Enumeration Algorithms for Colored and Rooted Outerplanar Graphs

WANG Jiexun

ENUMERATION ALGORITHMS FOR COLORED AND ROOTED OUTERPLANAR GRAPHS

WANG Jiexun

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Kyoto 606-8501, Japan

January 2010

Doctoral Dissertation

submitted to Graduate School of Informatics, Kyoto University

in partial fulfillment of the requirement for the degree of

DOCTOR OF INFORMATICS

(Applied Mathematics and Physics)

Preface

The problem of enumerating combinatorial objects with certain criteria is a fundamental and important problem in mathematics and theoretical computer science. In the literature, there are mainly three directions in the development of the enumeration of combinatorial objects: 1) to develop delicately mathematical methods for *counting* the number of all objects under criteria, 2) to develop algorithmic methods for *exhaustively generating* objects in a particular combinatorial class without repetition, and 3) to develop algorithms for *randomly generating* an object from a specific class under priori probability. While the first direction has been well studied in early years, the latter two have attracted a lot of attention with the advance of computer in recent years.

This thesis is associated with the second direction: exhaustive generation, which systematically generates all objects of a particular class rather than print out all objects into a paper or a computer file. In early years, several researchers have studied the exhaustive generation of objects in small combinatorial classes. In recent years, more and more questions are asked to generate larger lists of combinatorial objects. With the aid of a computer, it would be possible not only to count but also to list all objects in larger classes without duplications. In many cases, the number of objects under study increases exponentially as the problem size increases. It is high demanding to design efficient algorithms in terms of time and space complexities. Note that the time complexities of such algorithms are measured by the total amount of changes in the data structures, not the time required to print out all objects.

This thesis considers a special but very meaningful exhaustive generation problem which asks to systematically generate all colored and rooted outerplanar graphs with at most given number $n(\geq 1)$ of vertices without repetition. This problem is motivated by the fact that about 94.3% of molecules in the NCI database can be represented as outerplanar graphs [73], where nodes represent atoms and edges represent the bonds between two atoms. The exhaustive list of outerplanar graphs has many applications in various areas such as chemistry, medicine, biology and computer science. However, to the best of our knowledge, few papers have been published for the exhaustive generation of outerplanar graphs in any classes. The reasons are twofold: on the one hand, few researchers notice the extensive applications of the exhaustive list of outerplanar graphs, and on the other hand, it is difficult to design efficient algorithms to generate all outerplanar graphs without repetition because of the symmetry of graphical structures.

In this thesis, we design an efficient algorithm that can systematically generate all required outerplanar graphs in *constant time* per each in the *worse case* with only $O(n)$ space. The proposed algorithm does not require any duplication test when a new graph is generated. The key for designing this efficient algorithm is to choose a "canonical" representation for each colored and rooted outerplanar graph such that the canonical representation of any given outerplanar graph can be obtained from the canonical representation of another outerplanar graph with a constant-size change. This shares the spirit of most of the efficient generation algorithms of rooted trees such as [123].

To be more specific, the basic idea of our algorithm is presented as follows. We first introduce a *canonical* embedding as the representation for each outerplanar graph to avoid duplications. In doing so, the original problem of the thesis reduces to the problem of generating all *canonical* embeddings of colored and rooted outerplanar with at most n vertices. Then, for each canonical outerplanar embedding, we define a unique canonical outerplanar embedding as its parent-embedding so that each pair of parent-embedding and child-embedding has constant-size differences. Based on this relationship, all canonical outerplanar embeddings are arranged into a tree structure, called a *family tree* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a canonical embedding, and each edge in $\mathcal{F}$ corresponds to the parent-child relationship between two canonical embeddings. Finally, we generate all canonical outerplanar embeddings by traversing the family tree $\mathcal{F}$ with the depth-first search. In this way, we can systematically generate all colored and rooted outerplanar graphs without repetition, and moreover, the computation time for the changes between two successive graphs is constant in the worst case and the total space for the entire generation is $O(n)$.

The idea of our algorithm may be applied to the exhaustive generation problems of new families of tractable planar graphs or other classes of decomposable combinatorial structures. These algorithms are very useful and can find wide applications. For example, the exhaustive list of combinatorial objects can be used to search solutions under given constraints. The author hopes that the work in this thesis would be helpful to solve both practical and theoretical problems and stimulate future studies.

January 2010
Jiexun Wang

Acknowledgements

Completing a Ph.D likes a marathon race, and the rival is always myself. I would not have been able to complete this journey without the aid and support of many people over the past three years.

First of all, I am heartily grateful to my advisor, Professor Hiroshi Nagamochi, for recruiting and introducing me to this interesting field—discrete mathematics, for supervising me with great enthusiasm and insightful suggestions, and for spending a lot of time for discussion. The algorithm in this thesis is finally completed based on many versions of imperfect algorithms. I sincerely thank him for his inexhaustible patience in checking those works and in carefully reading the manuscript of this thesis and giving me valuable comments, for suggesting the ways of doing research, and for encouraging me to continue my work. I have made great progress on writing, speaking and presentation under his supervision. His passion on doing research, hard work and time management skill have set an example I hope to attain some day.

I would like to express my sincere appreciation to Professor Liang Zhao who acts as a mentor and friend. I thank him for helpful discussions, for enthusiastically sharing his experiences in living Japan, generously giving me suggestions whenever I meet problems either academically or personally, and for heartily concerning and encouraging me when I was at a low ebb.

I sincerely acknowledge Professor Tatsuya Akutsu for providing me good opportunities of academic communication and financial support, and for carefully reading on my manuscripts and giving critical comments even he is very busy.

I would like to thank Professor Masao Fukushima and Professor Yoshito Ohta for serving on my dissertation committee.

I would also like to thank Professor Takuro Fukunaga for his discussion and thank all members of our Discrete Mathematics Laboratory for the good atmosphere.

Last, but not the least, special gratitude goes to my family. I am forever indebted to my parents for bringing me to this colorful world, loving and supporting me through my entire life. In particular, I am deeply grateful to my boyfriend Dashan for his heartily support, endless patience, and encouragement when it is most required.

Contents

List of Figures

Chapter 1

Introduction

1.1 Combinatorial Enumeration

The problem of enumerating combinatorial objects with certain criteria is a fundamental and important problem in mathematics and theoretical computer science. The combinatorial objects to be enumerated are usually assumed to be discrete and finite such as sequences, subsets, combinations, permutations, partitions, and graphs of specific classes.

In the literature, there are mainly three directions in the development of enumeration theory of combinatorial objects. The first direction is to develop delicately mathematical methods for *counting* the number of all objects under criteria [20, 25, 39, 65, 70, 98, 117, 126, 129, 139, 160]. The second direction is to develop algorithmic methods for *exhaustively generating* combinatorial objects in a particular class without repetition [12, 13, 18, 19, 62, 76, 88, 89, 112, 114, 115, 119, 157, 156, 165, 172]. The third direction is to develop algorithmic methods for *randomly generating* a combinatorial object from a specific class with priori probability of being chosen [10, 11, 17, 34, 55, 69, 77, 81, 94, 111, 130, 132, 135, 176]. The counting of combinatorial objects has been well studied in early years. The exhaustive generation and random generation of combinatorial objects have been limitedly developed due to the restriction of computer and the ignorance of applications in early years. However, with the advance of computer in recent years, the combinatorial generation has been flourished.

1.1.1 Counting

The study of counting combinatorial objects has a long history in mathematical fields. As early as in 1751 or thereabouts, Euler counted the number of faces of a convex polyhedron by the number of the vertices and the number of the edges of the polyhedron. After that, the counting problems of objects of different combinatorial classes have been investigated. A typical problem in counting objects can be formulated as follows:

Counting Problem: Given an integer $n \in \mathbb{N}$, and a finite set S with $|S| = n$, how to count the number $f(n)$ of objects in the set S satisfying specific criteria?

There are four standard ways to calculate the counting function $f(n)$ according to Stanley [160]:

(1) The function $f(n)$ is an elegantly explicit closed formula involving only well-known functions, and free from summation symbols. For example, the number of subsets of the set S is 2^n. The number of the permutations of S is $n!$. A *vertex-labeled* graph is a graph whose vertices are distinguished by labels. The number of vertex-labeled trees with n vertices is n^{n-2} [39].

(2) The function $f(n)$ has a recursive form, which can be given in terms of previous calculated $f(i)$ for $1 \leq i < n$. Take a partition of an integer into a given number of parts for example. A *partition* of an integer n into k parts is a sequence $p_1 \geq p_2 \geq \cdots \geq p_k \geq 1$ such that $n = p_1 + p_2 + \cdots + p_k$. Let $p(n,k)$ be the number of partitions of n into k parts (note that $p(n,k) = f(n)$). It holds

$$p(n,i) = \begin{cases} 1 & \text{for } i = 1 \text{ or } n \\ p(n-1,i-1) + p(n-i,i) & \text{for } 1 < i < n. \end{cases}$$

Then $f(n) = p(n,k)$ can be calculated by this recursive formula.

(3) An approximation $g(n)$ such that $\lim_{n\to\infty} f(n)/g(n) = 1$ for the function $f(n)$ can be derived. For example, the counting problem of *vertex-unlabeled* tree, whose vertices are treated as the same, is much harder than the vertex-labeled case. So far, researchers have not yet found any formula for counting the exact number of unlabeled trees but only for the asymptotic number. Ott [126] proved that the approximation number of unlabeled trees with n vertices is $C\alpha^n n^{-5/2}$, where $C = 0.53495\ldots$ and $\alpha = 2.95576\ldots$.

(4) The most useful but most difficult to be understood method for evaluating $f(n)$ is to give a *generating function*. A generating function $G(x)$ is a formal power series

$$G(x) = \sum_{n=0}^{\infty} f(n)x^n$$

whose coefficients are the sequence $\{f(0), f(1), \ldots, f(n), \ldots\}$. Take the numerical partition for example. The goal is to count the number $f(n)$ of ways of writing an integer n as a sum of positive integers, where the order of these positive integers is not considered. A generating function $G(x)$ for the numerical partition is given by

$$G(x) = \frac{1}{(x)_\infty} = \sum_{n=0}^{\infty} f(n)x^n = 1 + x + 2x^2 + 3x^3 + 5x^4 + 7x^5 \ldots,$$

where $(q)_\infty$ is a q-series [6, 71]. The values of $f(n)$ for $n = 1, 2, 3, 4, 5, \ldots$ are $1, 2, 3, 5, 7, \ldots$, which can be founded by searching "ID A000041" on Sloane's online database [159].

1.1.2 Exhaustive Generation

The goal of the exhaustive generation of combinatorial objects is to systematically generate all objects rather than output all combinatorial objects into a paper or a computer file [32, 74, 96, 119, 168]. We present an abstract formulation for the exhaustive generation problem as follows:

Exhaustive Generation Problem: How to systematically generate all combinatorial objects in a specific class without repetition?

An algorithm for an exhaustive generation problem is also a counting algorithm since each object can be counted as it is generated. However, the reverse is not usually true. The exhaustive generation problem is more challenging than counting problem. In early years, several researchers have studied the exhaustive generation of objects in small combinatorial classes. See Read [138] for a survey of the generation of graphs in small classes. In recent years, more and more questions are raised to generate larger lists of combinatorial objects. With the aid of a computer, it would be possible not only to count but also to list all objects in larger combinatorial classes without duplications.

There are various approaches developed for solving the combinatorial generation problems. The common idea behind them is to first encode each object into a sequence, and then recursively generate larger candidate objects by augmenting elements on smaller objects. Such a strategy is usually named *orderly algorithm* by Read [137, 138]. Note that duplications are not allowed in the list of the generated objects. A naive way to test duplications is to maintain the whole list of generated objects and check whether each newly generated object is in the list or not. However, this method limits the size of generation problems we can solve because the list can be probably extremely long.

The main difference between the existing approaches is to define different orderings of the generation for avoiding the tests of the duplications in the list. In general, there are three popular approaches which can avoid brute-force checking. The first approach is *lexicographical method*, by which all combinatorial objects are arranged in the lexicographical order in terms of their codes [18, 76, 91, 112, 115, 157, 173]. Examples for generating classes of objects in the lexicographical order include combinations [112, 157], permutations [18, 76, 125, 149], set partition [47, 53, 155], and trees [21, 92, 97, 119, 120, 121, 122, 123, 152].

The second well-known approach is *combinatorial Gray codes*, a generalization of Gray codes (also called reflected binary Gray codes), which was applied to mathematical puzzles before they became known to engineers. In 1953, Gray code was patented by Frank Gray, which was used to deal with a communication problem [66]. Gray code is constructed in the following recursive way: starting from 1-bit with the code 0 and 1, respectively; for $n > 1$, constructing the n-bit Gray code by prefixing 0 to all $(n-1)$-bit Gray codes, and then by prefixing 1 to all $(n-1)$-bit Gray codes in the reverse order. Here we illustrate the construction of 3-bit Gray codes. Let G^i ($i \in [1, n]$) be the list of i-bit Gray codes, where the size of the list is 2^i. By definition, $G^1 = [0, 1]$, $G^2 = [00, 01, 11, 10]$ and $G^3 = [000, 001, 011, 010, 110, 111, 101, 100]$. From this illustration, we can see that two successive codes in the list G^i differ only in a single bit. Later Gray codes were applied in various areas such as circuit testing [142], signal encoding [102], data compression [141], statistics [45], and graphical and image processing [7].

Joichi et al. [80] first generalized Gray codes to combinatorial Gray codes, by which the objects are generated as a list such that two successive objects have small structural differences. After that, the combinatorial Gray codes became a popular strategy for generating combinatorial objects such that successive objects differ only slightly [14, 66, 86, 110, 148, 147, 169, 175]. This method has been widely applied for exhaustively generating permutations [78, 167], subsets of a set [22, 35, 51, 52, 124, 146], binary trees [100, 101], spanning trees of a graph [42, 72] and partition of an integer [150]. Savage [151] presented a detailed survey on the study of combinatorial Gray codes.

The third method is *reverse method* introduced by Avis and Fukuda [12]. To explain the basic idea of the reverse search, let G be a connected graph whose nodes represent the objects to be generated, and suppose that we have an objective function to be maximized over all nodes of G. A local search for a given node v in G is designed for exploring the neighborhood node with larger values from the node v in terms of the objective function until no better neighborhood vertex exists. For simplicity, assume that the local search derives a spanning directed tree T of G with a single sink x^* (an optimal solution). Thus if we trace backward T from x^* by systematically reversing its directed edges, then we can generate all objects. A more formal and detailed description of this method can be referred to the paper of Avis and Fukuda [12]. Examples for the reverse method include generating set partitions [84], generating all distributions of n objects to m bins [1] and generating different classes of trees [9, 120, 121, 122, 123].

1.1.3 Random Generation

In some cases, it is impractical to exhaustively generate objects in some combinatorial class. However, it is often useful to generate or sample an object from the class at random. The description of the random generation problem is presented as follows:

Random Generation Problem: How to randomly generate an object in a specific combinatorial class in such a way that each object has a priori probability of being chosen?

Based on the priori probability of object chosen, *uniformly* random generation is to randomly generate all objects with an equal priori probability. Otherwise, it is called *non-uniform* random generation.

The random generation of some classes of combinatorial objects has been studied in the last few years [10, 11, 17, 69, 77, 94, 111]. In the literature, there are mainly three methods for random generation of combinatorial objects: Markov chain method [81, 130, 135, 158], recursive method [124, 135, 175], and Boltzmann method [23, 49, 50, 54].

Markov chain method is a sampling method which constructs a Markov chain such that its equilibrium distribution is the desired distribution. This method was initially used in statistics and has been widely used in combinatorics. By this approach, a Markov chain is constructed in this way: the states of the chain correspond to the combinatorial objects in the class and the equilibrium distribution of the chain converges to the priori probability distribution over these objects. By simulating such a Markov chain for a sufficiently number of steps, an object corresponding to a state can be generated under a probability that is arbitrarily close to the priori probability. Kannan et al. [81] proposed algorithms based on Markov chains for generating bipartite graphs and tournaments. Rao et al. [135] applied the Markov chain method for generating random $(0, 1)$-matrices with given marginal. Milo et al. [113] proposed a Markov chain algorithm for generating a graph with arbitrary degree sequence uniformly at random. Note that a drawback of the Markov chain method is that in some cases, it is difficult to determine how many steps are needed to converge to the stationary distribution within an acceptable error.

Recursive method is another useful strategy which was proposed by Nijenhuis and Wilf [124], and then systematized and extended by Flajolet, Zimmermann and Van Cutsem [55]. Later Zimmermann [178] made a computer package to implement the algorithm of Flajolet et al. [55]. In this method, each object to be generated can be recursively decomposed into smaller parts. This decomposition leads to a set of recursive counting formulas. Given all the necessary recursive counting formulas, we can randomly generate an object in the class by using the reverse operation of the decomposition with the probability by the counting formula. Many researchers applied this method to study the random generation of words of context-free grammars [44, 55], trees [5] and planar maps and convex polyhedra [153].

