols. Let $W_* = \cup_{t \geq 1} W_t$.

We construct a splicing system H that produces W_*. Intuitively, for each $k \geq 0$, H builds W_{2^k} in k stages as follows: On stage $k = 0$, H produces all words in W_1, i.e., all words w of the form $\%_L u@v^r\%_R$ such that either $u \vdash_M v$ or $u = v$. On stage $k \geq 1$, H combines any pair (w, w'), $w, w' \in W_{2^{k-1}}$, such that $w = \%_L u@v^r\%_R$ and $w' = \%_L v@z^r\%_R$, and produces $\%_L u@w^r\%_R$. The union of the words thus produced and $W_{2^{k-1}}$ becomes W_{2^k}. Clearly, repeating this process indefinitely produces W_*. In addition, from a word $\%_L u@v^r\%_R \in W_*$ such that u is an initial configuration and v is an accepting configuration, H produces the input specified in u. This design ensures that exactly those words accepted by M are produced as words over the terminals. A large part of the idea in our construction comes from the one presented in [63].

Let

$$\begin{aligned} T_1 &= \{(p, q, a, b, c) \mid p, q \in Q, a, b, c \in \Gamma, \delta(p, a) = (q, b, \rightarrow)\} \text{ and} \\ T_2 &= \{(p, q, a, b, c) \mid p, q \in Q, a, b, c \in \Gamma, \delta(p, a) = (q, b, \leftarrow)\}. \end{aligned}$$

That is, T_1 (respectively, T_2) is the set of all legal transitions of M in which the head moves right (respectively, left). We construct our extended regular splicing system $\Gamma = (V, \Sigma, I, R)$ as follows. We define

$$\begin{aligned} V &= \Gamma \cup \Gamma' \\ &\cup \{\downarrow, \uparrow, \&_L, \&_R, \heartsuit_L, \heartsuit_R, Z\} \\ &\cup \{\&_{a,L}, \&_{a,R} \mid a \in \Gamma\} \\ &\cup \{\&_{1,t,L}, \&_{1,t,R} \mid t \in T_1\} \\ &\cup \{\&_{2,t,L}, \&_{2,t,R} \mid t \in T_2\} \\ &\cup \{\clubsuit_i \mid 1 \leq i \leq 4\} \\ &\cup \{\clubsuit_{\alpha,L}, \clubsuit_{\alpha,R}, \heartsuit_{\alpha,L}, \heartsuit_{\alpha,R} \mid \alpha \in \Gamma \cup \Gamma'\} \text{ and} \end{aligned}$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$$
\begin{aligned}
I &= \{\&_L@\&_R, \&_L\downarrow, \downarrow\&_R, \%_L\downarrow, \downarrow\%_R, \clubsuit_1\downarrow, \downarrow\clubsuit_2\clubsuit_2, \clubsuit_3\clubsuit_3\downarrow, \downarrow\clubsuit_4,\\
&\quad \heartsuit_L\downarrow, \downarrow\heartsuit_R, aZ\}\\
&\cup \{\clubsuit_{\alpha,L}\downarrow, \downarrow\clubsuit_{\alpha,R} \mid \alpha \in \Gamma \cup \Gamma'\}\\
&\cup \{\&_{a,L}a\downarrow, \&_{a,R} \mid a \in \Gamma\}\\
&\cup \{\&_{1,t,L}(p,a)c\downarrow, \downarrow(q,c)b\&_{1,t,R} \mid t \in T_1\}\\
&\cup \{\&_{2,t,L}c(p,a)\downarrow, \downarrow b(q,c)\&_{2,t,R} \mid t \in T_2\}.\\
R &= R_1 \cup R_2 \cup R_3.
\end{aligned}
$$

First, the words in W_1 are built from $\&_L@\&_R$ by using the rules in R_1. For each $a \in \Gamma$, R_1 contains the following eight rules, whose collectively insert an a immediately after $\&_L$ and an a immediately before $\&_R$. This process uses markers $\&_{a,L}$ and $\&_{a,R}$ that are specific to a. The use of these special symbols enable synchronization of the insertion at both ends. Of the eight, the first four are for $u@v$ that are yet to contain symbols from Γ' on each side (and so are non-members of C), and the second four are for words $u@v$ such that both u and v contain exactly one symbol from Γ' (and so are members of C).

$$
\begin{aligned}
\&_L\#C@C\&_R &\ \$\ \&_{a,L}a\#\downarrow,\\
\&_{a,L}aC@C\#\&_R &\ \$\ \downarrow\#a\&_{a,R},\\
\&_{a,L}\#aC@Ca\&_{a,R} &\ \$\ \&_L\#\downarrow,\\
\&_LaC@Ca\#\&_{a,R} &\ \$\ \downarrow\#\&_R,\\
\&_L\#\Gamma^*@\Gamma^*\&_R &\ \$\ \&_{a,L}a\#\&_{a,R},\\
\&_{a,L}a\Gamma^*@\Gamma^*\#\&_R &\ \$\ \&_{a,L}\#a\&_{a,R},\\
\&_{a,L}\#a\Gamma^*@\Gamma^*a\&_{a,R} &\ \$\ \&_L\#\downarrow,\\
\&_La\Gamma^*@\Gamma^*a\#\&_{a,R} &\ \$\ \downarrow\#\&_R.
\end{aligned}
$$

Also, R_1 contains rules for inserting symbols from Γ'. For each $t = (p,q,a,b,c) \in T_1$, R_1 contains:

$$
\begin{aligned}
\&_L\#\Gamma^*@\Gamma^*\&_R &\ \$\ \&_{1,t,L}(p,a)c\#\downarrow,\\
\&_{1,t,L}(p,a)c\Gamma^*@\Gamma^*\#\&_R &\ \$\ \downarrow\#(q,c)b\&_{1,t,R},\\
\&_{1,t,L}\#(p,a)c\Gamma^*@\Gamma^*(q,c)b\&_{1,t,R} &\ \$\ \&_L\#\downarrow,\\
\&_L(p,a)c\Gamma^*@\Gamma^*(q,c)b\#\&_{1,t,R} &\ \$\ \downarrow\#\&_R,
\end{aligned}
$$

and for each $t = (p,q,a,b,c) \in T_2$:

$$
\begin{aligned}
\&_L\#\Gamma^*@\Gamma^*\&_R &\ \$\ \&_{2,t,L}c(p,a)\#\downarrow,\\
\&_{2,t,L}c(p,a)\Gamma^*@\Gamma^*\#\&_R &\ \$\ \downarrow\#b(q,c)\&_{2,t,R},
\end{aligned}
$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$$\begin{aligned} \&_{2,t,L}\#c(p,a)\Gamma^*@\Gamma^*b(q,c)\&_{2,t,R} \quad \$ \quad \&_L\#\downarrow, \\ \&_Lc(p,a)\Gamma^*@\Gamma^*b(q,c)\#\&_{2,t,R} \quad \$ \quad \downarrow\#\&_R. \end{aligned}$$

The former group of four has the role of inserting $(p,a)c$ immediately after $\&_L$ and $(q,c)b$, which is the reverse of $b(q,c)$, immediately before $\&_R$. The latter gourd of four has the role of inserting $c(p,a)$ immediately after $\&_L$ and $b(q,c)$, which is the reverse of $(q,c)b$, immediately before $\&_R$. Furthermore, R_1 has the following rules, which are for replacing $\&_L$ with $\%_L$ and $\&_R$ with $\%_R$:

$$\begin{aligned} \&_L\#C@C\&_R \quad \$ \quad \%_L\#\downarrow, \\ \%_LC@C\#\&_R \quad \$ \quad \downarrow\#\%_R. \end{aligned}$$

For each n, and for each $w = \%_Lu@v^r\%_R \in W_1$ such that $|u| = |v| = n$, the time that it takes to produce w is $4(n-1)+2 = 4n-2$. All necessary auxiliary strings, all containing $\downarrow$, are provided in the initial language.

The rules in R_2 have the role of producing $\%_Lu@z^r\%_R \in W_*$ from any $\%_Lu@v^r\%_R \in W_*$ and any $\%_Lv@z^r\%_R \in W_*$. First, the rules

$$\begin{aligned} \%_L\#C@C\%_R \quad \$ \quad \clubsuit_1\#\downarrow, \\ \clubsuit_1C@C\#\%_R \quad \$ \quad \downarrow\#\clubsuit_2\clubsuit_2, \\ \%_L\#C@C\%_R \quad \$ \quad \clubsuit_3\clubsuit_3\#\downarrow, \\ \clubsuit_3\clubsuit_3C@C\#\%_R \quad \$ \quad \downarrow\#\clubsuit_4, \\ \clubsuit_1C@C\clubsuit_2\#\clubsuit_2 \quad \$ \quad \clubsuit_3\#\clubsuit_3C@C\clubsuit_4. \end{aligned}$$

produce

$$\clubsuit_1y@z^r\clubsuit_2\clubsuit_3u@v^r\clubsuit_4$$

from any $\%_Ly@z^r\%_R \in W_*$ and any $\%_Lu@v^r\%_R \in W_*$.

Second, R_2 contains for each $a \in \Gamma$ and $\alpha \in \Gamma \cup \Gamma'$ the following rules:

$$\begin{aligned} \clubsuit_1a\#C@C\clubsuit_2\clubsuit_3C@Ca\clubsuit_4 \quad \$ \quad \clubsuit_{a,L}\#\downarrow, \\ \clubsuit_{a,L}C@C\clubsuit_2\clubsuit_3C@C\#a\clubsuit_4 \quad \$ \quad \downarrow\#\clubsuit_{a,R}, \\ \clubsuit_{a,L}\#C@C\clubsuit_2\clubsuit_3C@C\clubsuit_{a,R} \quad \$ \quad \clubsuit_1\#\downarrow, \\ \clubsuit_1C@C\clubsuit_2\clubsuit_3C@C\#\clubsuit_{a,R} \quad \$ \quad \downarrow\#\clubsuit_4, \\ \clubsuit_1\alpha\#\Gamma^*@C\clubsuit_2\clubsuit_3C@\Gamma^*\alpha\clubsuit_4 \quad \$ \quad \clubsuit_{\alpha,L}\#\downarrow, \\ \clubsuit_{\alpha,L}\Gamma^*@C\clubsuit_2\clubsuit_3C@\Gamma^*\#\alpha\clubsuit_4 \quad \$ \quad \downarrow\#\clubsuit_{\alpha,R}, \\ \clubsuit_{\alpha,L}\#\Gamma^*@C\clubsuit_2\clubsuit_3C@\Gamma^*\clubsuit_{\alpha,R} \quad \$ \quad \clubsuit_1\#\downarrow, \\ \clubsuit_1\Gamma^*@C\clubsuit_2\clubsuit_3C@\Gamma^*\#\clubsuit_{\alpha,R} \quad \$ \quad \downarrow\#\clubsuit_4. \end{aligned}$$

These rules produce

$$\clubsuit_1@z^r\clubsuit_2\clubsuit_3u@\clubsuit_4$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

from $\clubsuit_1 y@z^r\clubsuit_2\clubsuit_3 u@v^r\clubsuit_4$ if and only if $y = v$.

Finally, R_2 has the following rules:

$$\begin{aligned}
\clubsuit_1@\#C\clubsuit_2\clubsuit_3C@\clubsuit_4 \quad &\$ \quad \heartsuit_L\#\downarrow,\\
\heartsuit_LC\clubsuit_2\clubsuit_3C@\#\clubsuit_4 \quad &\$ \quad \downarrow\#\heartsuit_R,\\
\heartsuit_L\clubsuit_2\clubsuit_3C@C\clubsuit_R \quad &\$ \quad \%_L\#\downarrow,\\
\%_LC@C\clubsuit_R \quad &\$ \quad \downarrow\#\%_R,
\end{aligned}$$

and for each $a \in \Gamma$ and $\alpha \in \Gamma \cup \Gamma'$,

$$\begin{aligned}
\heartsuit_La\#C\heartsuit_2\heartsuit_3C@\Gamma^*\heartsuit_4 \quad &\$ \quad \heartsuit_{a,L}\#\downarrow,\\
\heartsuit_{a,L}C\heartsuit_2\heartsuit_3C@\Gamma^*\#\heartsuit_4 \quad &\$ \quad \downarrow\#\heartsuit_{a,R},\\
\heartsuit_{a,L}\#C\heartsuit_2\heartsuit_3C@\Gamma^*\heartsuit_{a,R} \quad &\$ \quad \heartsuit_L\#\downarrow,\\
\heartsuit_LC\heartsuit_2\heartsuit_3C@\Gamma^*\#\heartsuit_{a,R} \quad &\$ \quad \downarrow\#a\heartsuit_R,\\
\heartsuit_L\alpha\#\Gamma^*\heartsuit_2\heartsuit_3C@\Gamma^*\heartsuit_4 \quad &\$ \quad \heartsuit_{\alpha,L}\#\downarrow,\\
\heartsuit_{\alpha,L}\Gamma^*\heartsuit_2\heartsuit_3C@\Gamma^*\#\heartsuit_4 \quad &\$ \quad \downarrow\#\heartsuit_{\alpha,R},\\
\heartsuit_{\alpha,L}\#\Gamma^*\heartsuit_2\heartsuit_3C@\Gamma^*\heartsuit_{\alpha,R} \quad &\$ \quad \heartsuit_L\#\downarrow,\\
\heartsuit_L\Gamma^*\heartsuit_2\heartsuit_3C@\Gamma^*\#\heartsuit_{\alpha,R} \quad &\$ \quad \downarrow\#\alpha\heartsuit_R.
\end{aligned}$$

These rules move the z^r at the beginning to immediately after the $u@$ and then add $\%_L$ at the beginning and $\%_R$ at the end, giving the desired result

$$\%_Lu@z^r\%_R \in W_*.$$

If $|u| = |v| = |z| = n$, the production of $w = \%_Lu@z^r\%_R$ from from $\%_Lu@v^r\%_R$ and $\%_Lv@z^r\%_R$ requires $5 + 4n + 2 + 4n + 2 = 8n + 9$ steps.

