7-1-2016

Logic Circuits Based on Extended Molecular Spider Systems

Dandan Mo

Dandan Mo

Candidate

Computer Science

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Darko Stefanovic, Chairperson

Matthew R. Lakin

Shuang Luan

Milan Stojanovic

Logic Circuits Based on Extended Molecular Spider Systems

by

Dandan Mo

B.E., Shanghai Jiao Tong University, 2010

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

July, 2016

Dedication

To my parents, Chengxiong Mo and Aixiang Wang, for their support and understanding. To my husband, Bo Li, for his company and encouragement.

"The Tao in its regular course does nothing (for the sake of doing it), and so there is nothing which it does not do. " – Laozi

Acknowledgments

I would like to thank my advisor, Professor Darko Stefanovic, for his support in my research and guiding me into the field of molecular computing. I would also like to thank Professor Matthew R. Lakin who gave me many important suggestions when I developed my research ideas and polished my work. I would like to thank my lab-mate Alireza Goudarzi, for his attendance at all my rehearsal talks, and his useful tips on improving presentation skills. I am grateful for the feedback of my dissertation committee: Shuang Luan, Milan Stojanovic, Matthew R. Lakin and Darko Stefanovic.

My work at the University of New Mexico was supported by the National Science Foundation under grants CCF-1318833, CCF-1422840, and CCF-1518861. I am grateful for this financial support.

Logic Circuits Based on Extended Molecular Spider Systems

by

Dandan Mo

B.E., Shanghai Jiao Tong University, 2010

Ph.D., Computer Science, University of New Mexico, 2016

Abstract

Spatial locality brings the advantages of computation speed-up and sequence reuse to molecular computing. In particular, molecular walkers that undergo localized reactions are of interest for implementing logic computations at the nanoscale. We use molecular spider walkers to implement logic circuits. We develop an extended multi-spider model with a dynamic environment wherein signal transmission is triggered via localized reactions, and use this model to implement three basic gates (AND, OR, and NOT) and a cascading mechanism. We develop an algorithm to automatically generate the layout of the circuit. We use a kinetic Monte Carlo algorithm to simulate circuit computations, and we analyze circuit complexity: our design scales linearly with formula size and has a logarithmic time complexity.

Contents

1 Introduction **1**

2 Model Description **5**

3 Design **7**

3.1 Normal Sites and Functional Sites 7

3.2 Designs of the AND and OR Gates 9

3.3 Design of the NOT gate 10

3.4 Gate Cascades 12

3.5 Simulation Results 13

4 Model Definition **17**

4.1 Site types and transition rules 17

4.2 Model definition. 20

5 Verification **22**

5.1 Geometric Layout Generation 22

5.2 Redesign of the Gates. 24

5.3 Circuit Tree Construction. 25

5.3.1 Case 1: the root node is a NOT gate 26

5.3.2 Case 2: the root node is an AND/OR gate 27

6 Performance **29**

6.1 Complexity Analysis 29

6.2 Simulation Results 30

7 Circuit Predictability **35**

7.1 Improving Circuit Predictability 36

7.2 Method 1: Height Reduction 37

7.2.1 Analysis 39

7.3 Method 2: One-Directional Movements 42

7.3.1 Analysis 42

8 Possible Implementations **59**

8.1 Possible Implementation I 59

8.1.1 Spiders and non-alterable sites. 60

8.1.2 Alterable sites and mechanisms. 60

8.2 Possible Implementation II 62

9 Discussion and Conclusion **71**

10 Future Work **74**

References **77**

Chapter 1

Introduction

Modern computers use silicon-based chips to conduct computation, but the principles of computation can be embodied in various other kinds of physical and biological material. For example, quantum computing uses quantum mechanics to represent data and operations, neural computation uses neurons to process and propagate data, and molecular computing uses DNA and small molecules to construct computation units. The use of novel materials requires new types of computation models. These models differ from standard models such as the *Turing Machine* and the *Von Neumann architecture*. They provide new abstractions for computation, and reformulate original concepts in computer science such as "programming" and "algorithm". This thesis investigates the computation potential of a complex molecular system by using it to implement a logic circuit.

Molecular walkers are synthetic molecular machines inspired by natural biological motors. Previous studies [5, 8, 13, 14, 19, 21, 24, 28, 29, 35] have shown that walkers can move directionally and autonomously on a pre-programmed track via localized reactions. In molecular computing where all the reactants diffuse freely in a well-mixed solution, a computation unit needs to search the entire space to find another

computation unit that is designed to be reacted with, which makes computation slow. In the real implementation in lab, slow computation means to a long wait for the experimental result. Since all reactants can reach each other in a well-mixed solution, any two computation units that are not expected to react with each other should have different sequences, which makes the sequence design difficult if the computation is complicated. Spatial locality can meet the challenges of computation speed-up and sequence reuse that are faced in molecular computing when all the reactants diffuse freely in a well-mixed solution [3, 11]. Hence, a walker system with inherent spatial locality has potential to perform more complex computational tasks. We investigate the computational power of a walker system by using it to implement scalable logic circuits.

We consider a molecular *spider* system, where a spider is a type of multi-legged molecular walker. A molecular spider [17], with a rigid body and several flexible legs, moves stochastically on a surface formed by sites containing DNA segments, and can exhibit biased behavior owing to different reactions with fresh sites (catalytic cleavage) and visited sites (dissociation). We extend previous models [1, 12, 23–26] to implement three basic logic gates (AND, OR, NOT), and cascade the gates to construct logic circuits. Unlike previous models where molecular spiders exhibit biased behaviors, our model uses multiple spiders that are assumed to behave in an unbiased manner with equal transition rates to all reachable sites. Sites are divided into *normal sites*, which are immutable, and *functional sites*, which can be altered via catalytic cleavage and/or strand displacement. We can encode signals into functional sites. Signal transmission [11, 15] is triggered locally when a spider interacts with a signal-carrying site, which dynamically changes the state of the spider or of the environment. We call this extended system an *active* molecular spider system.

In our design, each variable is represented by a moving spider with two legs and one arm. The arm has two possible states, 0 or 1, representing the Boolean value

of the signal the spider carries. Each gate is represented by a layout of different sites on a 2D lattice. In a single gate, spiders with different values will take different paths from their input locations. We arrange different functional sites along different paths, such that only a spider with the correct computation result will be directed to the output location via interactions between spiders and functional sites. On reaching the output location, a spider reports the computation result, and we call it the *reporting* spider. We cascade logic gates by connecting them such that only the reporting spider leaves the upstream gate and enters the downstream gate. We design a mechanism for *exit* from gates to implement gate cascades that allow parallel evaluation. As an example, Figure 3.4 will show a logic circuit where input spiders X and Y are initially placed at the input locations of two NOT gates, and the NOT gates are connected to the same AND gate via *exit* mechanisms. Spiders move within the circuit, and the spider reaching the final output location reports the computation result.

There has been much previous work on molecular logic circuits using different implementations. Molecular circuits using *DNA Strand Displacement* (DSD) [6, 18, 22] in a well-mixed solution use a high and a low relative concentration of a species to represent Boolean values 1 and 0, or they use two separate species in a dual-rail encoding. In deoxyribozyme-based circuits [2, 7, 16, 32–34], substrates are cleaved by active DNA enzymes if they bind together. Catalytic cleavage between a modified substrate and a DNA enzyme releases a sequence with fluorescence, which leads to a fluorescence increase in the environment. Detection of a fluorescence change represents Boolean value 1. Here, we use spiders with arm state 1 or 0 to represent Boolean values, which removes potential ambiguity from result reporting because we do not need to decide whether a concentration is high or low, or whether there indeed exists a fluorescence change.

Using an extended *active* multi-spider system, while keeping the advantages re-

lated to spatial locality, our design ensures modularity, unambiguity, and scalability. We will describe the model in Chapter 2, and show how to construct logic circuits in Chapter 3, along with simulation results to assess circuit computation times. A formal definition of the model is given in Chapter 4. We discuss the verification of larger circuits in Chapter 5, where we give an algorithm that automatically generates the circuit layout. We present simulation results of circuit computations in Chapter 6, discuss circuit predictability in Chapter 7, and offer possible implementations in Chapter 8. We give conclusions and discuss current challenges in Chapter 9. In Chapter 10 we discuss directions for future work.

This dissertation contains material from a conference publication and one journal article. Chapters 2 and 3 use material from IPCAT 2015 paper [9]. Chapters 4 to 6 and 8 use material from a published article in journal *BioSystems* [10].

Chapter 2

Model Description

Our long-term goal is to realize the circuits we describe here with a physical implementation based on molecular spiders [8, 17]. Therefore, our model draws from the existing models of molecular spiders [21, 24] and extends them to describe the richer functionalities of the walkers we hope to build. In spite of these extensions, we will use the evocative term "spider" throughout.

A molecular spider has a body and three limbs, two legs and an "arm", which it can use to attach to chemical sites on a surface. There is exclusion: at most one limb can be attached to a given site at a time. Different types of sites are laid out on a square lattice, $\mathbb{Z}^2$. A set of contiguous sites can form a *track* on which the spiders can move.

We model a spider's body as a single point, and the limbs as having equal length. This leads to the following postulated "hand-over-hand" gait [21]: at any given time, exactly two limbs are attached to the surface, and they are attached to nearest-neighbor sites. We call the sites a limb has bound to the *attachment points*. When a spider limb leaves a site, we assume it can quickly reattach to the surface, so there are always two attachment points for each spider, and they are adjacent to each other.

A transition step occurs when a spider detaches one of its limbs from an attachment point $p \in \mathbb{Z}^2$, and attaches to a site $p' \in \mathbb{Z}^2$. Figure 2.1 shows a transition step of a spider where there are four *reachable* sites that the limb can potentially transit to. However, a limb might not attach to a reachable site because whether a reachable site is *available* depends on the state of the site and of the limb, which will be discussed in Chapter 3 and Chapter 4. When multiple spiders are moving on the track, one spider cannot attach to a site occupied by another spider.

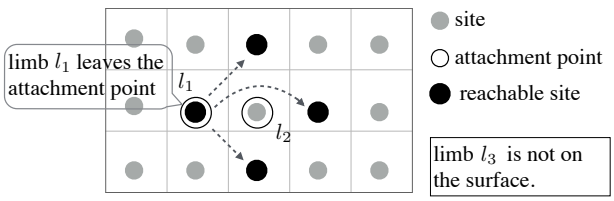


Figure 2.1: A spider has limb l_1 and limb l_2 attached to the surface. When limb l_1 detaches from the left attachment point, four sites represented by the black dots are reachable for limbs l_1 and l_3. The arrows show the transitions of a spider to other sites via *hand-over-hand* movement.

Spiders move stochastically on the track, interacting with the normal sites. If they attach to functional sites, signal transmission is triggered locally between two adjacent sites, or between a site and the spider attached to it. Changes to the sites and spiders may happen during a step, which is crucial in the operation of a logic circuit. In the next chapter, we will explain how to use different sites to construct three basic logic gates (AND, OR and NOT, thus complete for Boolean logic), and to cascade them to construct a logic circuit.

Chapter 3

Design

Each spider represents a Boolean variable. The value of the spider is indicated by its arm state, which is either 0 or 1. A logic circuit is formed by cascades comprising the basic logic gates. A logic gate is implemented as an arrangement of different sites on a square lattice, including an output location and input locations. When spiders begin moving from the input locations, their interactions with the sites lead to changes to the sites and the spider values, which ends with one spider reaching the output location. The value of that spider represents the computation result of the logic circuit.

3.1 Normal Sites and Functional Sites

We define the set of site types as $S = S_{norm} \cup S_{fun}$, where the *normal* sites $S_{norm} = \{s_l, s_1, s_0\}$ are non-alterable and the *functional* sites in S_{fun} are alterable. A normal site of type s_l binds to a spider's leg, and is used for the "wires" of a logic circuit. Sites of type s_0 and s_1 bind to the spider's arm if it has type 0 or 1, respectively. Sites of type s_0 and s_1 are typically placed at the beginning of two separate paths

that branch out from a junction, directing a spider with different values to different paths (Figure 3.1).

The junction design is used in the constructions for all gate types. Each logic gate has a set of functional sites placed on the paths branching out from the junction. After the spiders take their own paths at the junction according to their values, they will encounter different functional sites. The interactions between the spiders and the functional sites cause changes to the spider and the sites, directing one spider to the output location, to report the result of the gate computation.

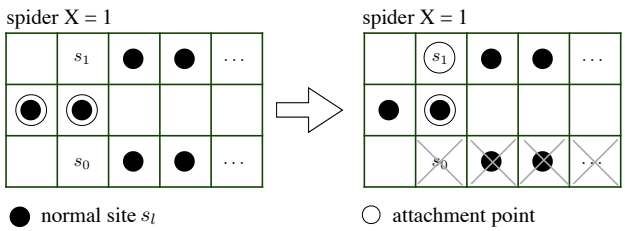


Figure 3.1: If a spider has an arm type of 1, it binds to site s_1 at a junction. If a spider has an arm type of 0, it binds to site s_0 at a junction. Here a spider $X = 1$ follows the upper path by attaching to site s_1. It cannot follow the lower path.

Before going into the details of each gate, we first introduce some important features of functional sites. (1) A functional site has a state among $\{on, off, trapped\}$. The spider can bind to an "on"-state site, cannot bind to an "off"-state site, and cannot leave a "trapped"-state site by itself. (2) A functional site may or may not trap a spider. When it traps a spider, the site's state becomes "trapped". (3) A functional site may contain a signal of "turning on/off" or "switching to 1/0". The signal held in a functional site is sent out once it is attached by a spider. When a spider attaches to a site holding a signal, the signal "turning on/off" is sent to another site, setting its state "on" or "off"; the signal "switching to 1/0" is sent to the spider, changing its value to 1 or 0. When a functional site sends out its signal, it has no signal remaining. Signal transmission is allowed between a site and

a spider that is attached to the site, or between two sites that are adjacent to each other. These features could be implemented via DNA strand displacement as we shall discuss further in Chapter 8. We will discuss the AND and OR gate designs in Section 3.2 and the NOT gate design in Section 3.3.

3.2 Designs of the AND and OR Gates

We use three types of functional sites s_t, s_p, and s_u in the designs of the AND and OR gates. Site s_t can trap the spider attaching to it, so we place a site s_t at the output location of the gate. The AND gate and OR gate each have two input spiders initially located at the two input locations, which are two junctions as shown in Figure 3.1. Each input spider selects one of two possible paths when computation begins, where one path leads to the output location without any functional sites and the other path is merged into a crossroads in the middle of the lattice. We place an initially "off"-state site s_p at the heart of the crossroads, which blocks the central path from the crossroads to the output location. We place a site s_u adjacent to site s_p, which will send a "turning-on" signal to unblock site s_p when a spider attaches to it, and trap that spider at the same time. The cooperation between sites s_u and s_p guarantees that only when both spiders meet at the crossroads can a spider take the central path to the output location.

Figure 3.2 shows the layout of the AND gate and OR gate. We now explain how the AND gate works under all four possible input assignments; the OR gate follows a similar design. In the AND gate, the two input spiders X and Y are initially placed at two junctions as their input locations. When spiders X and Y are both 0, they both take the path starting with site s_0, which leads to the output location without any functional sites. In this case, whichever spider reaches the output location will have value 0. The reporting spider reports that the result of $0 \wedge 0$ is 0. When spider

$X = 0$ and spider $Y = 1$, spider Y takes the path starting with site s_1, and becomes stuck at the crossroads because site s_p is "off". Spider X takes the path starting with site s_0, and will eventually reach the output location, reporting the result of $0 \wedge 1$ is 0. When spider $X = 1$ and spider $Y = 0$, spider X gets to the crossroads via the path starting with site s_1, and is trapped at the crossroads due to the sites s_t and s_u placed on that path. Spider Y is the only spider that can reach the output location in this case, reporting the result of $1 \wedge 0$ is 0. When both spiders are 1, they meet at the crossroads. Site s_p is turned on by the signal sent from site s_u, so spider Y can take the central path leading to the output location. Since spider X is trapped at the crossroads, only spider Y can reach the output location, reporting the result of $1 \wedge 1$ is 1. (The preceding description considers a gate in isolation. In Section 3.4, however, we will introduce gate cascades, in which a trapped spider at the output location of an upstream gate is set free and enters the downstream gate via a set of functional sites that connects the two gates.)

Following a similar design, the layout of the OR gate is shown in Figure 3.2. When both spiders are 0, they meet at the crossroads. Spider X is trapped on sites s_t and s_u, and spider Y takes the unblocked central path to the output location, reporting the result of $0 \vee 0$ is 0. Under other input assignments, the 0-valued spider takes the path to the crossroads and becomes stuck there: only the 1-valued spider can reach the output location, reporting the result of $1 \vee 0$, $0 \vee 1$, and $1 \vee 1$ is 1.

3.3 Design of the NOT gate

We use five types of functional sites in the NOT gate design. As shown in the layout of the NOT gate in Figure 3.3, site s_t which can trap a spider that attaches to it is placed on the output location. Sites $s_{1\to0}, s_r^I, s_r^{II}$ and sites $s_{0\to1}, s_r^I, s_r^{II}$ form two different *switch* mechanisms $SW_{1\to0}$ and $SW_{0\to1}$ that are laid on two separate paths.

The NOT gate has one input spider which is initially placed at a junction as the input location. Two separate paths branch out from the junction: one is taken by the 1-valued spider and contains mechanism $SW_{1\to0}$ that changes the spider value to 0, whereas the other is taken by the 0-valued spider and contains mechanism $SW_{0\to1}$ that changes the spider value to 1. When a spider moves through a *switch* mechanism, its value is switched and its backward route is cut off. We explain how mechanism $SW_{1\to0}$ works with a 1-valued spider as an example; mechanism $SW_{0\to1}$ works analogously.

Mechanism $SW_{1\to0}$ is formed by three neighboring functional sites along the horizontal direction: $s_{1\to0}, s_r^I, s_r^{II}$. We use a staging transition diagram in Figure 3.3 to describe how mechanism $SW_{1\to0}$ changes a 1-valued spider to be 0, and cuts off the backward route of the spider. A stage transition shows the change of the spider's location, value, or the site states. At stage (1), all sites are "on" initially. Site $s_{1\to0}$ can trap a spider, and contains a "switching to 0" signal that will be sent to its left site when a spider attaches to it. Therefore, when a 1-valued spider attaches to $s_{1\to0}$, it is trapped and receives the signal changing its value to 0, causing a transition to stage (2). At stage (2), since the limb trapped at site $s_{1\to0}$ cannot move back, the spider can only move forward by attaching to site s_r^I, which traps the spider and sends out a "turning off" signal to its left site. When site $s_{1\to0}$ receives that signal and turns itself "off", we get to stage (3). At stage (3), the limb trapped on s_r^I cannot move back, and the spider can only move forward by attaching to site s_r^{II}, which sends a "turning off" signal to its left site. When s_r^I receives that signal and turns itself "off", we get to stage (4). At stage (4), the limb on s_r^I can transit to a normal site on the right of s_r^{II}, while the limb on s_r^{II} cannot move back to $s_{1\to0}$ which is "off". The spider can only move forward to get to stage (5). At stage (5), sites s_r^I and s_r^{II} are "off", and the spider cannot walk back. When a spider goes through these five stages, its value is switched and its backward route is cut off. The mechanism $SW_{0\to1}$ comprising $s_{0\to1}, s_r^I, s_r^{II}$ follows similar staging transitions, the only difference

being that a 0-valued spider becomes 1 in the stage transition (1) to (2).

3.4 Gate Cascades

To construct a large logic circuit, we need to cascade logic gates of the three kinds defined in Section 3.2 and Section 3.3. A wire w connecting an upstream gate and a downstream gate is composed of contiguous normal sites s_l. To ensure that the spider that reaches the output location exits the upstream gate and never goes back to it, we place two additional sites s_r^I and s_r^{II} after site s_t on the output location, forming an *exit* mechanism which cuts off the backward route of a spider that moves through it.

The mechanism *exit* follows similar staging transitions to mechanism $SW_{1\to 0}$ shown in Figure 3.3. It consists of three neighboring functional sites along the horizontal direction: s_t, s_r^I, s_r^{II}. We explained the functionality of site s_r^I and s_r^{II} at the end of Section 3.3. Site s_t is designed to trap the spider. Therefore, a staging transition diagram for mechanism *exit* is similar to the one shown in Figure 3.3, with the only difference that the spider value is unchanged throughout the five stages. For a downstream gate with two inputs, its two input spiders may arrive at different times. Computation of the downstream gate begins when either input spider enters the gate, and the asynchronous arrival of input spiders will not influence the computation accuracy of the gate.

Figure 3.4 illustrates a simple logic circuit implemented by cascading two NOT gates as the inputs to an AND gate. The output location of each NOT gate is connected to an input location of the AND gate via the *exit* mechanism. Spider X and spider Y start to move in the two NOT gates concurrently. When the two spiders move out of the NOT gate, their backward routes are cut off due to the *exit* mechanisms, and they have their values changed to $\neg X$ and $\neg Y$. When either

spider enters the AND gate, gate computation begins, eventually yielding the result $\neg X \wedge \neg Y$.

3.5 Simulation Results

The movement of the spiders can be modeled as a continuous-time Markov process. We use a kinetic Monte Carlo algorithm to simulate gate computations. For each gate, under different assignments, we investigate the computation time using $10,000$ iterations in each simulation. We assume the transition rate (the rate at which a spider limb transits from one site to another) of each spider is 1.

Simulation results for the three basic gates AND, OR, and NOT are shown in Figure 3.5. The computation time of a logic circuit $\neg X \wedge \neg Y$ is shown in Figure 3.5 as well. Under a certain input assignment, the computation time follows a long-tailed distribution because spiders move stochastically. The computation time is the time spent on traversing the path taken by the reporting spider that reaches the output location. In all simulation runs, the output spider produced the correct output value.