Boltzmann method is an attractive framework for the random generation of combinatorics, proposed by Duchon, Flajolet, Louchard and Schaeffer [49, 50]. The idea is to assign a probability to each object proportional to an exponential of the size of the object, which relaxes the constraint of generating objects of a strictly fixed size. In other words, random objects under a Boltzmann model have a fluctuating size, but objects with the same size occur with the same probability. Boltzmann model has been applied both in the cases of unlabeled combinatorial objects [54] and labeled combinatorial objects [23, 50].

Finally, we briefly present the relationship between the combinatorial counting and combinatorial random generation problems. Jerrum [77, 158] pointed out that the uniform generation and counting are computationally equivalent. Besides, the problems of approximate counting and almost uniform generation are very closely related. More precisely, for most natural structures, a polynomial time procedure for approximate counting can be used to construct a polynomial time almost uniform generation algorithm, and vice versa. The only assumption we need to make is that the structures are self-reducible, which essentially means that they process a simple inductive construction in terms of similar structures of a smaller size. This implies that known efficient algorithms for counting can be used to obtain fast algorithms for uniform generation based on probabilistic Turing machines.

1.1.4 Computational Complexity

This section first reviews basic definitions of algorithms [93], and then surveys the computational results for combinatorial enumeration algorithms.

An algorithm solves an instance of a problem. In general, the input size is used to characterize the problem instance. Each algorithm consists of the input size and a series of instructions each of which consists of elementary steps including variable assignments, random access to a variable whose index is stored in another variable, conditional jumps (if-then-endif), loops (for-then, or, while-do), and simple arithmetic operations such as addition, substraction, multiplication, division, and comparison. The complexity (or efficiency) of an algorithm is traditionally evaluated by *time complexity* (or *running time*) which is the count of elementary steps and by *space complexity* which is the amount of computer memory during the execution of the algorithm. Usually we do not consider the exact computation of time and space but rather consider a good upper bound on them.

We review traditional and formal definitions of the time and space complexities of an algorithm, respectively, based on random access machine (RAM) model. Let A be an algorithm with input of size n. If there exists a function $T : \mathbb{N} \to \mathbb{N}$ with a constant $C > 0$ such that A terminates the computation after at most $C(T(n))$ elementary steps for each input of size n, then the algorithm A runs in $O(T(n))$ time. We also say that the *time complexity* (or *running time*) of A is $O(T(n))$. If there exists a function $S : \mathbb{N} \to \mathbb{N}$ with a constant $D > 0$ such that there are at most $DS(n)$ number of elementary objects required during the execution of A, then the algorithm A requires $O(S(n))$ space. We also say that the *space complexity* of A is $O(S(n))$. In particular, we say that A runs in *polynomial time* (resp., requires polynomial space) if given the size of input n, there exists an integer k such that A runs in $O(n^k)$ time (resp., requires $O(n^k)$ space). Especially if $k = 0$, then A runs in *constant time* (resp., requires constant space), i.e., the running time (resp., space) of A is independent of the input size n.

Note that the running time (resp., space) may vary according to different instances with the same input size n of the same problem. There are three measures for time (resp., space) complexity of algorithm. *Worst-case running time* (resp., *Worst-case space*) is the longest time (resp., most space) that the algorithm will use over all instances; *best-case running time* resp., *best-case space*) is the shortest time (resp., least space) that the algorithm will use over all instances; and *average-case running time* (resp., *average-case space*) is the average time (resp., space) that the algorithm will use over all instances, which relies on the probability distribution of instances of the problem. Note that usually people have no interest in the complexity measure in the best-case.

We are ready to review the complexities of combinatorial enumeration algorithms. Recall that for the combinatorial counting, most of researchers have focused on developing mathematical methods but not concerned on the computational aspect; and that for the combinatorial generation, researchers have been interested in exploring algorithmic methods and analyzed the complexities of algorithms. In the following, we will review the computational results of exhaustive generation algorithms and random generation algorithms, respectively.

For the combinatorial exhaustive generation, the running time of an algorithm reflects the total amount of changes in the data structures, but not the time required to print out all objects. Many researchers are interested in the amount of computation per each object in an amortized sense (i.e., the total amount of computation divided by the number of objects). This amortized computation time varies greatly. The amortized time of slow algorithms can be exponential of the input size. For example, a naive method for generating all subsets of a given set would require an exponential time of the input size for generating one subset. However, the amortized time of a faster algorithm can be polynomial with respect to the input size (called *polynomial delay*) such as [8, 68, 133, 134]. Especially, an algorithm is called a *constant amortized time* (CAT) algorithm if the delay between two successive outputs is constant. In the literature, CAT algorithms have been proposed for many classes of objects such as numerical partition [150], combinations [52, 124, 162], parenthesis strings [131, 148, 171], multiset permutations [163] and various types of trees [97, 123, 177].

For the combinatorial random generation, the time and space complexities of algorithms depend on the types of objects to be generated. Aronld et. al. [10] proposed an $O(n)$ time algorithm to generate balanced parenthesis strings uniformly at random. Bodirsky et al. [27] firstly designed an expected exponential time algorithm to generate labeled planar graphs uniformly at random. Bodirsky et. al. [30] designed algorithms for generating labelled and unlabeled outerplanar graphs with n vertices uniformly at random in polynomial time in n.

1.1.5 Applications

Counting results of combinatorial objects are frequently used to calculate probabilities. For example, the determination of the number of independent sets has several applications in statistical physics and in the estimation of the degree of reliability in communication networks.

Within the mathematical sciences, researchers are constantly trying to find patterns hidden in the structure of combinatorial objects. The growing trend of using computers and algorithms to produce lists of such objects allows researchers to obtain more information about the objects themselves. Often, this leads to a more thorough understanding of an object which may lead to new and interesting discoveries.

Exhaustive list of combinatorial objects in specific class can be used to get samples by randomly choosing from the list. Besides, some of important combinatorial optimization problems are called *NP-hard* problems, for which no polynomial-time algorithm is known. So far nobody can prove the non-existence of polynomial-time algorithms for the NP-hard problems. For these difficult combinatorial optimization problems, we can check all objects in the exhaustive list satisfying the constraints, until finding the optimal solution.

Besides, the random generation can be used to generate test data for other algorithms on these objects, or to experimentally verify conjectures about properties of this class. We could also use it to evaluate the average-case running times of algorithms on random instances.

1.2 Graphical Enumeration

Graph is a special class of combinatorial objects. This section will present the existing results for three problems in graphical enumeration: counting, exhaustive generation and random generation, respectively, after reviewing basic definitions in graph theory.

1.2.1 Preliminary

We will review some basic definitions in graph theory by Diestel [46].

A *graph* is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$, where the elements of V are the *vertices*, and the elements $e = (x, y)$ of E are the *edges*. The vertex set of G is referred to as $V(G)$, and its edge set as $E(G)$. Let $|V(G)|$ and $|E(G)|$ be the number of vertices and edges in G, respectively. Two vertices x and y of G are *adjacent* if (x, y) is an edge of G. The edge $e = (x, y)$ is *incident* to its end-vertex x or y. Pairwise non-adjacent vertices or edges are called *independent*. A set of vertices is called *independent set* if no two vertices of the set are adjacent.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs. We say that G and G' are isomorphic if there exists a bijection $\varphi : V \to V'$ with $(x, y) \in E \Leftrightarrow \varphi(x)\varphi(y) \in E'$ for all $x, y \in V$.

A graph $S = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. Especially, S is a *spanning* subgraph of G if $V' = V$. A *path* $P = (V', E')$ of $G = (V, E)$ is a subgraph of G such that $V' = \{v_0, v_1, \ldots, v_k\} \subseteq V$ and $E' = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)\} \subseteq E$, where all v_i for $i = 1, 2, \ldots, k$ are distinct. A *cycle* $C = (V', E')$ of $G = (V, E)$ is a subgraph of G such that $V' = \{v_0, v_1, \ldots, v_k, v_0\} \subseteq V$ and $E' = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k), (v_k, v_0)\} \subseteq E$, where all v_i for $i = 1, 2, \ldots, k$ are distinct. A *clique* in $G = (V, E)$ is a subgraph of G such that any two of its vertices are adjacent.

A non-empty graph G is called *connected* if any two of its vertices are linked by a path in G. Otherwise, G is called *disconnected*. A graph G is called k*-connected* (for $k \in \mathbb{N}$) if $|V(G)| > k$ and for any subset X of V with $|X| < k$, the graph obtained from G by removing all vertices of X and their incident edges remains connected.

A *directed* graph (or *digraph*) is a pair (V, E) of disjoint sets (of vertices and edges) with two maps, head: $E \to V$ and tail: $E \to V$ assigning to every edge e a head-vertex head(e) and a tail-vertex tail(e). The edge e is said to be directed from the vertex head(e) to the vertex tail(e).

Note that an undirected or directed graph may have several edges between the same two vertices x and y. Such edges are called *multiple edges*. A *loop* is an edge whose endvertices are the same vertex. A graph is a *multigraph* if it has multiple edges.

A *planar* graph is a graph that can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. A *plane embedding* is a graph that has been drawn on the plane without creating any edge-edge crossing.

1.2.2 Counting

To our best knowledge, the study of graphical counting can be tracked to Euler who counted the number of faces of a convex polyhedra. The major activity in graphical enumeration (mainly counting) was started in the preceding two centuries. Cayley [39] in 1889 counted the number of labeled trees with n vertices is n^{n-2}. Even earlier, Kirchhoff found the number of spanning trees in a given connected graphs, whose result was more general than that of Cayley [70]. However, the counting problem for the unlabeled trees is much harder than labeled one because there is more symmetry involved. So far, researchers have not found any formula for the exact number of unlabeled trees, but only has derived formulas for the asymptotic number [126].

Since Cayley [70], a great number of researchers have studied counting problems of graphs in more complicated classes. The counting problems for labeled graphs are more manageable than unlabeled ones because there is less symmetry involved. A well studied graph class is the class of labeled planar graphs. The exact and asymptotic number of labeled planar embeddings has been studied intensively [25, 24, 29, 65, 170], starting with Tutte's work on the number of rooted oriented planar maps [166]. The number of labeled 3-connected plane embeddings is related to the number of labeled 3-connected planar graphs [117] since a 3-connected planar graph has a unique embedding on the sphere [174]. Bender et al. [20] used this property to count labeled two-connected planar graphs and Gimenez and Noy [65] extended this work to the asymptotic enumeration of labeled planar graphs.

On the other hand, the pioneering work for the counting problems of unlabeled graphs was completed by Redfield [70, 140]. However, this work had not been noticed for about thirty years, but the counting problem of unlabeled graphs was solved independently by several mathematicians including Davis, Gleadson, Golomb, Slepian and Pólya [129]. In particular, Pólya evaluated the number of graphs by using alternating groups and symmetric groups. The classical enumeration theorem of Pólya can give the complete generating function for a class of graphs in terms of a cycle index and a polynomial. Comparing with other mathematicians' methods, Pólya's method is easier to be applied to most graphical problems, and hence becomes the most powerful tool in counting unlabeled graphs. For the class of unlabeled planar graphs, so far, the exact and asymptotic numbers of general graphs have not been known. An exception is that the exact numbers of unlabeled rooted 2-connected planar graphs and unlabeled rooted cubic planar graphs have been computed by Bodirsky et al. [26] and Gao and Wormald [63].

In recent years, several researchers, especially chemists, have extended the Pólya's powerful method to the counting problems of three-dimensional trees (3D-trees) [56, 57, 58, 103]. Robinson et al. [143] counted the number of stereoisomers of alkanes by modifying Pólya's cycle indices. Fujita wrote a series of papers on the enumeration problems of stereoisomers with three-dimensional tree structures. He developed the proligand method for counting stereoisomers [56, 57, 58, 59].

1.2.3 Exhaustive Generation

The goal of exhaustive generation of graphs in particular class is to systematically generate all graphs in the class without repetition. Tree is a connected graph with the simplest structure. To our best knowledge, the earliest work for exhaustive generation of rooted trees was done by Scions [152]. He represented a rooted tree by a depth sequence which recorded the depths (i.e., distance to the root) of each node in the depth-first search order, defined a unique "canonical" tree to be the tree among isomorphic rooted trees with the maximal depth sequence, and then generated the depth sequence of canonical trees in the lexicographical order. Based on the similar idea, Rusky and Hu [144] designed algorithms for all generating binary trees lexicographically as a list, and they were the first who proved that their algorithms satisfied CAT property, that is, they can generate all binary trees in constant time per each tree. Later Rusky [145] generalized a CAT algorithm for k-ary trees from the work of binary-tree of Rusky and Hu [144]. Beyer and Hedetniemi [21] generalized the work of Ruskey [145] for generating k-ary trees lexicographically to generating all rooted trees with n vertices, and also proved that their algorithms are CAT. Many algorithms have been developed for generating all free trees and their variants [97, 120, 121, 122, 123].

Most of efficient algorithms of various types of trees share a common idea. Take the generation of rooted unordered trees for example. One first defines a unique embedding for each rooted tree as its canonical representation, and then defines a parent-child relationship among all canonical representations, which is implicitly represented as the *family tree* each of whose nodes corresponds to the canonical representation of a colored and rooted tree. Then all canonical representations are enumerated one by one according to the depth-first traversal of the family tree in such a way that a new one is generated by attaching a new leaf vertex to the immediately previous output and/or by deleting a few leaf vertices from the previous one. The algorithms output only the constant-size difference between two consecutive trees in the series of all canonical representations, achieving a constant time enumeration per each output. The crucial point in the above enumeration approach is a choice of canonical representation such that the canonical representation of a rooted tree can be obtained from the canonical representation of another rooted tree with a constant-size change. Nakano and Uno [123] presented such a canonical representation as a unique embedding based on the fact that isomorphic duplication can be prevented by maintaining the order of subtrees at each vertex uniquely.

Later many researchers have intensively studied the exhaustive generation problems of graphs in more complicated classes such as cubic graphs [28, 36], series-parallel graphs [85] and monotonic graphs [134]. The general scheme for generating all non-isomorphic graphs in a specific class is similar with the tree case, that is, first define a canonical code to uniquely identify a set of isomorphic graphs, and then construct a canonical code of a graph with larger size from a canonical code of a graph with small size without duplications. Colbourn and Read [41] designed an algorithm base on this scheme to generate all unlabeled graphs in polynomial space but not polynomial delay. Later Goldberg proposed a polynomial delay and polynomial space algorithm for generating unlabeled graph [67]. Besides, there are other studies on the exhaustive generation of different classes of subgraphs of a single graph such as spanning trees [62, 82, 87, 114, 136, 156], cycles [99, 109], maximal cliques [2, 31, 104, 105, 116, 128], and maximal independent sets [37, 79, 83, 95, 165].

1.2.4 Random Generation

The random generation of a class of graphs is to randomly generate or sample a graph from the class under the priority distribution.

The random generation of different classes of graphs has been studied extensively. Nijenhuis and Wilf [124] designed algorithms for randomly generating completely labeled trees and unlabeled rooted free trees. Furnas [61] gave a complete survey of the different methods available for the random generation of several classes of binary trees. Quiroz [132] proposed improved algorithms for the random generation of several classes of trees with the uniform distribution, comparing to the work of Furnas [61] and Nijenhuis and Wilf [124]. Mckay [111] generated a k-regular graph with n vertices uniformly at random in expected time $O(nk^3)$, where $k = O(n^{1/3})$. Bodirsky [27] designed algorithms for uniformly drawing a graph from the class of simple planar graphs at random in time polynomial in the input size. The random generation of acyclic digraphs was studied based on the Markov chain method [107, 108]. Bodirsky [28] presented an expected polynomial time algorithm to generate an unlabeled connected cubic planar graph uniformly at random. Milo et al. [113] presented a Monte Carlo method for the uniform generation of random graphs with arbitrary degree sequences, which was motivated by complex networks. Besides, many papers have been studied on the random generation of different classes of subgraphs of a single graph such as spanning trees [34, 176].

1.2.5 Applications

Graphs are widely used as a modeling tool in various fields such as biology, social science and chemistry, and the graphical enumeration is served as very useful tool for solving the problems in these fields. For example, a protein interaction network can be represented as a simple graph, where nodes represent proteins, and an edge represents the interaction between two proteins. The problem of generating all clusters of densely interacting proteins (called dense modules) is an attractive problem in protein interaction networks. The enumerated dense modules can be useful for functional annotation of previously uncharacterized genes as well as for revealing additional functionality of known genes [64, 164].

In social sciences, a social network can be represented as a simple graph, where nodes correspond to individuals, and an edge corresponds to friendship between two individuals. The problem of listing all maximal groups of people all of whom know each other is one of most interesting problems. The enumerated groups can be used to study modularity or community. Such a group in the social network corresponds to a clique in the graph. Then problem can be formulated to list all maximal cliques in a given graph. Many algorithms for maximal clique enumeration have been applied to the study of social networks [48, 127].

In chemistry, each chemical compound can be represented as a multigraph, where nodes represent atoms and edges represent bonds between atoms. Enumeration of chemical graphs has been an interesting issue to chemists and mathematicians for more than 130 years, and still has been attracting them. Many papers such as [3, 4, 15, 33, 38, 40, 43, 60, 75, 90, 103, 106] have been devoted to the counting or generation of chemical graphs. Especially, Fujiwara et al. [60] applied a tree generation algorithm to infer tree-like chemical graphs under given constraint, and later Ishida et al. [75] proposed improved algorithms for the same problem based on the work of Nagamochi [118]. Implementations of the algorithms due to Fujiwara et al. [60] and Ishida et al. [75] are available as a web server[1].