The rules of R_3 have the role of extracting the input word from a word $\%_Lu@v^r\%_R \in W_*$ such that u is an initial configuration and v is an accepting configuration. The set R_3 contains for all $(a, b) \in \Sigma \times \Sigma$, the following rules:

$$\begin{aligned}
\%_L(q_0, a)\Sigma^*\#B^*@\Gamma^*(q_f, b)\Gamma^*\%_R \quad &\$ \quad \downarrow\#\uparrow,\\
\%_L(q_0, a)\#\Sigma^*\uparrow \quad &\$ \quad a\#Z,\\
a\Sigma^*\#\uparrow \quad &\$ \quad Z\#\epsilon.
\end{aligned}$$

The first kind is for producing the word of the form $\%_L(q_0, a)\Sigma^*\uparrow$. The second and the third kinds are for replacing (q_0, a) by a and eliminating the $\uparrow$, using auxiliary string aZ.

It is not hard to see that, to produce $w = \%_Lu@v^r\%_R \in W_t$ such that $|u| = |v| = n$ the number of rounds H demands is

$$(4n - 2) + \lceil\log(t)\rceil(8n + 9) \leq (4 + 8(\log(t) + 1))n + 9\log(t) + 7 \leq dn(\log(t) + 1)$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

for some fixed constant d. Since M is $s(n)$ space-bounded, M is $2^{es(n)}$ time-bounded for some fixed constant e. So, if $x \in L$ and $|x| = n$, the maximum number of rounds for producing x is bounded by $2des(n)^2 = O(s(n)^2)$.

This proves the theorem. □

Let SplTime[poly] = $\cup_{c>0}$SplTime[n^c]. By combining Theorem 6.5.1 and Theorem 6.3.5 via Savitch's Theorem [73], we have the following corollary.

Corollary 6.5.2. SplTime[poly] = PSPACE.

Let SplTime[polylog] = $\cup_{c>0}$SplTime[$(\log n)^c$]. Then we have the following corollary.

Corollary 6.5.3. SplTime[polylog] = POLYLOGSPACE.

In all, the notion of time complexity introduced in this chapter has proven a useful concept, which allowed comparisons with well known Turing machines classes and characterizations in terms of them. We believe that our concept can be a useful tool in understanding the intrinsic computational abilities of splicing systems.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

7 Space Complexity for Splicing Systems

7.1 Introduction

In Chapter 6 we introduced time complexity for splicing systems. This chapter further explores complexity for splicing systems by defining a notion of space complexity, which is based on the description size of the production tree of a word. We can then characterize the class SplSpace[$f(n)$] in terms of time-bounded nondeterministic Turing machines, Specifically, we show that there exists a finite k such that for every fully space-constructible function $f(n)$ the languages produced by extended splicing systems with a regular set of rules having space complexity $f(n)$ are accepted by $O(f(n))$ time bounded nondeterministic Turing machines. Also, it is shown that all languages accepted by $f(n)$ time-bounded nondeterministic Turing machines can be generated by extended regular splicing systems in space $O(f(n)^k)$. By combining these two results it is shown that the class of languages generated extended splicing systems with a regular set of rules in polynomial space is exactly NP and that in exponential space is exactly NEXPTIME.

Where in the previous chapter we defined and studied time complexity for splicing systems, here we propose and study a notion of space complexity of splicing systems, which is based on the *production tree* of a word. Like in other formal systems of languages, a production tree describes the word production in a splicing system using a tree with labeled nodes. In a production tree of a word in a splicing system, each non-leaf is labeled with a word and a rule such that the word is produced by applying the rule to the word labels of its children. Given the set of all such production trees for a word (i.e., all production trees having the word as the word label at the root), with respect to a given splicing system, the space complexity of the word is defined to be the size of the smallest tree in the tree set, where the size refers to the sum of the number of nodes plus the sum of the length of the node labels. In fact, thinking in terms of production trees, the time complexity of the word with respect to the system can be identified with the height of the shortest tree among the tree set.

As in Chapter 6, we define our notion in terms of splicing systems with a finite initial language and a regular set of rules. As before, we think of the set R of splicing rules as a set of quadruples of regular languages (A_i, B_i, C_i, D_i), $1 \leq i \leq m$, such that $R = \cup_{1 \leq i \leq m} \{a_i \# b_i \$ c_i \# d_i \mid a_i \in A_i, b_i \in B_i, c_i \in C_i, d_i \in D_i\}$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

7.2 Space Complexity for Splicing Systems

Let $\Gamma = (V, \Sigma, I, R)$ be an extended splicing system with rules defined by quadruples of regular languages. Let $L = L(\Gamma)$. Recall that a binary tree is full if every node has either zero or two children. We define a *production tree* with respect to Γ of a word $w \in L$ as follows.

Definition 7.2.1. *For each $w \in L$, a production tree T of w with respect to Γ is a node-labeled full binary tree with the following properties:*

1. *Each leaf is labeled by some word in I.*
2. *Each non-leaf is labeled by some word $x \in V^*$ and an automaton-quadruple $r \in R$ such that $(u, v) \vdash_r x$, where u and v are respectively the word label of the left child and that of the right child.*
3. *The word label of the root of T is w.*

The production tree represents a possible derivation that Γ can follow to produce w. Note that a word may have multiple production trees.

Definition 7.2.2. *For each production tree T, define the following:*

- *The* output *of a production tree T, denoted by output(T), is the word label of the root.*
- *The* height *of T, denoted by height(T), is the length of the path from the root node to its furthest leaf.*
- *The* size *of T, denoted by size(T), is the number of nodes in T plus the sum of the length of the word labels appearing in T.*

For an extended splicing system $\Gamma = (V, \Sigma, I, R)$ and $w \in V^*$, let $\mathcal{T}_\Gamma(w)$ denote the set of all production trees of w with respect to Γ. Now we define the measure of time complexity of splicing systems.

Definition 7.2.3. *Let $\Gamma = (V, \Sigma, I, R)$ be a regular extended splicing system. For each $w \in V^*$, define* $\mathrm{SplSpace}_\Gamma(w)$ *to be*

$$\begin{cases} \min_{\{} \mathit{size}(T_\Gamma) \mid T \in \mathcal{T}_\Gamma\} & \text{if } w \in L(\Gamma), \\ 0 & \mathit{otherwise.} \end{cases}$$

Note that the definition of space in the above is reminiscent of the the notion of size complexity of circuits.

Let $\mathbb{N}$ denote the set of all natural numbers.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Definition 7.2.4. *Let $f(n) : \mathbb{N} \to \mathbb{N}$ be a monotonically nondecreasing function. Then* SplSpace[$f(n)$] *is the set of all $L(\Gamma)$ such that*

$$(\forall w \in \Sigma^*)[\text{SplSpace}_\Gamma(w) \leq f(|w|)]$$

Definition 7.2.5. *For a class C of monotone nondecreasing functions from $\mathbb{N}$ to itself, define*

$$\text{SplSpace}[C] = \cup_{f(n)\in C}\text{SplSpace}[f(n)].$$

In the following, we will consider only monotone nondecreasing functions as complexity functions and simply say functions to mean monotone nondecreasing functions.

From these definitions we can make some initial observations. First, all finite languages have space complexity of n. Second, SplSpace[$O(2^{2f(n)})$] includes the class SplTime[$f(n)$].

Here we also observe that the time complexity studied in the previous chapter can also be defined equivalently in terms of the *height* of the production tree. Specifically, if $\Gamma = (V, \Sigma, I, R)$ is a regular extended splicing system:

Definition 7.2.6. *For each $w \in V^*$,*

$$\text{SplTime}_\Gamma(w) = \begin{cases} \min\{height(T) \mid T \in \mathcal{T}_\Gamma\} & \text{if } w \in L(\Gamma), \\ 0 & otherwise. \end{cases}$$

Definition 7.2.7. *Let $f(n) : \mathbb{N} \to \mathbb{N}$ be a monotonically nondecreasing function. Then* SplTime[$f(n)$] *is the set of all languages $L(\Gamma)$ such that*

$$(\forall w \in \Sigma^*)[\text{SplTime}_\Gamma(w) \leq f(|w|)].$$

Definition 7.2.8. *For a class C of monotone nondecreasing functions from $\mathbb{N}$ to itself, define*

$$\text{SplTime}[C] = \cup_{f(n)\in C}\text{SplTime}[f(n)]$$

7.3 Characterizing SplSpace[$f(n)$]

In this section, we characterize SplSpace[$f(n)$] in terms of nondeterministic time-bounded Turing machine computation.

First we show that all languages in SplSpace[$f(n)$] are accepted by a nondeterministic Turing machine in time $O(f(n))$.

Lemma 7.3.1. *For all fully time-constructible functions $f(n) = \Omega(n)$,*

$$\text{SplSpace}[f(n)] \subseteq \text{NTIME}[O(f(n))].$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Proof. Let L be a language SplSpace$[f(n)]$ produced by an FIRR extended splicing system $\Gamma = (V, \Sigma, I, R)$. Let $R = \{r_1, \ldots, r_m\}$, where for each i, $1 \leq i \leq m$, r_i is a quadruple of regular expressions. By convention, let r_0 be a hypothetical fixed rule that splices no pair of words. Let $\hat{R} = \{r_0, \ldots, r_m\}$. We introduce new symbols $r_0, \ldots, r_m$. Let N be a Turing machine that, on input x, behaves as follows: The machine N first guesses a full binary tree T having m nodes such that (i) each node is labeled by a word in $\hat{R}V^*$ and (ii) the sum of the length of the word labels is at most $f(|x|)$. The machine N then attempts to verify the following conditions for all nodes u of T.

- If u is a leaf node, then its label is of the form $r_0 w$ such that $w \in I$. If this test passes, record w as the word label of u.
- If u is a non-leaf node and y is the label of u, then y is of the form $r_i w$ satisfying the following properties:
  - $r_i \in \hat{R}$.
  - $w \in V^*$.
  - Let s be the word label of the left child of u and let t be the word label of the right child of u. Then applying r_i to (s, t) allows to cut s into s_1 and s_2 and cut into t_1 and t_2 so that $w = s_1 t_2$.
- If u is the root, then its label is of the form $r_i x$ for some $r_i \in \hat{R}$.

The machine accepts if and only if all the tests pass.

The machine N correctly accepts L. To wit, first suppose x is a member of L. Then, since $L \in$ SplSpace$[f(n)]$, there exists a production tree T of size at most $f(|x|)$. The machine N nondeterministically guesses such a tree and for that particular set of guesses N accepts. Conversely, if N on input x guesses a tree that passes all the tests for x, then x has a production tree of total size at most $f(|x|)$, and so $x \in L$. This means that for all x, $x \in L$ if and only if N accepts x.

To evaluate the running time of N, note that the part for guessing a tree description requires $O(f(n))$ and that for each node of the tree the word label of the node is scanned at most twice, for verifying the word label of the node itself and possibly for verifying the word label of the parent. This means that N can be made to run in time $O(f(n))$ and thus $L \in$ NTIME$[f(n)]$.

This proves the lemma. □

Also in the other directions we can show a general bound, namely that languages in NTIME$[f(n)]$ can be generated by a splicing system using at most SplSpace$[O(f(n)^4)]$.

Lemma 7.3.2. *For all fully time-constructible functions $f(n) = \Omega(n)$,*

$$\text{SplSpace}[O(f(n)^4)] \supseteq \text{NTIME}[(f(n)].$$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Proof. Let $L \in \text{NTIME}[f(n)]$ via an $f(n)$ time-bounded nondeterministic Turing machine M_0. Then L is accepted by an $O(f(n)^2)$ time-bounded one-tape nondeterministic Turing machine M_1. Let α be a constant such that M_1 is $\alpha f(n)^2$ time bounded. Let $g(n) = \alpha f(n)^2 + 1$.

Let $M_1 = (\Sigma, Q, \Upsilon, \delta_0, q_{ini}, q_{acc})$, where Σ is the input alphabet, Q is the state set, Υ is the work-tape alphabet, δ_0 is the transition function, q_{ini} is the initial state, and q_{acc} is the accept state. Without loss of generality, we can assume that

- the head of M_1 always moves,
- the only tape of M_1 is one-way infinite and starts from cell 1, and
- M_1 has a unique accepting configuration in which
  - the tape is blank everywhere,
  - the state is q_{acc}, and
  - the head position is at cell 1.