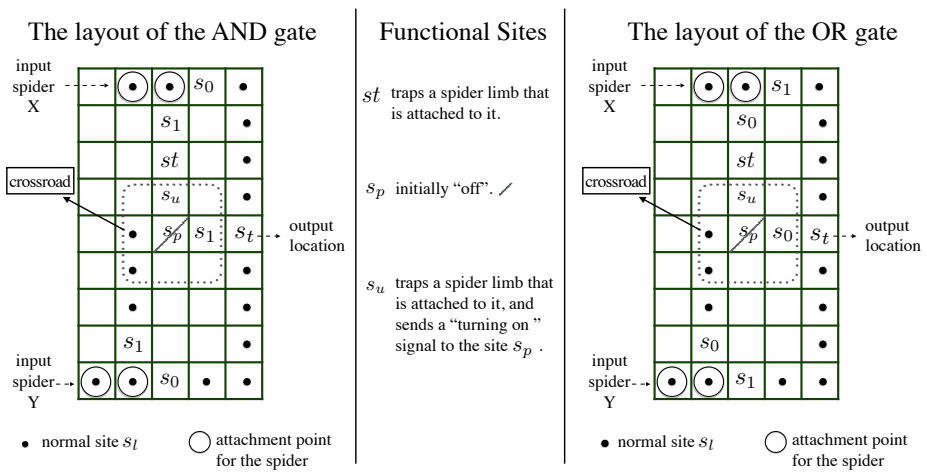


Figure 3.2: The layout of the AND gate and OR gate. Three functional sites s_t, s_p, and s_u used in the designs of these two gates are listed in the middle column. Normal site s_1 can only bind to an 1-valued spider and normal site s_0 can only bind to a 0-valued spider. In the AND gate, when both spiders are 1, they meet at the crossroads in the middle. Spider X gets trapped at sites s_t and s_u, site s_u sends a "turning-on" signal to unblock site s_p, allowing spider $Y = 1$ to take the unblocked central path from site s_p to the output location . Under other input assignments, the 1-valued spider gets stuck at the crossroads, so only the 0-valued spider can reach the output location. Therefore, the AND gate yields 1 when both spiders are assigned 1, and yields 0 in all other cases. Similarly, in the OR gate, when both spiders are 0, they meet at the crossroads in the middle and only spider $Y = 0$ can reach the output location. Under other input assignments, the 0-valued spider gets stuck at the crossroads, so only the 1-valued spider can reach the output location. Therefore, the OR gate yields 0 when both spiders are assigned 0, and yields 1 in all other cases.

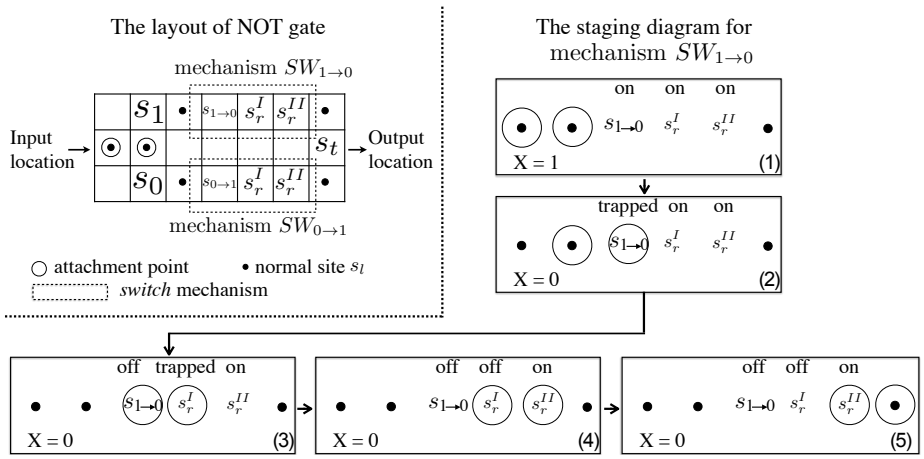


Figure 3.3: The layout of gate NOT is shown in the figure. The function of mechanism $SW_{1\to0}$ is to switch a spider's value from 1 to 0 and cuts off its backward route. We show how mechanism $SW_{1\to0}$ works in a staging transition diagram, where the spider value is expressed as X and the state of each functional site is shown above it.

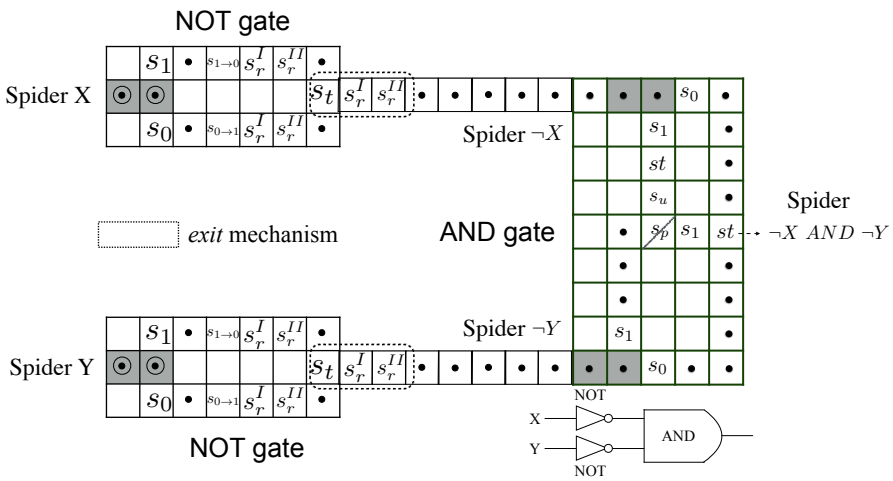


Figure 3.4: A logic circuit: $(\neg X \wedge \neg Y)$. The input locations of each gate are highlighted in grey. Spiders X and Y exit the NOT gate, becoming spider $\neg X$ and $\neg Y$ after passing through the *exit* mechanisms. The AND gate computation begins whenever a spider enters the AND gate. The spider reaching the output location of the AND gate represents the computation result $\neg X \wedge \neg Y$.

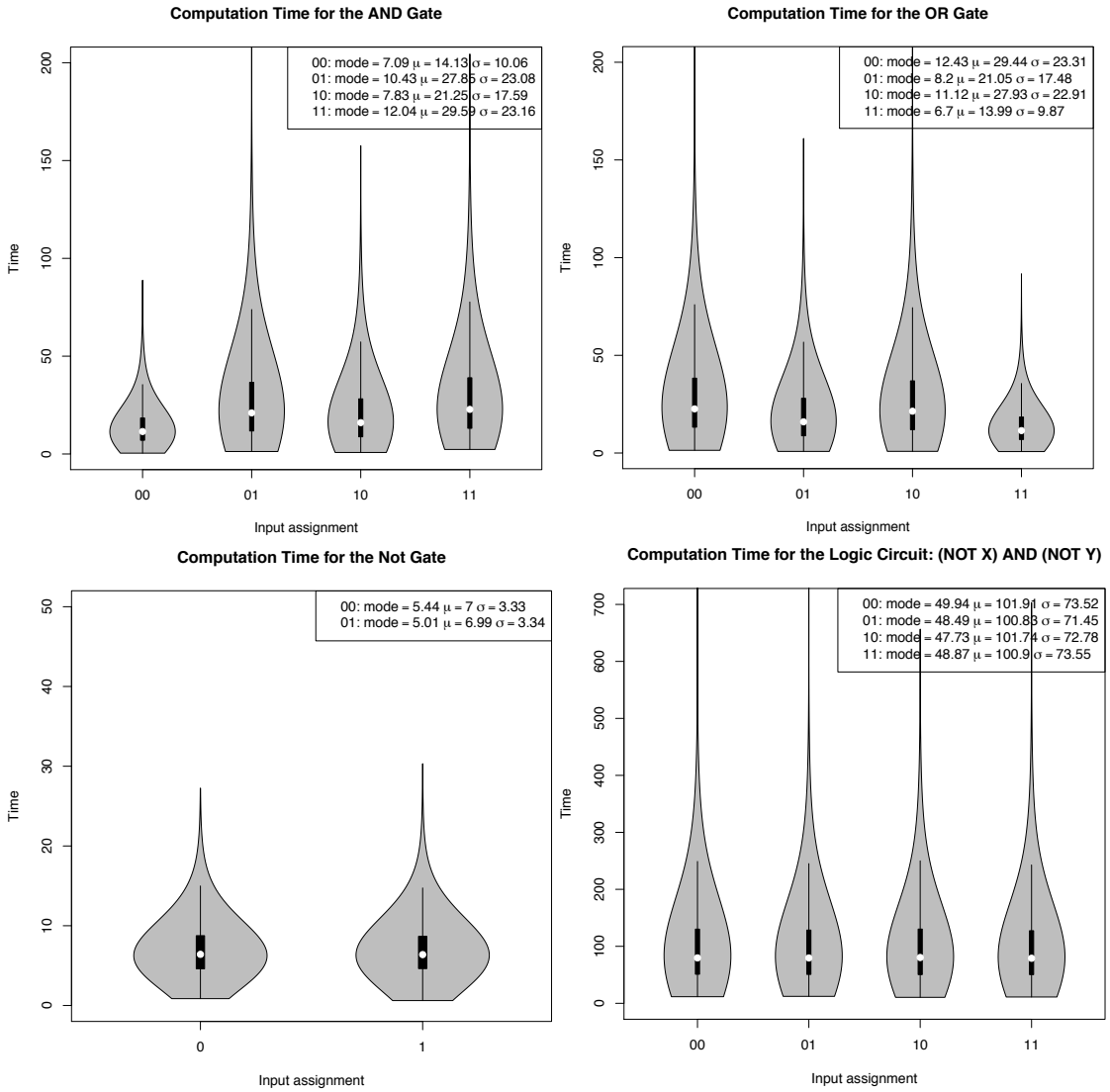


Figure 3.5: The computation time distributions for gates AND, OR and NOT under four possible input assignments are shown in the figure. Each group in one gate represents a time distribution under one assignment. The mode, mean time and standard deviation for each group is shown in the legend. The computation time distributions for the logic circuit $(\neg X \wedge \neg Y)$ in Figure 3.4 under four possible input assignments are shown in the figure. Each group of data plots is generated from $10,000$ simulation trajectories. We use the *vioplot* library in R to plot the data.

Chapter 4

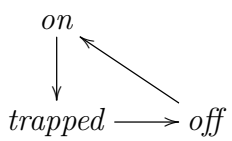
Model Definition

The molecular spider circuit proposed can be modeled as a continuous time Markov model. The environment in this model contains normal sites that are non-alterable and functional sites that are alterable via the interactions between the molecular spiders and the sites. Here we first give the definitions of the site types and transition rules of functional sites, and then the definition of the entire model.

4.1 Site types and transition rules

Sites are categorized into *normal sites* and *functional sites*. A normal site $s \in S_{norm} = \{s_l, s_0, s_1\}$ has no state. Site s_l binds to the spider's leg. Sites s_0 and s_1 bind to the spider's arm if it has type 0 or 1, respectively.

A functional site $s \in S_{fun}$ has a state which could be one of "on", "off" and "trapped". The site state transition diagram is:



A spider limb can only attach to an "on"-state site. An "off"-state site is non-alterable. The limb trapped on a "trapped"-state site cannot leave the site by itself. Whether a site can trap a spider is indicated by $TR \in \{0, 1\}$: a site with $TR = 1$ will trap a spider when a limb attaches to it. A functional site may change the spider's value, or the state of another site, by sending out a *signal* to the spider or another site. Define

$$signal = (val, d) \text{ or } null, \text{where } d \in \mathbb{Z}^2 \text{ and } val \in \{\text{on, off, trapped}, 1, 0\}. \quad (4.1)$$

Suppose a functional site is located at (x, y). If it holds a $signal = (val, (d_x, d_y))$ then it sends the signal to the location $(x + d_x, y + d_y)$, setting the state of the site located there to val. When $d = (0, 0)$, the val field of the signal is either 1 or 0, which is sent to the spider, setting the spider's value to 1 or 0.

Therefore, a functional site $s \in S_{fun}$ can be defined as

$$s = (state, TR, signal). \quad (4.2)$$

The signal held in a site is sent out once a spider limb attaches to the site. When a signal is sent out, the site has no signal remaining, which is express as $s = (state, TR, null)$. A functional site $s = (on, null)$ is equivalent to a normal site, which is non-alterable. Once a signal is received by a site or a spider, the site state or the spider's value is changed according to the signal.

In the logic circuit construction, we use two functional sites s_u and s_p in the AND gate and OR gate, and we design a set of functional sites that form different mechanisms in the NOT gate and the gate cascades. Table 4.1 gives the definitions of these functional sites and the *transition rules* applied to them. A functional site s transits to site s' in the second column, either by receiving a signal or being attached by a spider limb. If s holds a signal, it causes other changes in the last column. In table 4.1, the updated sites s' in the second column is either a normal site or a trapped site. According to the site state transition diagram, a trapped site can only transit

Table 4.1: Definitions of different functional sites used in the circuit construction and the *transition rules* applied to them. Suppose the location of the site is (x, y), define $(x', y') = (x + d_x, y + d_y)$. We use A to indicate the Boolean value of a molecular spider. The value of A is either 1 or 0.

Transition Rules		
functional site	updated site	other changes
$s_t = (\text{on}, 1, null)$	$s'_t = (\text{trapped}, 1, null)$	
$s_{1\to0} = (\text{on}, 1, (0, (0, 0)))$	$s'_{1\to0} = (\text{trapped}, 1, null)$	$A = 0$
$s_{0\to1} = (\text{on}, 1, (1, (0, 0)))$	$s'_{0\to1} = (\text{trapped}, 1, null)$	$A = 1$
$s_r^I = (\text{on}, 1, (\text{off}, d))$	$s_r^{I'} = (\text{trapped}, 1, null)$	site at (x', y') turns off
$s_r^{II} = (\text{on}, 0, (\text{off}, d))$	$s_r^{II'} = (\text{on}, 0, null) = s_l$	site at (x', y') turns off
$s_u = (\text{on}, 0, (\text{on}, d))$	$s'_u = (\text{on}, 0, null) = s_l$	site at (x', y') turns on
$s_p = (\text{off}, 0, null)$	$s'_p = (\text{on}, 0, null) = s_l$ when a "turning-on" signal is received	

to a "off"-state site that is non-alterable by itself. Since no signals are designed to turn on these "off"-state sites transited from the trapped sites, these "off"-state sites are non-alterable finally. Therefore, all the functional sites in Table 4.1 are alterable initially and become non-alterable finally. The functional sites used in our design are

$$\{s_t, s_{1\to0}, s_{0\to1}, s_r^I, s_r^{II}, s_u, s_p\},$$

where each site s among them includes its site transitions under the *transition rules* described in Table 4.1. The set of site types is $S = S_{norm} \cup S_{fun}$.

A *mechanism* is a set of neighboring mechanism sites along the same direction. We design three different mechanisms used in the logic circuit construction. The *switch* mechanism $SW_{1\to0}$ ($SW_0 \to 1$) contains sites $s_{1\to0}(s_{0\to1}), s_r^I, s_r^{II}$, where sites s_r^I, s_r^{II} contains the signal of $(\text{off}, (-1, 0))$ which can block its left site. When a spider moves over the *switch* mechanism, its value is flipped, and its backward route is cut off. The *exit* mechanism contains sites s_t, s_r^I, s_r^{II}. When a spider moves over this mechanism, its backward route is cut off.

When a spider limb leaves a site, this limb can reach 4 sites geometrically (shown in Figure 2.1). Since sites have different types, whether a site is available for a limb of a spider depends on the spider value and the site types.

Definition 1. Given a spider with value A and a reachable site, we check if the site is available by using the following conditions:

- if the site is already occupied by a limb, it is not available.
- else:
    1. if the site is a normal site:
        (a) if the site is s_l, it is available;
        (b) if the site is s_1 and $A = 1$, it is available;
        (c) if the site is s_0 and $A = 0$, it is available;
    2. else if the site is a functional site:
        (a) if the site is $s_{1\to 0}$ and $A = 1$, it is available;
        (b) if the site is $s_{0\to 1}$ and $A = 0$, it is available;
        (c) if the site is "on"-state, it is available;
    3. else, the site is not available.

Using Definition 1, every site is examined among the four reachable sites shown in Figure 2.1. Those available sites are put into a set $\mathcal{AV}$.

4.2 Model definition.

The *active* multi-spider system with normal sites and alterable sites can be modeled as a continuous-time Markov process. The state of the model is defined as

$$X = (S_1, S_2, \ldots, S_n, E), \tag{4.3}$$

where $S_i = (P_i, A_i)$ ($1 \leq i \leq n$, n is the number of spiders) describes the state of the i-th spider. Set $P_i = (p_a^i, p_b^i)$ contains attachment points for the i-th spider, and $A_i \in \{0, 1\}$ represents the Boolean value of the spider. The lattice configuration $E : \mathbb{Z}^2 \to S$ shows the layout of different sites, where S is the set of site types. Normal sites can be regarded as having state "on", $TR = 0$ and no signal, so the lattice configuration can be redefined as

$$E : \mathbb{Z}^2 \to \{\text{on, off, trapped}\} \times \{1, 0\} \times \mathbb{S},$$

where $\mathbb{S}$ represents the set of signals.

Given a model state $X = (S_1, S_2, \ldots, S_n, E)$ at time t, if a limb leaves an attachment point $p \in P_i \in S_i$, we use Definition 1 to obtain a set of available sites $\mathcal{AV}$. At time $t + \delta$, this limb transits to $p' \in \mathcal{AV}$, changing the set of attachment points to $P_i' = P_i - \{p\} \cup \{p'\}$. The *transition rules* are used to update A_i, so that the i-th spider state becomes $S_i' = (P_i', A_i')$. The transitions rules also updates E, thus the new state is

$$X' = (S_1, S_2, \ldots, S_{i-1}, S_i', S_{i+1}, \ldots, S_n, E').$$

Chapter 5

Verification

Verification is the procedure of checking if the given circuit yields the expected results under all possible input assignments. The verification of each single gate (AND, OR, NOT) discussed in previous sections checks if all spiders take the intended paths during the gate computation under all possible input assignments. Assuming all mechanisms work properly in each gate, spiders in each gate always take the intended paths under all input assignments, and only one spider can exit the gate. Therefore, all single gates are verified to work properly under all input assignments. Since our design guarantees modularity, and since all gates are verified to work properly, if all gates in a circuit are cascaded correctly and do not interfere with each other, the circuit is verified to work properly. Therefore, to verify a circuit, it remains to check if the layout of the circuit ensures correct cascading.

5.1 Geometric Layout Generation

We develop an algorithm to generate the circuit layout. Given a Boolean function, the algorithm converts the function into a circuit tree and generates a two-dimensional

layout according to the circuit tree. A Boolean function can be converted into a logic circuit composed of input variables and logic gates of AND, OR, and NOT. In our design, since all gates have one output and at most two inputs, the logic circuit (which is assumed to be feedforward) can be represented by a binary tree. Each tree node represents a gate. A connection between two nodes represents the cascade between two gates, where the output location of the upstream gate (child node) is connected to the input location of the downstream gate (parent node). Our design does not have the fan-out gate that takes one input and outputs multiple copies of the input; instead each variable occurrence is represented by a molecular spider. For a Boolean function in which a variable appears k times ($k \geq 1$), we must prepare the circuit by placing k molecular spiders, each carrying the Boolean value of that variable, at the appropriate leaf positions in the corresponding binary tree. Using the binary tree representation, the geometric layout problem of a circuit is converted to the problem of constructing a binary circuit tree that represents the given circuit such that gates do not interfere with each other. In the circuit tree construction, the layouts of the tree nodes should not have overlaps, and the layout of a connection between two nodes only joins the layout of the upstream gate at its output location with the layout of the downstream gate at its input location.

Our layout generation algorithm includes two parts: *circuit conversion* converts a given Boolean function into a circuit tree, whereas *circuit tree construction* constructs the circuit tree recursively. The circuit tree is a binary *Abstract Syntax Tree* (AST) representation of the Boolean formula. There are numerous tools that generate the AST for a given Boolean formula, so we only give details of *circuit tree construction* here, which re-designs all the gates to let them have the same size of geometric layout and constructs the circuit tree by recursively merging subtrees from the leaf nodes to the root node.

5.2 Redesign of the Gates.

Figure 5.1 shows the layouts of single gates for AND, OR, and NOT. Define $E_g : \mathbb{Z}^2 \to S$ where S is the set of site types and $g \in \{AND, OR, NOT\}$, E_g is the layout of gate g.

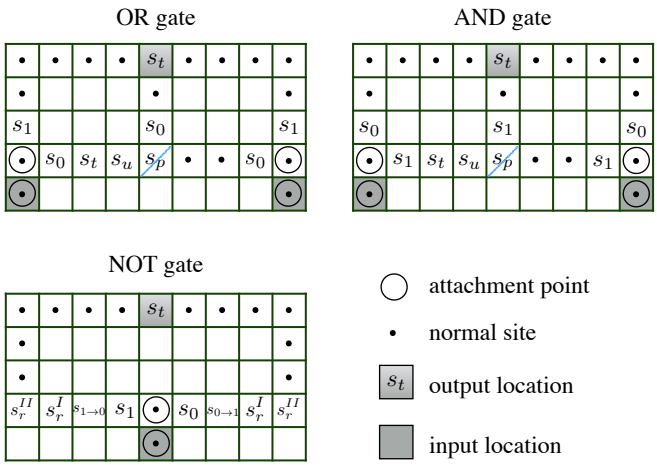


Figure 5.1: Geometric layouts of three basic gates with the same size.

A Boolean function can be expressed in a form of Boolean circuit that is composed of AND gates, OR gates, and NOT gates. Since all the gates have only one output location and at most two input locations, assuming each variable occurrence corresponds to a molecular spider, the Boolean circuit can be represented by a binary AST. A tree node represents a gate, its child nodes represent the upstream gates connected to it, and its parent node is the downstream gate it connects to. For example, a Boolean circuit $(x_1 \vee x_2) \wedge \neg x_3$ can be represented by a binary tree containing three nodes, where the root node is AND gate and it has two child nodes of OR gate and NOT gate. Each node can be mapped to its corresponding layouts shown in Figure 5.1. The child-parent relation is mapped to a contiguous set of sites connecting an upstream gate and a downstream gate. Therefore, given a Boolean circuit in a binary tree structure, the geometric layout of the circuit is converted to the

construction of the tree. Figure 5.2 shows an example of this conversion procedure.