1.3 Outerplanar Graph Enumeration

1.3.1 Existing Works

An *outerplanar* graph is a planar graph that can be embedded in the plane such that all vertices are in the outer face. Outerplanar graphs are characterized as those graphs not

[1]http://sunflower.kuicr.kyoto-u.ac.jp/~ykato/chem/

containing a subdivision of either K_4 or $K_{2,3}$. Syslo [161] pointed out that outerplanar graphs, after trees, form a next family of easy instances for almost all NP-complete problems on graphs.

Counting results for outerplanar graphs have been found in recent years [25, 24]. Bodirsky et al. proved that the number of labeled outerplanar graphs on n vertices is asymptotically equal to $h{\cdot}n^{-5/2}\sigma^{-n}n!$, where h is computable constant and $\sigma = 0.136593$ [25]. Later Bodirsky et al. considered a more difficult counting problem of unlabeled outerplanar graphs [24]. They counted the exact number g_n of unlabeled outerplanar graphs on n vertices in polynomial time, and derived the asymptotic estimation of g_n by $gn^{-5/2}\rho^{-n}$, where $g \approx 0.00909941$ and $\rho^{-1} \approx 7.50360$.

The random generation of labeled and unlabeled outerplanar graphs has also been well studied by Bodirsky and Kang [30]. To make the correct probabilistic choices in a recursive generation of uniformly distributed outerplanar graphs, they introduced a counting technique using the decomposition of a graph according to its block structure, and computed the exact number of labeled (resp., unlabeled) outerplanar graphs. Then they generated a labeled (resp., unlabeled) outerplanar graph with given number of vertices uniformly at random in the expected polynomial time. However, to our knowledge, few papers have been published for studying the exhaustive generation of outerplanar graphs in any class. This thesis attempts to fill in this gap.

1.3.2 Applications

Outerplanar graph is an important class of combinatorial structures, which has been studied extensively in graph theory. However, few researchers notice its practical application until Horváth [73] found that about 94.3% of the graphs in NCI molecular graph database are outerplanar graphs. In other words, outerplanar graphs can represent the majority of molecules. This result motivates us to investigate the problem of generating all outerplanar graphs in a specific class, which has potential applications in many fields such as drug design.

In practice, to reduce the increasing costs of drug development, pharmaceutical companies have shown great interest in designing new drugs with the aid of computer. It is well known that molecules with similar structures tend to have the same function [154]. The exhaustive generation of molecular graphs with a desired function can speed up the screening procedure of the target molecule.

1.3.3 Challenges

Recall that a CAT algorithm for the exhaustive generation of combinatorial objects in particular class is an optimal algorithm in terms of time complexity. Each CAT algorithm generates all objects without repetition without doing the test of duplications. Clearly the delay between two consecutive objects is not constant if we need to check whether each newly generated object has been generated before.

It is not easy to design a CAT algorithm with polynomial space for outerplanar graphs even for *rooted* outerplanar graphs because of complicated symmetric structures. The difficulty in designing such an algorithm is to how to avoid generation duplications without checking.

1.4 Overview of the Thesis

In this thesis, we study the exhaustive generation problem of simple and connected outerplanar graphs in a specific class as follows:

Problem: Given an integer $n \geq 1$ and a color set $\mathcal{C}$ with $K \geq 1$ colors, how to systematically generate all colored and rooted outerplanar graphs with at most n vertices without repetition?

We take advantage of the graphical symmetry of colored and rooted outerplanar graph, generalize the generation idea of rooted trees by Nakano and Uno [121], and obtain the following result in this thesis:

Result: The above problem admits an $O(n)$ space CAT algorithm.

The general idea of our algorithm is presented as follows. We first introduce a "canonical" embedding as the representative for each outerplanar graph to avoid duplications. In doing so, the original problem of the thesis reduces to the problem of generating all *canonical* embeddings of colored and rooted outerplanar with at most n vertices. Then for each canonical embedding, we define a unique canonical embedding as its parent-embedding such that each pair of parent-embedding and child-embedding have constant differences in terms of graphical structure. Based on the relationship, all canonical outerplanar embeddings can be arranged into a tree structure, called the *family tree* $\mathcal{F}$, where each node in $\mathcal{F}$ corresponds to a canonical embedding. Finally we generate all canonical outerplanar embeddings by traversing the family tree $\mathcal{F}$ in the depth-first search. Equivalently, we systematically generate all colored and rooted outerplanar graphs without repetition, where the computation time for the changes between two successive graphs is constant in the worst case, and the total space for the whole generation is $O(n)$.

The rest of this section is devoted to describe the idea of the algorithm with more details. Chapter 2 reviews some traditional definitions in graph theory and gives our new definitions, both of which are related to the problem under study. Chapter 3 introduces a delicate decomposition of a block in an outerplanar embedding, and gives a vertex-labeling and edge-

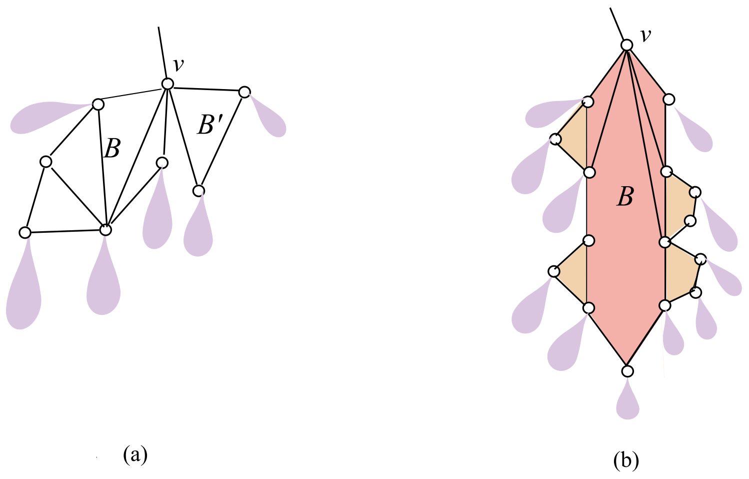

Figure 1.1: (a) An illustration of an ordering for sibling-blocks B and B' rooted at a cut-vertex v from left to right, whose descendant-blocks are shown by light purple; and (b) An illustration of the core, wings and all the descendants of a rooted block B, which are shown by dark pink, light pink and light purple, respectively.

labeling of an outerplanar embedding. Each outerplanar graph consists of blocks. The structure of an outerplanar embedding is fully determined by the structures of rooted blocks and the cut-vertices blocks intersect.

Recall that for a rooted tree, its embedding is determined by an ordering of the sibling-blocks at each cut-vertex, which we call the free symmetry at the cut-vertex, and canonical embeddings of rooted trees are introduced to avoid isomorphic duplication due to the free symmetry at all cut-vertices. See Figure 1.1 (a) for an illustration. For the outerplanar case, we need to avoid additional isomorphic duplications due to possible reflectional symmetry along each rooted block. This makes us to find a desired choice of embeddings inherently more difficult. Our idea to overcome the difficulty is to treat each rooted block as a pair of two rooted trees so that the basic idea for the rooted tree algorithm can be carried over. More specifically, we decompose each rooted block B into three parts: "core," "left wings" and "right wings." The core is a subgraph which is reflectionally symmetric in the block B (except for an assignment of colors to the vertices in the core). The left wings and all their descendant blocks play a role of the first tree rooted at the root of B, and the right wings and all their descendant blocks plays a role of the second tree rooted at the root of B. See Figure 1.1 (b) for an illustration. The core of a block B is shown by dark pink, wings of B are shown by light pink and the descendant blocks of the left and right wings are shown by light purple. By regarding each rooted block in this way, we generate the rooted outerplanar graphs in a similar way of the rooted tree algorithm. We define an integer as the depth of a vertex u in the "core" or the "wing" of the block B, which show the position of the vertex

u in B. In a recursive way, we assign integers to the vertices of all blocks in the embedding. Note that the depths of all vertices of an outerplanar embedding only maintain a partial information of the graphical structure of the embedding. In Chapter 4, we will see that these depths facilitate to fully describe the embedding.

Chapter 4 shows how to encode each outerplanar embedding into a sequence, called *signature*. We first define a parent-child relationship between two outerplanar embeddings, and then introduce the signature for an outerplanar embedding based on its parent-embedding. We define the *parent-embedding* $G' = P(G)$ of an embedding G with $N \in [2, n]$ vertices by removing a "specific" vertex u of G, called an operation $\texttt{remove}(u)$, where the choice of the vertex u among $V(G)$ is unique (note that how to choose such a unique vertex u will be explained in Chapter 4). Accordingly, G is called a *child-embedding* of G', which can be obtained from G' by applying a reverse operation of $\texttt{remove}(u)$. We attempt to define the signature $\sigma(G)$ of an embedding G such that G can be uniquely reconstructed from the signature. For this, we first encode the operation by which we gain G from G' into a sequence as a code $\gamma(u)$ of the vertex u. Based on the code $\gamma(u)$, we know how to attach the vertex u to G' to obtain G. Then we define the signature $\sigma(G)$ by the following recursive formula:

$$\sigma(G) = [\sigma(G'), \gamma(u)].$$

We have proved that the signature $\sigma(G)$ uniquely corresponds to the embedding G.

Chapter 5 chooses a unique embedding as *canonical* from the embeddings of a colored and rooted outerplanar graph. The canonical outerplanar embedding is treated to be the representative of all isomorphic embeddings. The problem studied in the thesis can be converted into the problem of generating all canonical outerplanar embeddings with at most n vertices without duplications. We choose the outerplanar embedding with the maximal code as the canonical embedding in similar with tree case [121, 123]. For the further investigation, we find that such a canonical outerplanar embedding G has two "left-heavy" properties, which are informally described as follows (note that the formal description will be presented in Chapter 5):

(1) *left-sibling-heaviness*: for any two sibling-blocks B and B' rooted at a cut-vertex $v \in V(G)$ from left to right, let $G(B)$ (resp., $G(B')$) be the subgraph of G consisting of the block B (resp., B') and and all its descendant-blocks. It holds that $G(B)$ is "heavier" than $G(B')$, i.e., the signature of $G(B)$ is lexicographically larger than that of $G(B')$; and

(2) *left-side-heaviness*: for any block B in G, all left wings and their descendants is "heavier" than all right wings and their descendants, i.e., the signature of all left wings and their descendants is lexicographically larger than that of all right wings and their descendants.

Chapter 6 explains how to generating all canonical child-embeddings from a given canonical embedding G without repetition and without testing duplication. Based on Chapters 4 and 5, an embedding G' is a canonical child-embedding of G if G' is obtained from G by attaching a new vertex v to an element ε (i.e., vertex or edge) in $V(G) \cup E(G)$ with a vertex code γ and G' satisfies the left-heavy properties. The systematical generation of all canonical child-embeddings of G depends on the determination of all possible elements ε of G (arranged by a sequence $\mathcal{E}^*(G)$) and all possible vertex codes (denoted by Γ). We can easily see that if all these valid elements and vertex codes can be automatically gained, then all canonical child-embeddings of G can be generated systematically without repetition. Fortunately, we characterize the element sequence $\mathcal{E}^*(G)$ and the set Γ of vertex codes such that an embedding obtained from G by applying a vertex code γ in Γ to an element in $\mathcal{E}^*(G)$ is a canonical child-embedding of G. This characterization guarantees that we do not need to check duplications but can generate all canonical child-embeddings of G without repetition.

Chapter 7 describes the algorithm with pseudo-codes and explains its implementation. To explain how to implement the algorithm, we present sufficient and compact data structures for each canonical outerplanar embedding and show the realization of the procedures associated with the data structures. We prove that the algorithm generates all non-isomorphic colored and rooted outerplanar graphs with at most given n number of vertices in constant time per each graph in the worst case and in $O(n)$ space.

Chapter 8 summarizes the results of this thesis, and gives some future directions that deserve further investigation.

Chapter 2

Preliminaries

Throughout the thesis, a graph stands for a simple undirected graph.

A component of a graph is a maximal subgraph in which every two vertices (if exist) are linked by a path (possibly a component consists of a single vertex). A graph is called *connected* if it has only one component. Otherwise, it is called *disconnected*. A vertex in a connected graph is called a *cut-vertex* if its removal results in a disconnected graph. A connected graph with at least three vertices is called *biconnected* if it has no cut-vertex. A maximal connected subgraph of a graph is called a *block* if it has no cut-vertex (i.e., it is biconnected or consists of a single edge or a single vertex). Two blocks in a graph are called *adjacent* if they share a vertex (which is a cut-vertex in the entire graph). By definition, we see that any connected graph can be decomposed into blocks such that any two blocks can share at most one vertex. A block in a graph is called a *leaf-block* if it has at most one adjacent block.

A graph is called *planar* if its vertices and edges can be drawn as points and curves on the plane so that no two curves intersect except for their endpoints. In such a drawing of a planar graph, the plane is divided into several connected regions, each of which is called a *face*. A face is called *outer face* if it is the unbound region, and it is called *inner face* otherwise. By definition, any drawing of a planar graph has only one outer face. A cycle of the graph is called a *facial cycle* if it is the boundary of a face. We call such a cycle the *outer facial cycle* (resp., an *inner facial cycle*) if it is the boundary of the outer (resp., an inner) face. A set F of facial cycles in a drawing defines a combinatorial embedding of a planar graph which gives an order of neighbors of each vertex. A planar graph with a fixed combinatorial embedding is called a *plane* graph if a facial cycle in the embedding is designated as the outer facial cycle. Note that two distinct plane graphs can be isomorphic to the same planar graph, and hence both of them can be treated as planar embeddings (i.e., drawings) of this planar graph. An *outerplanar* graph is a planar graph that admits a plane graph such that all vertices appear on its outer boundary.

A graph with a vertex r designated as the root is called a *rooted* graph or a graph rooted at r. The set of vertices and the set of edges of a graph H are denoted by $V(H)$ and $E(H)$, respectively.

For each block B of a graph rooted at a vertex r, the *root* $r(B)$ of B is defined to be the unique vertex $v \in V(B)$ closest to r. Let $V'(B)$ denote $V(B) - \{r(B)\}$. A block B is called the *parent-block* of all other vertices in $V'(B)$. A vertex u adjacent to a vertex v is called a *child-vertex* of v if u does not belong to the parent-block of v. A block B with $r(B) = v$ is a child-block of v. Let $\mathrm{Ch}(v)$ denote the set of all child-vertices of a vertex v. The *depth* $d(B)$ of a block B is defined by the number of blocks which edge sets intersect with a simple path from a vertex in $V'(B)$ to the root r. For notational convenience, let B_r be an imaginary block which is the parent-block of r_G, and define depth $d(B_r) = 0$. For two blocks B and B' with $r(B') \in V'(B)$, we say that B is the *parent-block* of B' and that B' is a *child-block* of B. Similarly, we define the *ancestor-blocks* and *descendant-blocks*.

Let $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$ be a set of colors. A *colored* graph is a graph in which each vertex v is assigned with a color $c(v) \in \mathcal{C}$ (different vertices can receive the same color). Two colored and rooted graphs H_1 and H_2 are *rooted-isomorphic* if and only if their vertex sets admit a bijection by which the root, the color classes, and the incidence-relation between vertices and edges in H_1 correspond to those in H_2. Let $H_1 \equiv H_2$ means that two colored and rooted graphs H_1 and H_2 are rooted-isomorphic.

A rooted outerplanar graph H can have several different embeddings in the plane. Note that there are two ways of embeddings of a rooted block B in the plane. Also there are $p!$ ways in the orderings of child-blocks of a vertex v, where p is the number of child-blocks of v. An embedding G of a rooted outerplanar graph H is determined by choosing one of the two ways of embeddings of each block and choosing one of the orderings of child-blocks of each cut-vertex. For each block B, let $\ell v(B)$ denote the leftmost vertex in $V(B)$ adjacent to $r(B)$, and for each vertex v, let $\mathcal{B}(v)$ denote a sequence $(B_1, B_2, \ldots, B_k)$ of all child-blocks of v such that $B_1, B_2, \ldots, B_k$ appear in this order from left to right under v. Thus an embedding G of a rooted graph H can be represented by

$$(V(H), E(H), \{\ell v(B) \mid \text{blocks } B \text{ in } H\}, \{\mathcal{B}(v) \mid v \in V(H)\}).$$

Let $\xi(H)$ denote the set of all embeddings of a colored and rooted outerplanar graph H. For two colored and rooted outerplanar graphs H_1 and H_2 (possibly $H_1 \equiv H_2$), we say that two embeddings $G_1 \in \xi(H_1)$ and $G_2 \in \xi(H_2)$ are *rooted-isomorphic* if $H_1 \equiv H_2$ and $G_1 = G_2$. Let $G_1 \equiv G_2$ mean that two embeddings G_1 and G_2 are rooted-isomorphic.