Consider a one-tape nondeterministic Turing machine M_2 that simulates M_1 backwards from this unique accepting configuration in the following way: Suppose that the symbol of the current head position is a and the current state is p. Let S be the set of all $(q, b, d) \in Q \times \Sigma \times \{L, R\}$ such that $\delta_0(q, b)$ contains (p, a, d).

- If $S = \emptyset$, then M_2 halts without accepting.
- If $S \neq \emptyset$, M_2 nondeterministically selects (q, b, d) from S.
  - If $d = L$, M_2 writes a b, moves the head to the right neighbor, and enters q.
  - If $d = R$, M_2 writes a b, moves the head to the left neighbor, and enters q; if the head falls off the left end of the tape M_2 halts without accepting.

This behavior of M_2 can be easily implemented as a nondeterministic transition function with the same state set Q and the same work-tape alphabet Υ. Clearly, for all x, M_2 produces the initial configuration of M_1 on input x if and only if $x \in L$. Also, for all x, if M_2 produces the initial configuration of M_1 on input x, then it can do so in at most $g(|x|)$ steps.

Let δ be the transition function of M_2 and let $\perp$ be the blank symbol of M_2, and thus, of M_1. We will construct an FIRR extended splicing system $\Gamma = (V, \Sigma, I, R)$ that produces L. Intuitively speaking, Γ simulates the moves of M_2 starting from the string $\vdash q_{acc}\% \dashv$, which encodes the initial configuration of M_2. We will design Γ so that from each word belonging to the set of initial configuration of M_1, i.e., $\vdash q_{ini}\Sigma^+\perp^*\% \dashv$, the part corresponding to Σ^+, which is a word in L, is extracted.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Specifically, we design Γ as follows. First,

$$\begin{aligned} V &= \Gamma \cup Q \\ &\cup \{@_{\tau,1}, @_{\tau,2}, @_{\tau,3}, @_{\tau,4} \mid \tau : (p, a', D) \in \delta(q, a)\} \\ &\cup \{\dashv_{L,d} \mid d \in \Gamma\} \\ &\cup \{\vdash_{L,p} \mid (p, a', L) \in \delta(q, a), d \in \Sigma\} \\ &\cup \{\vdash_{R,a'}, \dashv_{R,a'} \mid (p, a', R) \in \delta(q, a)\} \\ &\cup \{\vdash_{B,q}, @_{B,q,1}, @_{B,q,2}, @_{ini,1}, @_{ini,2}, @_{ini,3}\} \\ &\cup \{\tilde{a} \mid a \in \Sigma\} \end{aligned}$$

and

$$\begin{aligned} I &= \{\vdash q_{acc}\% \dashv, \vdash_{B,q} q\bot\%@_{B,q,1}, \vdash @_{B,q,2}, @_{ini,2}\} \\ &\cup \{\tilde{a}@_{ini,1}, a@_{ini,3} \mid a \in \Sigma\} \\ &\cup \{\vdash_{Ra'} p@_{\tau,1}, @_{\tau,2} \dashv_{R,a'}, \vdash @_{\tau,3}, @_{\tau,4} a' \dashv \\ &\qquad \mid \tau : (p, a', R) \in \delta(q, a)\} \\ &\cup \{\vdash_{L,p} a'@_{\tau,1}, @_{\tau,2} \dashv_{L,d}, \vdash pd@_{\tau,3}, @_{\tau,4} \dashv \\ &\qquad \mid \tau : (p, a', L) \in \delta(q, a), d \in \Gamma\}. \end{aligned}$$

Finally,

$$R = R_1 \cup R_2 \cup R_3 \cup R_4.$$

The four parts of R are each associated with a specific part of the simulation.

Simulating a transition with the right head move

Let τ be the transition $(p, a', R) \in \delta(q, a)$. To simulate this transition R_1 contains the following rules.

$$\begin{aligned} \vdash qa\#\Gamma^*\%\Gamma^* \dashv \quad &\$ \quad \vdash_{Ra'} p\#@_{\tau,1}, \\ \vdash_{R,a'} Q\Gamma^*\%\Gamma^*\# \dashv \quad &\$ \quad @_{\tau,2}\# \dashv_{R,a'}, \\ \vdash_{R,a'} \#Q\Gamma^*\%\Gamma^* \dashv \quad &\$ \quad \vdash \#@_{\tau,3}, \\ \vdash Q\Gamma^*\%\Gamma^*\# \dashv_{R,a'} \quad &\$ \quad @_{\tau,4}\#a' \dashv . \end{aligned}$$

On a word w of the form $\vdash qau\%v \dashv$ such that $u, v \in \Gamma^*$, these rules first replace the $\vdash qa$ at the left end with $\vdash_{R,a'} p$ and then $\dashv$ at the right end with $\dashv_{R,a'}$. Then the rules replace the two end markers with $\vdash$ and $a' \dashv$, respectively, to complete the transformation of w into $\vdash pu\%va' \dashv$.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Simulating a transition with the left head move

Let τ be a transition (p, a', L) in $\delta(q, a)$ and let d be an arbitrary symbol in Γ. To simulate τ R_2 contains the following rules:

$$\begin{aligned}
\vdash qa\#\Gamma^*\%\Gamma^*d\dashv \quad &\$ \quad \vdash_{L,p} a'\#@_{\tau,1},\\
\vdash_{L,p} a'\Gamma^*\%\Gamma^*\#d\dashv \quad &\$ \quad @_{\tau,2}\#\ \dashv_{L,d},\\
\vdash_{L,p} \#a'\Gamma^*\%\Gamma^*\dashv \quad &\$ \quad \vdash pd\#@_{\tau,3},\\
\vdash Qda'\Gamma^*\%\Gamma^*\#\ \dashv_{L,d} \quad &\$ \quad @_{\tau,4}\#\ \dashv .
\end{aligned}$$

On a word w of the form $\vdash qau\%vd\ \dashv$ such that $u, v \in \Gamma^*$, these rules first replace the $\vdash qa$ at the left end with $\vdash_{L,p} a'$ and then $\dashv$ at the right end with $\dashv_{L,d}$. Then the rules replace the two end markers with $\vdash pd'$ and $\dashv$, respectively, to complete the transformation of w into $\vdash pda'u\%v\ \dashv$.

Insertion of a blank symbol

Let $q \in Q$. Then R_3 consists of

$$\begin{aligned}
\vdash q\%\#\Gamma^*\dashv \quad &\$ \quad \vdash_{B,q} q\bot\%\#@_{B,q,1},\\
\vdash_{B,q} \#q\bot\%\Gamma^*\dashv \quad &\$ \quad \vdash\ \#@_{B,q,2}.
\end{aligned}$$

If M_2 needs to use more work tape space, these rules can extend the representation of the tape with a blank symbol. From a configuration $\vdash q\%u\ \dashv$ the rules produce $\vdash q\bot\%u\ \dashv$ via $\vdash_{B,q} q\bot\%u\ \dashv$.

Extraction of an input to M_1

For each $a \in \Sigma$, R_4 contains the following rules:

$$\begin{aligned}
\vdash q_{ini}a\#\Sigma^*\bot^*\%\ \dashv \quad &\$ \quad \tilde{a}\#@_{ini,1},\\
\tilde{a}\Sigma^*\#\bot^*\%\ \dashv \quad &\$ \quad @_{ini,2}\#\epsilon,\\
\tilde{a}\#\Sigma^* \quad &\$ \quad a\#@_{ini,3}.
\end{aligned}$$

When arriving at a configuration $\vdash q_{ini}auv\%\ \dashv$, such that $a \in \Sigma$, $u \in \Sigma^*$, and $v \in \bot^*$. these rules can extract the word au. This passes through the words $\tilde{a}uv\%\ \dashv$ and $\tilde{a}u$. The third rule may be applied also to $\tilde{a}uv\%\ \dashv$, but the word produced, $auv\%\ \dashv$, will never be spliced again.

In all derivations described above, the auxiliary strings that are used belong to I. It is not hard to see that the by-products of splicing cannot be spliced again. Thus, for all $x \in \Sigma^*$, Γ produces x if and only if $x \in L$.

To analyze the complexity of Γ, let x be a word in L having length n. Note that 4 rounds are required for simulating one step of M_2, 2 rounds are required for inserting

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

a $\perp$, and 3 rounds are required for extracting x. Since M_2 can produce the initial configuration with x as the input word in at most $g(n)$ steps, there is a production tree T of Γ in which one-step simulation of M_2 takes places at most $g(n)$ times, insertion of $\perp$ occurs at most $g(n)$ times, and extraction of x occurs once. Thus, the height of T is at most $6g(n) + 3$. In this tree every non-leaf has a child that is a leaf. This means that the total number of nodes is at most $12g(n) + 6$. Also, the word label of any node has length at most $g(n) + 4$. Therefore, the size of the tree is bounded by

$$(g(n) + 5)(12g(n) + 6) = O(g(n)^2) = O(f(n)^4).$$

Thus, $L \in \text{SplSpace}[O(f(n)^4)]$. □

Combining the last two lemmas, we obtain the following theorem.

Theorem 7.3.3. *There exists a $k \geq 1$ such that for all fully time-constructible functions $f(n) = \Omega(n)$,*

$$\text{SplSpace}[f(n)] \subseteq \text{NTIME}[O(f(n)^k)] \text{ and } \text{SplSpace}[O(f(n)^k)] \supseteq \text{NTIME}[(f(n)].$$

Define $\text{SplSpace}[\text{poly}] = \cup_{c>0}\text{SplSpace}[n^c]$ and $\text{SplSpace}[\text{exp}] = \cup_{c>0}\text{SplSpace}[2^{c \cdot n}]$. Then we have the following:

Corollary 7.3.4. *The following equalities hold:*

1. $\text{SplSpace}[\text{poly}] = \text{NP}$.
2. $\text{SplSpace}[\text{exp}] = \text{NEXPTIME}$.

Thus also our notion of space complexity for splicing systems and further explored time complexity, which allowed to obtain upper and lower bounds in terms of Turing classes. Moreover, these bounds are close together, which lead to an exact characterization of important complexity classes like NP and PSPACE in terms of splicing time and space classes.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

8 Accepting Splicing Systems as Problem Solvers

8.1 Introduction

In this chapter, we present a different look on splicing systems, namely as problem solvers. After defining the concept of an accepting splicing system we discuss how these systems can be used as problem solvers. Then we construct an accepting splicing system able to uniformly solve the satisfiability problem (SAT) in time $O(m + n)$ for a formula of length m over n variables. We also propose a uniform solution based on accepting splicing systems to the Hamiltonian path problem (HPP) that runs in time $O(n)$, where n is the number of vertices of the instance of HPP.

We define the concept of a problem solving H system starting from another concept introduced here, namely accepting splicing systems. It is rather strange that though the theory of splicing systems is mature and well developed, an accepting model based on the splicing operation has not considered so far, in spite of the fact that in practice we deal more with accepting processes than with generating ones. Here we do not consider the computational power and complexity of these accepting devices, but we propose efficient solutions - working in linear time - to two well-known NP-complete problems. Both solutions are based on accepting splicing systems with a finite initial language and a regular set of rules.

8.2 Accepting Splicing Systems as Problem Solvers

We now introduce the definitions and terminology for accepting splicing systems. An accepting splicing system is a tuple $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ where $\underline{YES}, \langle, \rangle \in V$ and $H_\Gamma = (V, A, R)$ is a splicing system. Let $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ be an accepting splicing system. We say that Γ *accepts* a word $w \in V^*$ if and only if the following condition holds:

$$\underline{YES} \in \sigma^k(A \cup \{\langle w \rangle\}) \text{ for some integer } k.$$

Thus the language accepted by Γ is defined as

$$L(\Gamma) = \{w \in V^* \mid \Gamma \text{ accepts } w\}.$$

An extended accepting splicing system $\Gamma = (V, T, A, R, \underline{YES}, \langle, \rangle)$ is defined similarly

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

as in the generating case. The language accepted by Γ is defined as $L(\Gamma) = \{w \in T^* \mid \Gamma \text{ accepts } w\}$.

The time complexity introduced in Chapter 6 for (generating) splicing systems can be easily extended to accepting splicing systems. The time complexity $\text{SplTime}_\Gamma(w)$ of a word w with respect to an accepting splicing system $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ is the minimal i such that $\underline{YES} \in \sigma^i(A \cup \{\langle w \rangle\})$.

Let $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ be an accepting splicing system; we say that Γ halts on a word $w \in V^*$ if one of the following conditions holds:

(*i*) $\underline{YES} \in \sigma_R^k(A \cup \{\langle w \rangle\})$ for some integer k,

(*ii*) $\sigma_R^k(A \cup \{\langle w \rangle\}) = \sigma_R^{k+1}(A \cup \{\langle w \rangle\})$ and $\underline{YES} \notin \sigma_R^k(A \cup \{\langle w \rangle\})$ for some integer k.