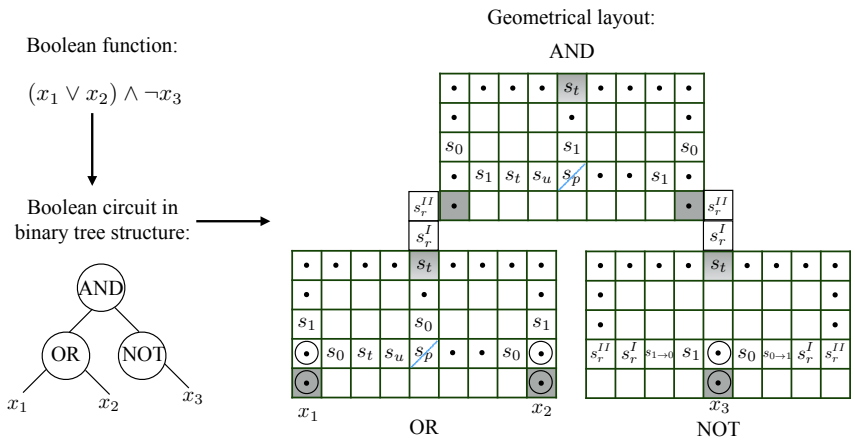


Figure 5.2: A Boolean function $(x_1 \vee x_2) \wedge \neg x_3$ can be represented by a binary tree. Each tree node is mapped to a gate layout among the layouts predefined in Figure 5.1. The geometric layout of the circuit can be converted into the construction of a binary tree using the predefined tree nodes.

5.3 Circuit Tree Construction.

Figure 5.1 has provided the layouts of all gates that will be used in the circuit construction. Given a circuit tree, the generation of the circuit layout is to recursively merge the layout of the root node with the layouts of its subtrees. The merging procedure follows a Depth-First-Search (DFS) routine. Leaf nodes representing the input variables do not have layouts. The result of merging a root node (gate) with a leaf node (variable) is the layout of the root node and additional information indicating the initial position of the variable. For example, circuit $(x_1 \vee x_2) \wedge \neg x_3$ in Figure 5.2 is the result of merging the layouts of NOT gate (right subtree) and OR gate (left subtree) with the layouts of downstream gate AND (root node). The layout of the merged circuit $(x_1 \vee x_2) \wedge \neg x_3$ includes the cascading paths between gate AND and gates NOT, OR. Initial positions of variables x_1, x_2 and x_3 are also

determined. We discuss the merging procedure between a root node (R) and its subtrees (left subtree $Left$ and right subtree $Right$) in two cases.

5.3.1 Case 1: the root node is a NOT gate

Suppose the NOT gate only has the right subtree. If $Right$ is a leaf node representing a variable, we place that variable at the input location of the NOT gate and return the layout of the NOT gate. Otherwise we create a new layout $E_{merged} : \mathbb{Z}^2 \to S$ containing the layouts of the root node E_R and the right subtree E_{Right}. In the merged layout, as shown in Figure 5.3, the subtree tree is placed below the root node and they keep unit distance to each other vertically. Cascading path between the output location of the subtree and the input location of the root node is constructed using the normal sites. An *exit* mechanism consisting of sites s_t, s_r^I and s_r^{II} is placed on the cascading path connecting the output location of the subtree circuit and the input location of the root node.

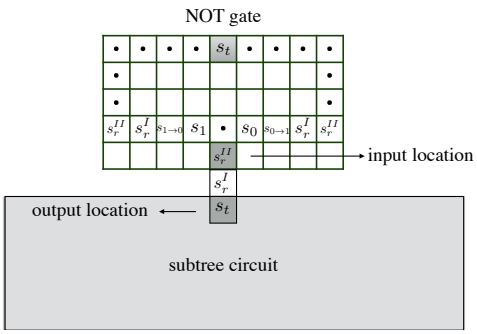


Figure 5.3: Merging the layout of a NOT gate with its subtree circuit.

5.3.2 Case 2: the root node is an AND/OR gate

If two subtrees are both variables, we return the layout of the root node gate and place the variables at the input locations of the root node gate. If one of the subtree circuits is a variable and the other is a circuit, we use the merging procedure shown in Figure 5.3 to place the subtree circuit below the root node gate and initialize the position of that variable. If both subtrees are circuits, we create a new layout $E_{merged} : \mathbb{Z}^2 \rightarrow S$ containing the layouts of the root node E_R and the subtrees E_{Left} and E_{Right}. In the merged layout, the top of the left subtree is aligned horizontally with the top of right subtree. The two subtrees keep unit distance to each other horizontally. The root node gate is placed in the middle between the output locations of subtrees $Left$ and $Right$. The root node gate keeps unit distance vertically to its subtrees. Figure 5.4 shows the procedure of merging a root node gate with its two subtree circuits. Two cascading paths containing the *exit* mechanisms connect the output locations of the subtree circuit and the input locations of the root node gate.

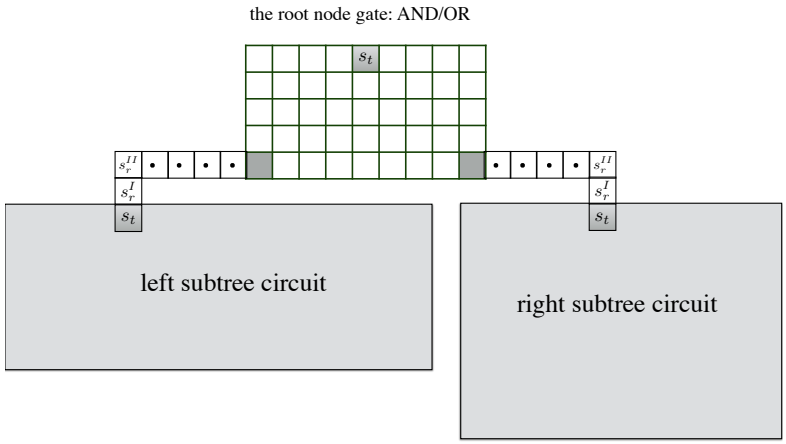


Figure 5.4: Merging the layout of an AND/OR gate with its subtree circuits.

Given a circuit tree that is represented by a binary AST, the procedure of the circuit tree construction ensures that every node in the tree is cascaded correctly with its subtrees. For each node of the tree, its layout and the layouts of its subtrees do

not have overlaps, thus all the gates in the tree do not influence each other. Since all single gates are verified to work properly under all input assignments, and since the circuit layout generation algorithm guarantees the correct cascades between different gates in the circuit, the entire circuit generated using our algorithm is verified to work properly under all input assignments.

Chapter 6

Performance

The performance of a circuit is usually measured by its computation speed and computation correctness, and the ability to accommodate the increasing size of the circuit (the number of input variables or the circuit depth). In this section, we analyze the complexity of our design and present some simulation results for a set of example circuits.

6.1 Complexity Analysis

In a single gate, the computation time t_{gate} is the traversal time of the spider that reaches the output location. Since the spider moves on the track stochastically, the computation time t_{gate} is a random variable following a long-tailed distribution, as shown in Figure 3.5.

When a spider leaves a gate or enters a gate, its backward route is cut off due to the functionality of the *exit* mechanism, so the computation time of a single gate t_{gate} can be used to estimate the computation time t of a circuit. For any n-variable Boolean function, it can be transformed into 3-CNF, which is a conjunction of m

clauses, each a disjunction of at most three literals. Since the circuit design allows parallel evaluation, for a clause $m_i = (l_1^i \vee l_2^i \vee l_3^i)$, the computation time of m_i is

$$t_{m_i} \leq 2 \times (t_{\text{OR}} + t_{\text{NOT}}) = O(1).$$

Since each clause needs time t_{m_i}, to evaluate m clauses in parallel, we need $\log m$ AND gate computations that cost $t_{\text{AND}} \times \log m$, which totally use time

$$t = t_{\text{AND}} \times \log m + t_{m_i} = O(\log m).$$

For any Boolean function in 3-CNF with m clauses, there are at most $3m$ spiders representing the literals. For each clause, at most three NOT gates and two OR gates are needed if all the literals are the negation of a variable, which is a constant number. For m clauses, there need to be $m - 1$ AND gates. Therefore, the total space complexity is $O(m)$. Hence, the circuit design is scalable because circuit size in the design scales linearly with formula size, and evaluation time is logarithmic in the formula size.

6.2 Simulation Results

Here we look at how the computation time varies with the size of the circuit on a set of examples. We investigate a small set of circuits that are in the form of 3-CNF where there are no NOT gates. We first investigate a single clause $(x_1 \vee x_2 \vee x_3)$. Computation time distributions of this single-clause circuit under all possible input assignments are shown as the violin plots in Figure 6.2. Each group of plots in Figure 6.2 represents the computation time distribution under one input assignment. The results show that computation time is influenced by the input assignment for a fixed circuit. Different input assignments lead to different computation paths for the

input spiders, where each computation path starts from the initial location of the spider and ends at the output location of the circuit. A spider taking the shorter path is more likely to be the *reporting* spider to reach the output location, which terminates the computation. Therefore, computation time is the time which the *reporting* spider takes to terminate the computation. Since different input assignments lead to different computation paths of the *reporting* spider, and different lengths lead to different computation time, computation time for a fixed circuit varies with different input assignments. For the single-clause circuit $(x_1 \vee x_2 \vee x_3)$ whose circuit structure is shown in Figure 6.1, there are two lengths of computation paths of the *reporting* spider. When $x_3 = 1$, spider x_3 has a shortcut to the output location (indicated by the dotted arrow); otherwise spider x_3 would wait in the root OR gate until either spider x_1 or spider x_2 takes the path indicated by the solid arrows to participate in the root-node gate computation. When $x_3 = 1$, spider x_3 is more likely to be the *reporting* spider. The right path taken by spider x_3 is relatively shorter, so the computation time is relatively shorter. In the results shown in Figure 6.2, four groups of plots with $x_3 = 1$ are similar to each other, and the mean time and standard deviations of these four groups are smaller compared with other four groups because x_3 is more likely to be the *reporting* spider and walks through a shorter path. When $x_3 = 0$, the *reporting* spider is more likely to take a longer path passing two gates, so the mean computation time and standard deviation are greater, as shown in the groups 000, 010, 100, and 110 of Figure 6.2.

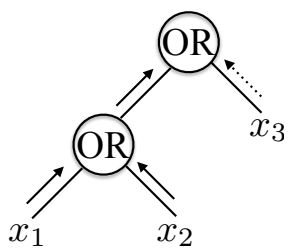


Figure 6.1: The circuit structure of $(x_1 \vee x_2 \vee x_3)$. In our design the maximum number of fan-in of a single gate is 2, and the fan-out is 1 for all gates.

Table 6.1: Example circuits evaluated in this section.

no.	(k, h)	Circuit Structure
1.	$(1, 1)$	$x_1 \vee x_2 \vee x_3$
2.	$(2, 2)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$
3.	$(3, 3)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$
4.	$(4, 4)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$
5.	$(4, 3)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)) \wedge ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4))$
6.	$(5, 5)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5)$
7.	$(5, 4)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)) \wedge ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5))$

Assigning 1 to all the variables, we investigate a group of circuits in the form of 3-CNF where the number of clauses of the circuit is from 1 to 5. Figure 6.3 shows the simulation results. We use (k, h) to represent a circuit having k clauses and height h. Table 6.1 lists 7 different circuit structures that we will simulate. The results in Figure 6.3 show that as the circuit size (the number of clauses) increases, the mode, mean computation time, and standard deviation increase. Having fixed the size of the circuit, the mode, mean computation time, and standard deviation increase as the circuit height increases, which is reflected in the groups $(4, 4)$ and $(4, 3)$, and the $(5, 5)$ and $(5, 4)$ groups shown in Figure 6.3.

The result of complexity analysis shows that our circuit design scales linearly as the circuit size increases. We use simulation to investigate what factors influence circuit performance, and we find that different input assignments lead to different computation time distributions for a fixed circuit, and the mode, mean computation time, and standard deviation increase as the circuit size and circuit height increase when we fix the input assignment.

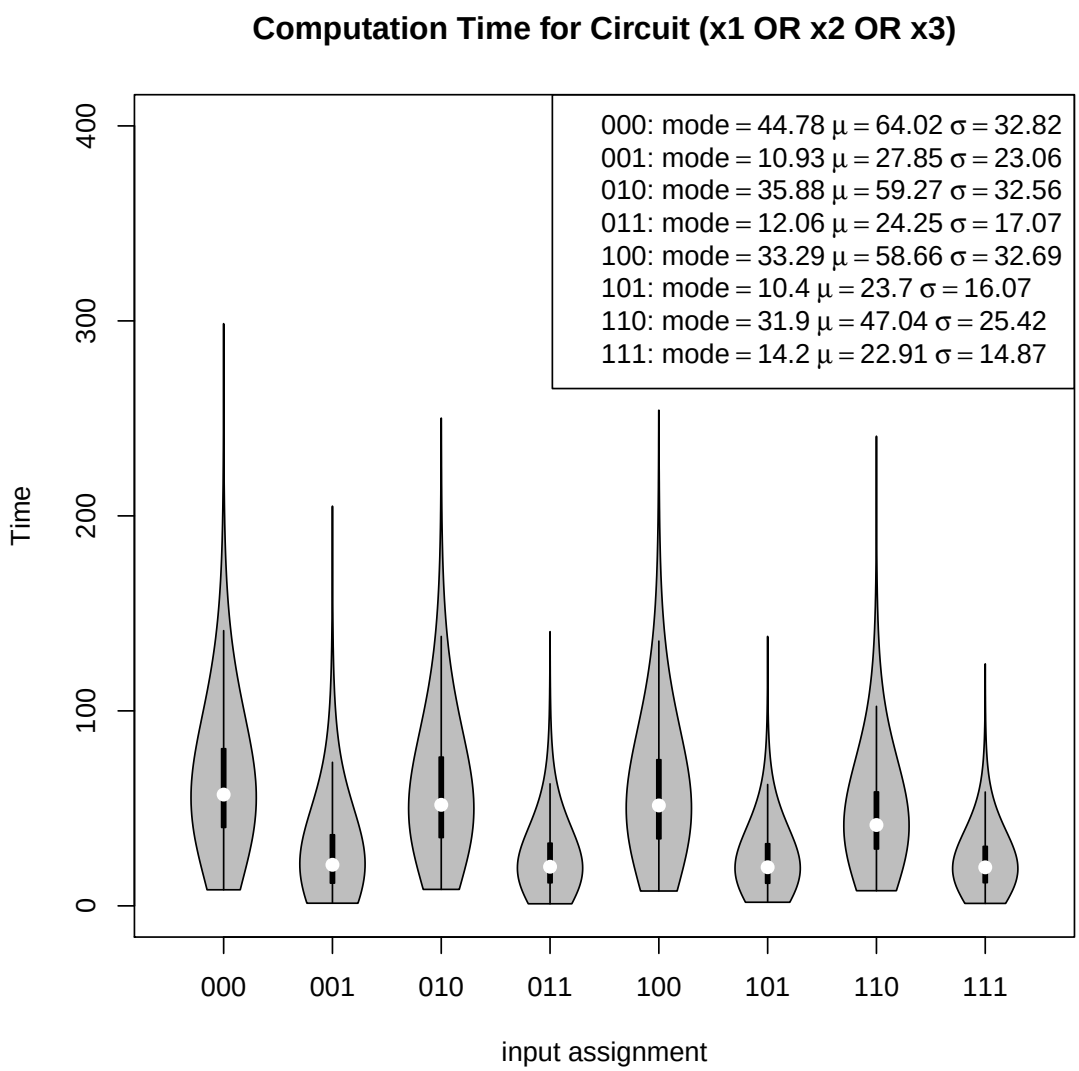


Figure 6.2: Simulation results of circuit $(x_1 \vee x_2 \vee x_3)$ under all input assignments. Each group of plots represents the computation time of the circuit under one input assignment. Input assignments for variables x_1, x_2 and x_3 are represented by binary strings as shown in the legend. For example, 101 represents the input assignment $x_1 = 1, x_2 = 0, x_3 = 1$. The legend shows the mode, mean and standard deviation for each group. Each group of data plots is generated from 5000 simulation trajectories. We use the *vioplot* library in R to plot the data.

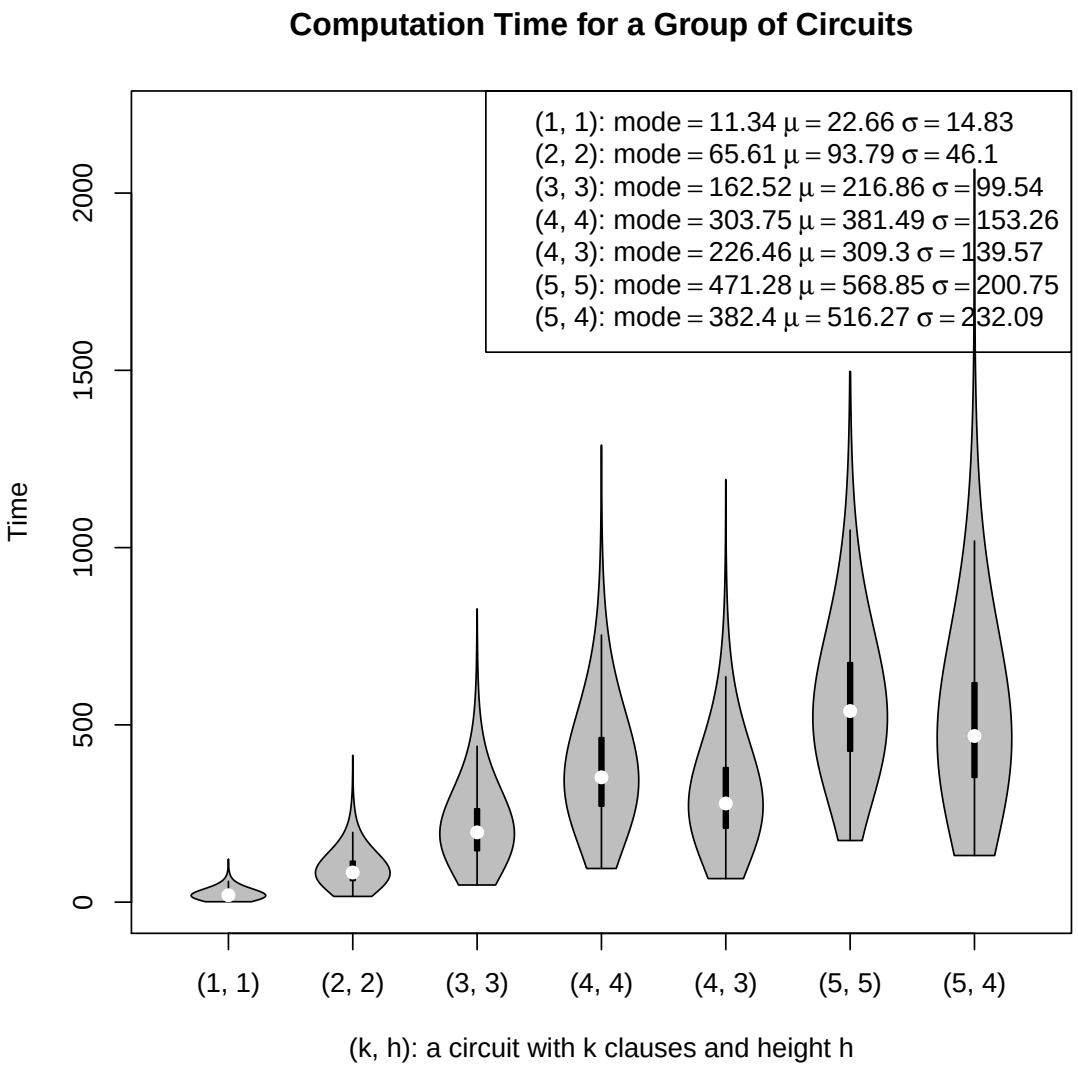


Figure 6.3: Simulation results of a group of circuits having clause number from 1 to 5 under the input assignments where all variables are 1. In the legend, each group (k, h) represents the computation time distribution of a circuit having k clauses and height h. We show the mode, mean time and standard deviation for each group in the legend. Each group of data plots is generated from 5000 simulation trajectories. We use the *vioplot* library in R to plot the data.

Chapter 7

Circuit Predictability

In Section 3.5 and chapter 6, we have shown simulation results for each single logic gate (AND, OR, NOT) and a group of circuits that are in 3-CNF (the clause number of these circuits is from 1 to 5). Simulation results show the computation time of a circuit follows a long-tailed distribution. As circuit size increases[1], computation time variance increases, making it hard to predict circuit performance. Therefore, we should consider circuit predictability as an important measure to evaluate the circuit performance using the extended molecular spider system besides computation time and computation correctness. For a circuit under a certain input assignment, the smaller standard deviation of computation time σ indicates it is easier to predict the circuit. Simulation results in Chapter 6 show that different input assignments would lead to different computation time, thus leading to different time deviations. For a circuit having n variables, there are $m = 2^n$ possible input assignments, and m different time deviations. Denote the maximum time deviation by σ_{max}, we use σ_{max} to measure the circuit predictability.

[1]Here we investigate the circuit in the form of 3-CNF, so the circuit size refers to the clause number k.

7.1 Improving Circuit Predictability

Improving circuit predictability means reducing σ_{max}. As discussed in Section 6.2, computation time is the time which the *reporting* spider takes to terminate the computation. Since molecular spiders move stochastically back and forth in the extended system, the less time they spend on walking backward the smaller time deviation will be obtained. To increase the circuit predictability, we need to reduce the time that the *reporting* spider spends on the backward route under all input assignments. To this end, we can shorten the lengths of all computation paths or change the behaviors of the molecular spiders to improve the circuit predictability. We propose two methods accordingly. **Method 1** reduces the height of the circuit tree. In this way, we believe the lengths of all computation paths can be shortened generally, and circuit predictability can be improved. **Method 2** makes molecular spiders move one-directionally on the preprogrammed track, thus the time spent on the backward route is eliminated and the time deviations under all input assignments are decreased.

We will show whether **Method 1** and **Method 2** are valid in Sections 7.2 and 7.3 via simulations. For convenience, the circuits being investigated are cascaded in the form of 3-CNF without negation. We will simulate a group of circuits having between 1 and 7 clauses. For each circuit, there are different heights of circuit trees. We will calculate σ_{max} for different heights of circuit trees. In Section 7.2, we will present the relation between the circuit height and the circuit predictability, and judge whether **Method 1** is valid. In Section 7.3, we will compare the computation time deviations between two models, the original model where molecular spiders move bi-directionally and the modified model where molecular spiders only move forward. We call the modified model *One-directional* model, and we will judge whether **Method 2** can improve the circuit predictability.