For an embedding $G \in \xi(H)$, $G' \subseteq G$ denote an embedding of a subgraph H' of H such that $G' \in \xi(H')$ is obtained from $G \in \xi(H)$ by deleting the vertices/edges not in H'. For notational convenience, a block B in an embedding G also means the embedding of B that is obtained from G by deleting the vertices/edges not in B. We let $G(B)$ denote the embedding

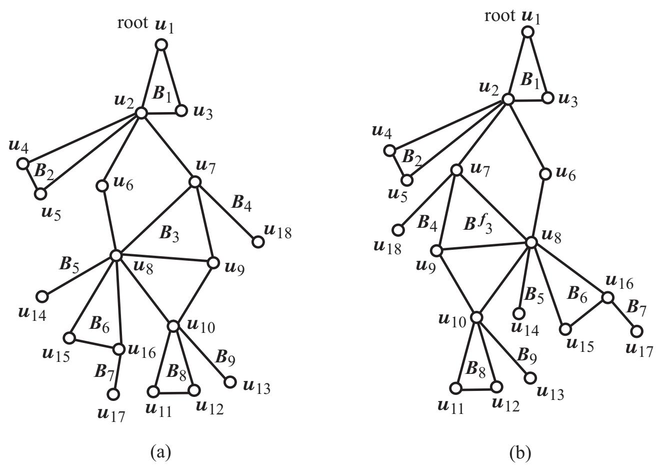

Figure 2.1: (a) An embedding G of a rooted outerplanar graph; (b) The embedding G' obtained by flipping block B_3 of G.

of that consists of embeddings of B and all descendant-blocks of B. For a block B, let $V_{\mathtt{cut}}(B)$ denote the set of cut-vertices of $v \in V'(B)$. For a vertex v, let $G(v)$ denote the embedding obtained from G by deleting the vertices which are not descendants of v. For an embedding G, let G^f denote the *flipped embedding* of G that is obtained by reversing the embedding G on the plane. For example, Figure 2.1(a) shows an embedding G of a rooted outerplanar graph, and Figure 2.1(b) shows the embedding G' obtained by flipping block B_3 in G.

We define an operation of eliminating a vertex u as follows. Let u' and u'' be the vertices adjacent to u.

> $\mathtt{remove}(u)$: remove vertex u together with edges (u, u') and (u, u''), and additionally introduce a new edge (u', u'') if u' and u'' are not adjacent.

Chapter 3

Rooted Outerplanar Graphs

Let $G \in \xi(H)$ be an embedding of a colored and rooted outerplanar graph H, and let r_G denote the root of G. We define the depth $d(r_G) = 0$ for the root r_G, and depth of other vertices in G recursively based on the following decomposition of blocks.

Structure of rooted blocks

For a block B in G, the vertices in $V'(B)$ adjacent to $r(B)$ are called the *head-vertices* of B, and the edges in B incident to $r(B)$ are called the *head-edges* of B. Let $V_{\texttt{head}}(B)$ denote the set of all head-vertices in B, and let $h = |V_{\texttt{head}}(B)|$. We denote the head-vertices in $V_{\texttt{head}}(B)$ by

$$x_1, x_2, \ldots, x_{h/2}, y_{h/2}, y_{h/2-1}, \ldots, y_2, y_1 \quad \text{(if } h \text{ is even)}$$

$$x_1, x_2, \ldots, x_{(h-1)/2}, z, y_{(h-1)/2}, \ldots, y_2, y_1 \quad \text{(if } h \text{ is odd)}$$

from left to right, where $\ell v(B) = x_1$. Define depth of head-vertices to be

$$d(x_i) = d(y_i) = d(r(B)) + i, \quad d(z) = d(r(B)) + (h+1)/2.$$

We will define "axial-faces" and "bottom" of block B as follows. Let h be odd. We call vertex z the *bottom vertex* of B and denote it by $bv(B)$. If $h = 1$, then no axial-face is defined for B. If $h \geq 3$, then an inner face of B containing edge $(r(B), z)$ is called an *axial-face* of B (there are exactly two such faces).

Let h be even. The inner face f_1 of B containing both edges $(r(B), x_{h/2})$ and $(r(B), y_{h/2})$ is called the *first axial-face* of B. If f_1 consists of an odd number of edges, then f_1 has a unique edge e^1 farthest from $r(B)$, and the other inner face containing e^1 (if any) is defined to be the *second axial-face* f_2. For each axial-face f_i, $i \geq 2$, if f_i consists of an even number of edges, then f_i has a unique edge e^i farthest from $r(B)$, and the other inner face containing e^i (if any) is defined to be the $(i+1)$*st axial-face* f_{i+1}. (Note that the definition of axial-faces is independent of choice of embeddings $G \in \xi(H)$.)

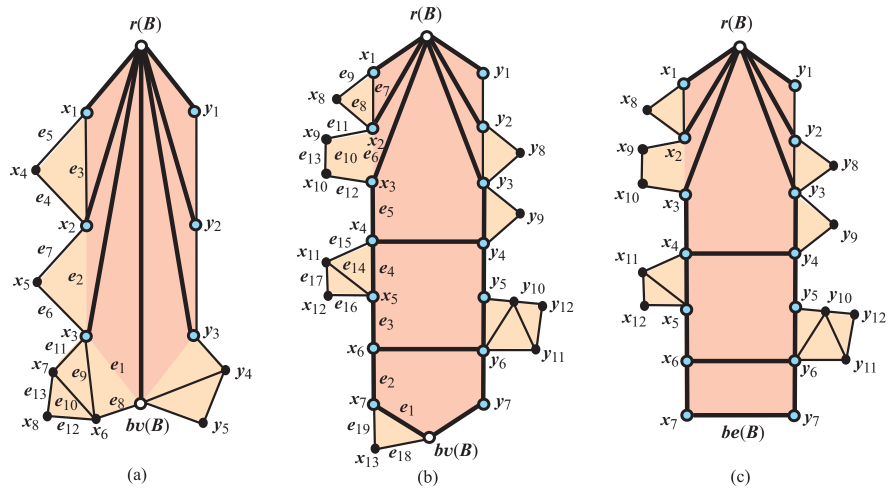

Figure 3.1: Structure of a rooted block.

The non-head-vertices in all axial-faces are called the *axial-vertices*, and the non-head-edges in all axial-faces are called the *axial-edges*. Let $V_{\texttt{axis}}(B)$ denote the set of axial-vertices in B, and $A(B)$ denote the set of axial-edges in B. The depth $d(u)$ of an axial-vertex u is defined to be the number of edges in a shortest path from u to a head-vertex v plus $d(v)$.

The last axial-face f_p has a unique vertex or edge farthest from $r(B)$, which we call the *bottom vertex* of B or *bottom edge* of B, and denote it by $bv(B)$ or $be(B)$, respectively.

A head- or axial-vertex is called a *core-vertex* of B. A non-core-vertex in B is called a *wing-vertex* of B. Let $V_{\texttt{core}}(B)$ and $V_{\texttt{wing}}(B)$ denote core-vertices and wing-vertices in B, respectively.

Any block B has either a bottom vertex $bv(B)$ or a bottom edge $be(B)$, which we call the *bottom* of B, where we let $bv(B) = \emptyset$ (resp., $be(B) = \emptyset$) mean that B has no bottom vertex (resp., edge).

If $h = |V_{\texttt{head}}(B)|$ is odd, where $V_{\texttt{core}}(B) = V_{\texttt{head}}(B)$ and $V_{\texttt{wing}}(B) = \emptyset$, then the *left* (*resp., right*) *side* of B is defined to be the sequence of vertices $x_1, x_2, \ldots, x_{(h-1)/2}$ (resp., $y_1, y_2, \ldots, y_{(h-1)/2}$).

Consider the case where h is even. Let

$$x_1, x_2, \ldots, x_{h/2+1}, \ldots, x_p \text{ (resp., } x_1, x_2, \ldots, x_{h/2+1}, \ldots, x_p, bv(B))$$

be the sequence of core-vertices on the shortest path from x_1 to the bottom if $(x_p, y_p) = be(B)$ (resp., $bv(B)$ exists). We define $y_1, y_2, \ldots, y_p$ (resp., $y_1, y_2, \ldots, y_p, bv(B)$) symmetrically.

Let x and x' be two consecutive core-vertices in the sequence $x_1, x_2, \ldots, x_p, bv(B)$ (possibly $bv(B) = \emptyset$), where $(x, x') = (x_i, x_{i+1})$ for some i or $(x, x') = (x_p, bv(B))$. Removal of these vertices from B leaves at most one subgraph B' which consists of wing-vertices. Let $B(x, x')$ denote such a subgraph B' if any.

We define a unique numbering for the wing-vertices in $B(x, x')$, i.e., the vertices in $V(B(x, x')) - \{x, x'\}$ $(t = |V(B(x, x'))| - 2)$, as the reverse order of the following vertex eliminations. Let w_t be the wing-vertex of degree 2 visited last when we traverse the boundary of $B(x, x')$ from x to x' (see Figure 3.1). Then we eliminate w_t by operation $\texttt{remove}(w_t)$. For each $j \leq t$, let w_{j-1} be the last wing-vertex of degree 2 along the boundary from x to x' in the plane graph obtained from $B(x, x')$ by removing wing-vertices $w_t, w_{t-1}, \ldots, w_j$ in the same manner (see Figure 3.1). This gives an ordering $w_1, w_2, \ldots, w_t$ for the wing-vertices in $B(x, x')$.

We define a unique numbering π for all wing-vertices in $B(x_i, x_{i+1})$, $i = 1, 2, \ldots, p - 1$ (and in $B(x_p, bv(B))$ if $bv(B)$ exists) by visiting $B(x_1, x_2)$, $B(x_2, x_3), \ldots,$ $B(x_{p-1}, x_p)$ (and $B(x_p, bv(B))$ if $bv(B)$ exists) in this order, where we visit the wing-vertices in each $B(x_i, x_{i+1})$ according to the above ordering. The depth of the jth wing-vertex w in π is defined to be $d(w) = d(r(B)) + p + j$ (see Figure 3.1). The sequence of core-vertices $x_1, x_2, \ldots, x_p$ and wing-vertices in the order π is called the *left side* of B. We define the right side of B in the same way (note that $bv(B)$ is not contained in the left or right side of B).

Let $V^{\mathtt{L}}(B)$ and $V^{\mathtt{R}}(B)$ denote the sets of vertices in the left and right sides of B, respectively. A vertex $u \in V^{\mathtt{L}}(B)$ (resp., $V^{\mathtt{R}}(B)$) is called a *left* (*resp., right*) *vertex* of B. Also denote $V^{\mathtt{L}}(B) \cap V_{\mathtt{core}}(B)$ by $V^{\mathtt{L}}_{\mathtt{core}}(B)$. Similarly for $V^{\mathtt{R}}_{\mathtt{core}}(B)$, $V^{\mathtt{L}}_{\mathtt{head}}(B)$ and $V^{\mathtt{R}}_{\mathtt{head}}(B)$ $V^{\mathtt{L}}_{\mathtt{axis}}(B)$, $V^{\mathtt{R}}_{\mathtt{axis}}(B)$, $V^{\mathtt{L}}_{\mathtt{wing}}(B)$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B)$.

For a vertex u in the left side of B, let $P_{\mathtt{L}}(u; B)$ denote the boundary of the left side of B from u to the bottom of B (excluding the bottom edge), and $E_{\mathtt{L}}(u; B)$ denote the sequence of edges in the path $P_{\mathtt{L}}(u; B)$.

We define $P_{\mathtt{R}}(u; B)$ and $E_{\mathtt{R}}(u; B)$ for the right side symmetrically with $P_{\mathtt{L}}(u; B)$ and $E_{\mathtt{L}}(u; B)$.

Let $\tilde{E}(B)$ denote the set of all edges (v, v') with $v, v \in V^{\mathtt{L}}(B) \cup \{bv(B)\}$ or $v, v \in V^{\mathtt{R}}(B) \cup \{bv(B)\}$, where we include edge (v, v') that appears as an edge when we remove the wing-vertex w adjacent to v and v' by $\texttt{remove}(w)$ to define the ordering π, but (v, v') is not an edge in B. A left (resp., right) edge $e = (v, v')$ is an edge such that $\{v, v'\} \subseteq V^{\mathtt{L}}(B) \cup \{bv(B)\}$ (resp., $\{v, v'\} \subseteq V^{\mathtt{R}}(B) \cup \{bv(B)\}$).

We define depth $d(e)$ for all left edges $e \in \tilde{E}(B)$.

Let $L_1 = |V^{\mathtt{L}}_{\mathtt{core}}(B) \cup \{bv(B)\}|$ (possible $bv(B) = \emptyset$), and $L_2 = |V^{\mathtt{L}}_{\mathtt{wing}}(B)|$. For the left wing-vertex w with the largest depth and the two edges e and e' incident to w, where e is closer to x_1 than e' along $P_{\mathtt{L}}(x_1; B)$, we let $d(e) = 2L_2 + L_1 - 1$ and $d(e') = 2L_2 + L_1 - 2$, and then remove w by applying $\texttt{remove}(w)$. We repeat this procedure of assigning pair of numbers

$$(2(L_2-1) + L_1-1, 2(L_2-1) + L_1-2),\ (2(L_2-2) + L_1-1, 2(L_2-2) + L_1-2),\ \ldots,\ (L_1+1, L_1)$$

until no left wing-vertices remain. After removing all left wing-vertices, we assign $d(e_i) = i$ for the ith edge e_i along $P_{\mathtt{L}}(x_1; B')$ when we traverse $P_{\mathtt{L}}(x_1; B')$ reversely from the bottom to the first left head-vertex x_1 in the resulting block (see Figure 3.1). The reason why we define the depth of left edges in this way is to attain the following property. For the left wing-vertex x_i with the *largest* depth i (x_i is the left wing-vertex of degree 2 that appears last along $P_{\mathtt{L}}(x_1; B)$ by definition), let x^* be the vertex that precedes x_i along $P_{\mathtt{L}}(x_1; B)$. Then

$$\begin{array}{l}\text{the sequence } (e_{q+1}, e_q, \ldots, e_1) \text{ of edges in } E_{\mathtt{L}}(x^*; B) \text{ satisfies} \\ d(e_{q+1}) > d(e_q) > \cdots > d(e_1).\end{array} \tag{3.1}$$

We define depth $d(e)$ for all right edges $e \in \tilde{E}(B)$ symmetrically.

Note that the definition of depth of vertices $v \in V(B)$ and edges $e \in \tilde{E}(B)$ are independent of a choice of embeddings B and B^f.

Chapter 4

Signatures of Embeddings

4.1 Tips of Rooted Blocks

We define the "tip" $t(B)$ of a block B as follows.

Let $\{x_i \mid i = 1, 2, \ldots, p_{\mathtt{L}}\}$, $p_{\mathtt{L}} = |V^{\mathtt{L}}(B)|$ (resp., $\{y_j \mid j = 1, 2, \ldots, p_{\mathtt{R}}\}$, $p_{\mathtt{R}} = |V^{\mathtt{R}}(B)|$) denote the set of vertices in the left (resp., right) side of B, where $d(x_i) = d(r(B)) + i$ and $d(y_j) = d(r(B)) + j$.

Case-1 $V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$ (see Figure 4.1(a)): Define $t(B)$ to be the right vertex $y \in V^{\mathtt{R}}_{\mathtt{cut}}(B)$ with the *largest* depth $d(y)$.

Case-2 $V^{\mathtt{R}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$ (see Figure 4.1(b)): Define $t(B)$ to be the right wing-vertex $y \in V^{\mathtt{R}}_{\mathtt{wing}}(B)$ with the *largest* depth $d(y)$. By definition, y is the right wing-vertex $y_{p_{\mathtt{R}}}$ of degree 2 that appears last along $P_{\mathtt{R}}(y_1; B)$.

Case-3 $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$, where possibly $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$ (see Figure 4.1(c)-(d)): Define $t(B)$ to be the left vertex $x \in V^{\mathtt{L}}_{\mathtt{cut}}(B)$ with the *largest* depth $d(x)$.

Case-4 $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$ (see Figure 4.1(e)): Define $t(B)$ to be the left wing-vertex $x \in V^{\mathtt{L}}_{\mathtt{wing}}(B)$ with the *largest* depth $d(x)$. By definition, x is the left wing-vertex $x_{p_{\mathtt{L}}}$ of degree 2 that appears last along $P_{\mathtt{L}}(x_1; B)$.

Case-5 $|V(B)| = 2$ or $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$, where possibly $\mathcal{B}(bv(B)) \neq \emptyset$ (see Figure 4.1(f)-(g)): Define $t(B)$ to be the core-vertex $u \in V'(B)$ with the *largest* depth $d(u)$. Let $t(B)$ be the right endvertex of $be(B)$ if any.