In both cases we say that Γ halts on w in k steps. We say that Γ decides the language L iff Γ halts on every word $w \in L$ such that condition (i) is satisfied.

Let $\Gamma = (V, A, R, \underline{YES}\langle, \rangle)$ be a splicing system that halts on every word in V^*; for a word $w \in V^*$ and $n \in \mathbb{N}$ we say that

$$\begin{aligned} \text{SplTime}_\Gamma(w) &= \min\{k \mid \Gamma \text{ halts on } w \text{ in } k \text{ steps}\} \\ \text{SplTime}_\Gamma(n) &= \max\{\text{SplTime}_\Gamma(w) \mid |w| = n\}. \end{aligned}$$

We say that Γ decides L in time $T(n)$ if Γ decides L and $\text{SplTime}_\Gamma(n) \leq T(n)$ for all $n \geq 1$.

We now propose a way of using accepting splicing systems as problem solvers. A possible correspondence between decision problems and languages can be done via an encoding function which transforms an instance of a given decision problem into a word, see, e.g., [17]. We say that a decision problem P is solved in time $O(f(n))$ by accepting splicing systems if there exists a family $\mathcal{H}$ of accepting H systems such that the following conditions are satisfied:

1. The encoding function of any instance p of P having size n can be computed by a deterministic Turing machine in time $O(f(n))$.

2. For each instance p of size n of the problem one can effectively construct, in time $O(f(n))$, an accepting splicing system $\Gamma(p) \in \mathcal{H}$ which decides, again in time $O(f(n))$, the word encoding the given instance. This means that the word is decided if and only if the solution to the given instance of the problem is "YES". This effective construction is called an $O(f(n))$ time solution to the considered problem.

If an accepting splicing system $\Gamma(n)$ decides the language of words encoding all instances of the same size n, then the construction of $\mathcal{H}$ is called a uniform solution. Intuitively, a solution is uniform if for problem size n, we can construct a unique H system solving all instances of size n taking the (reasonable) encoding of instance as "input".

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

8.3 A Linear-time Uniform Solution to SAT

In this section we illustrate the use of accepting splicing systems as problem solvers by showing that accepting H systems with regular sets of rules and finite initial languages can uniformly solve SAT in linear time.

A *Boolean expression* is an expression composed of variables, parentheses and the operators $\bar{\cdot}$, $\wedge$ and $\vee$. The variables can take values 0 (false) and 1 (true). An expression is satisfiable if there is some assignment of variables such that the expression is true. The *satisfiability problem*, commonly denoted as *SAT*, is to determine, given a Boolean expression, whether it is satisfiable. SAT is a well known NP-complete problem (see e.g. [31] for more details). A Boolean expression is said to be in *conjunctive normal form (CNF)* if it is of the form $E_1 \wedge E_2 \wedge ...E_k$, where each E_i, called a *clause*, is of the form $\alpha_{i1} \vee \alpha_{i2} \vee ... \vee \alpha_{ir_i}$, where each α_{ij} is a literal, that is either x or $\bar{x}$, for some variable x. Here, we assume that all boolean formulas are in CNF.

Theorem 8.3.1. *SAT can be uniformly solved in linear time by splicing systems with regular sets of rules.*

Proof. For all formulas over n variables, we construct an accepting splicing system $\Gamma = (V, A, R, \underline{YES}, \langle\!\langle_0, \rangle\!\rangle)$, where

$$\begin{aligned} V &= \{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \cup \{\langle\!\langle_0, \langle\!\langle_1, \rangle\!\rangle, X, W, \underline{YES}, \vee, \wedge, \bot, (,), \top\} \cup \\ &\quad \{[x_i = b] \mid 1 \leq i \leq n, b \in \{0,1\}\} \\ A &= \{X\#[x_i = b]\rangle\!\rangle \mid 1 \leq i \leq n, b \in \{0,1\}\} \cup \{\langle\!\langle_0 \bot, \langle\!\langle_1 \bot, \top\ \bot, W\underline{YES}\}, \end{aligned}$$

and the set of splicing rules is defined as follows:

$$\begin{aligned} (1)\quad & \{)\#\rangle\!\rangle\$X\#[x_1 = a]\rangle\!\rangle \mid a \in \{0,1\}\} \cup \\ & \{[x_i = a]\#\rangle\!\rangle\$X\#[x_{i+1} = b]\rangle\!\rangle \mid 1 \leq i \leq n-1, a, b \in \{0,1\}\}, \\ (2)\quad & \{\langle\!\langle_0(\#z\$\langle\!\langle_0\# \bot \mid z \in \{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}\} \cup \\ & \{\langle\!\langle_0 x_i\#\alpha\$\langle\!\langle_1\# \bot \mid \alpha \in V^*[x_i = 1]V^*, 1 \leq i \leq n\} \cup \\ & \{\langle\!\langle_0 \bar{x}_i\#\alpha\$\langle\!\langle_1\# \bot \mid \alpha \in V^*[x_i = 0]V^*, 1 \leq i \leq n\} \cup \\ & \{\langle\!\langle_0 x_i\#\alpha\$\langle\!\langle_0\# \bot \mid \alpha \in V^*[x_i = 0]V^*, 1 \leq i \leq n\} \cup \\ & \{\langle\!\langle_0 \bar{x}_i\#\alpha\$\langle\!\langle_0\# \bot \mid \alpha \in V^*[x_i = 1]V^*, 1 \leq i \leq n\} \cup \\ & \{\langle\!\langle_1 Y\#Z\$\langle\!\langle_1\# \bot \mid Y, Z \in \{x_i \mid 1 \leq i \leq n\} \cup \{\bar{x}_i \mid 1 \leq i \leq n\} \cup \{\vee\}\} \cup \\ & \{\langle\!\langle_1 Y\#)\$\langle\!\langle_0\# \bot \mid Y \in \{x_i \mid 1 \leq i \leq n\} \cup \{\bar{x}_i \mid 1 \leq i \leq n\}\} \cup \\ & \{\langle\!\langle_0 \vee\#(\$\langle\!\langle_0\# \bot\}, \\ (3)\quad & \{\langle\!\langle_0\#)\$\top\# \bot\}, \\ (4)\quad & \{\lambda\#\langle\!\langle_0[x_1 = b]\$W\#\underline{YES} \mid b \in \{0,1\}\}. \end{aligned}$$

Clearly, given n the splicing system Γ can be constructed in $O(n)$ time. Now, given an

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

instance of SAT over n variables, that is a formula $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ for some $m \geq 1$, we show that Γ accepts ϕ, that is generates $\underline{YES}$ on input $\langle\langle_0\phi\rangle\rangle$, if and only if ϕ is satisfiable.

We discuss how the splicing system Γ works on $\langle\langle_0\phi\rangle\rangle$, where ϕ is a word over the alphabet $\{x_1, x_2, \ldots, x_n\} \cup \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \cup \{\vee, \wedge, (,)\}$. First we assume that ϕ is satisfiable, that is there exists an assignment of variable that satisfies each clause. Let $x_i = b_i$, $b_i \in \{0, 1\}$, $1 \leq i \leq n$, be such an assignment. By using the splicing rules in the set (1) the word w is transformed into $\langle\langle_0\phi[x_1 = b_1][x_2 = b_2]\ldots[x_n = b_n]\rangle\rangle$. This process takes n splicing steps. Then, the rules of (2) remove the current leftmost symbol of the formula ϕ at every step. Since each clause is a disjunction, for each $1 \leq k \leq m$, there exists $1 \leq i_k \leq n$ such that $x_{i_k} = b_{i_k}$ satisfies the clause C_k. Moreover, we assume that x_{i_k} is the leftmost variable appearing in C_k that satisfies C_k. When x_{i_k} is the current leftmost symbol of the formula, a rule of type $\langle\langle_0 x_i\#\alpha\$\langle\langle_1\# \perp$ applies, removing $\langle\langle_0 x_{i_k}$ and replacing it by $\langle\langle_1$, which we interpret as a marker that the current clause is satisfied. The process resumes with $\langle\langle_0$ for every clause. Therefore, after $|\phi|$ splicing steps, Γ generates the word $\langle\langle_0[x_1 = b_1][x_2 = b_2]\ldots[x_n = b_n]\rangle\rangle$. In the next splicing step, by using the rule $\lambda\#\langle\langle_0[x_1 = b_1]\$W\#\underline{YES}$ in the set (4), one gets $\underline{YES}$.

On the other hand, it is easy to note that if Γ accepts a word ϕ (if this happens, then it happens in $O(n + |\phi|)$ time), then the rule in the singleton (3) can never be used in the computation of Γ on input $\langle\langle_0\phi\rangle\rangle$. It follows that every clause is satisfiable, therefore ϕ is satisfiable. From these explanations it follows that Γ accepts ϕ in linear time if and only if ϕ is satisfiable.

Now, to conclude the proof, let us argue that Γ halts on every word $\langle\langle_0\phi\rangle\rangle$ where ϕ is not satisfiable. Then for every possible assignment, there is a clause that is not satisfied by the assignment. Let C_{s_i} be the first clause of ϕ which is not satisfied by the assignment $1 \leq i \leq 2^n$. When reaching the closing bracket of C_{s_i}, rule (3) applies, introducing the symbol $\top$, after which no rule can be applied. In the worst case, $C_{s_i} = C_m$ so that $\sigma_R^k(A \cup \{\langle\langle_0\phi\rangle\rangle\}) = \sigma_R^{k+1}(A \cup \{\langle\langle_0\phi\rangle\rangle\})$ for $k \leq n + |\phi|$. □

8.4 A Linear-time Uniform Solution to HPP

The Hamiltonian path problem (HPP) is to decide whether or not a given directed graph has a Hamiltonian path. A Hamiltonian path in a directed graph is a path which contains all vertices exactly once. It is known that HPP is an NP-complete problem.

Theorem 8.4.1. *HPP can be uniformly solved in linear time by splicing systems with regular sets of rules.*

Proof. Let us consider a directed graph $G = (X, E)$, with $X = \{x_1, x_2, \ldots, x_n\}$ for which we are looking for a Hamiltonian path starting with x_1. We construct the accepting

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

splicing system $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ with

$$\begin{aligned} V &= \{x_1, x_2, \ldots, x_n\} \cup \{[1], [2], \ldots, [n]\} \cup \{(,), Y, \langle\langle, \underline{YES}, <, >\}, \\ A &= \{Yx_1[1], \underline{YES}\langle\langle\} \cup \{Yx_i[j] \mid 2 \leq i, j \leq n\}, \end{aligned}$$

and the set R defined as follows:

$$\begin{aligned} (1) \quad & \{)\# > \$Y\#x_1[1]\} \cup \{\alpha x_t[j]\# > \$Y\#x_k[j+1] \mid 1 \leq t \neq k \leq n, \\ & 1 \leq j \leq n-1, \alpha \in V^*(x_t, x_k)V^* \setminus V^*x_kV^*\}, \\ (2) \quad & \{\underline{YES}\#\langle\langle\$[n] > \#\lambda\}. \end{aligned}$$

The instance $G = (X, E)$ of HPP is encoded into the word

$$\begin{aligned} w = \; & (x_1, x_{i_1}^{(1)})(x_1, x_{i_2}^{(1)}) \ldots (x_1, x_{i_{k_i}}^{(1)})(x_2, x_{i_1}^{(2)})(x_2, x_{i_2}^{(2)}) \ldots (x_2, x_{i_{k_i}}^{(2)}) \ldots \\ & (x_n, x_{i_1}^{(n)})(x_n, x_{i_2}^{(n)}) \ldots (x_n, x_{i_{k_i}}^{(n)}), \end{aligned}$$

over V^*, where $(x_j, x_{i_1}^{(j)}), (x_j, x_{i_2}^{(j)}), \ldots, (x_j, x_{i_{k_i}}^{(j)})$ are all edges going out from the node x_j, for some $1 \leq j \leq n$. This means we have a word $< w >$ as the input of Γ.

Clearly, given n the splicing system Γ can be constructed in $O(n)$ time. The rules of (1) extend $< w >$, constructing a path starting in x_1 and appending an edge at each step, provided w contains an edge from the current node x_i to some node x_j, $j \neq i$. It is easy to note that if there exists a Hamiltonian path in G, say $x_1, x_{s_2}, x_{s_3}, \ldots, x_{s_n}$, then the word

$$< wx_1[1]x_{s_2}[2]x_{s_3}[3] \ldots x_{s_n}[n] >$$

is generated by Γ in n splicing steps, hence $\underline{YES}$ is obtained in the next splicing step. On the other hand, the only possibility to get $\underline{YES}$ is to apply the splicing rule (2) to a pair of words formed by the axiom $\underline{YES}\langle\langle$ and a word of the form $< wx_1[1]x_{s_2}[2]x_{s_3}[3] \ldots x_{s_n}[n] >$. By the form of the splicing rules in the set (1), the word $< wx_1[1]x_{s_2}[2]x_{s_3}[3] \ldots x_{s_n}[n] >$ is obtained only if $x_1, x_{s_2}, x_{s_3}, \ldots, x_{s_n}$ is a Hamiltonian path in G.