For a circuit having k clauses, the possible height h of the circuit tree is

$$\lceil \log k \rceil + 1 \leq h \leq k, \tag{7.1}$$

so there are $k - \lceil \log k \rceil$ different circuit heights for a k-clause circuit. Suppose the circuit has n variables, then there are $2^n(k - \lceil \log k \rceil)$ number of computations that need to be simulated. Because we target exploring the relation between the circuit height and the circuit predictability, there is no need to explore all possible conditions of the input variables for a k-clause circuit. In order to make it computationally feasible, we assume each variable only appears once in a clause, and choose n to be the smallest possible value that could guarantee there are no duplicated clauses in the circuit. To avoid duplicated clauses, k and n must satisfy:

$$\binom{n}{3} \geq k. \tag{7.2}$$

The circuits being simulated in **Method 1** and **Method2** can be expressed as a set $\{(k, h)\}$, where (k, h) denotes a circuit tree having k clauses and height h. The number of variables n and k, h satisfy constraint $1 \leq k \leq 7$ and Equations (7.1) and (7.2).

7.2 Method 1: Height Reduction

For a 3-CNF circuit denoted by (k, h) where k denotes the clause number and h denotes the circuit height, if k is fixed, the height of the circuit tree is from $\lceil \log k \rceil + 1$ to k. We simulate circuits having between one and seven clauses. For each circuit, we use 1000 trajectories to calculate the time deviation under each input assignment. We calculate σ_{max} for each circuit, which are shown in Table 7.1. Standard deviation of computation time for a fixed circuit (k, h) varies with the change of input assignments. Boolean formulas for the circuits (k, h) are presented in Table 7.2. Simulation results are presented in Figures 7.8 to 7.11 in the next section

Table 7.1: The values of σ_{max} for circuits having between 1 and 7 clauses. Circuit (k, h) has k clauses and height h.

no.	(k, h)	σ_{max}	no.	(k, h)	σ_{max}
1.	$(1, 1)$	32.69	8.	$(6, 4)$	310.94
2.	$(2, 2)$	57.57	9.	$(6, 5)$	264.72
3.	$(3, 3)$	109.9	10.	$(6, 6)$	256.07
4.	$(4, 3)$	143.94	11.	$(7, 4)$	356.04
5.	$(4, 4)$	162.83	12.	$(7, 5)$	465.95
6.	$(5, 4)$	262.1	13.	$(7, 6)$	657.61
7.	$(5, 5)$	221.55	14.	$(7, 6)$	296.78

(Section 7.2). Each diagram contains plots in two different colors. The black plots represent the the standard deviation under all possible input assignments. The blue plots represent simulation results in a modified model where spiders only move forward, which will be discussed in Section 7.3. Figure 7.8 shows the time deviations for circuits $(2, 2), (3, 3), (4, 3), (4, 4)$. Figure 7.9 shows the time deviations for circuits $(5, 4), (5, 5)$. Figure 7.10 shows the time deviations for circuits $(6, 4), (6, 5), (6, 6)$. Figure 7.11 shows the time deviations for circuits $(7, 4), (7, 5), (7, 6), (7, 7)$.

In Figure 7.1, we show the relation between circuit height and σ_{max}. In each diagram in Figure 7.1, we use different colors to indicate the circuits with different number of clauses. In Figure 7.1, we can tell that σ_{max} increases as the height increases in the magenta-color plots, which represent the circuits with 4 clauses. However, in the black and blue plots (5-clause circuit and 6 clause circuit), σ_{max} decreases as the circuit height increases. In the red plots (7-clause circuit), σ_{max} goes up and down as the circuit height increases. Therefore, there exists no linear relation between the circuit height and the measure of the circuit predictability (σ_{max}), **Method 1** is **NOT** valid.

Table 7.2: Boolean formulas for the 3-CNF circuits having between 1 and 7 clauses.

no.	(k, h)	Circuit Structure
1.	$(1, 1)$	$x_1 \vee x_2 \vee x_3$
2.	$(2, 2)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)$
3.	$(3, 3)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$
4.	$(4, 3)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)) \wedge ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4))$
5.	$(4, 4)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)$
6.	$(5, 4)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)) \wedge ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5))$
7.	$(5, 5)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5)$
8.	$(6, 4)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)) \wedge ((x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5))$
9.	$(6, 5)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4)) \wedge ((x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5))$
10.	$(6, 6)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5)$
11.	$(7, 4)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)) \wedge (((x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5)) \wedge ((x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5)))$
12.	$(7, 5)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)) \wedge ((x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5))$
13.	$(7, 6)$	$((x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4)) \wedge ((x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5))$
14.	$(7, 7)$	$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_3 \vee x_5)$

7.2.1 Analysis

At the beginning of this section, we state that shortening the computation paths of the *reporting spiders* can lead to an improvement of the circuit predictability. Since different input assignments lead to different *reporting spiders* that take different computation paths, we hypothesize if the total length of all the cascading paths are reduced, the total length of all computation paths under different input assignments are reduced generally, which we believe can reduce the values of σ_{max}. Based on this hypothesis, we propose that reducing the height of the circuit tree can reduce the

Table 7.3: Total length of all cascading paths for the circuits having clauses from 1 to 7. Circuit (k, h) has k clauses and height h.

no.	(k, h)	Total Length	no.	(k, h)	Total Length
1.	$(1, 1)$	1	8.	$(6, 4)$	88
2.	$(2, 2)$	10	9.	$(6, 5)$	84
3.	$(3, 3)$	26	10.	$(6, 6)$	91
4.	$(4, 3)$	42	11.	$(7, 4)$	108
5.	$(4, 4)$	46	12.	$(7, 5)$	120
6.	$(5, 4)$	68	13.	$(7, 6)$	135
7.	$(5, 5)$	68	14.	$(7, 6)$	114

total length of all cascading paths. Simulation results in Figure 7.1 has shown that height reduction is **NOT** a valid method to improve the circuit predictability. We list the total length of all cascading paths for the circuits having clauses from 1 to 7 in Table 7.3, we can see that height reduction does not reduce the total length of all cascading paths, which results from our geometrical layout generation algorithm discussed in Section 5.1.

The circuits being simulated in this section are generated using our geometrical layout generation algorithm. The algorithm is discussed in previous section (Section 5.1). When we fix the number of clauses for a circuit and vary the height of the circuit, when the circuit tree is shorter, the number of gates arranged in the same layer is increased. To void the overlaps between the gates at the same layer, our algorithm makes them aligned according to the tops of their layouts, keep unit distance to the next upper layer vertically, and keep at least unit distance to each other horizontally. When the number of gates at the same layer is increased, the layout of this layer becomes wider, thus increasing the lengths of the cascading paths between each layer and the layer above. Therefore, for a circuit having a fixed number of clauses, reducing the height of the circuit tree cannot shorten the total length of all cascading paths in all cases. So we turn to directly explore the relation between

the total length of all cascading paths and σ_{max}, trying to figure out whether our previous hypothesis is correct. Our previous hypothesis is that shortening the total length of all cascading paths can improve the circuit predictability.

In Figure 7.2, we plot how σ_{max} varies with the total length of all the cascading paths. Circuits with different number of clauses are indicated using different types of points. From this figure, we can see that σ_{max} does not increase monotonically as the length of all the cascading paths increases. Therefore, our previous hypothesis that shortening the length of the cascading paths can improve the circuit predictability is **WRONG**. That is because shortening the total length of all cascading paths cannot reduce the probability that the *reporting spiders* walk backward. We know that spiders are less far back to go on a shorter path. However, the *reporting spider* under one certain input assignment does not move on a contiguous path from the input location to the output location. The cascading path between any two connected gates are individual segments with different lengths. Molecular spiders move bi-directionally on each individual segment. Traversing time for the *reporting spider* on these individual segments is the sum of the traversing time on all segments. Therefore, the standard deviation of computation time under one certain input assignment is related to the the number of different segments and the length of each different segment. Reducing the total length of all cascading paths does not guarantee the reduction of σ_{max}.

It is difficult to improve the circuit predictability by changing the circuit structure when the circuit size (the number of clauses) is fixed. However, if we use sequential cascades to construct the circuits having between 1 and 7 clauses, σ_{max} increases as the circuit size increases, which is shown in Figure 7.3. Gates are connected along a single path in a sequentially cascaded circuit. A sequentially cascaded circuit has the highest circuit tree. From the current simulation results shown in Figures 7.1 and 7.2 and the analysis on the current results, we can conclude that **Method 1** is **NOT** valid for the circuit predictability improvement. Instead, we could try reducing the

circuit size and shortening all the cascading paths. This method would require a new algorithm to generate the layout of the circuit, which is beyond the scope of this thesis.

7.3 Method 2: One-Directional Movements

In this section, we modify the original model to make spiders only move forward, and compare the modified model with the original model in terms of circuit predictability. The modified model is still a continuous time Markov chain. We compare the gate computation between two models using simulations. Each simulation produces 1000 trajectories in both models under one input assignment. Simulation results of gate computation are shown in Figures 7.4 to 7.6. Simulation results of the clause circuit are shown in Figure 7.7. In Figures 7.8 to 7.11, we show the standard deviations under all input assignments for circuits having clauses from 2 to 7. The values of σ_{max} for those circuits in the modified model are shown in Table 7.4. We show the relation between the circuit height and σ_{max} in Figure 7.12. In Figure 7.13, we show the relation between the total length of all cascading paths and the values of σ_{max}. In Figure 7.14, we compare how circuit predictability is influenced by the circuit size in sequentially cascaded circuits in both models.

7.3.1 Analysis

From the comparisons shown in Figures 7.4 to 7.6 and simulations results shown in Figures 7.12 and 7.13, we can see:

1). For a circuit (k, h), the standard deviation of computation time in the modified model does not change as much as in the original model under different input assignment;

2). In both models, σ_{max} do not increase monotonically with the height of the circuit tree. Compared with the original model, σ_{max} does not change much as the height of a circuit three varies;
3). The circuit predictability in the original model is influenced by the circuit size more than the modified model;
4). Compared with the original model, σ_{max} is decreased significantly in the modified model.

Summary From the simulation results and analysis in Sections 7.2 and 7.3, we conclude that **Method 1** is **NOT** valid for improving the circuit predictability since reducing the height of the circuit three does not guarantee reducing the probability that spiders move backward. **Method 2** improves the circuit predictability significantly because this method directly changes the behaviors of the spiders to avoid spending time on the backward route. The circuit predictability can be improved by reducing the circuit size and shortening all the cascading paths in the circuit tree. Therefore, given a Boolean formula, we can apply Boolean algebra simplification rules to convert the original formula to a logically equivalent one using fewer gates and variables to improve the circuit predictability. We can also use the modified model where spiders only move forward to implement the circuit.

Table 7.4: The values of σ_{max} for circuits having between 1 and 7 clauses in the modified model. Spiders only move forward in the modified model. Circuit (k, h) has k clauses and height h.

no.	(k, h)	σ_{max}	no.	(k, h)	σ_{max}
1.	$(1, 1)$	4.15	8.	$(6, 4)$	8.73
2.	$(2, 2)$	5.49	9.	$(6, 5)$	9.69
3.	$(3, 3)$	6.69	10.	$(6, 6)$	10.47
4.	$(4, 3)$	7.56	11.	$(7, 4)$	8.92
5.	$(4, 4)$	8.19	12.	$(7, 5)$	9.88
6.	$(5, 4)$	8.50	13.	$(7, 6)$	11.31
7.	$(5, 5)$	9.35	14.	$(7, 6)$	11.38

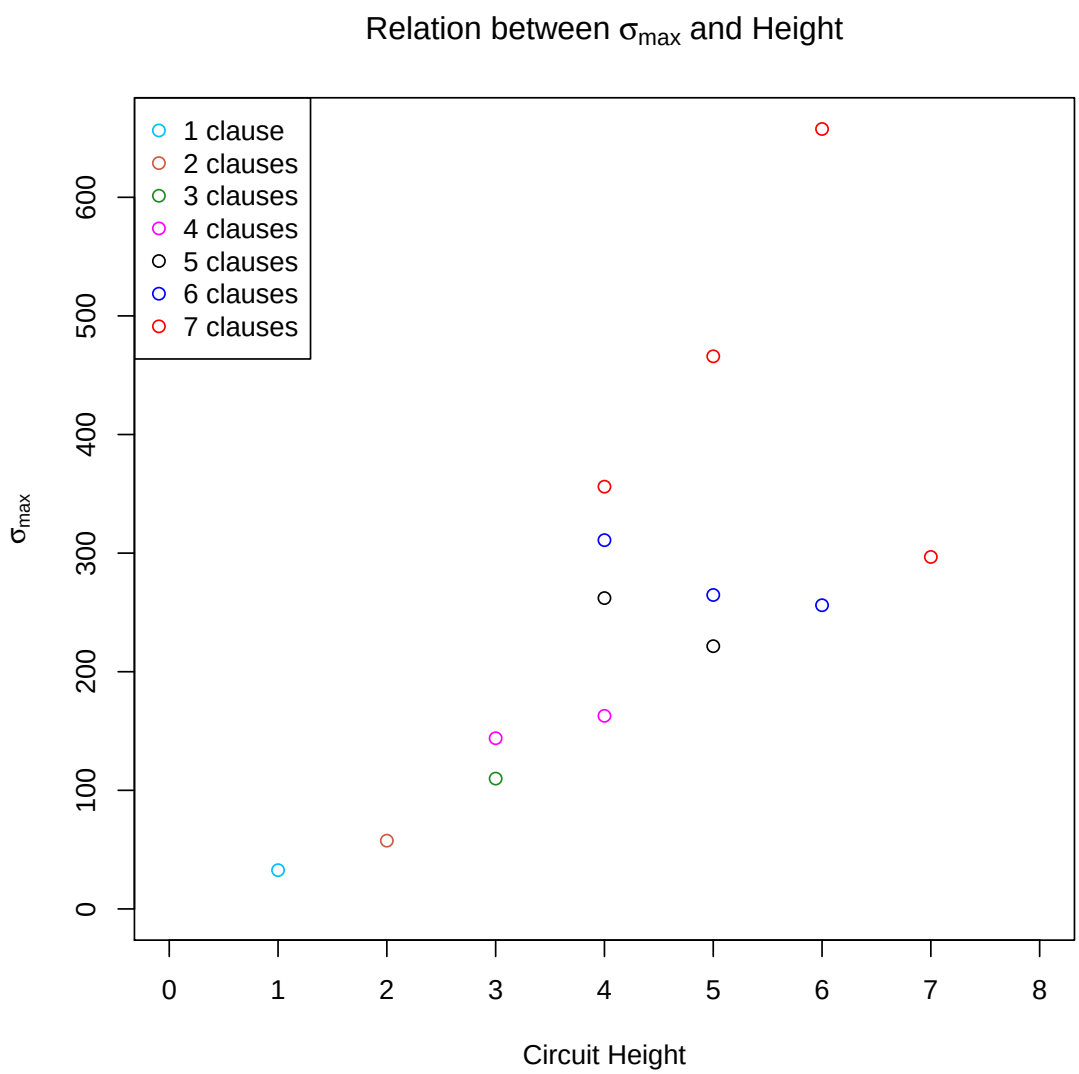


Figure 7.1: The figure shows the relation between σ_{max} and the circuit height. We use different colors to represent the circuits with different number of clauses, as indicated in the legend.

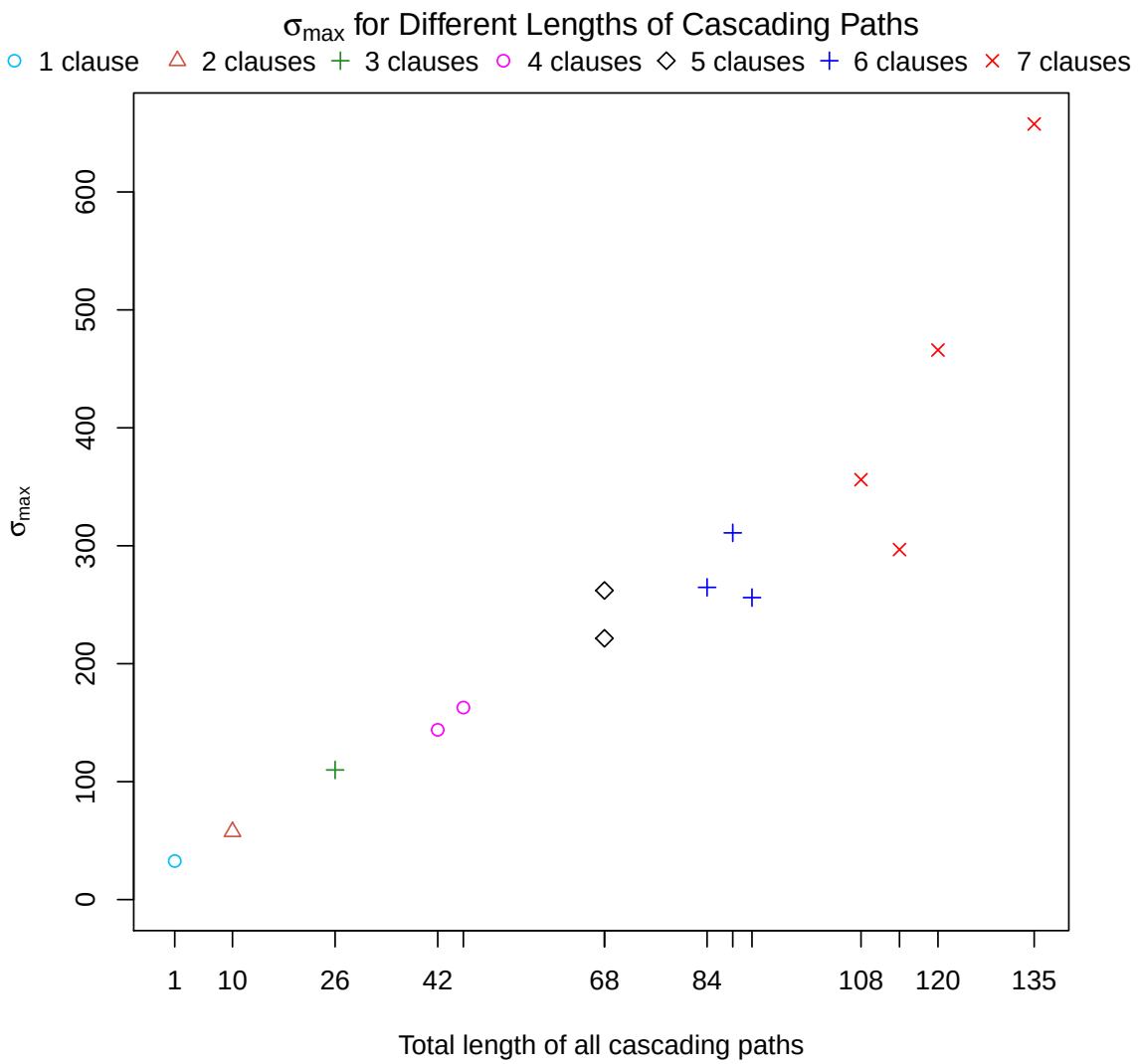


Figure 7.2: For each circuit (k, h), we calculate the total lengths of all the cascading paths in it. We show the relation between σ_{max} and the total cascading path lengths for the circuits having between 1 and 7 clauses. Circuits with different number of clauses are represented by different types of points, as indicated in the legend.

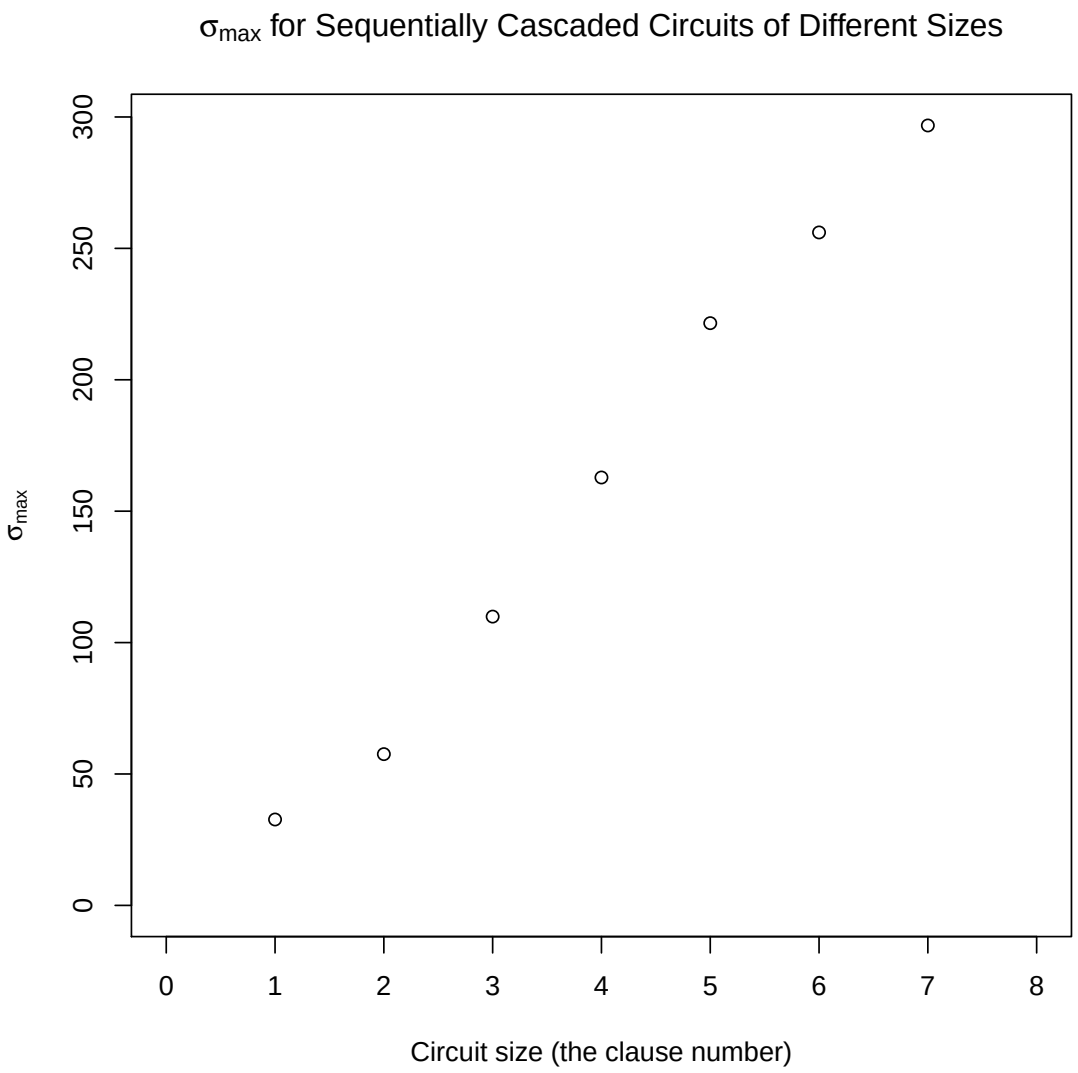


Figure 7.3: The figure shows the relation between σ_{max} and circuit size. Circuit size here refers to the number of clauses in a circuit in the form of 3-CNF. We explore circuits having between one and seven clauses, and these circuits are cascaded in the same manner, sequential cascades. The value of σ_{max} increases as the circuit size increases.