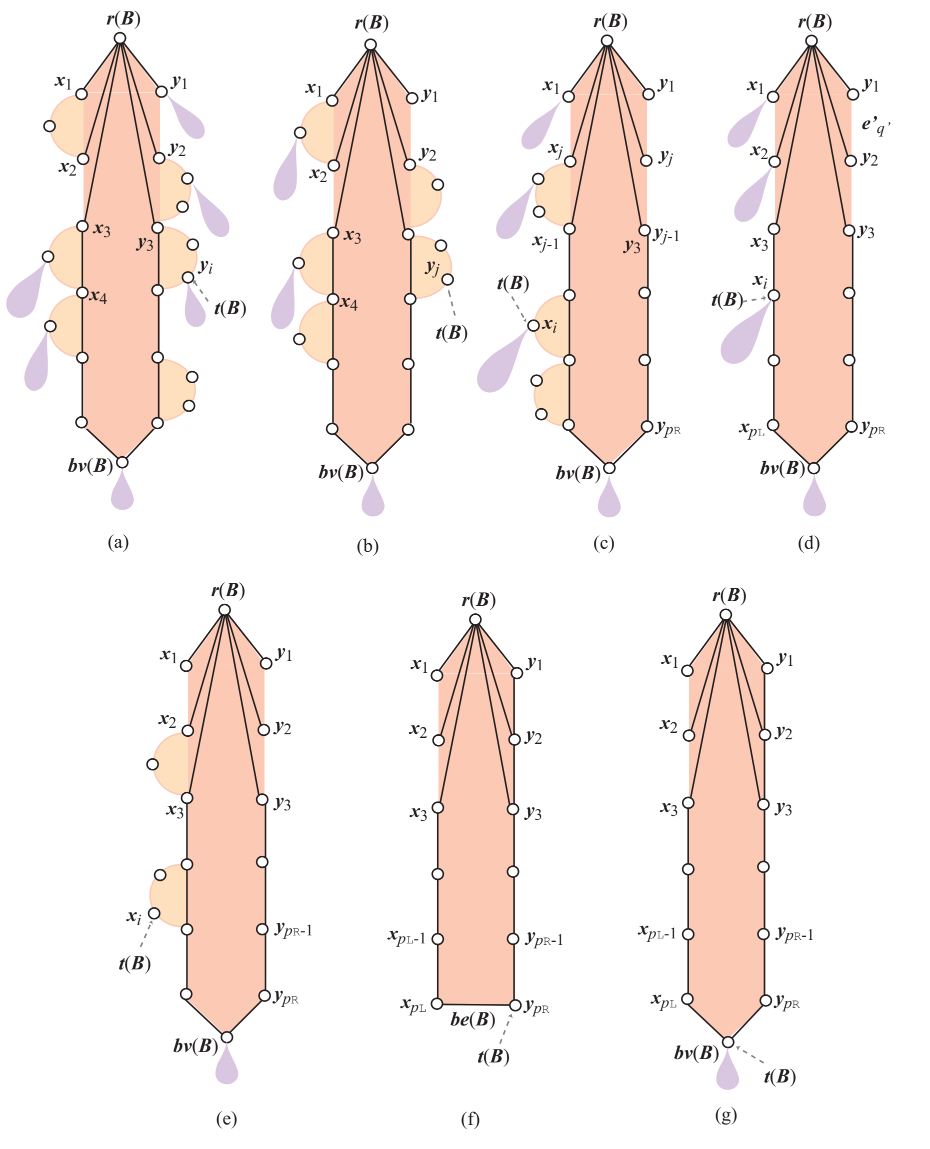

Figure 4.1: Tip $t(B)$ of a rooted block B: (a) Case-1; (b) Case-2; (c) Case-3; (d) Case-3; (e) Case-4; (f) Case-5; and (d) Case-5.

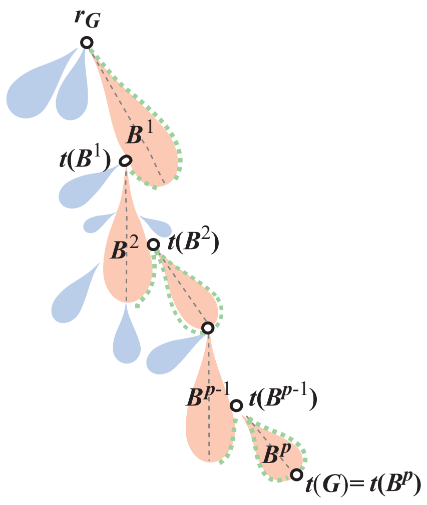

Figure 4.2: An illustration for a sequence of blocks between r_G and $t(G)$, which forms a spine.

For a block B such that $\mathcal{B}(t(B)) \neq \emptyset$, the *successor* of B is defined to be the rightmost block in $\mathcal{B}(t(B))$. The *spine* of G is defined to be the sequence of all successors starting from the rightmost block $B^1 \in \mathcal{B}(r_G)$ by

$$B^1, B^2, \ldots, B^p,$$

where B^1 is the rightmost block in $\mathcal{B}(r_G)$, and each B^i $(i \geq 2)$ is the *successor* of B^{i-1}. See Figure 4.2. The tip $t(G)$ of G is defined to be the tip $t(B^p)$ of block B^p, and the last block B^p is called the *tip-block* of G. Note that the tip-block is not necessarily a leaf-block.

4.2 Parent-embedding and Signature

For an embedding G with $|V(G)| \geq 2$ and $t^p = t(G)$, we define the *parent-embedding* $P(G)$ of G to be the embedding G' of a graph obtained from G by operation $\texttt{remove}(u)$ for $u = t^p$. An embedding G is called a *child-embedding* of G'. A child-embedding G is obtained from $P(G)$ by applying a "reverse operation" of $\texttt{remove}(u)$. We encode such a reverse operation as a "vertex code" $\gamma(u)$, which tells how to attach a new vertex u to $P(G)$ to obtain G. We also define a "signature" of G as a sequence of such vertex codes.

If $|V(G)| = 1$, then we define the *signature* $\sigma(G)$ of G to be a sequence of a single code $c(u)$,

$$\sigma(G) = [c(u)].$$

We define *signature* $\sigma(G)$ of G from the signature $\sigma(P(G))$ of $P(G)$ by attaching a vertex code $\gamma(u)$ of u, i.e.,

$$\sigma(G) = [\sigma(P(G)), \gamma(u)].$$

A vertex code γ is a sequence

$$(d_1, \texttt{at}, d_2, \texttt{op}, c)$$

of five entries such that d_1 and d_2 are nonnegative integers, $c \in \mathcal{C}$,

$$\texttt{at} \in \{\texttt{h}^{\texttt{L}}, \texttt{w}^{\texttt{L}}, \texttt{h}^{\texttt{R}}, \texttt{w}^{\texttt{R}}, *\},$$

called an *attachment-label*, and

$$\texttt{op} \in \{\texttt{new-block}, \texttt{star}, \texttt{triangle}, \texttt{subdivide}\},$$

called an *operation-label*. The vertex code $\gamma(u)$ of a vertex $u = t(G)$ is defined as follows. Let B be the tip-block of G. Note that $t(B) = t^p = t(G)$ and $\mathcal{B}(t^p) = \emptyset$.

(P-1) Let $u = t(G)$ be a head-vertex of B: Let $h = |V'(B)|$.

If $h = 1$, i.e., B consists a leaf edge $(v = r(B), u)$ of G, then for the block B' with $v \in V'(B')$, define

$$\gamma(u) = \begin{cases} (d(B'), \texttt{h}^{\texttt{L}}, d(v), \texttt{new-block}, c(u)) & \text{if } v \text{ is a left vertex of } B' \\ (d(B'), \texttt{h}^{\texttt{R}}, d(v), \texttt{new-block}, c(u)) & \text{if } v \text{ is a right vertex of } B' \\ (d(B'), *, d(v), \texttt{new-block}, c(u)) & \text{otherwise (i.e., } v \in \{r_G, bv(B')\}), \end{cases} \tag{4.1}$$

where $B' = B_r$ with $d(B_r) = 0$ if $v = r_G$.

If $h = 2$, i.e., B consists of a triangle $(r(B), \ell v(B), u)$ of G and $(r(B), \ell v(B))$ is an edge in B, then define

$$\gamma(u) = (d(B), *, d(\ell v(B)), \texttt{triangle}, c(u)).$$

For $h \geq 3$, let v and v' be the vertices in $V'(B)$ adjacent to u, and let $d(v') \geq d(v)$, where (v, v') is an edge e in $P(G)$. Define

$$\gamma(u) = (d(B), *, d(v'), \texttt{star}, c(u)).$$

(P-2) Let $u = t(G)$ be an axial-vertex of B, where u is of degree 2 in G: Let v and v' be the vertices in $V'(B)$ adjacent to u, and let $d(v') \geq d(v)$, where (v, v') is an edge e in $P(G)$, but (v, v') can be an edge in B only when $|V(B)|$ is even. Define

$$\gamma(u) = \begin{cases} (d(B), *, d(v'), \texttt{triangle}, c(u)) & \text{if } u, v \text{ and } v' \text{ form a triangle in } G \\ (d(B), *, d(v'), \texttt{subdivide}, c(u)) & \text{if } v \text{ and } v' \text{ are not adjacent in } G. \end{cases}$$

(P-3) Let $u = t(G)$ be a left wing-vertex of B: Let x and x' be the two vertices in B adjacent to u, where (x, x') is an edge e in $P(G)$ and $e \in \tilde{E}(B)$ holds. Define

$$\gamma(u) = \begin{cases} (d(B), \texttt{w}^{\texttt{L}}, d(e), \texttt{triangle}, c(u)) & \text{if } u, x \text{ and } x' \text{ form a triangle in } G \\ (d(B), \texttt{w}^{\texttt{L}}, d(e), \texttt{subdivide}, c(u)) & \text{if } x \text{ and } x' \text{ are not adjacent in } G. \end{cases}$$

(P-4) Let $u = t(G)$ be a right wing-vertex of B: Let y and y' be the two vertices in B adjacent to u, where (y, y') is an edge e in $P(G)$ and $e \in \tilde{E}(B)$ holds. Define

$$\gamma(u) = \begin{cases} (d(B), \mathtt{w}^{\mathtt{R}}, d(e), \mathtt{triangle}, c(u)) & \text{if } u,\ y \text{ and } y' \text{ form a triangle in } G \\ (d(B), \mathtt{w}^{\mathtt{R}}, d(e), \mathtt{subdivide}, c(u)) & \text{if } y \text{ and } y' \text{ are not adjacent in } G. \end{cases}$$

Let an *element* denote a vertex or an edge. In case (P-1) with $h = 1$, we say that G is obtained from $P(G)$ by creating a new block at B with an application of $\gamma(u)$ to vertex $v = r(B)$ in $P(G)$. In case (P-1) with $h \geq 2$ and cases (P-2)-(P-4), we say that G is obtained from $P(G)$ by expanding block B with an application of $\gamma(u)$ to edge e in $P(G)$. Such a vertex v and an edge e in $P(G)$ are called *applicable* elements in $P(G)$.

4.3 Generating Operations

Let $Ch(G)$ denote the set of all child-embeddings of an embedding G. A child-embedding G' of G has signature $\sigma(G') = [\sigma(G), \gamma(u)]$. We define a graph transformation associated with each type of vertex codes $\gamma(u)$ to construct G' from G. Let $(B^1, B^2, \ldots, B^p)$ be the spine of G. Note that a newly introduced vertex u by an application of $\gamma(u)$ must be the tip of the resulting embedding G'. From this and definition of tips, we can observe the next.

Lemma 4.1. *Any applicable elements in G appear on some block B^i in the spine of G.*

We define a graph transformation for each type of vertex codes as follows. For a block B^h in the spine of G, let e^b denote the bottom edge $be(B^h)$ or the right edge incident to the bottom vertex $bv(B^h)$, where e^b is the unique edge in B^h if $|V(B^h)| = 2$.

- Vertex code $(d_1, \mathtt{at}, d_2, \mathtt{new\text{-}block}, c)$ introduces a new head-vertex u with $c(u) = c$ and a new edge (u, v) for the vertex $v \in V'(B^h)$ with $d(v) = d_2$ of in the block B^h with $d(B^h) = d_1$ in the spine, where `at` needs to satisfy (4.1) for $B' = B^h$. The new edge (u, v) forms a new block B with $d(B) = d_1 + 1$ and $\ell v(B) = u$ in G'.

- Vertex code $(d_1, *, d_2, \mathtt{star}, c)$ deletes the edge $e^b = (v, v')$ with $v, v' \in V(B^h)$ and $d(v) \leq d(v') = d_2$ in the block B^h with $d(B^h) = d_1$ in the spine, and introduces a new head-vertex u with $c(u) = c$ and three new edges $(r(B), u)$, (v, u) and (u, v').

- Vertex code $(d_1, *, d_2, \mathtt{triangle}, c)$ introduces a new core-vertex u with $c(u) = c$ and two new edges (u, v) and (u, v') for the edge $e^b = (v, v')$ with $v, v' \in V(B^h)$ and $d(v) \leq d(v') = d_2$ in the block B^h with $d(B^h) = d_1$ in the spine. Note that $(d_1, *, d_2, \mathtt{triangle}, c)$ is not applied when $|V(B^h)|$ is an even number greater than 2.

- Vertex code $(d_1, *, d_2, \texttt{subdivide}, c)$ deletes $e^b = (v, v')$ with $v, v' \in V(B^h)$ and $d(v) \leq d(v') = d_2$ in the block B^h with $d(B^h) = d_1$ in the spine, and introduces a new axial-vertex u with $c(u) = c$ and two new edges (v, u) and (u, v'). Note that $(d_1, *, d_2, \texttt{subdivide}, c)$ is not applied to the right edge e^b incident to $bv(B)$ if $V'(B) = V_{\texttt{head}}(B)$ and $|V(B)|$ is even.

- Vertex code $(d_1, \texttt{w}^\texttt{L}, d_2, \texttt{triangle}, c)$ (resp., $(d_1, \texttt{w}^\texttt{R}, d_2, \texttt{triangle}, c)$) introduces a new left (resp., right) wing-vertex u with $c(u) = c$ and two new edges (v, u) and (u, v') for the left (resp., right) edge $e = (v, v')$ with $d(e) = d_2$ in the block B^h with $d(B^h) = d_1$ in the spine.

- Vertex code $(d_1, \texttt{w}^\texttt{L}, d_2, \texttt{subdivide}, c)$ (resp., $(d_1, \texttt{w}^\texttt{R}, d_2, \texttt{subdivide}, c)$) deletes the left (resp., right) edge $e = (v, v')$ with $d(e) = d_2$ in the block B^h with $d(B^h) = d_1$ in the spine, and introduces a new left (resp., right) wing-vertex u with $c(u) = c$ and two new edges (v, u) and (u, v') for the edge $e = (v, v')$. Note that $(d_1, \texttt{w}^\texttt{L}, d_2, \texttt{subdivide}, c)$ (resp., $(d_1, \texttt{w}^\texttt{R}, d_2, \texttt{subdivide}, c)$) is not applied to edge e such that (i) $e \in A(B^h)$, or (ii) e is the left (resp., right) edge with the largest depth $d(e)$ incident to the left (resp., right) wing-vertex with the largest depth.

Note that B^h, v and v' in the above cases are uniquely determined by d_1 and d_2.

From the above argument, each type of vertex codes uniquely determines the resulting graph augmented with a new vertex. Therefore, this means that an embedding G can be reconstructed uniquely from its signature $\sigma(G)$ by performing the graph transformations implied by the vertex codes in $\sigma(G)$. Then signature σ has the following property.

Lemma 4.2. *Let G_1 and G_2 be two embeddings of colored and rooted outerplanar graphs H_1 and H_2, respectively. Then $G_1 \equiv G_2$ if and only if $\sigma(G_1) = \sigma(G_2)$.*

Proof. For an embedding $G \in \xi(H)$ of a colored and rooted outerplanar graph H, $\sigma(G)$ is uniquely determined from G independently of the vertex/edge names of H. Hence if $G_1 \equiv G_2$ then $\sigma(G_1) = \sigma(G_2)$. As observed in the above, given a sequence σ of vertex codes, an embedding G with $\sigma(G) = \sigma$ is determined uniquely. Therefore, if $\sigma(G_1) = \sigma(G_2)$ then $G_1 \equiv G_2$. □

Chapter 5

Canonical Embeddings

For two sequences A and B, let $A > B$ mean that A is lexicographically larger then B, and let $A \geq B$ mean that $A > B$ or $A = B$. Let $A \sqsupset B$ mean that B is a prefix of A and $A \neq B$, and let $A \gg B$ mean that $A > B$ but B is not a prefix of A. Let $A \sqsupseteq B$ mean that $A \sqsupset B$ or $A = B$, i.e., B is a prefix of A.

For two embeddings G_1 and G_2 of a graph H, we compare two signatures $\sigma(G_1)$ and $\sigma(G_2)$ by comparing their codes lexicographically code-wise. We compare two vertex codes γ and γ' by comparing their entries lexicographically, treating colors and labels as negative integers such that

$$0 > c_K > c_{K-1} > \cdots > c_1 > * > \mathtt{w}^{\mathtt{L}} > \mathtt{h}^{\mathtt{L}} > \mathtt{w}^{\mathtt{R}} > \mathtt{h}^{\mathtt{R}}$$
$$> \mathtt{subdivide} > \mathtt{triangle} > \mathtt{star} > \mathtt{new\text{-}block}.$$

For each block $B \in \mathcal{B}(v)$, the signature $\sigma(G)$ of an embedding G contains a subsequence which consists of the codes of vertices in $V(G(B)) - \{v\}$, which we denote by $\sigma(G(B); G)$.