If G has no Hamiltonian path, then Γ halts on w after at most n splicing steps without generating $\underline{YES}$, which concludes the proof. □

Thus, we showed how the complexity notion of Chapter 6 opened the way to considering H systems as efficient problem solvers. This passed through the definition of an accepting variant of splicing systems, which we feel is a natural extension to the concept of a splicing system, worthy of further study. In fact, we will address some aspects of accepting splicing systems in the next chapter.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

9 Descriptional Complexity of Splicing Systems

In this chapter, the descriptional complexity of extended finite splicing systems is studied. These systems are known to generate exactly the class of regular languages. Upper and lower bounds in both directions are shown relating the size of these splicing systems to the size of their equivalent minimal nondeterministic finite automata (NFA). In addition, an accepting model of extended finite splicing systems is studied. Using this variant one can obtain systems which are more than polynomially smaller than the equivalent NFA or generating extended finite splicing system.

9.1 Introduction

Although some descriptional complexity issues of splicing systems have been previously studied, the existing work has been mainly focused on limiting a specific resource or "structural" parameter. This typically involved increasing other resources or parameters. For instance, in [63], Păun showed that extended splicing systems with regular rules are still universal with only one initial word, or with initial words of length at most 1, but that these measures cannot be simultaneously minimized. Also the *radius* (the biggest u_i for a rule $u_1\#u_2\$u_3\#u_4$) of splicing rules has been studied in the context of H systems with additional control mechanisms, e.g. [57]. Finally, in distributed H systems, the question of reducing the number of components is well studied, see for instance [15, 50], sometimes together with radius considerations [58].

While measuring specific resources or parameters provides important insights into the role or the weight which these parameters play or carry, it does not necessarily say that much about the overall description necessary to specify a particular system, nor does it always allow to realistically compare the descriptional complexity of splicing systems with other specification methods, at least not when the various description methods use different and incompatible resources or parameters. We therefore take a different approach here by studying the total size of the system. This will allow us to compare H systems to other formalisms in terms of descriptional complexity and shed some light on the behavior of splicing systems when faced with specific tasks or problems. As a first step in this direction, we will study extended finite splicing systems, i.e., systems having a finite number of rules and starting with a finite number of initial words. It is known that such systems generate exactly the class of regular languages. This means that all regular languages can be obtained by recombining

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

finite sets while applying a finite number of different rules only.

Thus, it is interesting from a descriptional complexity perspective to compare the descriptive power of such systems with the standard model for regular languages, namely deterministic and nondeterministic finite automata (henceforth abbreviated by DFA and NFA, respectively). Many succinctness results between several models describing regular languages are known in the literature and are summarized in [19]. There are, e.g., exponential trade-offs between NFA and DFA, doubly exponential trade-offs between alternating finite automata and DFA, and polynomial trade-offs between NFA with very small finite amounts of nondeterminism and DFA.

Here, we will complement the above list by investigating the relative succinctness of representations between extended finite splicing systems and NFA. We prove upper and lower bounds in both directions to show that the size needed is similar for both systems. However, we show that using an accepting model of extended finite splicing systems, one can obtain systems which are more than polynomially smaller than the equivalent NFA or generating extended finite splicing system.

9.2 Complexity Measures

Defining fair and meaningful complexity measures is not always straightforward. The descriptional complexity of finite automata has been well studied (see [19] and its references). So far, the primary complexity measure considered is the number of states. Although this measure does not always provide a complete picture, it is normally a good indicator of total size, since the number of transitions of an n-state NFA over Σ is bounded by $n^2|\Sigma|$. Moreover, it gives fair comparisons when comparing with other formalisms having states or a comparable resource (e.g. non-terminals in context-free grammars). However, since for splicing systems there does not seem to be such a resource and since we are interested in the total size of the systems, here we will also consider the number of transitions of the NFA. Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. Then we denote

$$\mathcal{Q}(M) = |Q|,$$
$$\mathcal{T}(M) = |\delta|.$$

Since the states are implicit in the description of the transitions, we define the total size of M as $\mathsf{Size}(M) = \mathcal{T}(M)$.

For splicing systems, there are several measures we could consider. For instance, one could consider the number of rules of the system. But because of its structure consisting of an initial language and a set of rules, even a system with an empty set of rules can be arbitrarily complex in terms of the size of the equivalent NFA (as complex as the finite initial language). Even if we also consider the size of A, the length of the words in A can make the equivalent NFA arbitrarily complex. Also, as we will see, the length of the rules is relevant. This is why we will measure the size of a splicing system

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

by considering both the number and the length of the rules and the initial language. Given an (A)EH(FIN) $\Gamma = (V, T, A, R)$ we define the following measures:

$$\mathcal{R}(\Gamma) = \sum_{r \in R} |r|,$$
$$\mathcal{A}(\Gamma) = \sum_{w \in A} |w|,$$
$$Q_A(\Gamma) = min\{Q(M) \mid M \text{ is an NFA and } L(M) = A\}.$$

So, for the initial language we have two measures, the total size of A and the size of the minimal automaton accepting A. The total size of Γ is defined as $\mathsf{Size}(\Gamma) = \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma)$.

9.3 Describing Regular Languages by EH(FIN) and NFA

9.3.1 From EH(FIN) to NFA

In this section, we prove upper and lower bounds for the increase in descriptional complexity when changing from an EH(FIN) to an equivalent NFA. We will first show an upper bound.

Lemma 9.3.1. *For each EH(FIN) $\Gamma = (V, T, A, R)$, there exists an equivalent NFA M such that $Q(M) \leq Q_A(\Gamma) + \mathcal{R}(\Gamma)$.*

Proof. We use a slightly improved variant of Pixton's proof of regularity [70]. The idea behind the construction is that we create an NFA equivalent to a splicing system $\Gamma = (V, A, R)$ by starting from the NFA accepting the initial language A. Then, we add the states and transitions describing the strings u_1u_4 and u_3u_2 for each rule $r = u_1\#u_2\$u_3\#u_4$ in R. This involves adding $|u_1u_4| + |u_3u_2| + 2$ states, for each $r \in R$. Then, by connecting the two parts with λ-transitions, an NFA can be constructed accepting $L(\Gamma)$. Specifically, for each path corresponding to u_1u_2 that is reachable from q_0 and from where one can reach a final state, we introduce two λ-transitions. One from the beginning of the path to the beginning of the path for u_1u_4, and one from the end of the path for u_3u_2 to the end of the path. Similarly for u_3u_4. We refer to [70] for more details. To obtain the NFA for an EH(FIN) $\Gamma = (V, T, A, R)$, we only need to remove all transitions (q_i, a, q_j) such that $a \in V - T$. Following this construction, for an extended splicing system $\Gamma = (V, T, A, R)$ the resulting NFA has at most $Q_A(\Gamma) + \sum_{r \in R}(|r| + 2) = Q_A(\Gamma) + \mathcal{R}(\Gamma) + 2|R|$ many states.

In fact, we can improve this construction by realizing that the first state of each path describing each u_1u_4 and u_3u_2 is only reached by λ-transitions and that all outgoing transitions of the last state of the path are also λ-transitions. This means that removing λ-transitions in the usual way (see e.g.[31]) and removing all superfluous states, we can eliminate those states. In this way, the resulting NFA M has $Q_A(\Gamma) +$

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

$\sum_{r\in R}(max\{0, |u_1u_4| - 1\} + max\{0, |u_3u_2| - 1\})$ states where $r = u_1\#u_2\$u_3\#u_4$. Thus $Q(M) \leq Q_A(\Gamma) + \mathcal{R}(\Gamma)$. □

Of course, for an empty set of rules, the upper bound is tight. For non-empty rules, at first sight it looks as if this construction is not very efficient and might introduce unnecessary states. However, we are able to show a lower bound which is close to this upper bound.

Lemma 9.3.2. *There is an infinite sequence of languages $L_n, n \geq 1$, such that each L_n is generated by an EH(FIN) Γ_n and each NFA accepting L_n has at least $Q_A(\Gamma_n) + \mathcal{R}(\Gamma_n) - 8$ states.*

Proof. We define $L_n = L(\Gamma_n)$, where $\Gamma_n = (V, A, R_n)$, with

- $V = \{a, b, c, d\}$,
- $A = \{cabc, dd\}$,
- $R_n = \{b\#c\$c\#a, (ab)^n\#c\$\lambda\#d\}$.

Applying the first rule repeatedly to $cabc$ yields $c(ab)^*c$. The second rule can only be applied to strings $c(ab)^jc$ such that $j \geq n$. This gives $c(ab)^jd$ and $c(ab)^jdd$ for all $j \geq n$. From this, also $c(ab)^jdc$ is obtained by the second rule. In addition, the strings c, dc, dd are produced. Thus, $L_n = \{c(ab)^ic \mid i \geq 0\} \cup \{c(ab)^jd, c(ab)^jdd, c(ab)^jdc \mid j \geq n\} \cup \{c, dc, dd\}$. The smallest NFA M_n accepting this language has $2n + 4$ states, as can be shown using the extended fooling set method of [5]. As fooling set we can take $\{(\lambda, cc), (d, d), (dd, \lambda)\} \cup \{(c(ab)^i, (ab)^{n-i}dc) \mid 0 \leq i \leq n\} \cup \{(c(ab)^{i-1}a, b(ab)^{n-i}dc) \mid 1 \leq i \leq n\}$. As an example, a minimal NFA for L_3 is shown in Figure 9.1. For all Γ_n, $Q_A(\Gamma_n) = 6$ and $\mathcal{R}(\Gamma_n) = 2n + 6$. This means $Q(M_n) = Q_A(\Gamma_n) + \mathcal{R}(\Gamma_n) - 8$. □

So far, we have only considered the measure Q_A. In fact, from the previous results the values using the measure $\mathcal{A}$ are straightforward.

Corollary 9.3.3. *For the conversion of an EH(FIN) Γ to an equivalent NFA M the following bounds hold:*

1. *Upper bound: $Q(M) \leq \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) + 1$.*
2. *Lower bound: $Q(M) \geq \mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) - 8$.*

Proof. For (I), this follows from Lemma 9.3.1 and the fact that the minimal NFA accepting A has at most $2 + \sum_{a\in A} |a| - 1 \leq \mathcal{A}(\Gamma) + 1$ states. For (II), this follows from Lemma 9.3.2 and the fact that the minimal NFA accepting A has 6 states for each n. □

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

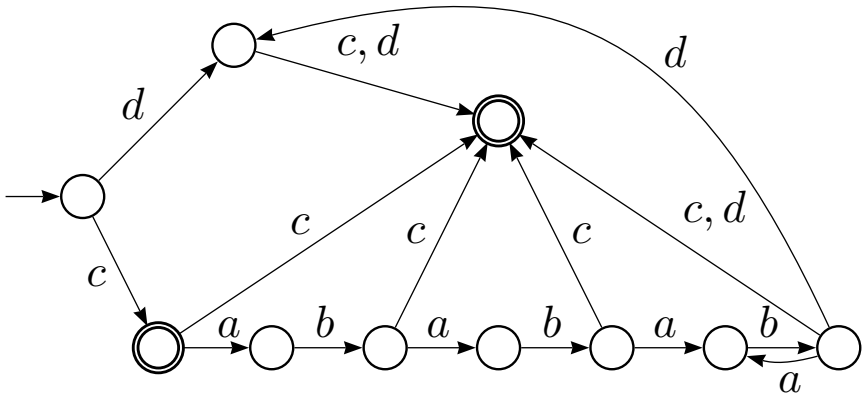

Figure 9.1: A minimal NFA accepting L_3.

9.3.2 From NFA to EH(FIN)

We now turn to the other direction, converting an NFA to an equivalent EH(FIN). Again, we show an upper and a lower bound.

Lemma 9.3.4. *For each NFA M, there exists an equivalent EH(FIN) Γ with $\mathcal{A}(\Gamma) = 5 \cdot \mathcal{T}(M) + 2$ and $\mathcal{R}(\Gamma) \leq 8 \cdot Q(M) + 4$.*

Proof. Let L be a regular language and $M = (Q, \Sigma, \delta, q_0, F)$ be a minimal NFA accepting L, where Q is the set of states, Σ the input alphabet, q_0 the initial state, F the set of final states and δ the set of transitions. We construct the extended finite splicing system $\Gamma = (V, \Sigma, A, R)$, where

- $V = \Sigma \cup Q \cup \{Z\}$, where Z is a new symbol not in $\Sigma \cup Q$,
- $A = \{ZZ\} \cup \{Zq_iaq_jZ \mid q_i, q_j \in Q, a \in \Sigma \cup \{\lambda\}, (q_i, a, q_j) \in \delta\}$,
- and R consists of the following rules:
  - $\lambda\#qZ\$Zq\#\lambda$ for all $q \in Q$,
  - $Zq_0\#\lambda\$\lambda\#ZZ$,
  - $\lambda\#q_fZ\$ZZ\#\lambda$ for all $q_f \in F$.