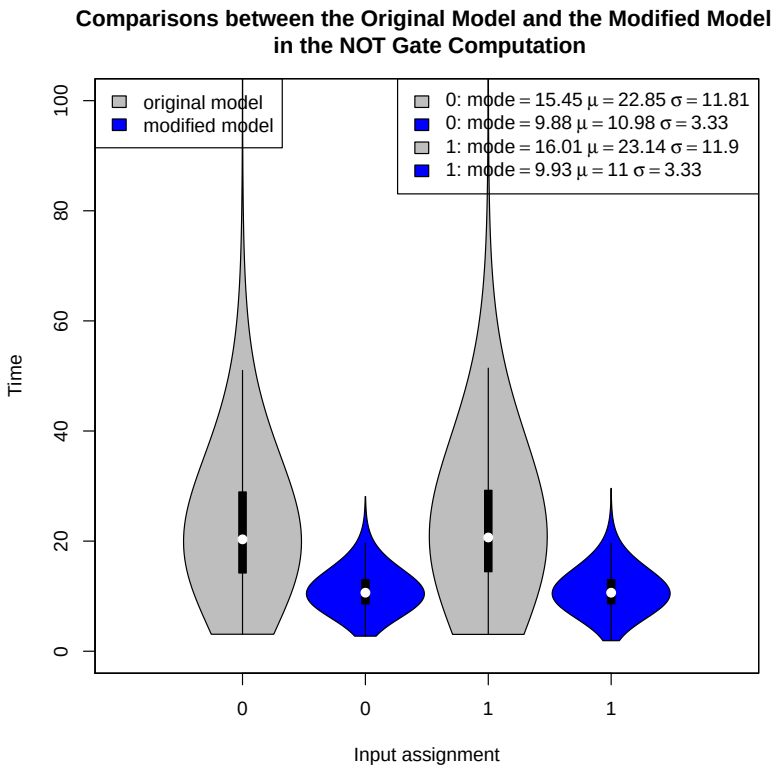


Figure 7.4: Comparisons between the original model and the modified model in the AND gate computation. The original model data is in black, the modified model data is in blue. Spiders only move forward in the modified model. Computation time distributions under all possible input assignments are shown in the figure. Standard deviation, mean and mode for both models under all input assignments are shown in the legend.

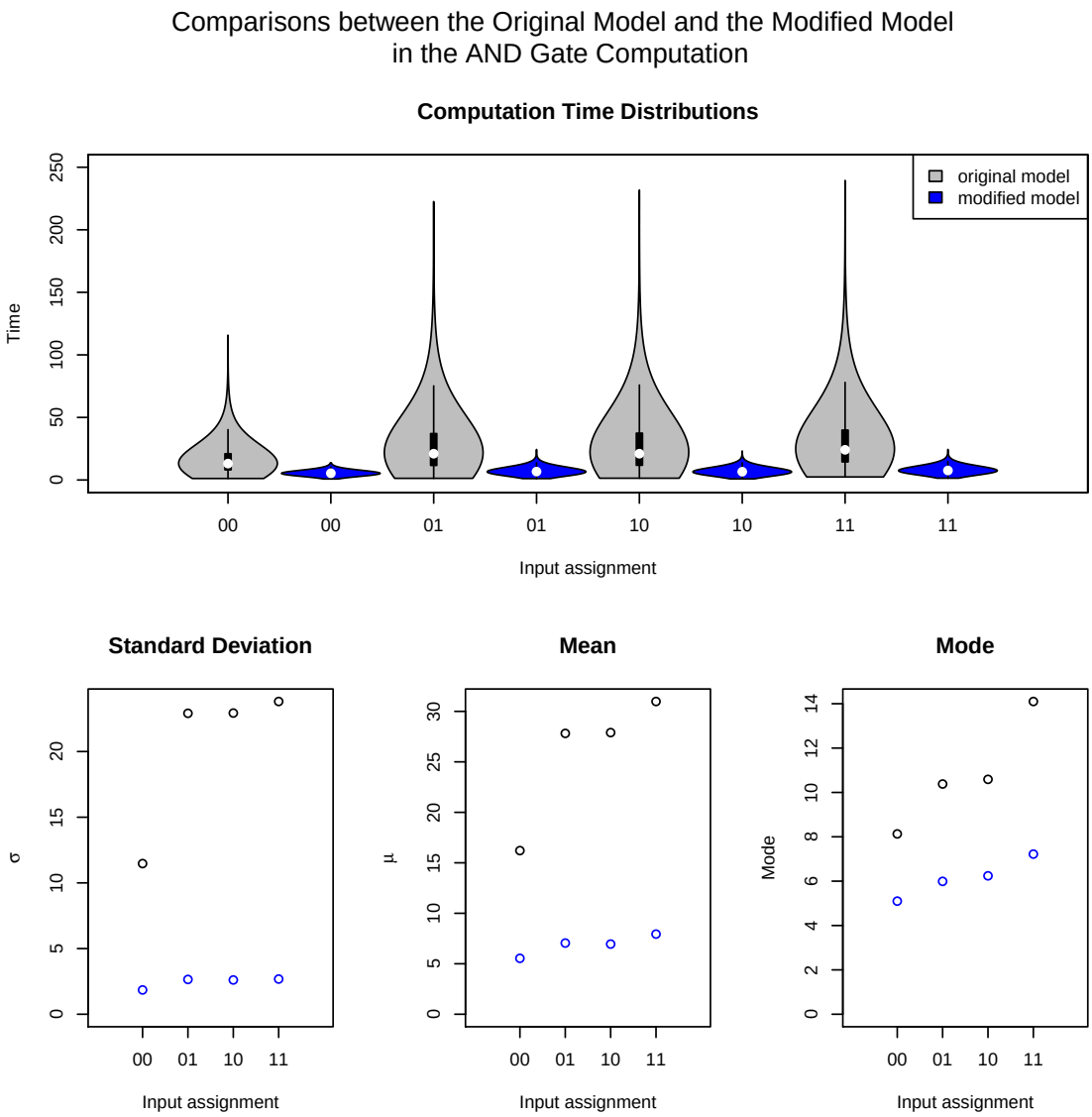


Figure 7.5: Comparisons between the original model and the modified model in the AND gate computation. The original model data is in black, the modified model data is in blue. Spiders only move forward in the modified model. Computation time distributions under all possible input assignments are shown in the top diagram. Standard deviation, mean and mode for both models under all possible input assignments are shown at the bottom.

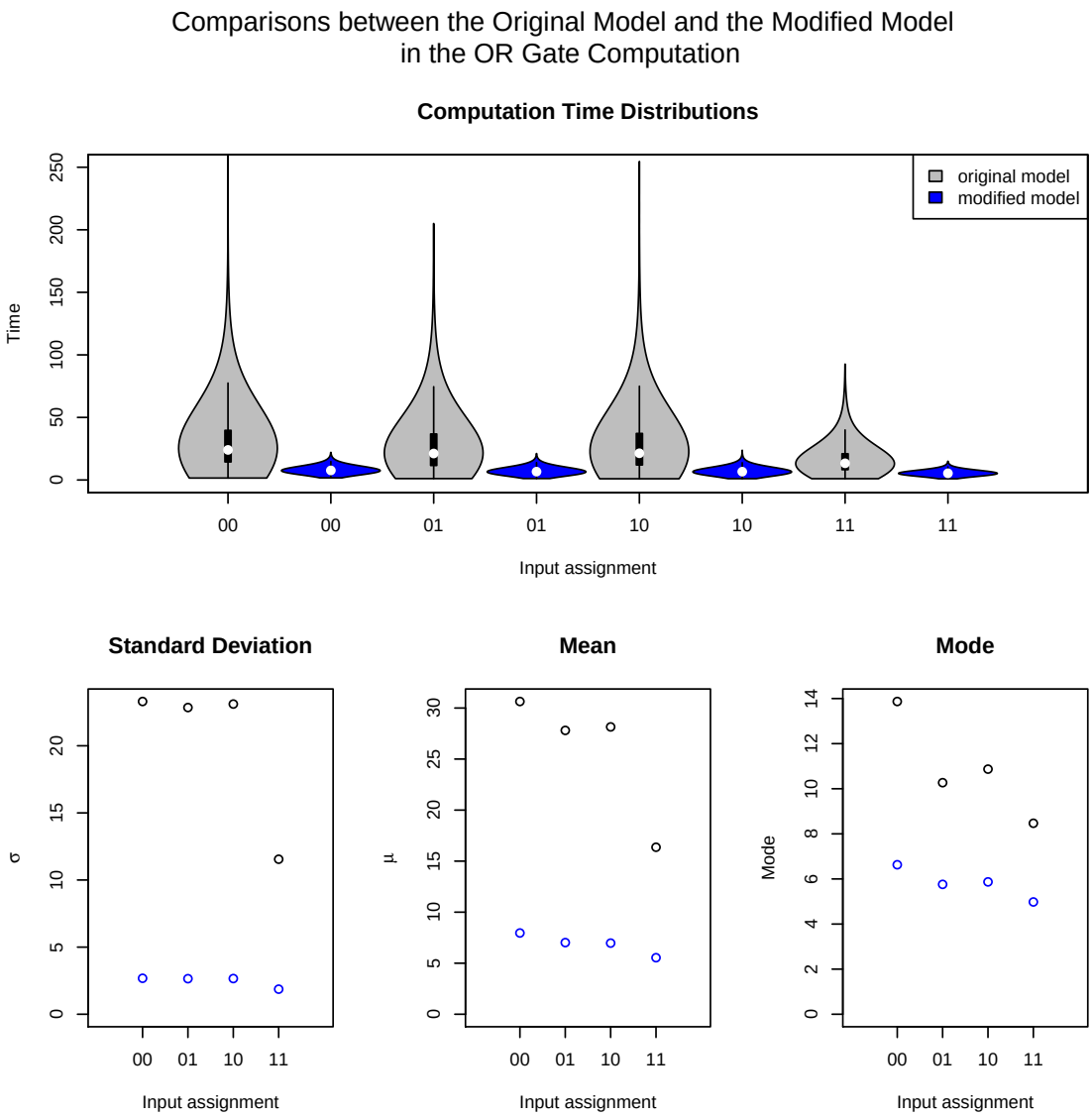


Figure 7.6: Comparisons between the original model and the modified model in the OR gate computation. The original model data is in black, the modified model data is in blue. Spiders only move forward in the modified model. Computation time distributions under all possible input assignments are shown in the top diagram. Standard deviation, mean and mode for both models under all possible input assignments are shown at the bottom.

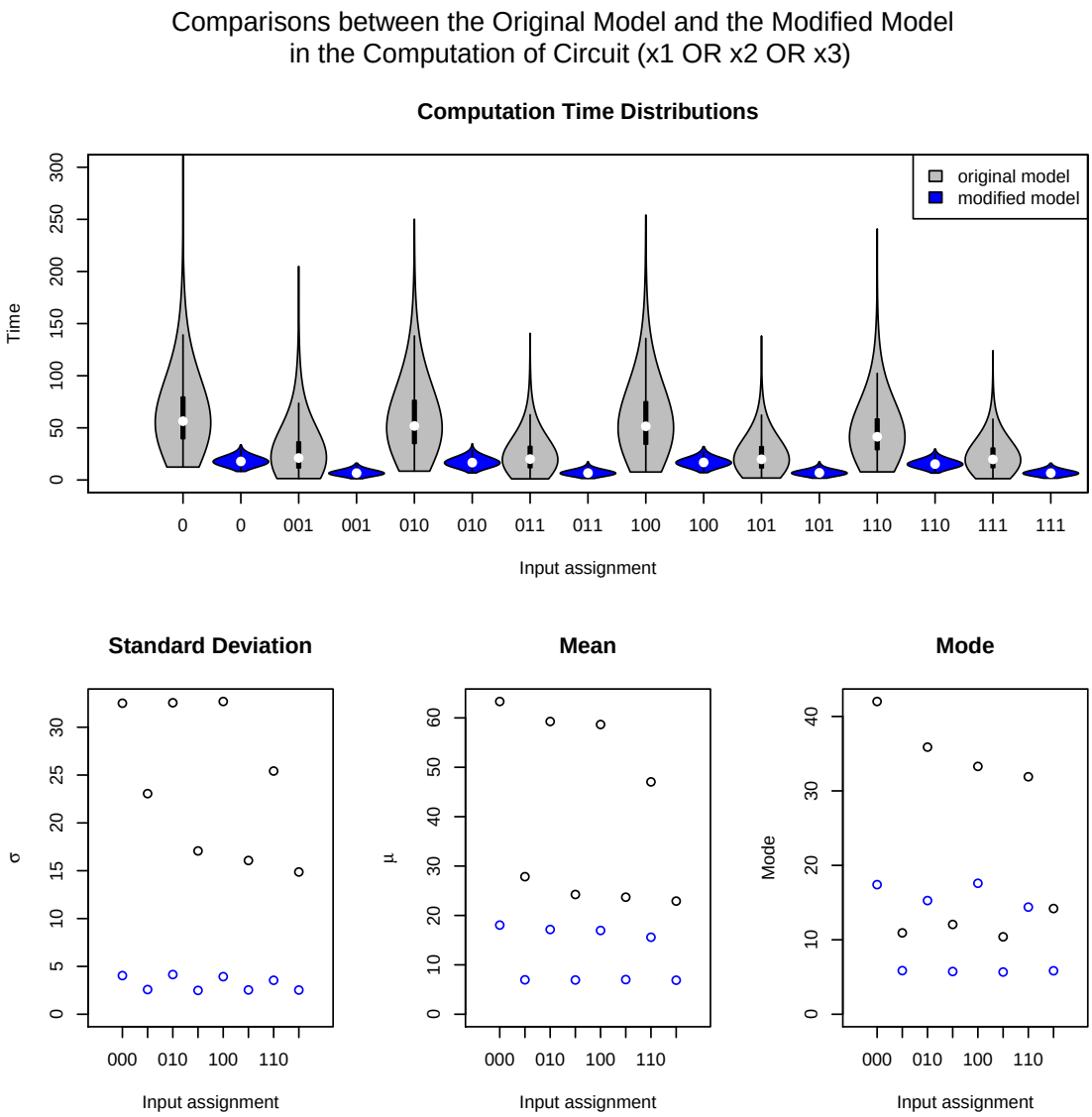


Figure 7.7: Comparisons between the original model and the modified model in the clause circuit. The original model data is in black, the modified model data is in blue. Spiders only move forward in the modified model. Computation time distributions under all possible input assignments are shown in the top diagram. Standard deviation, mean and mode for both models under all possible input assignments are shown at the bottom.

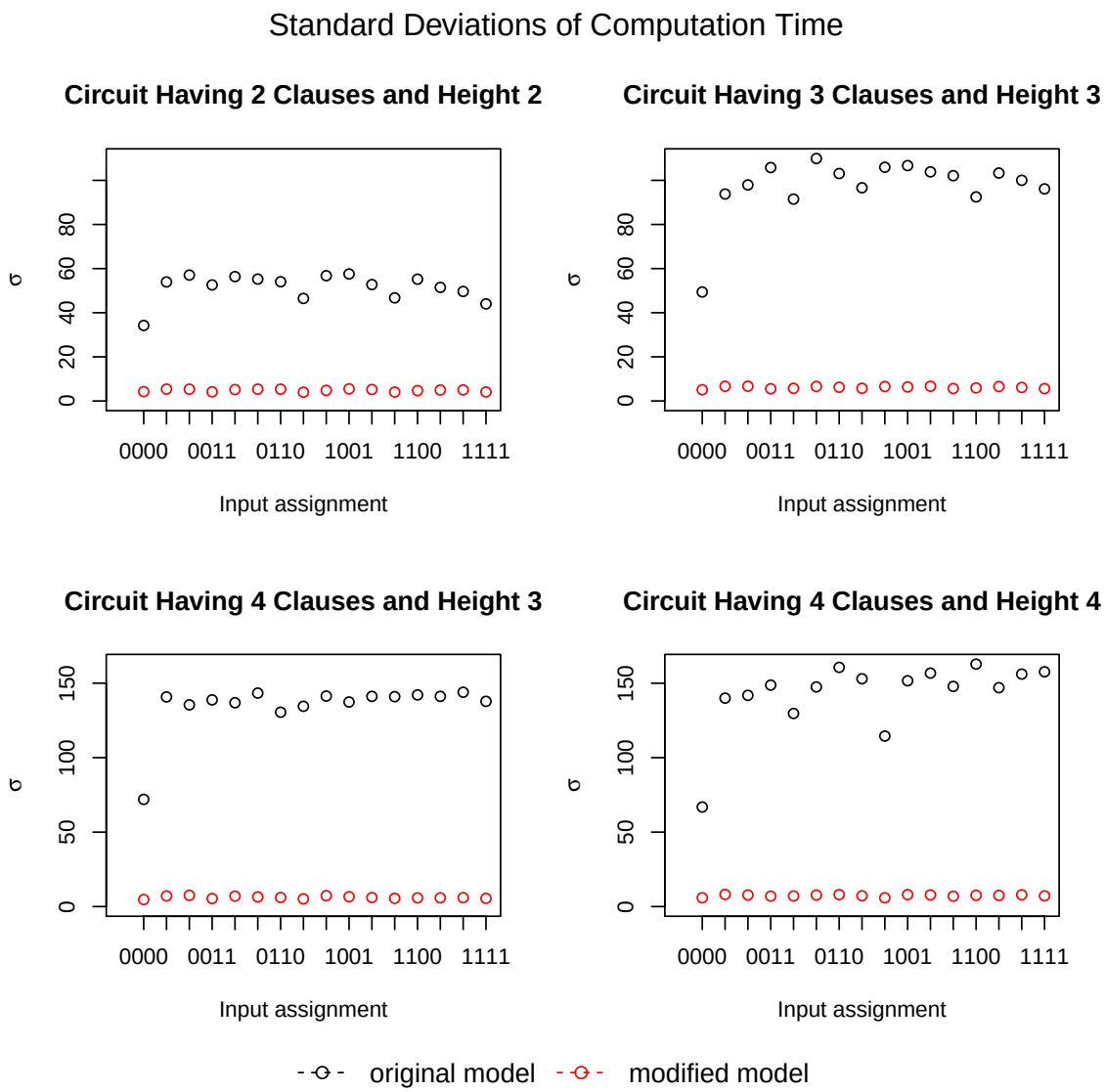


Figure 7.8: Standard deviations under all possible input assignments for circuits having clauses from 2 to 4. The original model data is in black color, while the data in the modified model are in blue. Spiders only move forward in the modified model.

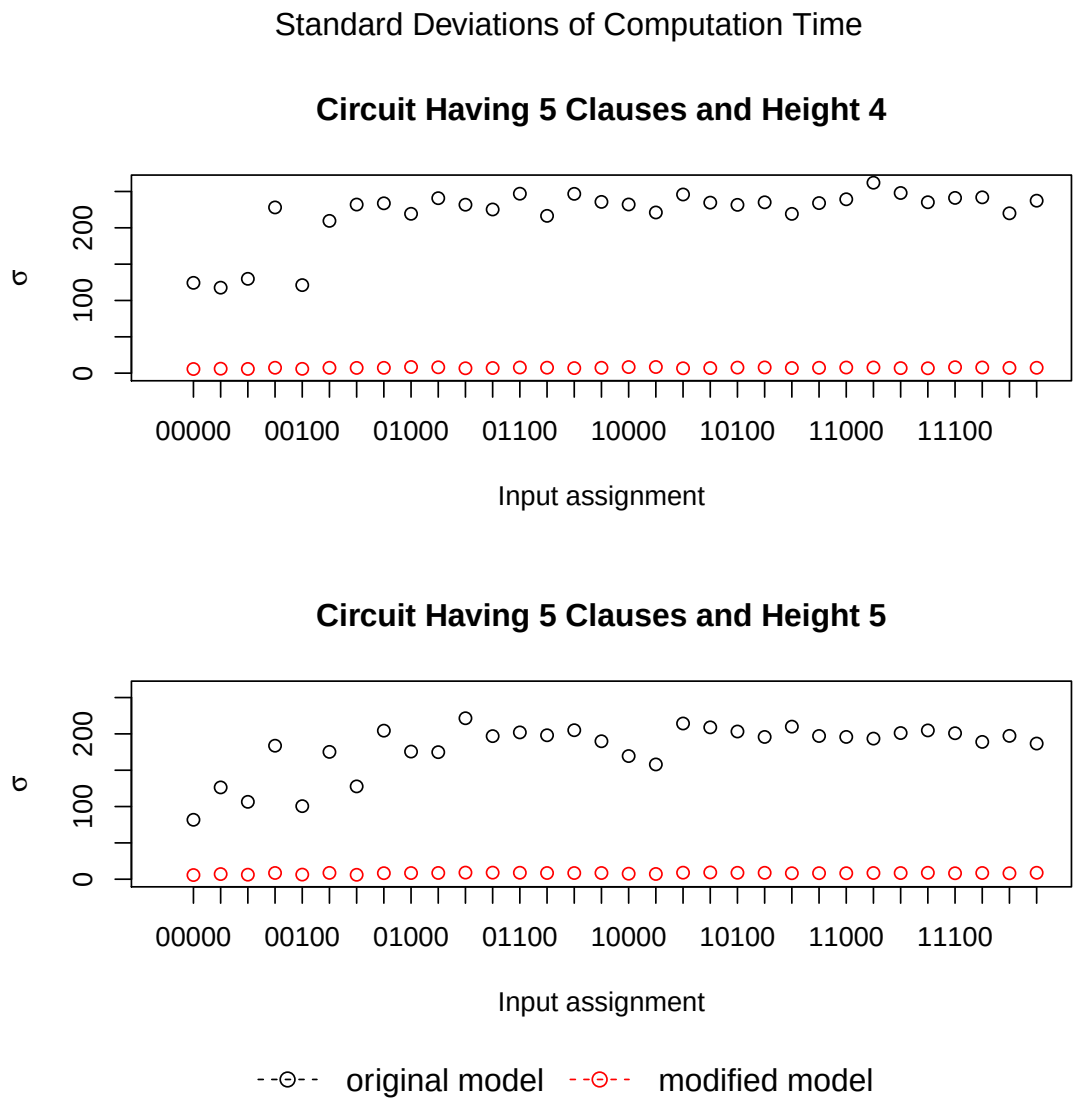


Figure 7.9: Standard deviations under all possible input assignments for 5-clause circuits: $(5, 4), (5, 5)$. The original model data is in black color, while the data in the modified model are in blue. Spiders only move forward in the modified model.