Left-sibling-heaviness An embedding G is called *left-sibling-heavy* at a block $B \in \mathcal{B}(v) = (B_1, B_2, \ldots, B_p)$ if $B = B_1$ or

$$\sigma(G) \geq \sigma(G')$$

holds for the embedding G' obtained from G by exchanging the order of B_{i-1} and $B_i = B$ in $\mathcal{B}(v)$.

Lemma 5.1. *An embedding G is left-sibling-heavy at a block $B_i \in \mathcal{B}(v) = (B_1, B_2, \ldots, B_p)$ with $i \geq 2$ if and only if $\sigma(G(B_{i-1}); G) \geq \sigma(G(B_i); G)$ holds.*

Proof. Let G' be the embedding obtained from G by exchanging the order of B_{i-1} and B_i in $\mathcal{B}(v)$. Signatures $\sigma(G)$ and $\sigma(G')$ have a common subsequence before the subsequences $[\sigma(G(B_{i-1}); G), \sigma(G(B_i); G)]$ and $[\sigma(G(B_i); G'), \sigma(G(B_{i-1}); G')]$, respectively.

Note that $\sigma(G(B_i);G') = \sigma(G(B_i);G)$ and $\sigma(G(B_{i-1});G') = \sigma(G(B_{i-1});G)$. Hence $\sigma(G) \geq \sigma(G')$ holds if and only if

$$[\sigma(G(B_{i-1});G), \sigma(G(B_i);G)] \geq [\sigma(G(B_i);G), \sigma(G(B_{i-1});G)].$$

Since the lemma holds when $\sigma(G(B_{i-1});G) = \sigma(G(B_i);G)$, it suffices to show that $\sigma(G(B_{i-1});G) > \sigma(G(B_i);G)$ implies

$$[\sigma(G(B_{i-1});G), \sigma(G(B_i);G)] > [\sigma(G(B_i);G), \sigma(G(B_{i-1});G)], \tag{5.1}$$

and that $\sigma(G(B_i);G) > \sigma(G(B_{i-1});G)$ implies $[\sigma(G(B_i);G), \sigma(G(B_{i-1});G)] > [\sigma(G(B_{i-1});G), \sigma(G(B_i);G)]$. By symmetry, it is sufficient to show the former.

Assume that $\sigma(G(B_{i-1});G) > \sigma(G(B_i);G)$. If $\sigma(G(B_{i-1});G) \gg \sigma(G(B_i);G)$, then we have $[\sigma(G(B_{i-1});G), \sigma(G(B_i);G)] > [\sigma(G(B_i);G'), \sigma(G(B_{i-1});G')]$. We assume $\sigma(G(B_{i-1});G) \sqsupset \sigma(G(B_i);G)$. In this case, $|\sigma(G(B_{i-1});G)| > |\sigma(G(B_i);G)|$ holds, and the $(|\sigma(G(B_i);G)|+1)$st code $\gamma(v)$ in $\sigma(G(B_{i-1});G)$ is compared with the first code $\gamma(x)$ in $\sigma(G(B_{i-1});G)$.

Let B_x (resp., B_v) be the block such that $x \in V'(B_x)$ (resp., $v \in V'(B_v)$). Then the first entry $d_1(x)$ of $\gamma(x)$ is $d_1(x) = d(B_x) = d(B_{i-1}) - 1$, whereas the first entry $d_1(v)$ of $\gamma(v)$ satisfies $d_1(v) \geq d(B_{i-1})$. Hence $\gamma(v) > \gamma(x)$, as required. □

Let $\hat{B} \in \mathcal{B}(r(B))$ denote the sibling preceding B, where we let $\hat{B} = \emptyset$ indicate that there is no such sibling (i.e., B is the leftmost block in $\mathcal{B}(r(B))$). We define the *sibling-state* $\texttt{sbl}(B;G)$ of a block B in G as follows.

$$\texttt{sbl}(B;G) = \begin{cases} \texttt{stc} & \text{if } \hat{B} = \emptyset \text{ or } \sigma(G(\hat{B});G) \gg \sigma(G(B);G) \\ \texttt{pfx} & \text{if } \hat{B} \neq \emptyset \text{ and } \sigma(G(\hat{B});G) \sqsupset \sigma(G(B);G) \\ \texttt{eqv} & \text{if } \hat{B} \neq \emptyset \text{ and } \sigma(G(\hat{B});G) = \sigma(G(B);G). \end{cases} \tag{5.2}$$

Note that an embedding G is left-sibling-heavy at a block B if and only if $\texttt{sbl}(B;G) \in \{\texttt{stc}, \texttt{pfx}, \texttt{eqv}\}$. If $\hat{B} \neq \emptyset$ and $\texttt{sbl}(B;G) = \texttt{stc}$, then the first pair of codes $\gamma(\hat{v}) \in \sigma(G(\hat{B});G)$ and $\gamma(v) \in \sigma(G(B);G)$ such that $\gamma(\hat{v}) > \gamma(v)$ is called the *witness pair* of $\texttt{sbl}(B;G)$, and the vertex $v \in V(G(B))$ is called the *witness vertex* of $\texttt{sbl}(B;G)$.

For a block $B_i \in \mathcal{B}(v) = (B_1, B_2, \ldots, B_q)$ and a vertex $u \in G(B_i) - \{r(B_i)\}$, let $k(u;v)$ denote the integer such that $\gamma(u)$ appears as the kth vertex code in $\sigma(G(B_i);G)$. Let $\mu^-(u;v)$ denote the vertex $w \in V(G(B_{i-1}))$ with $k(w;v) = k(u;v)$, and let $\mu^+(u;v)$ denote the vertex $w \in V(G(B_{i+1}))$ with $k(w;v) = k(u;v)$, where we let $\mu^-(u;v) = \emptyset$ and $\mu^+(u;v) = \emptyset$ if no such vertex w exists.

Left-side-heaviness An embedding G is called *left-side-heavy* at a block $B \in \mathcal{B}(v)$ if

$$\sigma(G) \geq \sigma(G')$$

holds for the embedding G' obtained from G by replacing B with B^f (thus flipping the embedding B along the axis through v and the bottom of B).

The code subsequence $\sigma(G(B);G)$ consists of six subsequences: the first consists of the codes of left or right core-vertices (excluding $bv(B)$), the second consists of the code of the descendants of the bottom vertex $bv(B)$ (if any), the third consists of the code of left wing-vertices, the fourth consists of the code of descendants of left vertices, the fifth consists of the code of right wing-vertices, and the sixth consists of the code of descendants of right vertices. We denote these subsequences by $\sigma_{\tt core}(G(B);G)$, $\sigma_{\tt b}(G(B);G)$, $\sigma^{\tt L}_{\tt wing}(G(B);G)$, $\sigma^{\tt L}_{\tt dscd}(G(B);G)$, $\sigma^{\tt R}_{\tt wing}(G(B);G)$, and $\sigma^{\tt R}_{\tt dscd}(G(B);G)$, respectively.

The subsequence of $\sigma(G(B);G)$ consisting of $\sigma^{\tt L}_{\tt wing}(G(B);G)$ and $\sigma^{\tt L}_{\tt dscd}(G(B);G)$ (resp., $\sigma^{\tt R}_{\tt wing}(G(B);G)$ and $\sigma^{\tt R}_{\tt dscd}(G(B);G)$) is denoted by $\sigma_{\tt L}(G(B);G)$ (resp., $\sigma_{\tt R}(G(B);G)$).

Let $\sigma^{\tt L}_{\tt core}(G(B);G)$ (resp., $\sigma^{\tt R}_{\tt core}(G(B);G)$) denote the sequence obtained from $\sigma_{\tt core}(G(B);G)$ by eliminating the codes of right (resp., left) core-vertices and of the bottom vertex $bv(B)$ (if any) after deleting the first four entries of each code in $\sigma_{\tt core}(G(B);G)$, respectively. Thus, $\sigma^{\tt L}_{\tt core}(G(B);G)$ (resp., $\sigma^{\tt R}_{\tt core}(G(B);G)$) is the sequence of color entries of left (resp., right) core-vertices of B.

For each left wing-vertex u of B (resp., a child-vertex $u \in Ch(v)$ of a vertex v in the left side of B), we define the *flipped code* $\overline{\gamma}(u)$ of vertex code $\gamma(u)$ to be the code obtained from $\gamma(u)$ by replacing the second entry ${\tt w}^{\tt L}$ (resp., ${\tt h}^{\tt L}$) with ${\tt w}^{\tt R}$ (resp., ${\tt h}^{\tt R}$). Symmetrically, for each right wing-vertex u of B (resp., a child-vertex $u \in Ch(v)$ of a vertex v in the right side of B), we define the *flipped code* $\overline{\gamma}(u)$ of vertex code $\gamma(u)$ to be the code obtained from $\gamma(u)$ by replacing the second entry ${\tt w}^{\tt R}$ (resp., ${\tt h}^{\tt R}$) with ${\tt w}^{\tt L}$ (resp., ${\tt h}^{\tt L}$). For a notational convenience, we set $\overline{\gamma}(u) = \gamma(u)$ for the other vertices $u \in V(G(B)) - \{r(B)\}$.

Let $\overline{\sigma_{\tt L}}(G(B);G)$ (resp., $\overline{\sigma_{\tt R}}(G(B);G)$) denote the sequence obtained from $\sigma_{\tt L}(G(B);G)$ (resp., $\sigma_{\tt R}(G(B);G)$) by replacing each vertex code $\gamma(u)$ with $\overline{\gamma}(u)$.

Lemma 5.2. *An embedding G is left-side-heavy at a block $B \in \mathcal{B}(v)$ if and only if it holds $[\sigma^{\tt L}_{\tt core}(G(B);G), \sigma_{\tt L}(G(B);G)] \geq [\overline{\sigma^{\tt R}_{\tt core}}(G(B);G), \overline{\sigma_{\tt R}}(G(B);G)]$.*

Proof. Let G' be the embedding obtained from G by replacing B with B^f. Signatures $\sigma(G)$ and $\sigma(G')$ have a common subsequence before their subsequences $[\sigma_{\tt core}(G(B);G), \sigma_{\tt L}(G(B);G), \sigma_{\tt R}(G(B);G)]$ and $[\sigma_{\tt core}(G(B);G'),\ \sigma_{\tt L}(G(B);G'), \sigma_{\tt R}(G(B);G')]$ start, respectively.

Note that $\sigma_{\tt L}(G(B);G') = \overline{\sigma_{\tt R}}(G(B);G)$ and $\sigma_{\tt R}(G(B);G') = \overline{\sigma_{\tt L}}(G(B);G)$. Also $\sigma_{\tt core}(G(B);G) \geq \sigma_{\tt core}(G(B);G')$ holds if and only if $\sigma^{\tt L}_{\tt core}(G(B);G) \geq \overline{\sigma^{\tt R}_{\tt core}}(G(B);G)$, since $\sigma_{\tt core}(G(B);G)$ is an alternating sequence of vertices in the left and right sides of B. Hence

$$\begin{aligned}\sigma(G) \geq \sigma(G') \Leftrightarrow\ & [\sigma^{\tt L}_{\tt core}(G(B);G), \sigma_{\tt L}(G(B);G), \sigma_{\tt R}(G(B);G)] \\ & \geq [\overline{\sigma^{\tt R}_{\tt core}}(G(B);G), \overline{\sigma_{\tt R}}(G(B);G), \overline{\sigma_{\tt L}}(G(B);G)].\end{aligned} \tag{5.3}$$

For simplicity, let $\sigma^{\tt L}_{\tt core}$ denote $\sigma^{\tt L}_{\tt core}(G(B);G)$. Similarly for $\overline{\sigma^{\tt R}_{\tt core}}$, $\sigma_{\tt L}$, $\sigma_{\tt R}$, $\overline{\sigma_{\tt L}}$ and $\overline{\sigma_{\tt R}}$.

Since (5.3) holds when "$\sigma^{\mathtt{L}}_{\mathtt{core}} \neq \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}$" or "$\sigma^{\mathtt{L}}_{\mathtt{core}} = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}$ and $\sigma_{\mathtt{L}} = \overline{\sigma_{\mathtt{R}}}$," it suffices to show that $\sigma_{\mathtt{L}} > \overline{\sigma_{\mathtt{R}}}$ (resp., $\overline{\sigma_{\mathtt{R}}} > \sigma_{\mathtt{L}}$) implies $[\sigma_{\mathtt{L}}, \sigma_{\mathtt{R}}] > [\overline{\sigma_{\mathtt{R}}}, \overline{\sigma_{\mathtt{L}}}]$ (resp., $[\overline{\sigma_{\mathtt{R}}}, \overline{\sigma_{\mathtt{L}}}] > [\sigma_{\mathtt{L}}, \sigma_{\mathtt{R}}]$). We prove the former (the latter can be treated symmetrically).

Assume $\sigma_{\mathtt{L}} > \overline{\sigma_{\mathtt{R}}}$. If $\sigma_{\mathtt{L}} \gg \overline{\sigma_{\mathtt{R}}}$, then we have $[\sigma_{\mathtt{L}}, \sigma_{\mathtt{R}}] > [\overline{\sigma_{\mathtt{R}}}, \overline{\sigma_{\mathtt{L}}}]$. We assume $\sigma_{\mathtt{L}} \sqsupset \overline{\sigma_{\mathtt{R}}}$. In this case, $|\sigma_{\mathtt{L}}| > |\overline{\sigma_{\mathtt{R}}}|$ holds, and the $(|\overline{\sigma_{\mathtt{R}}}| + 1)$st code $\gamma(x)$ in $\sigma_{\mathtt{L}}$ is compared with the first code $\overline{\gamma}(u)$ in $\overline{\sigma_{\mathtt{L}}}$, where $\overline{\gamma}(u)$ is the flipped code of the first code $\gamma(u)$ in $\sigma_{\mathtt{L}}$. It suffices to show that $\gamma(x) > \overline{\gamma}(u)$, since this shows $[\sigma_{\mathtt{L}}, \sigma_{\mathtt{R}}] > [\overline{\sigma_{\mathtt{R}}}, \overline{\sigma_{\mathtt{L}}}]$. Let B_u (resp., B_x) be the block such that $u \in V'(B_u)$ (resp., $x \in V'(B_x)$), and let $(d_1(u), \mathtt{at}(u))$ and $(d_1(x), \mathtt{at}(x))$ be the first two entries in $\overline{\gamma}(u)$ and $\gamma(x)$, respectively. We show that

$$(d_1(x), \mathtt{at}(x)) > (d_1(u), \mathtt{at}(u)),$$

which implies $\gamma(x) > \overline{\gamma}(u)$.

If u (resp., x) is a left wing-vertex of B, then $B_u = B$, $d_1(u) = d(B)$ and $\mathtt{at}(u) = \mathtt{w}^{\mathtt{R}}$ (resp., $B_x = B$, $d_1(x) = d(B)$ and $\mathtt{at}(x) = \mathtt{w}^{\mathtt{L}}$) by definition of γ.

If u (resp., x) is not a left wing-vertex of B, then $d(B_u) = d(B) + 1$, $d_1(u) = d(B)$ and $\mathtt{at}(u) = \mathtt{h}^{\mathtt{R}}$ (resp., $d(B_x) \geq d(B) + 1$, $d_1(x) \geq d(B)$, and if $d_1(x) = d(B)$ then $\mathtt{at}(x) = \mathtt{h}^{\mathtt{L}}$).

Hence $d_1(x) \geq d_1(u)$. Clearly $d_1(x) > d_1(u)$ implies $(d_1(x), \mathtt{at}(x)) > (d_1(u), \mathtt{at}(u))$. If $d_1(x) = d_1(u)$, then $\mathtt{at}(x) \in \{\mathtt{w}^{\mathtt{L}}, \mathtt{h}^{\mathtt{L}}\}$ and $\mathtt{at}(u) \in \{\mathtt{w}^{\mathtt{R}}, \mathtt{h}^{\mathtt{R}}\}$, implying $(d_1(x), \mathtt{at}(x)) > (d_1(u), \mathtt{at}(u))$. □

For simplicity, denote $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B); G)$ by $\sigma^{\mathtt{L}}_{\mathtt{core}}$. Similarly for $\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}$, $\sigma_{\mathtt{R}}$ and $\sigma_{\mathtt{L}}$. We define the *side-state* $\mathtt{sd}(B; G)$ of a block B in G as follows.

$$\mathtt{sd}(B; G) = \begin{cases} \mathtt{stc} & \text{if "}[\sigma^{\mathtt{L}}_{\mathtt{core}}, \sigma_{\mathtt{L}}] \gg [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}, \overline{\sigma_{\mathtt{R}}}] \\ \mathtt{nil} & \text{if } \sigma^{\mathtt{L}}_{\mathtt{core}} = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}} \text{ and } \sigma_{\mathtt{R}} = \emptyset \\ \mathtt{pfx} & \text{if } \sigma^{\mathtt{L}}_{\mathtt{core}} = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}} \text{ and } \sigma_{\mathtt{L}} \sqsupset \overline{\sigma_{\mathtt{R}}} \neq \emptyset \\ \mathtt{eqv} & \text{if } \sigma^{\mathtt{L}}_{\mathtt{core}} = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}} \text{ and } \sigma_{\mathtt{L}} = \overline{\sigma_{\mathtt{R}}} \neq \emptyset. \end{cases} \tag{5.4}$$

Note that an embedding G is left-side-heavy at a block B if and only if $\mathtt{sd}(B; G) \in \{\mathtt{stc}, \mathtt{nil}, \mathtt{pfx}, \mathtt{eqv}\}$. If $\mathtt{sd}(B; G) = \mathtt{stc}$, then the first pair of codes $\gamma(x) \in [\sigma^{\mathtt{L}}_{\mathtt{core}}, \sigma_{\mathtt{L}}]$ and $\gamma(y) \in [\sigma^{\mathtt{R}}_{\mathtt{core}}, \sigma_{\mathtt{R}}]$ such that $\gamma(x) > \overline{\gamma}(y)$ is call the *witness pair* of $\mathtt{sd}(B; G) = \mathtt{stc}$, and the vertex $y \in V(G(B))$ is called the *witness vertex* of $\mathtt{sd}(B; G) = \mathtt{stc}$.