The initial language A contains all the words of the form Zq_iaq_jZ such that M can pass from q_i to q_j on reading a. Thus, A is the set of all valid paths of length 1. The rules of the form $\lambda\#qZ\$Zq\#\lambda$ connect two paths sharing the same state in the middle. Thus we build all words in Zq_iwq_jZ such that $\delta(q_i, w) = q_j$. The last two rules eliminate the initial state appearing at the beginning and the final state appearing at the end, obtaining

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

words in L. This is the only way to obtain words in Σ^*. It is easily verified that $\mathcal{A}(\Gamma) = 5 \cdot \mathcal{T}(M) + 2$ and $\mathcal{R}(\Gamma) = 4 \cdot (Q(M) + |F| + 1) \leq 4 \cdot (2Q(M) + 1) = 8 \cdot Q(M) + 4$. □

Lemma 9.3.5. *There is an infinite sequence of languages $L_n, n \geq 1$, such that each L_n is generated by an NFA M_n, and for each EH(FIN) Γ_n generating L_n it holds that $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) \geq \mathcal{T}(M_n)$.*

Proof. We define L_n to be $L_n = \{a^n\}$. A minimal NFA accepting L_n has $n + 1$ states and n transitions. Obviously, there exists an EH(FIN) Γ with an empty set of rules and $\mathcal{A}(\Gamma) = n$ generating L_n. We now show that no smaller EH(FIN) can exist. Assume that such a smaller EH(FIN) Γ_n exists. This means that $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) \leq n - 1$. Now, by Corollary 9.3.3 we can construct an equivalent NFA with $\mathcal{A}(\Gamma) + \mathcal{R}(\Gamma) + 1 \leq n$ states. Since we chose L_n to need $n + 1$ states, this is a contradiction. □

Finally, we can summarize the results obtained in terms of the size of the respective systems.

Theorem 9.3.6. *The following upper and lower bounds hold, for some constant c:*

1. *When converting an NFA M to an equivalent EH(FIN) Γ:*
   a) $\mathsf{Size}(\Gamma) \leq c \cdot \mathsf{Size}(M)$
   b) $\mathsf{Size}(\Gamma) \geq \mathsf{Size}(M)$
2. *When converting an EH(FIN) Γ to an equivalent NFA M:*
   a) $\mathsf{Size}(M) \leq c \cdot \mathsf{Size}(\Gamma)^2$
   b) $\mathsf{Size}(M) \geq \mathsf{Size}(\Gamma) - 9$.

Proof. Statement (I) follows directly from Lemmas 9.3.4 and 9.3.5. For (II), Lemmas 9.3.1 and 9.3.2 refer only to the number of states of M. But since $Q(M) - 1 \leq \mathcal{T}(M) \leq O(Q(M)^2)$, we obtain the given bounds. □

We leave here as an open question whether the second upper bound can be lowered to $O(n)$ or, on the contrary, the lower bound can be improved.

9.3.3 Decidability Questions

It is known that some decidability questions for NFA such as, e.g., membership or emptiness are solvable in polynomial time whereas the problems of equivalence or inclusion are known to be hard, namely PSPACE-complete [74]. It is an easy observation that EH(FIN) systems can be converted to equivalent NFA and vice versa in polynomial time. Thus, every decidability question, which is solvable for NFA in polynomial time, is solvable in polynomial time for EH(FIN) systems as well. On the other hand, problems being hard for NFA are hard for EH(FIN) systems as well. Altogether, we obtain

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Theorem 9.3.7. *The following problems are solvable in polynomial time for a given EH(FIN) system:*

1. *membership*
2. *emptiness*
3. *finiteness*

The following problems are not solvable in polynomial time for EH(FIN) systems unless P = PSPACE*:*

1. *equivalence*
2. *inclusion*
3. *universality*

9.4 Representing Regular Languages by AEH(FIN) and NFA

As mentioned in Section 9.2, accepting splicing systems are a recently introduced variant of splicing systems. In fact, in the context of biomolecular computing the accepting version seems to be more natural than the generating one. So far, little is known about the power of accepting splicing systems. It is fairly easy to show that all regular languages can be recognized by accepting splicing systems. An upper bound is harder to show, but based on the close similarity between the accepting and generating variants, we conjecture the following.

Conjecture 9.4.1. *$\mathcal{L}(AEH(FIN)) = REG$, i.e., the class of languages accepted by accepting extended splicing systems is exactly the class of regular languages.*

It is clear that any upper bound on the size when passing from an AEH(FIN) to an NFA provides a proof of the conjecture.

Turning to the descriptional complexity of AEH(FIN), we first show an upper and a lower bound for the other direction.

Lemma 9.4.2. *For each NFA M, there exists an equivalent AEH(FIN) Γ with $\mathcal{R}(\Gamma) \leq 4 \cdot \mathcal{T}(M) + 4 \cdot \mathcal{Q}(M) + 3$ and $\mathcal{A}(\Gamma) \leq 2 \cdot \mathcal{Q}(M) + 2$.*

Proof. Given an NFA $M = (Q, \Sigma, \delta, q_0, F)$, we construct the AEH(FIN) $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ with

- $V = \Sigma \cup Q \cup \{Z, \underline{YES}, \langle, \rangle\}$, where Z is a new symbol not in $\Sigma \cup Q$,
- $A = \{Z\underline{YES}\} \cup \{qZ \mid q \in Q\}$,

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- and R consists of the following rules:
  - $q_i a\#\lambda\$q_j\#Z$ for all $q_i, q_j \in Q, a \in \Sigma, (q_i, a, q_j) \in \delta$,
  - $\langle\#\lambda\$q_0\#Z$,
  - $\lambda\#q_f\rangle\$Z\#\underline{YES}$ for all $q_f \in F$.

On a given input $\langle w\rangle$, the second rule gives $q_0 w\rangle$. From this word, Γ simulates the moves of M on w using the first type of rules. If M accepts w, we get to a word $q_f\rangle$, for some $q_f \in F$. Then, the last rule can be applied to give $\underline{YES}$. $\mathcal{R}(\Gamma) = 4 \cdot \mathcal{T}(M) + 4 \cdot |F| + 3 \leq 4 \cdot \mathcal{T}(M) + 4 \cdot Q(M) + 3$ and $\mathcal{A}(\Gamma) = 2 \cdot Q(M) + 2$. □

Lemma 9.4.3. *There is an infinite sequence of languages $L_n, n \geq 1$, such that each L_n is accepted by an NFA M_n and for each AEH(FIN) Γ_n generating L_n, $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) \geq \mathcal{T}(M_n) - c$ where $c \geq 0$ is some constant.*

Proof. For the proof we use a standard incompressibility argument. General information on Kolmogorov complexity and incompressibility arguments may be found in [36]. A similar argument in the context of descriptional complexity has been given in [48].

Let L_n be a singleton of length n and Γ_n an AEH(FIN) accepting L_n. Then $C(L_n|n)$ denotes the minimal size of a program describing L_n and knowing the length n. Clearly, the size of this minimal description is lower than or equal to the size of a certain encoding $cod(\Gamma_n)$ of Γ_n and the size $|P|$ of a program P which describes how an AEH(FIN) is encoded and how an AEH(FIN) describes L_n. Obviously, $|P|$ does not depend on L_n, Γ_n and n. An encoding $cod(\Gamma_n)$ of Γ_n consists of encodings $cod(T)$, $cod(A)$, and $cod(R)$.

It is known due to Theorem 2.3. of [36] that there exists an incompressible string w_n of length n such that $C(w_n|n) \geq n$. Let $L_n = \{w_n\}$ and $c = |T| + |P|$. Now, assume by way of contradiction that there exists an AEH(FIN) Γ_n accepting L_n such that $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) < n - c$. Then,

$$\begin{aligned} C(L_n|n) &\leq |cod(\Gamma_n)| + |P| \\ &\leq |cod(A)| + |cod(R)| + |cod(T)| + |P| \\ &\leq \mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) + c \\ &< n - c + c = n. \end{aligned}$$

This is a contradiction to $C(L_n|n) \geq n$. Thus, $\mathcal{A}(\Gamma_n) + \mathcal{R}(\Gamma_n) \geq n - c$ which shows the lemma since L_n is accepted by an NFA M_n having n transitions. □

As mentioned before, we do not yet know an upper bound when passing from an AEH(FIN) to an NFA (or an EH(FIN)). At first sight, it can be expected that AEH(FIN) can be more concise than EH(FIN) in certain cases, since an EH(FIN) as a generating device must explicitly generate the whole word, whereas the AEH(FIN) as an accepting device, roughly speaking, just needs to test the "relevant parts" of the word. This

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

is illustrated by the the following very simple AEH(FIN) which accepts the language $abc\{a,b,c\}^*$.

Example 9.4.4. *Let* $\Gamma = (V, \{a,b,c\}, A, R, \underline{YES}, \langle, \rangle)$ *be an AEH(FIN) with*

- $V = \{a,b,c,\underline{YES},\langle,\rangle\}$,
- $A = \{c\underline{YES}\}$,
- $R = \{c\#\underline{YES}\$\lambda\#\langle abc\}$.

In this case, an equivalent EH(FIN) needs extra information to generate the $\{a,b,c\}^*$-part, whereas the AEH(FIN) only has to check whether the input starts with abc. Thus, an EH(FIN) needs additional elements whose size is in the order of the size of the alphabet. However, even for languages over a one-letter alphabet, we can show a significant increase in size when passing from an AEH(FIN) to an EH(FIN).

Specifically, we prove that the increase in size when passing from an AEH(FIN) to an NFA (or an EH(FIN)) cannot be bounded by any polynomial function.

We first show an auxiliary result stating that the intersection of two regular languages can be accepted by an AEH(FIN) of size $O(m+n)$, where m and n are the sizes of the AEH(FIN) for each of the languages.

Lemma 9.4.5. *For two regular languages* L_1 *and* L_2*, there exist two AEH(FIN)* Γ_1 *and* Γ_2 *accepting* L_1 *and* L_2*, such that there is an AEH(FIN)* Γ *with* $L(\Gamma) = L_1 \cap L_2$ *and* $\mathcal{A}(\Gamma) = \mathcal{A}(\Gamma_1) + \mathcal{A}(\Gamma_2) + 3$ *and* $\mathcal{R}(\Gamma) = \mathcal{R}(\Gamma_1) + \mathcal{R}(\Gamma_2) + 6$.

Proof. Let Γ_1 and Γ_2 be two AEH(FIN) constructed from the NFA for L_1 and L_2 as in Lemma 9.4.2, where $\Gamma_1 = (V_1, A_1, R_1, \underline{YES}_1, \langle, \rangle)$ and $\Gamma_2 = (V_2, A_2, R_2, \underline{YES}_2, \langle, \rangle)$. We assume without loss of generality that the state sets of the original automata are disjoint. We construct $\Gamma = (V, A, R, \underline{YES}, \langle, \rangle)$ with

- $V = V_1 \cup V_2 \cup \{\underline{YES}, Z\}$, where Z is a new symbol not in $V_1 \cup V_2$,
- $A = A_1 \cup A_2 \cup \{Z\underline{YES}Z\}$,
- $R = R_1 \cup R_2 \cup \{\underline{YES}\#Z\$\underline{YES}_1\#\lambda, Z\#\underline{YES}\$\lambda\#\underline{YES}_2\}$.

Since Γ_1 and Γ_2 are constructed as in Lemma 9.4.2, all rules involve a state symbol of the original automata. This means that no interference between the derivations of Γ_1 and Γ_2 can take place. With this in mind, it is easy to see that Γ generates $\underline{YES}$ on input $\langle w\rangle$ if and only if both Γ_1 and Γ_2 accept w and that Γ has the required size. □

It is more or less a folklore result (see for instance [80]) that for two minimal NFA M_n and M_m accepting the languages $(a^n)^*$ and $(a^m)^*$, where n and m are coprimes, any NFA accepting the intersection of these languages has at least $Q(M_n) \cdot Q(M_m)$ states. We use this fact to show the following result.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Theorem 9.4.6. *There is no polynomial function f such that for any AEH(FIN) Γ there exists an equivalent NFA M such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(M)$.*

Proof. For each $k \in \mathbb{N}$, let L^k denote the language $(a^k)^*$. Moreover, let p_i denote the ith prime number. We now define an infinite sequence of languages $L_n, n \geq 1$, where

$$L_n = \bigcap_{i=1}^{n} L^{p_i}.$$

Any NFA accepting L^k needs at least k states. By the intersection result stated above, for any NFA M_n accepting L_n

$$Q(M_n) \geq \prod_{i=1}^{n} p_i.$$

Using Lemmas 9.4.2 and 9.4.5, we can construct an AEH(FIN) Γ_n accepting L_n. Starting from the NFA for L^{p_i}, $i \geq 1$, each having p_i states and p_i transitions, we construct the equivalent AEH(FIN) using Lemma 9.4.2. Each AEH(FIN) has at most size $10p_i + 5$. We then iteratively apply the construction of Lemma 9.4.5 to obtain the AEH(FIN) Γ_n for L_n, such that

$$\mathsf{Size}(\Gamma_n) \leq \sum_{i=1}^{n}(10p_i + 14) = 10\sum_{i=1}^{n} p_i + 14n.$$

Now, assume there exists a polynomial function f such that $f(\mathsf{Size}(\Gamma)) \geq Q_M$. Then, since $\mathsf{Size}(M) \leq Q_M^2$, there is a polynomial f' such that $f'(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(M)$. Let m be the degree of the largest polynomial in f'. Then $f'(x) \leq c \cdot x^m$ for some c. Since $\sum_{i=1}^{n} p_i \leq n^3$,

$$f'(\mathsf{Size}(\Gamma_n)) \leq c \cdot (\mathsf{Size}(\Gamma_n))^m \leq c \cdot (10n^3 + 14n)^m \leq c' \cdot n^{3m}.$$

This means that $Q(M_n)$ is polynomial in n. But since $Q(M_n) \geq \prod_{i=1}^{n} p_i \geq 2^n$, this is a contradiction. So there exists no such f. □

Given the bounds proved in Section 3, this result also extends to EH(FIN).