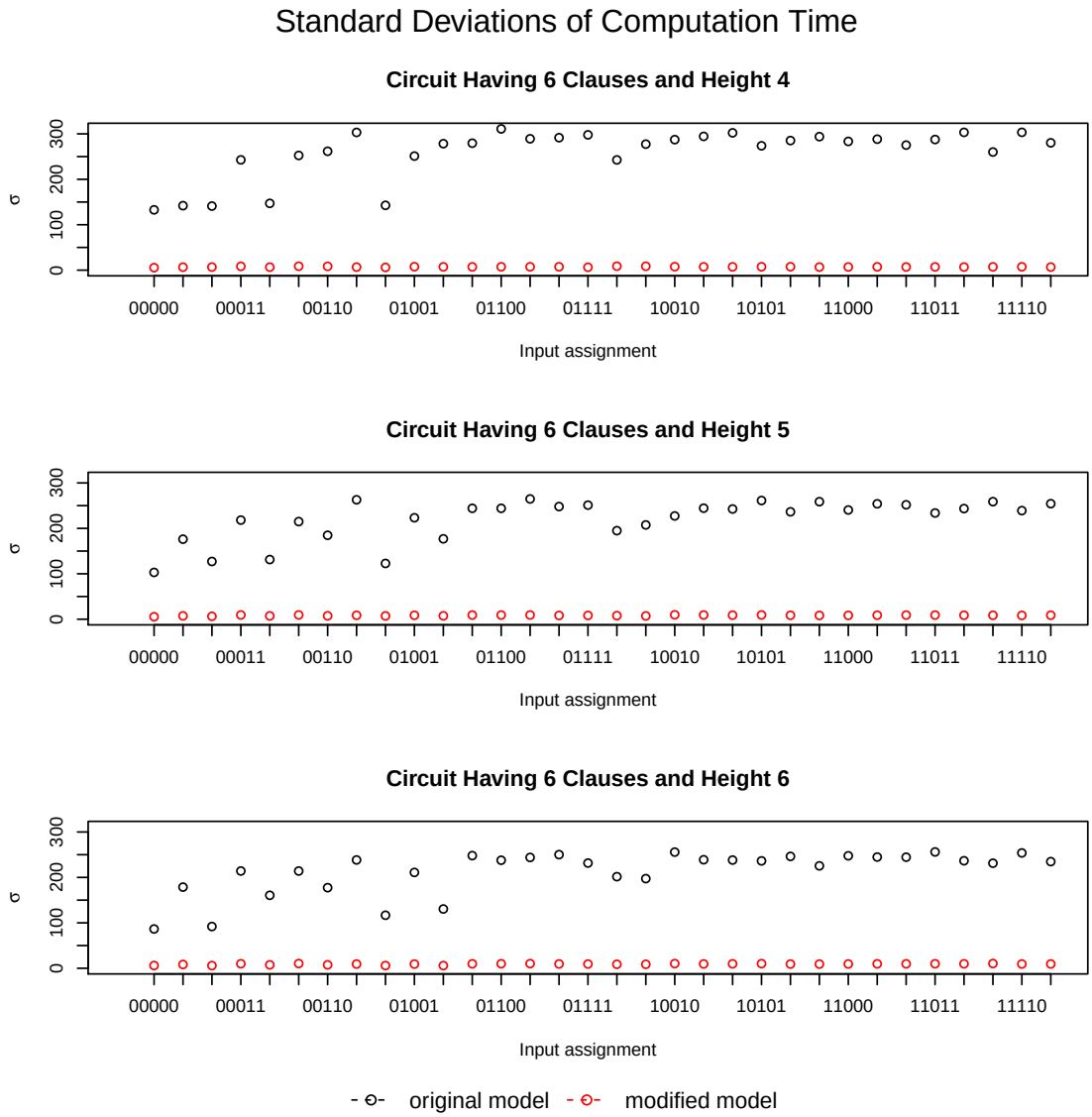


Figure 7.10: Standard deviations under all possible input assignments for 6-clause circuits: $(6, 4), (6, 5), (6, 6)$. The original model data is in black color, while the data in the modified model are in blue. Spiders only move forward in the modified model.

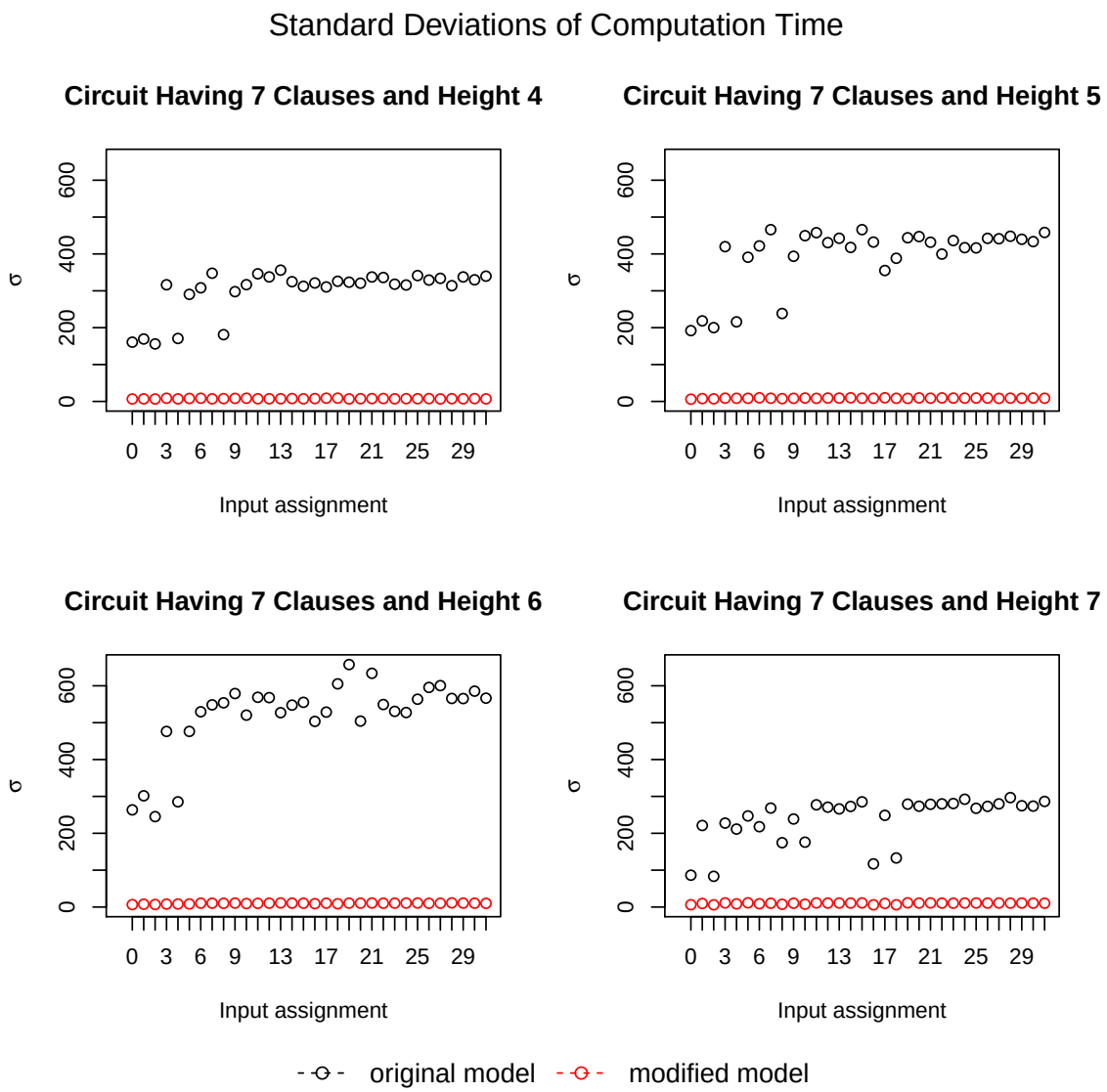


Figure 7.11: Standard deviations under all possible input assignments for 7-clause circuits: $(7,4),(7,5),(7,6),(7,7)$. Input assignments from "00000" to "11111" are represented as decimal numbers in this figure. For example, input assignment "25" represents "11001". The original model data is in black color, while the data in the modified model are in blue. Spiders only move forward in the modified model.

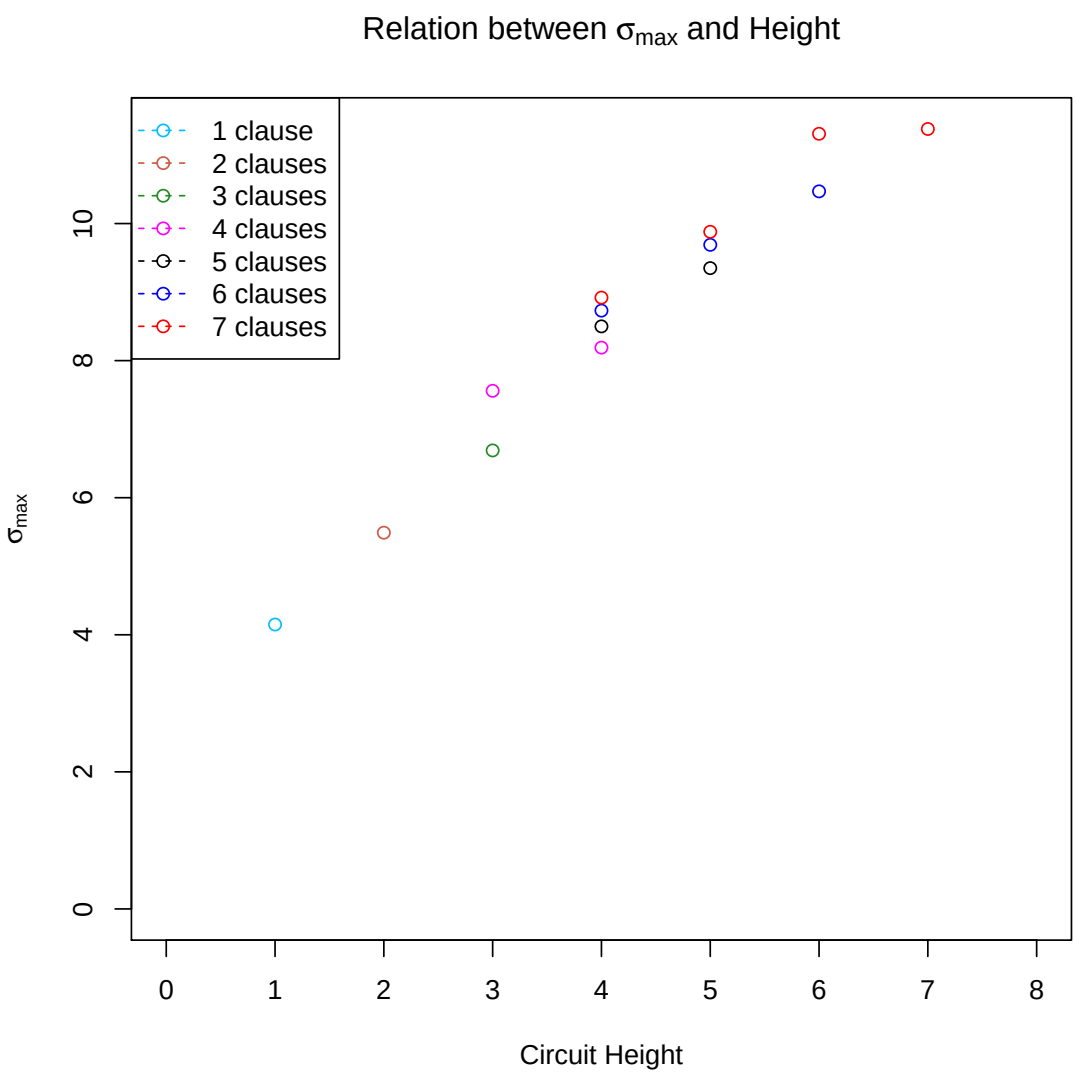


Figure 7.12: The figure shows the relation between σ_{max} and the circuit height. We use different colors to represent the circuits with different number of clauses, as indicated in the legend.

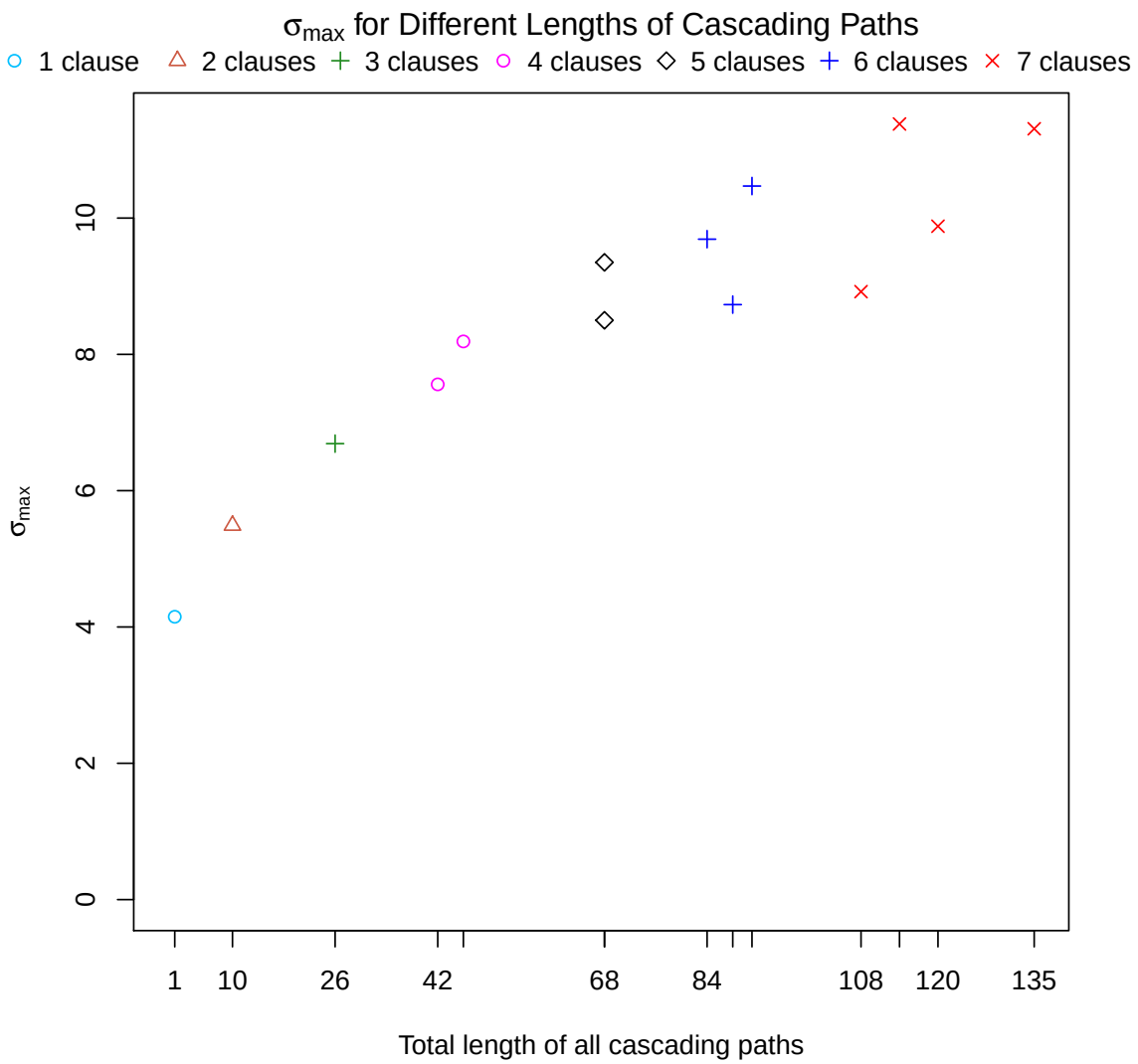


Figure 7.13: For each circuit (k, h), we calculate the total lengths of all the cascading paths in it. We show the relation between σ_{max} and the total cascading path lengths for the circuits having between 1 and 7 clauses. Circuits with different number of clauses are represented by different types of points, as indicated in the legend.

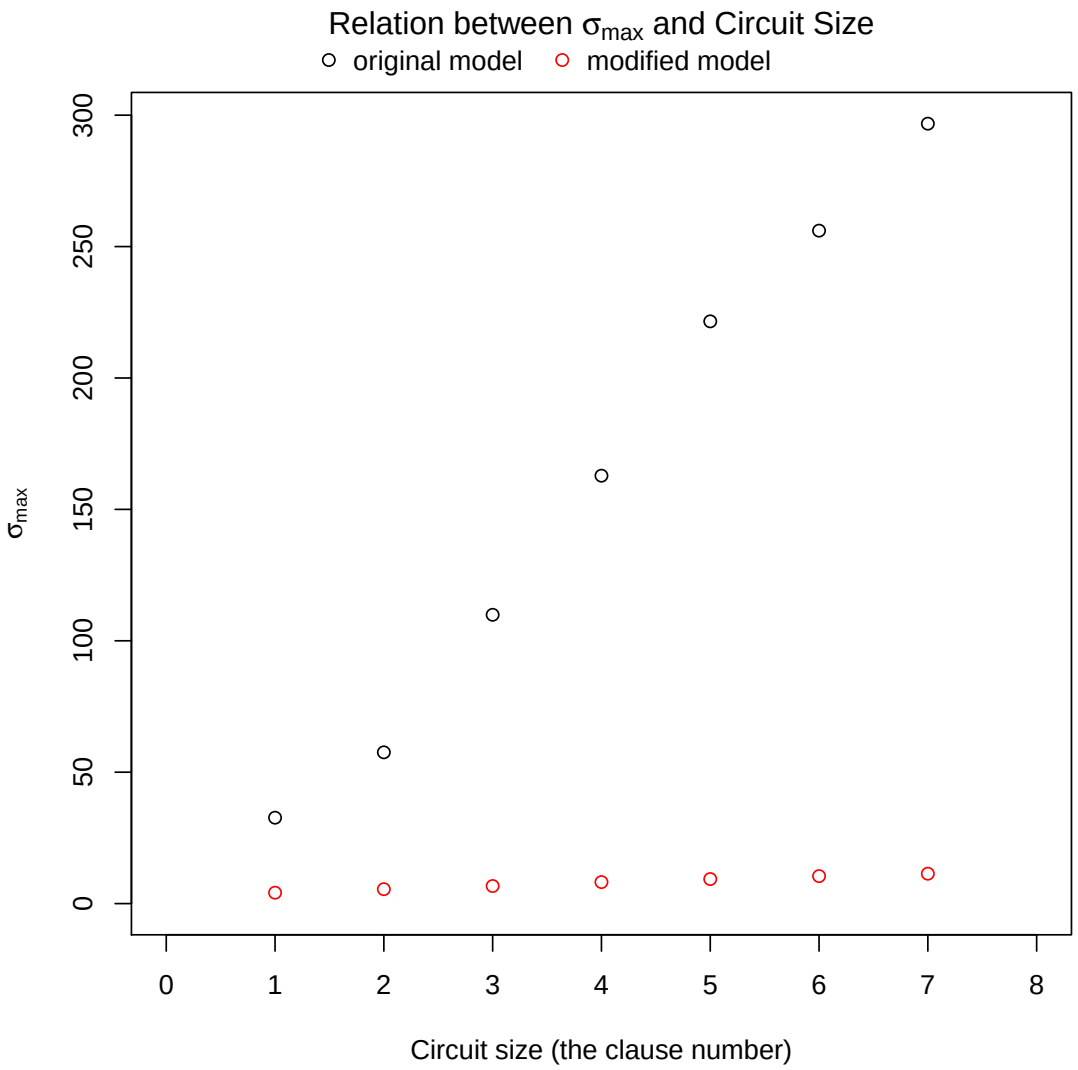


Figure 7.14: Comparison between the original model and the modified model. The figure shows the relation between σ_{max} and circuit size in both models. The original model data is in black, the modified model data is in blue. The circuit size refers to the number of clauses in a circuit in the form of 3-CNF. We explore the circuits having between 1 and 7 clauses, and these circuits are cascaded using the same manner, sequential cascades. The value of σ_{max} increases as the circuit size increases in both models. The increase in σ_{max} in the modified model is less than in the original model.

Chapter 8

Possible Implementations

We have sketched two different possible implementations of the model used in the logic circuit construction. We will give details of these two implementations in Section 8.1 and Section 8.2.

8.1 Possible Implementation I

We use DNA strand displacement and catalytic cleavage to implement the mechanisms $SW_{1\to0}$ and $SW_{0\to1}$ in a NOT gate and mechanism $exit$ in the gate cascades. Non-alterable sites form the main parts of the track, and their structures are single-stranded DNA. Alterable sites form the functional parts of the track, whose structures are either single-stranded DNA with long complementary domains to the spider limbs, or the hairpin structures and bridge structures. We assume that when the binding part between a limb and a site is more than two complementary domains, the bound limb cannot leave that site by itself, and is trapped on that site.

8.1.1 Spiders and non-alterable sites.

The spider has two DNAzyme legs [1, 24, 25] and a single-stranded DNA arm containing a sequence that is hybridized with an indicator strand. The domains of the legs and the arm are shown in Figure 8.1. A site is a DNA structure tethered to the surface of a $2D$ rectangular lattice. A normal site in $S_{norm} = \{s_l, s_1, s_0\}$ is a single-stranded DNA. Site s_l contains the complementary domain (a^*) to the spider leg (a); sites s_1 and s_0 each contains a complementary domain (1^* and 0^*) to the 1-typed arm and 0-typed arm respectively.