For each vertex $u \in V(G(B)) - \{r(B)\}$ in a block $B \in \mathcal{B}(v)$ for a vertex v, we define its corresponding vertex $\mu(u; v)$ as follows.

If u is a left (resp., right) core-vertex of B, then let $\mu(u; v)$ be the right (resp., left) core-vertex $w \in V(B)$ with $k(w; v) = k(u; v) + 1$ (resp., $k(w; B) = k(u; v) - 1$).

If $\gamma(u)$ appears in $\sigma_{\mathtt{b}}(G(B); G)$ (i.e., vertex $z = bv(B)$ exists and $u \in V(G(z))$), then $\mu(u; v) = u$.

If $\gamma(u)$ appears as the kth code in $\sigma_{\mathtt{L}}(G(B);G)$ (resp., $\sigma_{\mathtt{R}}(G(B);G)$), then let $\mu(u;v)$ be the vertex $w \in V(G(B))$ such that $\gamma(w)$ appears as the kth code in $\sigma_{\mathtt{R}}(G(B);G)$ (resp., $\sigma_{\mathtt{L}}(G(B);G)$), where we let $\mu(u;v) = \emptyset$ if no such w exists.

An embedding G is called *canonical* if it is left-sibling-heavy and left-side-heavy at all blocks in G.

Lemma 5.3. *Let G be an embedding of a colored and rooted outerplanar graph H. Then G is canonical if and only if $\sigma(G)$ is lexicographically maximum among all $\sigma(G')$ of embeddings $G' \in \xi(H)$.*

Proof. (i) Only if part: Let G be an embedding of H such that $\sigma(G)$ is lexicographically maximum. To derive a contradiction, assume that G is not canonical.

If G is not left-sibling-heavy at some block $B_i \in \mathcal{B}(v) = (B_1, B_2, \ldots, B_p)$, then $\sigma(G(B_i);G) > \sigma(G(B_{i-1});G)$ holds by Lemma 5.1. Hence by the definition of left-sibling-heaviness, the embedding G' obtained from G by exchanging the order of B'_{i-1} and B'_i in $\mathcal{B}(v)$ has signature $\sigma(G')$ which is lexicographically larger than G.

If G is not left-side-heavy at some block B, then it holds $\overline{[\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B);G), \overline{\sigma_{\mathtt{R}}}(G(B);G)] > [\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B);G), \sigma_{\mathtt{L}}(G(B);G)]$ by Lemma 5.2. Hence by the definition of left-sibling-heaviness, the embedding G' obtained from G by replacing B with B^f has signature $\sigma(G')$ which is lexicographically larger than G.

(ii) If part: Let G be a canonical embedding. To prove that $\sigma(G)$ is lexicographically maximum, it suffices to show that a canonical embedding is unique up to rooted-isomorphism $\equiv$, since any embedding with the lexicographically maximum signature is canonical by (i) and each signature represents a rooted-isomorphically unique embedding of H by Lemma 4.2. Let v be a cut-vertex with the largest depth in G. For each block $B \in \mathcal{B}(v)$, its embedding maximizes $\sigma(G(B);G)$ (where $G(B) = B$), and hence the embedding is unique, since any code $\gamma(u)$ is lexicographically larger or smaller than other code $\gamma(v)$ with $\gamma(u) \neq \gamma(v)$ by definition. Also the ordering of blocks $B_1, B_2, \ldots, B_q \in \mathcal{B}(v)$ maximizes $[\sigma(G(B_1);G), \sigma(G(B_2);G), \ldots, \sigma(G(B_q);G)]$, and is unique. By applying the argument in a bottom-up manner along G, we see that a canonical embedding is rooted-isomorphically unique. □

Lemma 5.4. *For a canonical embedding G with $|V(G)| \geq 2$, its parent-embedding $P(G)$ is a canonical embedding.*

Proof. Let $B^1, B^2, \ldots, B^p$ be the spine of G, and let $t^p = t(G)$, and $t^k = r(B^{k+1})$ for $k = 0, 1, \ldots, p-1$. Let $G' = P(G)$.

Since each block B^k is the rightmost one among its siblings in G, $G' = P(G)$ remains left-sibling-heavy at any block other than these blocks B^k, where possibly B^p is eliminated in $G' = P(G)$. For each block B^k which has a preceding sibling $\hat{B}^k$, we have $\sigma(G(\hat{B}^k); G) \geq \sigma(G(B^k); G)$ by Lemma 5.1, since G is left-sibling-heavy at B^k. By the definition of parent-embeddings, we have $\sigma(G(B^k); G) \sqsupset \sigma(G'(B^k); P(G))$, implying that $\sigma(G'(\hat{B}^k); P(G)) \geq \sigma(G'(B^k); P(G))$, i.e., $P(G)$ remains left-sibling-heavy at B^k.

It is easy to see that $G' = P(G)$ remains left-side-heavy at any block other than these blocks B^k.

For each block B^k, it holds $[\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^k); G), \sigma_{\mathtt{L}}(G(B^k); G)] \geq [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^k); G), \overline{\sigma_{\mathtt{R}}}(G(B^k); G)]$ by Lemma 5.1, since G is left-side-heavy at B^k.

If t^k is the bottom vertex $bv(B^k)$ of B_k or a right wing-vertex of B_k, then $\sigma_{\mathtt{core}}(G(B^k); G) = \sigma_{\mathtt{core}}(G'(B^k); P(G))$, $\sigma_{\mathtt{L}}(G(B^k); G) = \sigma_{\mathtt{L}}(G'(B^k); P(G))$ and $\sigma_{\mathtt{R}}(G(B^k); G) \sqsupseteq \sigma_{\mathtt{R}}(G'(B^k); P(G))$, implying that $G' = P(G)$ remains left-side-heavy at B^k.

If t^k is a left wing-vertex of B_k, then $\sigma_{\mathtt{core}}(G(B^k); G) = \sigma_{\mathtt{core}}(G'(B^k); P(G))$, $\sigma_{\mathtt{L}}(G(B^k); G) \sqsupset \sigma_{\mathtt{L}}(G'(B^k); P(G))$ and $\sigma_{\mathtt{R}}(G(B^k); G) = \sigma_{\mathtt{R}}(G'(B^k); P(G)) = \emptyset$, implying that $G' = P(G)$ remains left-side-heavy at B^k.

Note that t^k is not a left core-vertex of B^k by definition of tips. If t^k is a right core-vertex of B^k, then $\sigma_{\mathtt{L}}(G(B^k); G) = \sigma_{\mathtt{R}}(G(B^k); G) = \emptyset$ and $\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^k); P(G)) = \sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^k); G)$ hold and $\sigma^{\mathtt{R}}_{\mathtt{core}}(G'(B^k); P(G))$ is obtained from $\sigma^{\mathtt{R}}_{\mathtt{core}}(G(B^k); G)$ by deleting the last code, implying that $G' = P(G)$ remains left-side-heavy at B^k. □

Chapter 6

Generating Canonical Child-embeddings

We now consider how to generate all canonical child-embeddings from a canonical embedding.

6.1 Side-state Change of Applied Blocks

We first consider how to generate all child-embeddings G' from a canonical embedding G such that G' is left-side-heavy at the block applied a vertex code γ.

Let G be a canonical embedding with $|V(G)| = N$, and let B be a block in G. For an element ε in $V'(B) \cup E(B)$ and a vertex code $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$, let $G + \gamma(u_{N+1})$ with $\gamma(u_{N+1}) = \gamma$ denote the embedding G' such that $\sigma(G') = [\sigma(G), (d_1, \texttt{at}, d_2, \texttt{op}, c)]$, i.e., $G + \gamma(u_{N+1})$ is obtained from G by introducing a new vertex u_{N+1} by applying operation $(d_1, \texttt{at}, d_2, \texttt{op}, c)$ to ε.

Now we identify all the elements ε in $V'(B) \cup E(B)$ that admit a vertex code γ such that $G' = G + \gamma(u_{N+1})$ remains left-side-heavy at block B. We give the set of such elements in $V'(B) \cup E(B)$ as a sequence of these elements, called the *element sequence* $\mathcal{E}(B)$.

Note that the condition for G' to remain canonical (i.e., left-sibling-heavy and left-side-heavy at any other blocks) will be investigated in the next section.

We assume that $\texttt{sd}(B; G) \neq \texttt{eqv}$, since otherwise no application of $\gamma(u_{N+1})$ is applied any element in $V'(B) \cup E(B)$ without violating left-side-heaviness of B. Let $\mathcal{E}(B) = \emptyset$ if $\texttt{sd}(B; G) = \texttt{eqv}$.

Let $\{x_i \mid i = 1, 2, \ldots, p_{\texttt{L}}\}$ (resp., $\{y_j \mid j = 1, 2, \ldots, p_{\texttt{R}}\}$) denote the set of vertices in the left (resp., right) side of B, where $d(x_i) = d(r(B)) + i$ and $d(y_j) = d(r(B)) + j$. To identify $\mathcal{E}(B)$, we consider the same five cases used to define the tip $t(B)$ of a block B.

Case-1. $V^{\texttt{R}}_{\texttt{cut}}(B) \neq \emptyset$, where $\texttt{sd}(B; G) \in \{\texttt{stc}, \texttt{pfx}\}$: See Figure 6.1(a)-(d). In this case, $t(B)$ is the vertex $y_i \in V^{\texttt{R}}_{\texttt{cut}}(B)$ with the *largest* depth $d(y_i)$. If $\texttt{sd}(B; G) = \texttt{pfx}$, then let

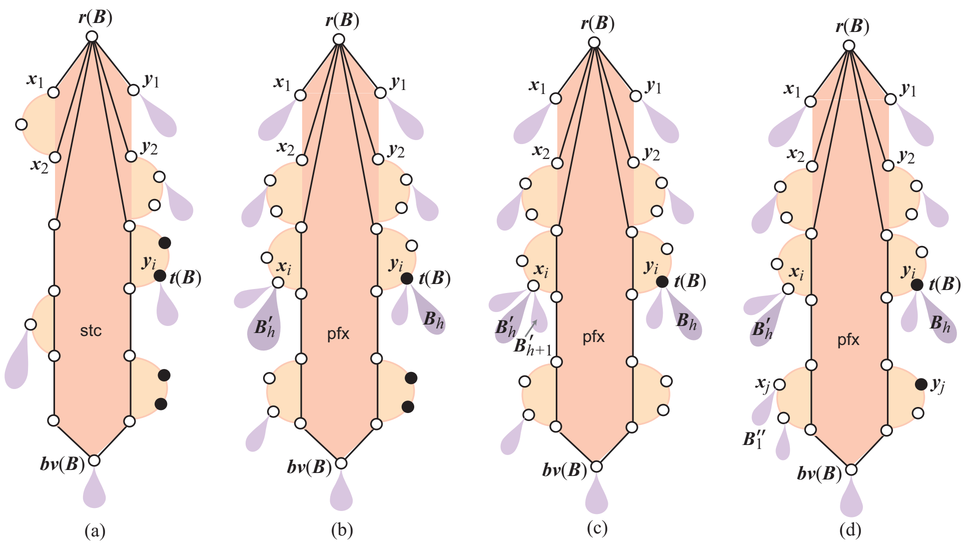

Figure 6.1: Illustrations of Case-1 (i.e., $V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$): (a) $\mathtt{sd}(B;G) = \mathtt{stc}$; (b) $\mathtt{sd}(B;G) = \mathtt{pfx}$ and $|V(B'_h)| > |V(B_h)|$; (c) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $|V(B'_h)| = |V(B_h)|$ and $|\mathcal{B}(x_i)| > |\mathcal{B}(y_i)|$; (d) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $|V(B'_h)| = |V(B_h)|$ and $|\mathcal{B}(x_i)| = |\mathcal{B}(y_i)|$.

$h = |\mathcal{B}(y_i)|$, and B_h be the hth rightmost block (i.e., the rightmost one) in $\mathcal{B}(y_i)$, and let B'_h and B'_{h+1} (if any) denote the hth and $(h+1)$st blocks in $\mathcal{B}(x_i)$ (see Figure 6.1(b)-(c)).

(a) $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.1(a)): Define $\mathcal{E}(B)$ to be

$$\mathcal{E}(B) = [y_i, y_{i+1}, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}],$$

where $d(y_i) < d(y_{i+1}) < \cdots < d(y_{p_{\mathtt{R}}-1}) < d(y_{p_{\mathtt{R}}})$ holds.

(b) $\mathtt{sd}(B;G) = \mathtt{pfx}$ and $|V(B'_h)| > |V(B_h)|$ (see Figure 6.1(b)): Define $\mathcal{E}(B)$ to be

$$\mathcal{E}(B) = [y_i, y_{i+1}, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}].$$

(c) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $|V(B'_h)| = |V(B_h)|$ and $|\mathcal{B}(x_i)| > |\mathcal{B}(y_i)|$ (see Figure 6.1(c)): Define

$$\mathcal{E}(B) = [y_i], \quad \mathrm{c}_{\max}(y_i) = c(\ell v(B'_{h+1})).$$

(d) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $|V(B'_h)| = |V(B_h)|$ and $|\mathcal{B}(x_i)| = |\mathcal{B}(y_i)|$ (see Figure 6.1(d)): Let x_j be the left vertex $x \in V^{\mathtt{L}}_{\mathtt{cut}}(B)$ having the smallest depth $d(x_j)$ such that $d(x_j) > d(x_i)$ (i.e., x_j is the next vertex in $V^{\mathtt{L}}_{\mathtt{cut}}(B)$ after x_i), and B''_1 be the leftmost block in $\mathcal{B}(x_j)$. Define

$$\mathcal{E}(B) = [y_i, y_{i+1}, \ldots, y_{j-1}, y_j], \quad \mathrm{c}_{\max}(y_j) = c(\ell v(B''_1)).$$

Case-2. $V^{\mathtt{R}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$, where $\mathtt{sd}(B;G) \in \{\mathtt{stc}, \mathtt{pfx}\}$: See Figure 6.2(a)-(d).

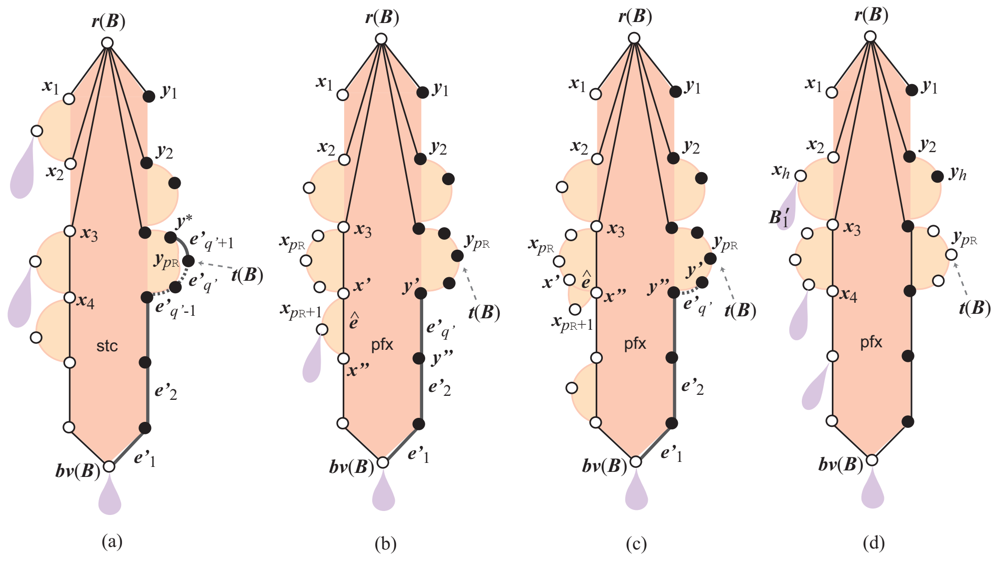

Figure 6.2: Illustrations of Case-2 (i.e., $V^{\mathtt{R}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$): (a) $\mathtt{sd}(B;G) = \mathtt{stc}$; (b) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) \sqsupset \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$, and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$; (c) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) \sqsupset \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$, and $V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$; (d) $\mathtt{sd}(B;G) = \mathtt{pfx}$ and $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$.