Corollary 9.4.7. *There is no polynomial function f such that for any AEH(FIN) Γ there exists an equivalent EH(FIN) Γ' such that $f(\mathsf{Size}(\Gamma)) \geq \mathsf{Size}(\Gamma')$.*

9.5 Final Remarks

We have investigated the descriptional complexity of EH(FIN) and AEH(FIN). It turned out that EH(FIN) and NFA are not only equally powerful concerning their generative capacity, but are also nearly equally powerful concerning their descriptional complexity. Similar results have been obtained for simulating NFA by AEH(FIN). It is not

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

known whether every AEH(FIN) can be simulated by some NFA, but there is a lower bound for that trade-off which is not bounded by a polynomial.

Thus, one important open question we would like to pose is whether one can find an upper bound for the increase in size when passing from an AEH(FIN) to an NFA. This would also prove our conjecture about the power of these systems.

As mentioned in the introduction, this work should be seen only as a first step in studying descriptional complexity aspects of splicing systems. So we focused here on regular languages, while there are many types of splicing systems with more than regular power. Especially describing context-free languages using splicing systems could be of interest. Our results for AEH(FIN) suggest the possibility of significant savings also for other language classes.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

10 Conclusions and Further Research

10.1 Conclusions

This thesis studied several new aspects of splicing systems, a well-known formalism of molecular computation. There are several good reasons to consider that the splicing system is an attractive model to study. On one hand, the biochemical operation that is the basis of splicing is well understood and widely used in biochemical engineering. Moreover, there is a close match between the biochemical operation and its formal model. All this causes splicing to be a prime candidate for actual experimental implementation of theoretical algorithms. In fact, we have seen in Section 2.2 that many important advances in experimental molecular computing have been based on the splicing operation. On the other hand, the formalization is simple and elegant in its definition. Even disregarding its biochemical origin, the operation is very natural and fits in seamlessly with other operations on strings studied in theoretical computer science. Also computation using this operation is interesting since, being based solely on *recombination* of given elements, it is fundamentally different from other modes of computation, which are in most cases based on rewriting.

The work in this thesis reflects the versatility of the splicing formalism. The first part of the the thesis, comprising Chapters 3 through 5, highlights its character as formalization of a biochemical process, whereas the second part, consisting of Chapters 6 through 9, focuses on splicing as a formal language operation.

The main contribution of the first part is that it challenges the common assumption that a finite, more realistic definition of splicing systems is necessary computationally weak, and that universality can only be obtained at the price of additional unrealistic control features.

Indeed, in Chapter 3 we introduced an alternative definition of the evolution and language of a splicing system, which we can claim to have a degree of biological realism comparable to that of Head's original definition. We showed that in this definition, extended finite splicing systems can be as powerful as Turing machines. The essence of the new definition is that we allow for the possibility of *replacing* existing strings by new ones (which is possible in biochemistry) instead of only *adding* new strings. The language is defined as those strings which are created and do not disappear definitively.

Also, if we weaken these systems by imposing gradual replacement via the new notion of *delay* (the replaced strings 'survive' for some time after replacement), we can shed a new light on the limited computational power of basic finite splicing systems:

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

For arbitrary large, but finite delay splicing systems can generate non-context-free languages. Only *infinite* survival leads to systems that generate only regular languages.

This approach can be generalized, with similar results, to encompass the language definition used in time-varying H systems, which shares the property that not all strings are preserved from one splicing step to the next. Thus we can see both models as two types of *non-preserving* splicing, and add a third type combining properties of both. A useful result arising from this generalization is that in systems based on our new definition of the evolution of splicing systems we can equivalently use another more traditional definition of the language, namely as the union of all words obtained at any step.

In Chapter 4, we reexamined distributed time-varying H systems in light of the fact that their language definition alone is powerful enough for computational completeness, making the distributed architecture unnecessary. We proposed two alternative, weaker language definitions based on the usual definition of splicing systems: No strings disappear and only new strings are added. We showed that if all strings pass through the components, these systems are regular. However, if the newly created strings are passed to the next component, and the existing strings remain accessible to the current one, then we obtain computational completeness for systems with at least 4 components.

Chapter 5, we showed that allowing more than one rule to be simultaneously applied to the same string, reflecting what actually happens in biochemistry, gives rise to several possible formalizations, most of which, in its finite version, have more than the regular power of normal finite splicing systems. In fact, k, p-multiple splicing, meaning that k rules out of at most p can be applied to a string at the same time, is universal for $k = p = 2$ and for all $k \geq p \geq 3$.

The second part of the thesis explores other territory. The splicing model has been well studied with respect to its computational power. But surprisingly enough, complexity considerations have hardly been addressed, even though these are a vital aspect of a computational model, especially when it comes to using or applying the model for practical purposes. In this thesis we initiated the study of these issues, and provided many new insights into the nature of splicing systems.

In Chapters 6 and 7, we introduced a notion of time and space complexity for splicing systems. Time complexity was defined as the minimal number of rounds of rule applications necessary to generate the word. Space complexity was defined as the size of the production tree of a word. These definitions were used to define and characterize complexity classes for splicing systems. Among other results, we presented a number of exact characterizations of known Turing machine classes in terms of splicing classes:

- SplTime[poly] = PSPACE.
- SplSpace[poly] = NP.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

- SplSpace[exp] = NEXPTIME.

The constructions showed that splicing systems provide a very natural medium for divide-and-conquer strategies.

Also, as we showed in Chapter 8, these notions opened the way to regarding splicing systems as problem solvers. Using a new accepting variant of splicing systems, we showed that known NP-complete problems SAT and HPP can be solved in linear time using splicing systems.

Finally, Chapter 9 addressed the descriptional complexity of splicing systems. We defined fair measures, taking into account the total size of both the rule set and the initial language. We showed that representations of regular languages by finite extended splicing systems are approximately of the same size as the corresponding NFAs. However, using the accepting variant of accepting finite extended splicing systems savings can be obtained which are not bounded by any polynomial function.

10.2 Directions for Future Research

Since most of the chapters open new areas of research on splicing systems, a range of directions of future research deriving from this thesis can be identified. Several of these issues have been mentioned in the respective chapters. Moreover, the new features can be studied in any combination, providing another source of research directions. Especially, it makes sense to study the complexity issues introduced in the second part for the new systems introduced in the first part. Here we will not discuss in detail all these possibilities, but rather highlight a few directions of further research that we feel are particularly interesting or promising.

10.2.1 Computational Complexity of Non-preserving Systems

As finite, universal systems, non-preserving H systems are an attractive model and an obvious candidate for the study of their complexity. As a first observation, we can say that for each system with the traditional language definition, we can easily construct an equivalent (also in complexity) non-preserving system, just by adding rules to keep supplying all words in the initial language. On the other hand, we may be able to extend the construction of Theorem 6.3.5 to these systems to prove a similar upper bound. At the same time, it is not very hard to construct a non-preserving system generating a^{2^n}, which is a non-context-free language, in logarithmic time. This suggests that the class SplTime[$O(\log n)$] is likely to be an interesting class for these systems. In this respect, we note that for traditional systems with regular rules we have so far not found evidence of any non-regular languages in this class.

Also the space complexity of these systems is worthy of further study. Indeed, for traditional systems the definition of space in terms of the size of the production tree makes sense, since all intervening and intermediate words are retained. For non-preserving systems, however, we can consider the space to be 'reused' when a string

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

is replaced by another. In fact, especially the non-reflexively evolving definition represents a very pure way of recombination, where a given set of elements is only reorganized, not enlarged. In this case, the space may be more adequately expressed by all initial words used in the derivation, this is, the sum of the lengths of the leaf labels in the production tree.

10.2.2 Descriptional Complexity and the Characterization of Basic Splicing Systems

While the work in Chapter 9 yielded several very interesting results, it did in some way only scratch the surface of the descriptional complexity issues which can be considered. One obvious direction of research is extending this work to more powerful models of splicing. Here a very interesting question would be if we can define descriptional parameters which limit the computational power of complete systems to yield some smaller class. On the other hand, even in the realm of the regular languages, we actually know very little. Given a regular language, we have no way to construct a splicing systems to generate it, other than via the corresponding FA. Also we have no idea whether the resulting splicing system is minimal, or have any technique to show such properties. Looking at descriptional complexity issues in this way, it seems that these are closely related to the main long-standing open issue in splicing theory: a characterization of the family of languages generated by non-extended finite splicing systems (see [7]). Looking at these issues from both perspectives might be a good way to approach them. For instance, one might ask what languages can be generated by splicing systems with only one rule? An answer to this or similar questions is likely to give an important insight in how splicing systems work. It is probable that advances in either of the two questions will have an immediate importance for the other.

10.2.3 Accepting Splicing Systems

While accepting splicing systems are a natural and straightforward variant of generating splicing systems, it does not seem that the two types are trivially related as in many other formalisms, for instance Chomsky grammars. The specific nature of splicing systems, using two strings to generate new ones, makes that proof techniques for the generating case do not necessary carry over to the accepting case. A good example is the seemingly simple question whether accepting finite extended systems generate only regular languages or more. From the complexity bounds we know that if there exists any construction converting such a system into a generating one or an FA, it will involve a blow-up in size of the system. In general, we believe there are many interesting and challenging questions related to these systems. Of course the languages accepted by the non-extended case can be studied, a class which may well be different from the corresponding generating class, as well as reflexive accepting systems (for which we do have a characterization in the generating case). Also, here we defined these systems providing the input word in brackets. This makes sense, since in the

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

unbracketed case the system accepts a word w, it will accept all words having w as a subword (it is interesting to realize that a splicing system has no way of identifying the ends of a word). Nevertheless, the study of the unbracketed case may also be of interest from a formal language point of view.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

List of Publications

Journal Articles

1. R. Loos, An alternative definition of splicing (2006), *Theoretical Computer Science*, 358: 75–876.
2. R. Loos, Time-varying H systems revisited (2006), *Journal of Universal Computer Science*, 12:10, 1439–1463.
3. R. Loos and V. Mitrana (2007), Non-preserving splicing with delay, *International Journal of Computer Mathematics*, 84(4): 427–436.
4. R. Loos and M. Ogihara (2007), Complexity theory for splicing systems, *Theoretical Computer Science*, 386: 132–150.
5. T. Ishdorj, R. Loos and I. Petre (2007), Computational efficiency of intermolecular gene assembly, *Fundamenta Informaticae*, in press.
6. R. Loos and M. Ogihara (2007), Time and space complexity for splicing systems, submitted for journal publication.
7. R. Loos, A. Malcher and D. Wotschke (2007), Descriptional complexity of splicing systems, submitted for journal publication.
8. R. Loos, F. Manea and V. Mitrana (2007), On small, reduced and fast universal accepting networks of splicing processors, submitted for journal publication.

Articles in LNCS Volumes

1. R. Loos, C. Martín-Vide and V. Mitrana (2006), Solving SAT and HPP with Accepting Splicing Systems, *PPSN IX, Lecture Notes in Computer Science* 4193, Springer-Verlag, 771–777.
2. R. Loos (2007), On accepting networks of splicing processors of size 3, *CiE 2007, Lecture Notes in Computer Science* 4497, 497–506.
3. R. Loos and M. Ogihara (2007), Complexity theory for splicing systems, *DLT 2007, Lecture Notes in Computer Science* 4588, Springer-Verlag, 300–311.
4. R. Loos, V. Mitrana and M. Ogihara (2007), Multiple splicing, *DNA 13, Lecture Notes in Computer Science 4848*, Springer-Verlag, in press.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Articles in Refereed Conference Proceedings

1. R. Loos and V. Mitrana (2005), Non-preserving splicing with delay, *Pre-proceedings of the 11th International Meeting on DNA computing*, London, Canada, 354–363.

2. B. Nagy and R. Loos (2007), Parallelism in DNA and membrane computing, *Proceedings of Computability in Europe: Computation and Logic in the Real World*, Siena, Italy, 283–287.

3. R. Loos, A. Malcher and D. Wotschke (2007), Descriptional complexity of splicing systems, *Proceedings of the 9th International Workshop on Descriptional Complexity of Formal Systems*, High Tatras, Slovakia, 93–104.