8.1.2 Alterable sites and mechanisms.

An alterable site has three features: (1) it can be in one of three states, "on", "off", or "trapped"; (2) it may or may not trap a spider; (3) it may contain a signal that can be sent out once a spider has attached to the site. In the designs of NOT gate and gate cascades, we use the *switch* mechanisms to implement a NOT gate and use the *exit* mechanism to implement the gate cascades. A mechanism consists of a set of neighboring alterable sites along the same direction. An alterable site is a DNA strand with different structures tethered to the surface of the lattice. In the definitions of different mechanisms, we use the same label to represent the sites that have the same functionality, but the DNA structures for these sites with the same label may be different in different mechanisms. For example, mechanisms $SW_{1\to0}, SW_{0\to1}$ and *exit* all contain sites s_r^I and s_r^{II}. These two sites have different DNA structures in different mechanisms.

Define mechanisms as:

$$SW_{1\to0} = \{(s_{1\to0}, \mathbf{p}_1), (s_r^I, \mathbf{p}_2), (s_r^{II}, \mathbf{p}_3)\}$$
$$SW_{0\to1} = \{(s_{0\to1}, \mathbf{p}_1), (s_r^I, \mathbf{p}_2), (s_r^{II}, \mathbf{p}_3)\}$$

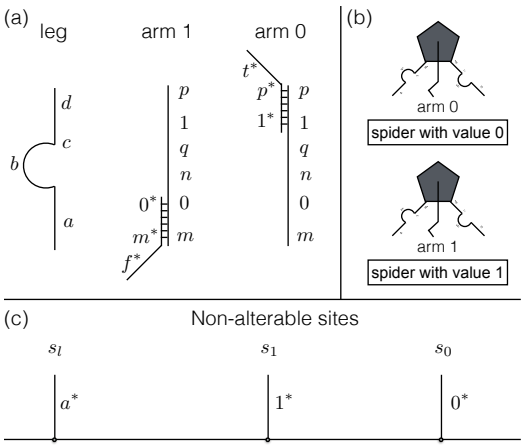


Figure 8.1: Possible implementation I. The spider structure and non-alterable sites. (a) The leg is a DNAzyme where the domain b catalyzes cleavage of a substrate in mechanisms that will be shown in later figures. The a domain of the leg is complementary to the a^* domain of site s_l, thus the leg can attach to site s_l. Arm 1 is partially hybridized with an indicator strand $(0^*, m^*, f^*)$. A spider having an arm of 1 can attach to a site s_1 since the domain 1 of its arm is complementary to the domain 1^* of site s_1. Similarly, a spider having an arm of 0 can attach to a site s_0. (b) Two types of spiders are shown at top right. Each spider has two legs and an arm. (c) The non-alterable sites are single-stranded DNA with a single domain. They are tethered to the surface of a lattice, where the small gray dots in the figure represent the tethered points. We assume the binding strength between a pair of complementary domains is weak, so the limb bound to these non-alterable sites can dissociate from them. When the binding part between a limb and a site is more than two complementary domains, we assume the limb cannot leave the site by itself (the limb is trapped on that site).

$$exit = \{(s_t, \mathbf{p}_1), (s_r^I, \mathbf{p}_2), (s_r^{II}, \mathbf{p}_3)\}$$

where $(s, \mathbf{p}_i)$ in a mechanism indicates the location of site s is $\mathbf{p}_i$, and $\mathbf{p}_i \in \mathbb{Z}^2$ must satisfy

$$|\mathbf{p}_i - \mathbf{p}_{i-1}| = 1, \text{ where } i = 2, 3.$$

When a spider moves on site s_t, it becomes trapped. When a spider moves to a site $s_{1\to0}$ or $s_{0\to1}$, it becomes trapped and its value becomes 0 or 1. When the spider

goes on moving over the latter two sites in these mechanisms, site s_r^I releases the limb trapped on the first site, and traps the spider; site s_r^{II} releases the limb trapped on the second site. Since the first site has different implementations, sites s_r^I and s_r^{II} must have different implementations in different mechanisms even though their functionality is the same in three mechanisms.

Figure 8.2 shows the implementations of mechanism *exit* and its operating procedure. Figure 8.3 shows the implementations and operating procedure for $SW_{1\to0}$. Since $SW_{0\to1}$ follows a similar design to $SW_{1\to0}$, we only show the implementation for it in Figure 8.4.

In the designs of AND gate and OR gate, we use sites s_u and s_p to control the spiders' behavior at the crossroads in the AND gate and OR gate. Figure 8.5 shows how site s_u sends a "turning-on" signal to unblock the site s_p when a spider attaches to it.

8.2 Possible Implementation II

In the second implementation, only DNA strand displacement is used. We add an excess amount of external strands in the environment to aid the movement of molecular spiders when they move through the *exit* mechanism and *switch* mechanisms. Each spider has two legs and one arm. There are two types of arm. An arm of type 1 can only attach to the normal site s_1 and an arm of type 0 can only attach to the normal site s_0. Implementations of spider leg and arms are shown in Figure 8.6. We add one type of external strand containing two domains 4 and b in this implementation. Figure 8.7 shows the implementation of the *exit* mechanism. Figure 8.8 and Figure 8.9 show the implementations of a *switch* mechanisms $SW_{1\to0}$ and $SW_{0\to1}$. Sites s_u and s_p used in the AND/OR gate implementation retain the same design as in the first implementation.

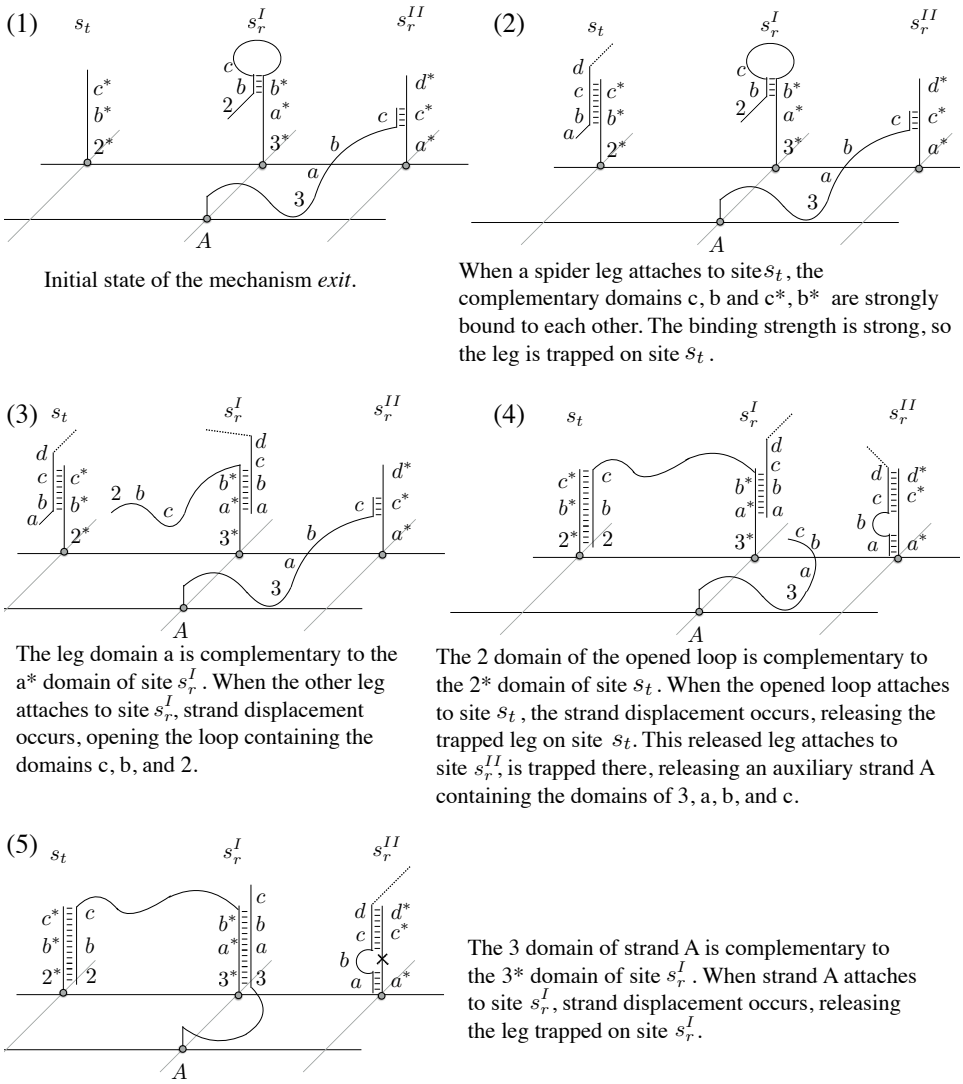


Figure 8.2: Possible implementation I. An implementation of mechanism *exit*. Diagram (1) is the initial condition of mechanism *exit*. Diagram (1) through (5) show the operating procedure of *exit* that cuts off the backward route of the spider. Each diagram has an explanation to the right of it. We omit the spider body in these diagrams. A dotted line of a limb represents the omitted part that extends to connect to the spider body. In diagram (5), the spider cannot move to sites s_t and s_r^I since no complementary domains exist between the leg and these sites. Therefore, the backward route of the spider is cut off when the spider moves through the *exit* mechanism. In diagram (5), the DNAzyme leg on site s_r^{II} cleaves the site at the position labeled with a small cross, making the spider move forward.

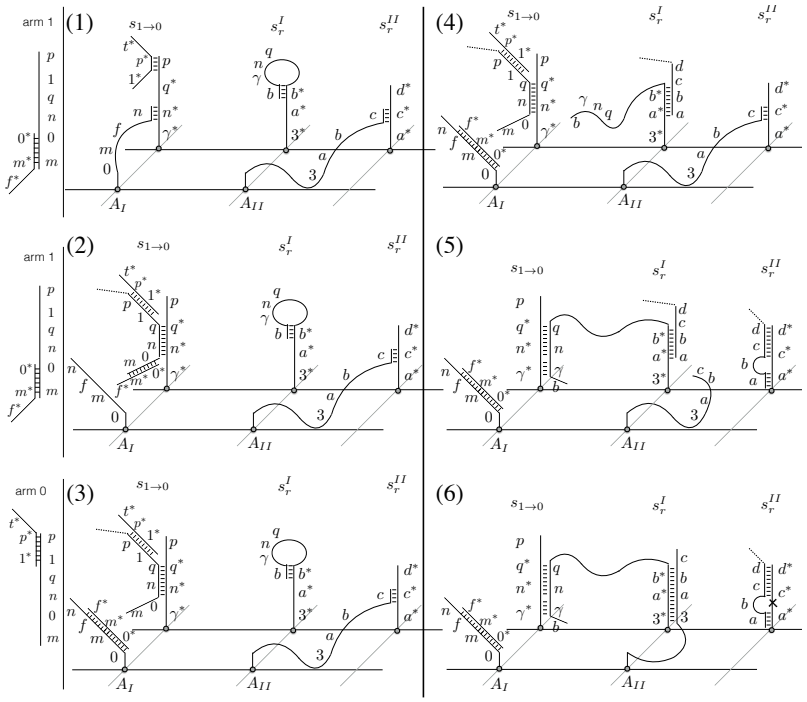


Figure 8.3: Possible implementation I. An implementation of the *switch* mechanism $SW_{1\to 0}$. Diagram (1) shows the initial condition. When a spider with arm type of 1, as shown to the left of diagram (1), reaches site $s_{1\to 0}$, the arm type becomes 0 and the spider is trapped on that site. This procedure consists of two DNA strand displacements. Domain 1 of arm 1 binds to 1^* domain, the strand displacement occurs to take off the strand $t^*p^*1^*$ from the site and binds it to the arm; Domain q of arm 1 binds to q^* domain, the strand displacement occurs to release the strand A_I having domains of $n, f, m, 0$. The arm binds to the site via two complementary domains (q, q^* and n, n^*), the binding strength is high so the arm can not dissociate from the site by itself, as shown in diagram (2). Since the released strand A_I contains an f domain which can bind to the open end f^* of the trapped arm, strand displacement occurs to take off the strand $f^*m^*0^*$ from the arm, making the arm 0-typed, as shown in diagram (3). In diagram (4), one leg of the spider attaches to site s_r^I, the strand displacement opens the loop, and this leg is trapped. In diagram (5), the opened loop releases the arm trapped on $s_{1\to 0}$, and the other leg attaches to site s_r^{II} releasing strand A_{II}. In diagram (6), strand A_{II} displaces the leg trapped on site s_r^I and the leg on site s_r^{II} can cleave the site at the position labeled with a small cross, thus the spider is free to move forward. Since the legs and the arm of the spider do not have any complementary domains to the current site $s_{1\to 0}$ and s_r^I, the spider cannot move back. The arm type is 0 in diagram (4), (5), and (6).

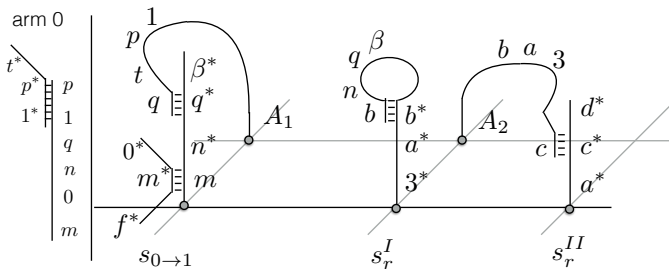


Figure 8.4: Possible implementation I. An implementation of the *switch* mechanism $SW_{0\to1}$. The operating procedure of this mechanism is similar to the one in $SW_{1\to0}$ shown in Figure 8.3. When arm 0 attaches to site $s_{0\to1}$, the strand displacement caused by binding of $0, 0^*$ domains takes off the strand $f^*m^*0^*$ from the site and binds this strand to the arm, the strand displacement caused by the biding of n, n^* domains releases the strand A_1. The released strand A_1 takes off the strand $t^*p^*1^*$ from the arm, switching the type of the arm to 1. The arm is trapped on $s_{0\to1}$, which could be released when one spider leg attaches to site s_r^I. When the arm is released, one spider leg is trapped on s_r^I. When the other leg of the spider attaches to site s_r^{II}, strand A_2 is released, which can release the trapped leg on s_r^I.

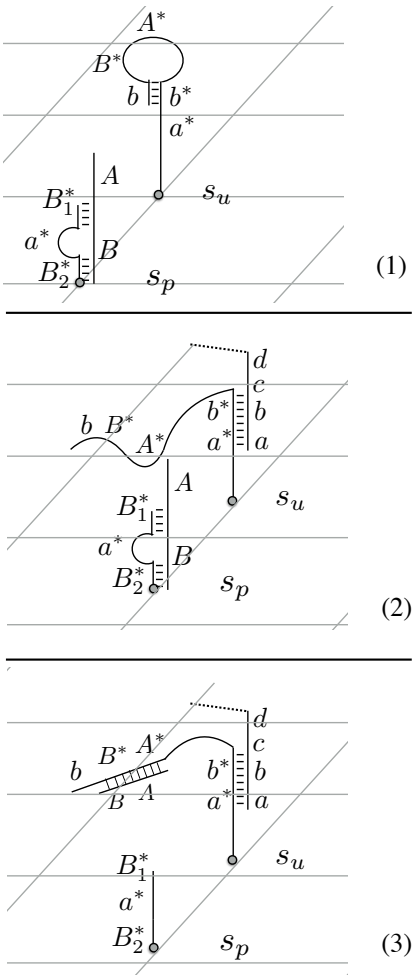


Figure 8.5: Possible implementation I. An implementation of sites s_u and s_p used in the AND gate and the OR gate. Diagram (1) shows the initial condition. Domain a^* forms a small loop on site s_p, which is hard to bind to a spider limb, so a spider cannot attach to site s_p initially. In diagram (2), when a spider leg attaches to site s_u, strand displacement occurs, opening the loop A^*B^*b. The binding parts between the spider leg and site s_u are two pairs of complementary domains, so the binding strength is high enough to trap the spider leg. The opened loop A^*B^*b has the complementary part A on site s_p. In Diagram (3), strand displacement takes place when the opened loop binds to the A domain on site s_p, which removes strand AB from site s_p, thus domain a^* becomes available for the spider limb. The state of site s_p is turned on in diagram (3).

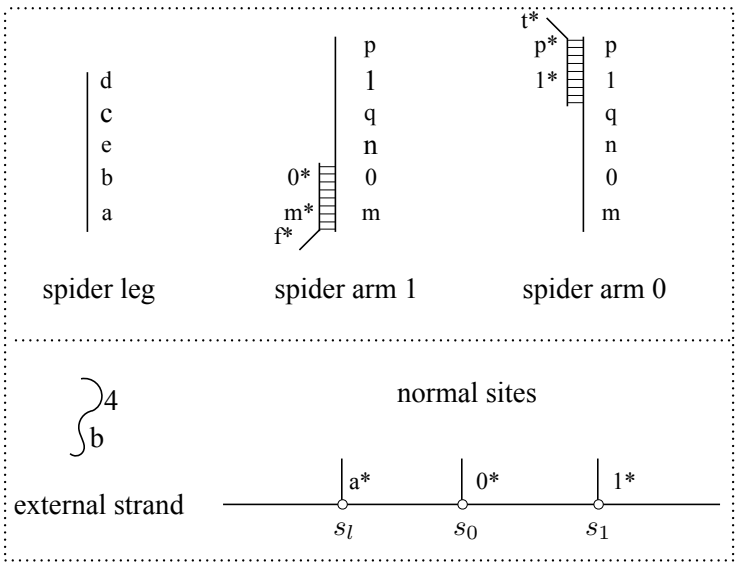


Figure 8.6: Possible implementation II. Implementations of spider limbs, external strands, and normal sites. A spider leg contains five domains. Normal sites keep the same structures as in the first implementation. An excess amount of external strands diffuse freely in the environment. External strands only participate in reactions of DNA strand displacement when a spider is attached to a functional site s_r^{II}.

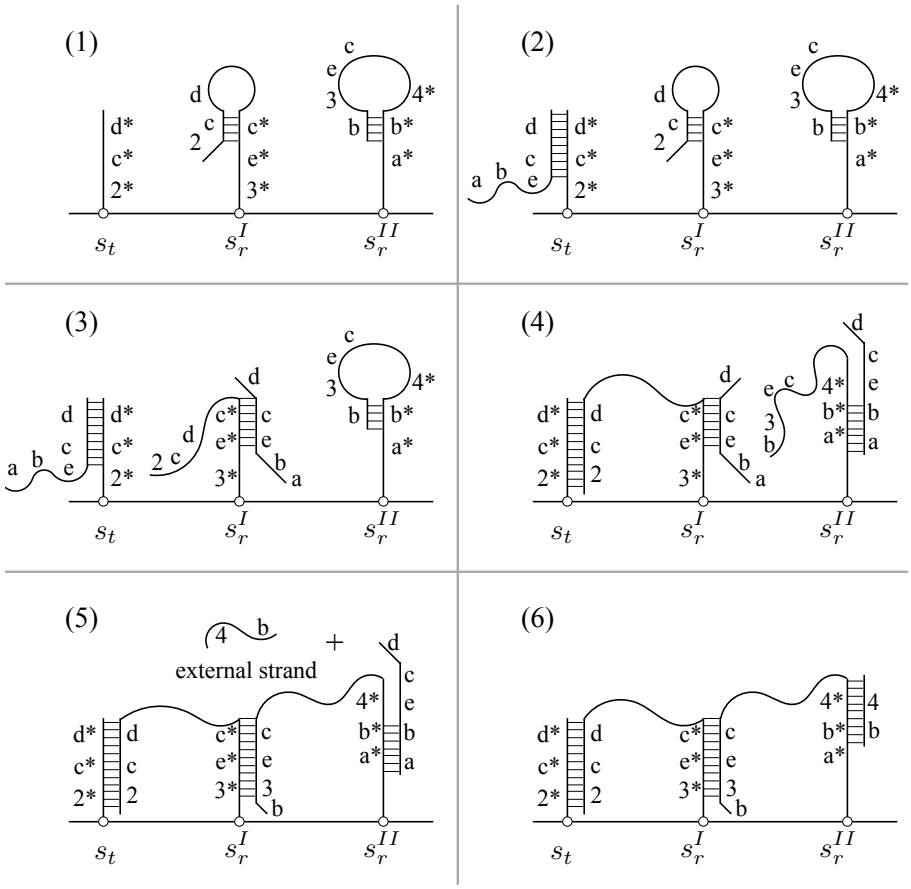


Figure 8.7: Possible implementation II. An implementation of mechanism *exit* using DNA strand displacement. Diagram (1) shows initial conditions. In Diagram (2), a spider leg attaches to site s_t and is trapped on that site. In Diagram (3), when the other leg of the spider attaches to site s_r^I, strand displacement occurs to open the loop of site s_r^I. Spider is trapped on site s_r^I, and a signal containing domains $2, c$ and d can reach the neighboring sites of site s_r^I. In diagram (4), the released signal containing domains $2, c$ and d reacts with site s_t to cause strand displacement, which frees the trapped leg on site s_t. The released leg attaches to site s_r^{II} and is trapped on site s_r^{II}. Strand displacement occurs between the spider leg and site s_r^{II}, domain 4^* becomes active because the loop is opened, and a signal containing domains $b, 3, e$ and c is released. In diagram (5), the released signal containing domains $b, 3, e$ and c reacts with site s_r^I to free the trapped leg. Since domain 4^* is exposed, external strands containing domains $4, b$ can react with site s_r^{II}, which would cause strand displacement. In diagram (6), the trapped leg on site s_r^{II} is released. Both legs of the spider are free to move forward, and the spider cannot move back because sites s_t and s_r^I are blocked.