4. T. Ishdorj, R. Loos and I. Petre (2007), Computational Efficiency of Intermolecular Gene Assembly, *Workshop on Language Theory in Biocomputing*, Kingston, Canada, to appear.

Volumes Edited

1. R. Loos, S. Fazekas and C. Martín-Vide eds. (2007), *Proceedings of the International Conference on Language and Automata Theory and Applications*, Reports of the Research Group on Mathematical Linguistics 35/07, Universitat Rovira i Virgili, Tarragona, Spain.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

Bibliography

[1] L.M. Adleman, Molecular Computation of Solutions To Combinatorial Problems, *Science*, 266: 1021–1024, 1994.

[2] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J. D. Watson, *Molecular Biology of the Cell*, 3rd ed., Garland Publishing, New York, 1994.

[3] Y. Benenson, B. Gil, U. Ben-Dor, R. Adar and E. Shapiro, An autonomous molecular computer for logical control of gene expression, *Nature*, 429: 423–429, 2004.

[4] Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh and E. Shapiro, Programmable and autonomous computing machine made of biomolecules, *Nature*, 414: 430–434, 2001.

[5] J-C. Birget, Intersection and union of regular languages and state complexity, *Information Processing Letters*, 43: 185–190, 1992.

[6] P. Bonizzoni, C. Ferretti, G. Mauri and R. Zizza, Separating some splicing models, *Information Processing Letters*, 76(6): 255–259, 2001.

[7] P. Bonizzoni and G. Mauri, Regular splicing languages and subclasses, *Theoretical Computer Science*, 340: 349–363, 2005.

[8] R. V. Book, Time-bounded grammars and their languages, *Journal of Computer and System Sciences*, 5(4): 397–429, 1971.

[9] C.S. Calude and Gh. Păun, *Computing with Cells and Atoms : An Introduction to Quantum, DNA and Membrane Computing*, Taylor & Francis, London, 2001.

[10] M. Cavaliere, N. Jonoska and N.C. Seeman, Biomolecular implementation of Computing Devices with Unbounded Memory, *DNA10, Lecture Notes in Computer Science* 3384, Springer-Verlag, 35–49, 2005.

[11] A. Condon, Automata make antisense, *Nature* 429: 351–352, 2004.

[12] K. Culik, II and T. Harju, Splicing semigroups of dominoes and DNA, *Discrete Applied Mathematics*, 31: 261–277, 1991.

[13] A. Ehrenfeucht and G. Rozenberg, Forbidding-Enforcing Systems, *Theoretical Computer Science*, 292: 611–638, 2003.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

[14] J. Dassow and Gh. Păun, *Regulated Rewriting in Formal Language Theory*, Spinger-Verlag, 1989.

[15] R. Freund and F. Freund, Test tube systems: when two tubes are enough. *Developments in Language Theory 1999*, World Scientific, 338–350, 2000.

[16] P. Frisco and C. Zandron, On variants of communicating distributed H systems, *Fundamenta Informaticae* 21: 1001–1012, 2001.

[17] M.R. Garey and D.S. Johnson, *Computers and Intractability - A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[18] A. V. Gladkiĭ, On the complexity of derivations in phase-structure grammars, *Algebra i Logika Seminar*, 3(5-6): 29–44, 1964 (in Russian).

[19] J. Goldstine, M. Kappes, C.M.R. Kintala, H. Leung, A. Malcher and D. Wotschke, Descriptional complexity of machines with limited resources, *Journal of Universal Computer Science*, 8(2): 193–234, 2002.

[20] E. Goode and D. Pixton, Splicing to the limit, in: [34], 189–201.

[21] T. Harju and M. Margenstern, Splicing systems for universal Turing machines, *DNA10, Lecture Notes in Computer Science* 3384, Springer-Verlag, 151–160, 2005.

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

[23] T. Head, Hamiltonian paths and double stranded DNA, in [59], 80–92.

[24] T. Head, Splicing systems, aqueous computing and beyond, *Unconventional Models of Computation, UMC'2K, Proceedings of the Second International Conference*, Brussels, Belgium, 13-16 December 2000. Springer-Verlag, 68–84, 2001.

[25] T. Head, X. Chen, M.J. Nichols, M. Yamamura and S. Gal, Aqueous solutions of algorithmic problems: emphasizing knights on a 3X3, *DNA7, Lecture Notes in Computer Science* 2340, Springer-Verlag, Berlin, 191–202, 2002.

[26] T. Head, X. Chen, M. Yamamura and S. Gal, Aqueous computing: a survey with an invitation to participate, *Journal of Computer Science and Technology*, 17: 672–681, 2002.

[27] T. Head, D. Pixton and E. Goode, Splicing Systems: Regularity and below, *DNA8, Lecture Notes in Computer Science* 2568, Springer-Verlag, 262–268, 2003.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

[28] T. Head, G. Rozenberg, R. Bladergroen, C.K.D. Breek, P.H.M. Lommerse and H. Spaink, Computing with DNA by operating on plasmids, *Bio Systems* 57: 87–93, 2000.

[29] J. Hartmanis, On the Succinctness of Different Representations of Languages, *SIAM Journal of Computing* 9(1): 114–120, 1980.

[30] J. Hartmanis and S. R. Mahaney, Languages simultaneously complete for one-way and two-way log-tape automata, *SIAM Journal of Computing*, 10(2): 383–390, 1981.

[31] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[32] L. Ilie and V. Mitrana, Crossing-over on languages: a formal representation of chromosome recombination, in *Grammars and Automata for String Processing*, Taylor and Francis, London, 391–401, 2003.

[33] M. Ito, Gh. Paun and S. Yu eds., *Words, Semigroups, and Transductions*, World Scientific Publishing, Singapore, 2001.

[34] N. Jonoska, Gh. Paun and G. Rozenberg eds., *Aspects of Molecular Computing, Lecture Notes in Computer Science* 2950, Springer-Verlag, 2004.

[35] R. E. Ladner and N. A. Lynch, Relativization of questions about logspace computability, *Mathematical Systems Theory*, 10(1): 19–32, 1976.

[36] M. Li and P. Vitány, *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, New York, 1993.

[37] R. Lipton, Using DNA to solve NP-complete problems, *Science*, 268: 542-545, 2005.

[38] R. Loos, An alternative definition of splicing, *Theoretical Computer Science*, 358: 75–87, 2006.

[39] R. Loos, Time-varying H systems revisited, *Journal of Universal Computer Science* 12:10, 1439–1463, 2006.

[40] R. Loos, A. Malcher and D. Wotschke, Descriptional complexity of splicing systems, *Proceedings of the 9th International Workshop on Descriptional Complexity of Formal Systems*, High Tatras, Slovakia, 93–104, 2007.

[41] R. Loos, A. Malcher and D. Wotschke, Descriptional complexity of splicing systems, submitted for journal publication, 2007.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

[42] R. Loos, C. Martín-Vide and V. Mitrana, Solving SAT and HPP with Accepting Splicing Systems, *PPSN IX, Lecture Notes in Computer Science* 4193, Springer-Verlag, 771–777, 2006.

[43] R. Loos and V. Mitrana, Non-preserving splicing with delay, *International Journal of Computer Mathematics*, 84(4): 427–436, 2007.

[44] R. Loos, V. Mitrana and M. Ogihara, Multiple splicing, *DNA 13, Lecture Notes in Computer Science*, 4848, Springer-Verlag, in press, 2007.

[45] R. Loos and M. Ogihara, Complexity theory for splicing systems, *Proceedings DLT 2007, Lecture Notes in Computer Science* 4588, Springer-Verlag, 300–311, 2007.

[46] R. Loos and M. Ogihara, Complexity theory for splicing systems, *Theoretical Computer Science* 386: 132–150, 2007.

[47] R. Loos and M. Ogihara, Time and space complexity for splicing systems, submitted for journal publication, 2007.

[48] A. Malcher, On two-way communication in cellular automata with a fixed number of cells, *Theoretical Computer Science* 330(2): 325–338, 2005.

[49] M. Margenstern and Y. Rogozhin, Time-varying distributed H systems of degree 1 generate all recursively enumerable languages, in [33], 329–340.

[50] M. Margenstern, Y. Rogozhin and S. Verlan, Time-varying distributed H systems of degree 2 can carry out parallel computations, *DNA8, Lecture Notes in Computer Science* 2568, Springer-Verlag, 326–336, 2003.

[51] M. Margenstern, Y. Rogozhin and S. Verlan, Time-varying distributed H systems with parallel computations: the problem is solved, *DNA9, Lecture Notes in Computer Science* 2943, Springer-Verlag, 48–53, 2004.

[52] V. Mitrana, Crossover systems: a generalization of splicing systems, *Journal of Automata, Languages, Combinatorics*, 2: 151–160, 1997.

[53] M. Ogihara, Relating the minimum model for DNA computation and Boolean circuits, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, 1817–1821, Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[54] M. Ogihara and A. Ray, The minimum DNA computation model and its computational power, *Unconventional Models of Computation*, 309–322, Springer, Singapore, 1998.

[55] Q. Ouyang, P. D. Kaplan, S. Liu and A. Libchaber, DNA solution of the maximal clique problem, *Science*, 278: 446–449, 1997.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

[56] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[57] A. Păun, Extended H systems with permitting contexts of small radius, *Fundamenta Informaticae*, 31: 185–193, 1997.

[58] A. Păun, On time-varying H systems, *Bulletin of the EATCS*, 67: 157–164, 1999.

[59] Gh. Păun, ed., *Computing with Bio-Molecules. Theory and Experiments*, Springer-Verlag, Singapore, 1998.

[60] Gh. Păun, DNA Computing; Distributed splicing systems, *Structures in Logic and Computer Science, Lecture Notes in Computer Science*, 1261, Springer-Verlag, 309–327, 1997.

[61] Gh. Păun, DNA computing based on splicing: universality results, *Proc. of Second Intern. Colloq. Universal Machines and Computations*, Metz, Vol I, 67–91, 1998.

[62] Gh. Păun, DNA computing: Distributed splicing systems, *Structures in Logic and Computer Science, Lecture Notes in Computer Science*, 1261, Springer-Verlag, 351-370, 1997.

[63] Gh. Păun, Regular extended H systems are computationally universal, *Journal of Automata, Languages, Combinatorics*, 1(1): 27–36, 1996.

[64] Gh. Păun, G. Rozenberg, and A. Salomaa, Computing by splicing, *Theoretical Computer Science*, 168(2): 32–336, 1996.

[65] Gh. Păun, G. Rozenberg, and A. Salomaa, *DNA Computing - New Computing Paradigms*, Springer-Verlag, Berlin, 1998.

[66] K. Rinaudo, L. Bleris, R. Maddamsetti, S. Subramanian, R. Weiss and Y. Benenson, A universal RNAi-based logic evaluator that operates in mammalian cells, *Nature Biotechnology*, to appear (available on-line), 2007.

[67] P. Rothemund, Folding DNA to create nanoscale shapes and patterns, *Nature* 440: 297–302, 2006.

[68] G. Rozenberg and A. Salomaa, *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.

[69] G. Rozenberg and H. Spaink, *DNA computing by blocking*, Theoretical Computer Science, 292: 653–665 2003.

[70] D. Pixton, Regularity of splicing languages, *Discrete Applied Mathematics*, 69:101–124, 1996.

UNIVERSITAT ROVIRA I VIRGILI
FINITE MODELS OF SPLICING AND THEIR COMPLEXITY
Remco Loos
ISBN:978-84-691-9750-9/DL:T-1250-2008

[71] J. H. Reif, Parallel molecular computation, *Proceedings of the 7th ACM Symposium on Parallel Algorithms and Architecture*, 213–223, ACM Press, New York, NY, 1995.

[72] W. Ruzzo, On uniform circuit complexity, *Journal of Computer and System Sciences*, 22:365–383, 1981.

[73] W. J. Savitch, Relationships between nondeterministic and deterministic tape complexities, *Journal of Computer and System Sciences*, 4: 77–192, 1970.

[74] L. Stockmeyer and A.R. Meyer, Word problems requiring exponential time: preliminary report, *Fifth Annual ACM Symposium on Theory of Computing*, 1–9, 1973.

[75] H. Venkateswaran, Properties that characterize LOGCFL, *Journal of Computer and System Sciences*, 43:380–404, 1991.

[76] S. Verlan, A frontier result on enhanced time-varying distributed H systems with parallel computations, *Theoretical Computer Science* 344(2-3): 226–242, 2005.

[77] S. Verlan, Communicating distributed H systems with alternating filters, in [34], 367–384.

[78] S. Verlan and M. Margenstern, *About splicing P systems with one membrane*, Fundamenta Informaticae 65, 279–290 (2005)

[79] S. Verlan and R. Zizza, 1-splicing vs. 2-splicing: Separating results, *Proceedings of WORDS'03, 4th International Conference on Combinatorics on Words, Turku, Finland, TUCS General Publication* 27, 320-331, 2003.

[80] D. Wotschke, *Descriptional Complexity*, Lecture Notes, Universität Frankfurt, 1984.