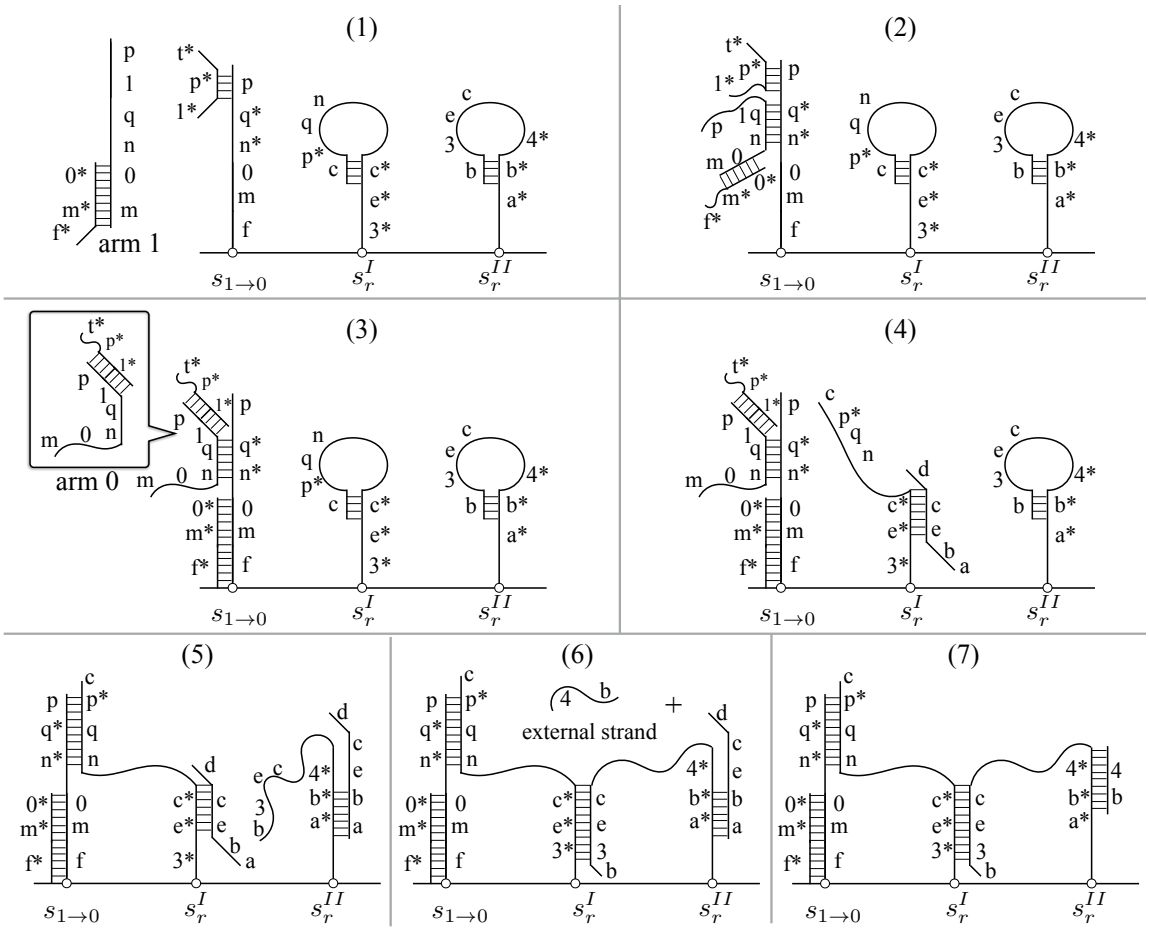


Figure 8.8: Possible implementation II. An implementation of *switch* mechanism $SW_{1\to 0}$. Diagram (1) shows initial conditions. The arm of type 1 binds to site $s_{1\to 0}$ via two complementary domains (q, q^* and n, n^*), and is trapped on site $s_{1\to 0}$ because the binding strength is high, as shown in diagram (2). In diagram (3), two kinds of strand displacements occur between the trapped arm and site $s_{1\to 0}$. Domains p and 1 of the arm strip off the strand having domains $p^*, 1^*, t^*$ from site $s_{1\to 0}$. Domains $0, m$ and f of site $s_{1\to 0}$ strip off the strand having domains $0^*, m^*, f^*$ from the arm. The type of the arm becomes 0 after these two strand displacements. Diagrams (4) – (7) show similar reactions between sites s_r^I, s_r^{II} and spider as the reactions shown in the *exit* mechanism implementation (diagrams (3) – (6) in Figure 8.7). When a spider leaves site s_r^{II}, as shown in diagram (7), its backward route is cut off because sites s_r^I and s_r^{II} are blocked.

dd

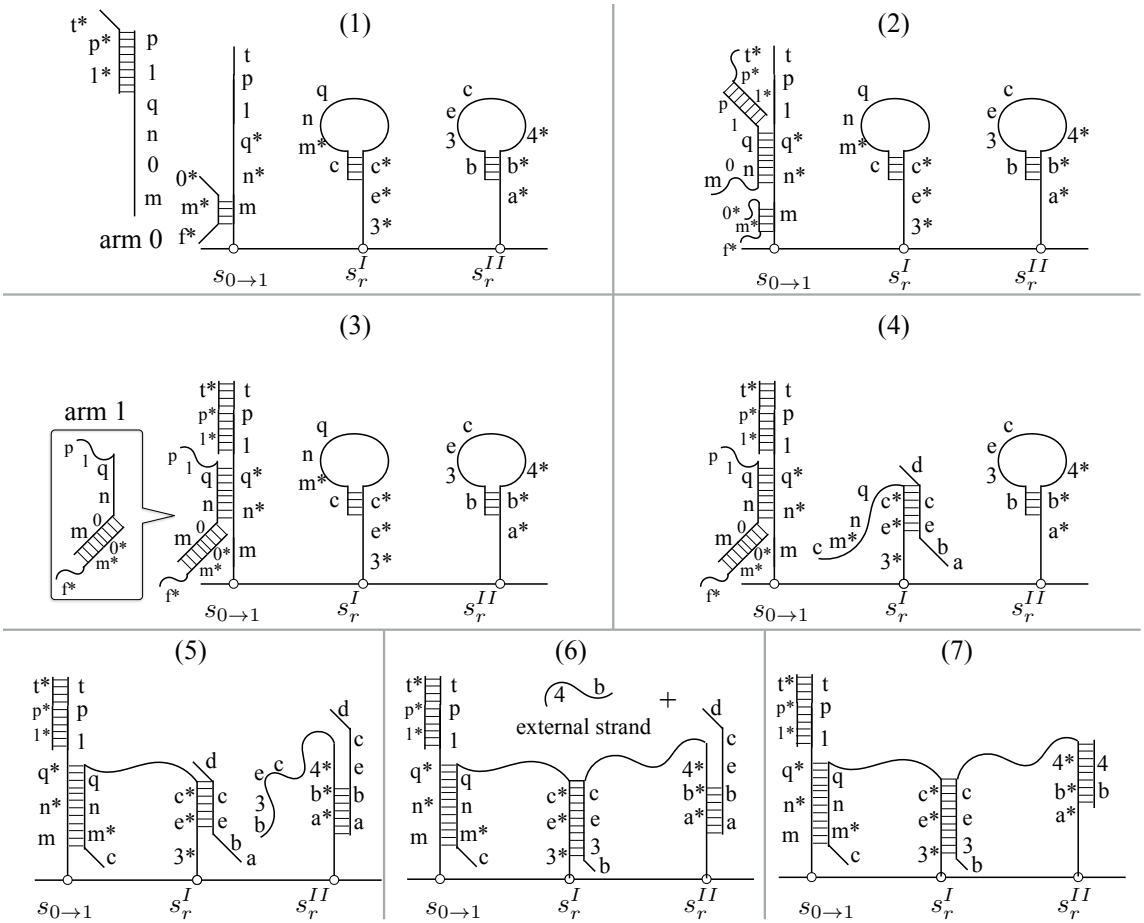


Figure 8.9: Possible implementation II. An implementation of *switch* mechanism $SW_{0\to1}$. Diagram (1) shows initial conditions. The arm type 0 binds to site $s_{0\to1}$ via two complementary domains (q, q^* and n, n^*), and is trapped on site $s_{0\to1}$, as shown in diagram (2). In diagram (3), two kinds of strand displacement occur between the trapped arm and site $s_{0\to1}$. Domains 0 and m of the arm strip off the strand having domains $0^*, m^*, f^*$ from site $s_{0\to1}$. Domains $t, p, 1$ of site $s_{0\to1}$ strip off the strand having domains $t^*, p^*, 1^*$ from the arm. The type of the arm becomes 1 after these two strand displacements. Diagrams (4) – (7) are similar reactions as the procedures shown in the implementation of *switch* mechanism $SW_{1\to0}$ (diagrams (4) – (7) in Figure 8.8). When a spider leaves site s_r^{II}, as shown in diagram (7), its backward route is cut off because sites s_r^I and s_r^{II} are blocked.

Chapter 9

Discussion and Conclusion

Using the extended multi-spider model with spider cooperation and localized signal transmission, we have implemented the basic logic gates (AND, OR, NOT). We have shown how to implement gate cascades, in which each upstream gate G_u is connected to a downstream gate G_d using the *exit* mechanism. We use $O(1)$ types of spiders and sites. To evaluate an n-variable Boolean function that is in 3-CNF with m clauses, the evaluation time is $O(\log m)$ and the size of the circuit is $O(m)$. Therefore, our design supports scalable computation and ensures spatial locality.

Molecular circuits with spatial locality overcome the challenges of computation speed-up and sequence reuse in molecular computing in a well-mixed environment, but there are still other issues. Previous work on tethered circuits [3, 11] spatially isolates different gates on a surface, e.g., a DNA origami tile [20], such that only gates in close proximity can interact with each other, and two computation units that are not adjacent to each other can use the same sequence. However, the tethered circuits [3, 11] lack a direct implementation of the NOT gate, and circuit verification needs to verify many complicated reactions involved in the computation. Previous work [4] has used a walker system to construct logic circuits with spatial locality, but

it lacks modularity and is limited to sequential evaluation due to its design where the circuit constructed is in the form of a *Binary Decision Diagram* (BDD). A walker initially placed at the root node walks along a path unblocked by externally-added strands to reach a leaf node representing True or False, causing a fluorescence change to report the computation result. For practical reasons, this reporting strategy needs two parallel circuits that detect fluorescence change at the True nodes and False nodes respectively to avoid ambiguity. Our design uses the reporting spider to avoid reporting problems [4], and we support parallel evaluation. As a result, to evaluate an m-clause 3-CNF circuit, we need time $O(\log m)$ while the circuit [4] needs time $O(m)$. We use the same linear space complexity $O(m)$ as in the circuit [4], and it is easier to construct large circuits using our design because of its modularity. Compared with previous work [3, 4, 11], our design better addresses the following issues:

Geometric layout. Molecular circuits with spatial locality arrange different computing components on a 2D plane where the distance between different components should be set carefully to avoid interference across components. Reducing the number of gates used in a circuit can ease the geometric layout problem. Our design implements a NOT gate to avoid dual-rail logic conversion used in previous work [3,11], which simplifies the circuit and its layout. Compared with the circuit [4] in a form of BDD where the layout of different branching paths requires appropriate angles and lengths, our design only considers connections between gates because each gate has a fixed layout.

Data encoding. In previous work, variable representation is encoded into the circuit [3,4,11], so each variable corresponds to a distinct sequence. This complicates sequence design if the circuit has a large number of variables. Our design separates variable representation from circuit design, only using two types of spiders placed at different input locations to represent all variables.

Our design surpasses other existing molecular circuits in the aspects of geometrical layout and data encoding, but there are disadvantages existed in our current design compared with other previous work. There is no experimental result showing how likely the extended molecular spider system can be implemented in lab, and we do not know how well the circuit works. We do not implement the fan-out facility. The simulator used in this thesis assumes that the state transition rates in the continuous time Markov model are 1, which can not reflect the real reactions between the molecular spiders and the sites. In our design, the circuit is verified within linear time on the premise that all the mechanisms work properly. If mechanisms do not work properly, we should consider the failure rate of the circuit. We have a further discussion on some of theses disadvantages in Chapter 10.

The performance of our circuit is influenced by the input assignments, the circuit size, and the circuit height (the height of a binary AST representing the circuit). We explore the relation between the circuit performance and the circuit size and circuit height in Chapter 6.

For a large circuit, computation time would span a large range, which makes it hard to predict the overall circuit performance. We explore the relation between circuit predictability and the circuit size, circuit height and total length of all cascading paths in Chapter 7. We propose two methods to improve the circuit predictability in Chapter 7, and we prove the validness of **Method 2** that makes the spiders move one-directionally.

Chapter 10

Future Work

We lack an experimental implementation of our designs, thus here we use a simulator that simulates the circuit at the site level, assuming spiders have equal transition rates to all reachable sites. We have sketched two kinds of implementation where normal sites are short DNA strands so that molecular spiders can attach to or detach from the normal sites freely, and functional sites transmit signals to neighboring sites via strand displacement. In the future, we will complete a plausible implementation and focus on a simulator that can better reflect how different sites react with spiders according to that implementation.

While in this presentation we do not implement or use fan-out, it would be desirable to have that facility. Similar mechanisms as in the NOT gate could be used to implement two-way fan-out. Two previously-trapped spiders with different values are located on two separated paths. The incoming spider selects one of the path and releases the trapped spider. The original spider and the released spider have the same value, and they take two separated paths leading to the different cascaded gates. We shall visit this problem in a future study.

In previous work, we have considered emergent superdiffusive transient behavior

in molecular spiders without functional sites as defined here [27, 30, 31]; it will be interesting to examine how such behaviors can be enriched with suitably placed functional sites.

In circuit verification, we do not consider the failures of mechanisms that are implemented using the hairpin structures in Chapter 8. The loop of a hairpin structure may be opened spontaneously without interacting with other strands. If the signal encoded in the loop is thus released, it may cause unintended changes to the environment, which may cause errors in computation (if a spider with the wrong value reaches the output location) or lead to incomplete computation (if the walking paths for spiders are blocked). In the future, we will explore circuit robustness by incorporating failure of the mechanisms into our model using the following error modeling.

Previous work [4] on the circuit that used a molecular walker system probabilistically analyzed the circuit performance by including possible errors in the model. Since the circuit in this thesis is based on a molecular spider system which is a type of molecular walker system, we will refer to the error modeling method in that previous work [4]. In the circuit based on a walker system [4], there are three kinds of error sources considered in the model. (1) Wrong steering: the pre-blocked sites laid at a junction might be unblocked spontaneously without the externally-added strands, which may direct the walker to the wrong path leading to the incorrect result. (2) Leakage: the walker may fall off from an anchorage site of a path, leaking to step on a neighboring path, which may lead to an incorrect result. (3) Deadlock: there may be no available anchorages within the reach of the walker, which leads to incomplete computation. When a molecular spider selects a path at junction, its arm type determines which path to take. The selection procedure only involves hybridization between two complementary domains, so the molecular spider has very low probability to choose the wrong way, and we can also omit the "wrong steering" error source

in the error modeling. Deadlock may occur when molecular spiders interact with the *exit* mechanism and *switch* mechanism, which is caused by unintended loop opening of the hairpin structures. An incorrect result may be produced in the AND/OR gate if the unblocking site s_u spontaneously opens its own loop and turns on the passive site s_p without interacting with the 1-valued spider. The loop of the passive site s_p may be unblocked spontaneously as well, which may lead to the incorrect computation result. Therefore, the proposed circuit only has one single error source that is caused by the spontaneous opening of the loop structures. This spontaneous loop-opening error source could be modeled into a state transition that could be added to the original Markov chain process. Since the probability that leakage occurs is highly related to the real implementations in lab, we can omit the leakage error source in the theoretical analysis if we do not have the real-data support.

This thesis targets a scalable design of molecular circuits based on extended molecular spider system. This novel design can better address the existing challenges in terms of **geometrical layout** and **data encoding**. Possible implementations for the extended molecular spider system described in this thesis are discussed. This thesis also investigates the computation potential of the molecular spider system, giving other possible applications for such system besides mimicking natural molecular motors.

References

[1] T. Antal and P. L. Krapivsky. Molecular spiders with memory. *Physical Review E*, 76(2):021121, 2007.

[2] C. W. Brown, M. R. Lakin, D. Stefanovic, and S. W. Graves. Catalytic molecular logic devices by DNAzyme displacement. *ChemBioChem*, 15(7):950–954, 2014.

[3] H. Chandran, N. Gopalkrishnan, A. Phillips, and J. Reif. *Localized Hybridization Circuits*, volume 6937, pages 64–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[4] F. Dannenberg, M. Kwiatkowska, C. Thachuk, and A. Turberfield. DNA walker circuits: Computational potential, design, and verification. In D. Soloveichik and B. Yurke, editors, *DNA Computing and Molecular Programming*, volume 8141 of *Lecture Notes in Computer Science*, pages 31–45. Springer International Publishing, 2013.

[5] H. Gu, J. Chao, S.-J. Xiao, and N. C. Seeman. A proximity-based programmable DNA nanoscale assembly line. *Nature*, 465(7295):202–205, 2010.

[6] M. R. Lakin and D. Stefanovic. Supervised learning in an adaptive DNA strand displacement circuit. In *DNA Computing and Molecular Programming*, volume 9211, pages 154–167. Springer, 2015.

[7] H. Lederman, J. Macdonald, D. Stefanovic, and M. N. Stojanovic. Deoxyribozyme-based three-input logic gates and construction of a molecular full adder. *Biochemistry*, 45(4):1194–1199, 2006.

[8] K. Lund, A. J. Manzo, N. Dabby, N. Michelotti, A. Johnson-Buck, J. Nangreave, S. Taylor, R. Pei, M. N. Stojanovic, N. G. Walter, et al. Molecular robots guided by prescriptive landscapes. *Nature*, 465(7295):206–210, 2010.

[9] D. Mo, M. R. Lakin, and D. Stefanovic. *Scalable Design of Logic Circuits Using an Active Molecular Spider System*, volume 9303, pages 13–28. Springer International Publishing, Cham, 2015.

[10] D. Mo, M. R. Lakin, and D. Stefanovic. Logic circuits based on molecular spider systems. *Biosystems*, 2016.

[11] R. A. Muscat, K. Strauss, L. Ceze, and G. Seelig. DNA-based molecular architecture with spatially localized components. In *Proceedings of the 40th Annual International Conference on Computer Architecture (ISCA)*, pages 177–188. ACM, 2013.

[12] M. J. Olah and D. Stefanovic. Multivalent random walkers – a model for deoxyribozyme walkers. In *DNA Computing and Molecular Programming*, volume 6937, pages 160–174. Springer, 2011.

[13] M. J. Olah and D. Stefanovic. Superdiffusive transport by multivalent molecular walkers moving under load. *Phys. Rev. E*, 87:062713, Jun 2013.

[14] T. Omabegho, R. Sha, and N. C. Seeman. A bipedal DNA brownian motor with coordinated legs. *Science*, 324(5923):67–71, 2009.

[15] J. E. Padilla, W. Liu, and N. C. Seeman. Hierarchical self assembly of patterns from the Robinson tilings: DNA tile design in an enhanced tile assembly model. *Natural Computing*, 11(2):323–338, 2012.

[16] R. Pei and S. K. Taylor. Deoxyribozyme-based autonomous molecular spiders controlled by computing logic gates. *IPCBEE Proceedings*, 2009.

[17] R. Pei, S. K. Taylor, D. Stefanovic, S. Rudchenko, T. E. Mitchell, and M. N. Stojanovic. Behavior of polycatalytic assemblies in a substrate-displaying matrix. *Journal of the American Chemical Society*, 128(39):12693–12699, 2006.

[18] L. Qian and E. Winfree. Scaling up digital circuit computation with DNA strand displacement cascades. *Science*, 332(6034):1196–1201, 2011.

[19] M. Rank, L. Reese, and E. Frey. Cooperative effects enhance the transport properties of molecular spider teams. *Physical Review E*, 87:032706, 2013.

[20] P. W. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, 2006.

[21] L. Samii, G. A. Blab, E. H. C. Bromley, H. Linke, P. M. G. Curmi, M. J. Zuckermann, and N. R. Forde. Time-dependent motor properties of multipedal molecular spiders. *Physical Review E*, 84:031111, Sep 2011.

[22] G. Seelig, D. Soloveichik, D. Y. Zhang, and E. Winfree. Enzyme-free nucleic acid logic circuits. *Science*, 314(5805):1585–1588, 2006.

[23] O. Semenov, D. Mohr, and D. Stefanovic. First-passage properties of molecular spiders. *Physical Review E*, 88(1):012724, 2013.

[24] O. Semenov, M. J. Olah, and D. Stefanovic. Mechanism of diffusive transport in molecular spider models. *Physical Review E*, 83(2):021117, 2011.

[25] O. Semenov, M. J. Olah, and D. Stefanovic. Multiple molecular spiders with a single localized source-the one-dimensional case. In *DNA Computing and Molecular Programming*, volume 6937, pages 204–216. Springer, 2011.

[26] O. Semenov, M. J. Olah, and D. Stefanovic. Cooperative linear cargo transport with molecular spiders. *Natural Computing*, 12(2):259–276, 2013.

[27] O. Semenov, D. Stefanovic, and M. N. Stojanovic. The effects of multivalency and kinetics in nanoscale search by molecular spiders. In *WIVACE2012, Italian Workshop on Artificial Life and Evolutionary Computation*, 2012.

[28] W. B. Sherman and N. C. Seeman. A precisely controlled DNA biped walking device. *Nano Letters*, 4(7):1203–1207, 2004.

[29] J.-S. Shin and N. A. Pierce. A synthetic DNA walker for molecular transport. *Journal of the American Chemical Society*, 126(35):10834–10835, 2004.

[30] D. Stefanovic. Maze exploration with molecular-scale walkers. In *Theory and Practice of Natural Computing*, 2012.

[31] D. Stefanovic, M. N. Stojanovic, M. J. Olah, and O. Semenov. Catalytic molecular walkers: Aspects of product release. In *12th European Conference on Artificial Life*, 2013.

[32] M. N. Stojanovic, T. E. Mitchell, and D. Stefanovic. Deoxyribozyme-based logic gates. *Journal of the American Chemical Society*, 124(14):3555–3561, 2002.

[33] M. N. Stojanovic and D. Stefanovic. Chemistry at a higher level of abstraction. *Journal of Computational and Theoretical Nanoscience*, 8(3):434–440, 2011.

[34] M. N. Stojanovic, D. Stefanovic, and S. Rudchenko. Exercises in molecular computing. *Accounts of Chemical Research*, 47(6):1845–1852, 2014.

[35] S. F. Wickham, J. Bath, Y. Katsuda, M. Endo, K. Hidaka, H. Sugiyama, and A. J. Turberfield. A DNA-based molecular motor that can navigate a network of tracks. *Nature Nanotechnology*, 7(3):169–173, 2012.