(a) $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.2(a)): Let y^* be the right vertex that precedes $y_{p_{\mathtt{R}}}$ along $P_{\mathtt{R}}(y_1;B)$, and denote the sequence of edges in $E_{\mathtt{R}}(y^*;B)$ by $(e'_{q'+1}, e'_{q'}, \ldots, e'_2, e'_1)$. Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \ldots, e'_{q'}, e'_{q'+1}],$$

where $d(e'_1) < d(e'_2) < \cdots < d(e'_{q'}) < d(e'_{q'+1})$ holds by(3.1).

(b) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) \sqsupset \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$, and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$ (see Figure 6.2(b)): There is a left wing-vertex $x_{p_{\mathtt{R}}} = \mu(y_{p_{\mathtt{R}}}; r(B))$, and let $\hat{e} = (x', x'') \in \tilde{E}(B)$ be the left edge to which

$$\gamma(x_{p_{\mathtt{R}}+1}) = (d(B), \mathtt{w}^{\mathtt{L}}, d(\hat{e}), \mathtt{op}(x_{p_{\mathtt{R}}+1}), c(x_{p_{\mathtt{R}}+1}))$$

is applied, where $\mathtt{op}(x_{p_{\mathtt{R}}+1}) \in \{\mathtt{triangle}, \mathtt{subdivide}\}$. Let $y' = \mu(x'; r(B))$ and $y'' = \mu(x''; r(B))$, and let $e'_{q'} = (y', y'')$ be the corresponding right edge, where $e'_{q'}$ appears along $E_{\mathtt{R}}(y_1;B)$. Denote the sequence of edges in $E_{\mathtt{R}}(y';B)$ from $e'_{q'}$ to the bottom of B by $(e'_{q'}, e'_{q'-1}, \ldots, e'_2, e'_1)$. Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \ldots, e'_{q'}], \quad \mathrm{opc}_{\max}(e'_{q'}) = (\mathtt{op}(x_{p_{\mathtt{R}}+1}), c(x_{p_{\mathtt{R}}+1})).$$

(c) $\mathtt{sd}(B;G) = \mathtt{pfx}$, $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) \sqsupset \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$, and $V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ (see Figure 6.2(c)): Define $\mathcal{E}(B)$ and $\mathrm{opc}_{\max}(e'_{q'})$ in the same way of the above (b).

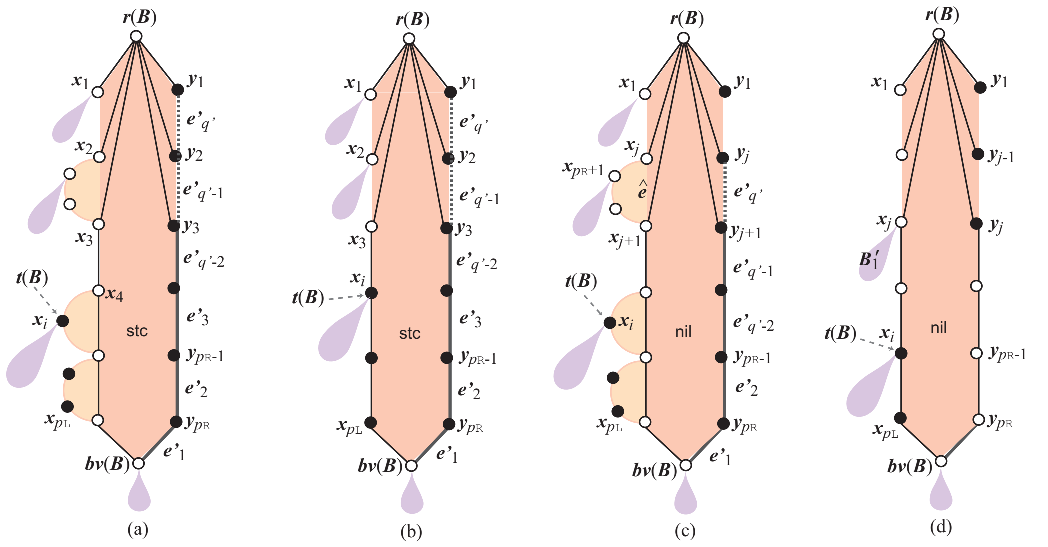

Figure 6.3: Illustrations of Case-3 (i.e., $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$): (a) $\mathtt{sd}(B;G) = \mathtt{stc}$ with $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$; (b) $\mathtt{sd}(B;G) = \mathtt{stc}$ with $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$; (c) $\mathtt{sd}(B;G) = \mathtt{nil}$ with $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$; (d) $\mathtt{sd}(B;G) = \mathtt{nil}$ with $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$.

(d) $\mathtt{sd}(B;G) = \mathtt{pfx}$ and $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$ (see Figure 6.2(d)): Let x_h be such the vertex $x \in V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$ with the smallest depth, and let B'_1 be the leftmost block in $\mathcal{B}(x_h)$. For the corresponding left vertex $y_h = \mu(x_h; r(B))$, define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_h], \quad \mathrm{c}_{\max}(y_h) = c(\ell v(B'_1)).$$

Case-3. $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$, where $\mathtt{sd}(B;G) \in \{\mathtt{nil}, \mathtt{stc}\}$: Let x_i be the left vertex in $V^{\mathtt{L}}_{\mathtt{cut}}(B)$ with the *largest* depth $d(x_i)$, where $x_i = t(B)$. See Figure 6.3(a)-(d).

(a) $\mathtt{sd}(B;G) = \mathtt{stc}$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$: Let $(e'_{q'}, e'_{q'-1}, \ldots, e'_2, e'_1)$ denote the sequence of edges in $E_{\mathtt{R}}(y_1; B)$, as shown in Figure 6.3(a). Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_i, x_{i+1}, \ldots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}].$$

(b) $\mathtt{sd}(B;G) = \mathtt{stc}$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$: Let $(e'_{q'}, e'_{q'-1}, \ldots, e'_2, e'_1)$ denote the sequence of edges in $E_{\mathtt{R}}(y_1; B)$, as shown in Figure 6.3(b). Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_i, x_{i+1}, \ldots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}].$$

(c) $\mathtt{sd}(B;G) = \mathtt{nil}$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$: Let $\hat{e} = (x_j, x_{j+1})$ be the two left core-vertices adjacent to the left wing-vertex $x_{p_{\mathtt{R}}+1}$ with the smallest depth $d(x_{p_{\mathtt{R}}+1})$, let $\gamma(x_{p_{\mathtt{R}}+1}) =$

$(d(B), \mathtt{w}^{\mathrm{L}}, d(\hat{e}), \mathtt{op}(x_{p_{\mathrm{R}}+1}), c(x_{p_{\mathrm{R}}+1}))$ and $y_j = \mu(x_j; r(B))$. Let $(e'_{q'}, e'_{q'-1}, \ldots, e'_2, e'_1)$ denote the sequence of edges in $E_{\mathtt{R}}(y_j; B)$, as shown in Figure 6.3(c). Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathrm{R}}-1}, y_{p_{\mathrm{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_i, x_{i+1}, \ldots, x_{p_{\mathrm{L}}-1}, x_{p_{\mathrm{L}}}],$$

$$\mathrm{opc}_{\max}(e'_{q'}) = (\mathtt{op}(x_{p_{\mathrm{R}}+1}), c(x_{p_{\mathrm{R}}+1})).$$

(d) $\mathtt{sd}(B; G) = \mathtt{nil}$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$: Let x_j be the left core-vertex with $\mathcal{B}(x_j) \neq \emptyset$ having the smallest depth $d(x_j)$ in B, as shown in Figure 6.3(d). Let B'_1 be the leftmost block in $\mathcal{B}(x_j)$. Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{j-1}, y_j, x_i, x_{i+1}, \ldots, x_{p_{\mathrm{L}}-1}, x_{p_{\mathrm{L}}}], \quad \mathrm{c}_{\max}(y_j) = c(\ell v(B'_1)).$$

Case-4. $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$, where $\mathtt{sd}(B; G) \in \{\mathtt{nil}, \mathtt{stc}\}$: See Figure 6.4(a)-(b). Let x^* be the vertex that precedes $x_{p_{\mathrm{L}}}$ along $P_{\mathrm{L}}(x_1; B)$, and denote the sequence of edges in $E_{\mathrm{L}}(x^*; B)$ by $(e_{q+1}, e_q, \ldots, e_1)$.

(a) $\mathtt{sd}(B; G) = \mathtt{stc}$: Let $(e'_{q'}, e'_{q'-1}, \ldots, e'_1)$ denote the sequence of edges in $E_{\mathtt{R}}(y_1; B)$ in G, as shown in Figure 6.4(a). Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathrm{R}}-1}, y_{p_{\mathrm{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_1, \ldots, x_{p_{\mathrm{L}}-1}, x_{p_{\mathrm{L}}}, e_1, e_2, \ldots, e_q, e_{q+1}].$$

(b) $\mathtt{sd}(B; G) = \mathtt{nil}$: Let $\hat{e} = (x_j, x_{j+1})$ be the two left core-vertices adjacent to the left wing-vertex $x_{p_{\mathrm{R}}+1}$ with the smallest depth $d(x_{p_{\mathrm{R}}+1})$, as shown in Figure 6.4(b). Let $\gamma(x_{p_{\mathrm{R}}+1}) = (d(B), \mathtt{w}^{\mathrm{L}}, d(\hat{e}), \mathtt{op}(x_{p_{\mathrm{R}}+1}), c(x_{p_{\mathrm{R}}+1}))$ and $y_j = \mu(x_j; r(B))$. Let $(e'_{q'}, e'_{q'-1}, \ldots, e'_1)$ denote the sequence of edges in $E_{\mathtt{R}}(y_j; B)$. Define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathrm{R}}-1}, y_{p_{\mathrm{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_1, \ldots, x_{p_{\mathrm{L}}-1}, x_{p_{\mathrm{L}}}, e_1, e_2, \ldots, e_q, e_{q+1}],$$

$$\mathrm{opc}_{\max}(e'_{q'}) = (\mathtt{op}(x_{p_{\mathrm{R}}+1}), c(x_{p_{\mathrm{R}}+1})).$$

Case-5. $|V(B)| = 2$ or $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$ (possibly $\mathcal{B}(bv(B)) \neq \emptyset$), where $\mathtt{sd}(B; G) \in \{\mathtt{nil}, \mathtt{stc}\}$: See Figures 6.5 and 6.6. In this case, $t(B)$ is the core-vertex $u \in V'(B)$ with the *largest* depth $d(u)$, where $t(B)$ is $bv(B)$ or the right endvertex of $be(B)$.

(a) $|V(B)| = 2$, where $t(B) = bv(B)$ holds. Let $e^b = (r(B), bv(B))$ be the edge in B:

If $\mathcal{B}(bv(B)) = \emptyset$ (see Figure 6.5(f)), then define

$$\mathcal{E}(B) = [bv(B), e^b], \quad \mathrm{opc}_{\max}(e^b) = (\mathtt{triangle}, c(bv(B))).$$

If $\mathcal{B}(bv(B)) \neq \emptyset$ (see Figure 6.5(g)), then let B'_1 be the rightmost block in $\mathcal{B}(bv(B))$, and define

$$\mathcal{E}(B) = [bv(B)], \quad \mathrm{c}_{\max}(bv(B)) = c(\ell v(B'_1)).$$

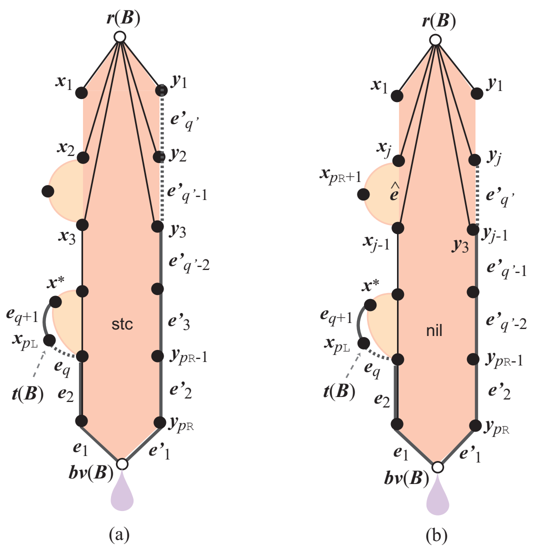

Figure 6.4: Illustrations of Case-4 (i.e., $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$): (a) $\mathtt{sd}(B;G) = \mathtt{stc}$; (b) $\mathtt{sd}(B;G) = \mathtt{nil}$.

(b) $|V(B)| \geq 3$ and $\mathtt{sd}(B;G) = \mathtt{nil}$ (see Figure 6.5(a)-(e)): Denote the sequence of edges in $E_{\mathtt{L}}(x_1;B)$ by $(e_q, e_{q-1}, \dots, e_1)$. Let e^b denote the bottom edge $be(B)$ or the right edge incident to the bottom vertex $bv(B)$.

If $\mathcal{B}(bv(B)) = \emptyset$ (see Figure 6.5(a)-(d)), where possibly $bv(B) = \emptyset$, then define

$$\mathcal{E}(B) = [x_1, \dots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}, e_1, e_2, \dots, e_q, bv(B), e^b],$$

where we define

$$\mathrm{opc_{max}}(e^b) = (\mathtt{subdivide}, c(bv(B))) \text{ if } bv(B) \neq \emptyset \text{ (i.e., } |V(B)| \geq 4 \text{ is even).}$$

If $\mathcal{B}(bv(B)) \neq \emptyset$ (see Figure 6.5(e)), then let B'_1 be the rightmost block in $\mathcal{B}(bv(B))$, and define

$$\mathcal{E}(B) = [x_1, \dots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}, e_1, e_2, \dots, e_q, bv(B)], \quad \mathrm{c_{max}}(bv(B)) = c(\ell v(B'_1)).$$

(c) $|V(B)| \geq 3$ and $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.6(a)-(e)): Denote the sequence of edges in $E_{\mathtt{L}}(x_1;B)$ and $E_{\mathtt{R}}(y_1;B)$ by $(e_q, e_{q-1}, \dots, e_1)$ and $(e'_q, e'_{q-1}, \dots, e'_1)$, respectively. Let e^b denote the bottom edge $be(B)$ or the right edge incident to the bottom vertex $bv(B)$.

If $\mathcal{B}(bv(B)) = \emptyset$ (see Figure 6.6(a)-(d)), then define

$$\mathcal{E}(B) = [y_1, y_2, \dots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \dots, e'_{q'}, x_1, \dots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}, e_1, e_2, \dots, e_q, bv(B), e^b],$$

where possibly $bv(B) = \emptyset$, and different operations are applicable to edges e^b and e'_1, although they have the same endvertices.

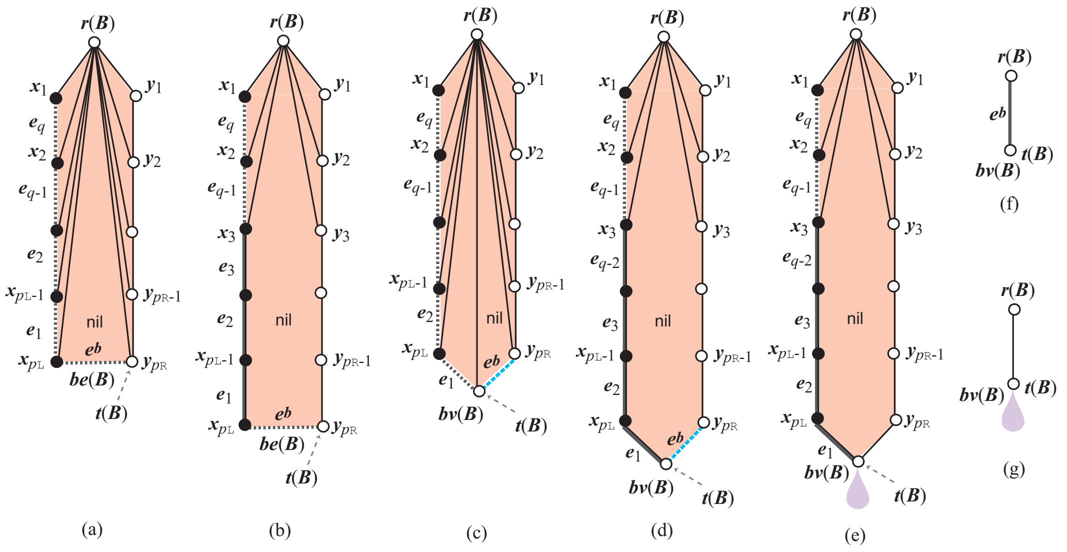

Figure 6.5: Illustrations of Case-5 with $\mathtt{sd}(B;G) = \mathtt{nil}$: (a) $be(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) = \emptyset$; (b) $be(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$; (c) $bv(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) = \emptyset$; (d) $bv(B) \neq \emptyset$, $\mathcal{B}(bv(B)) = \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$; (e) $\mathcal{B}(bv(B)) \neq \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$; (f) $|V(B)| = 2$ and $\mathcal{B}(bv(B)) = \emptyset$; (g) $|V(B)| = 2$ and $\mathcal{B}(bv(B)) \neq \emptyset$.

If $\mathcal{B}(bv(B)) \neq \emptyset$ (see Figure 6.6(e)), then define

$$\mathcal{E}(B) = [y_1, y_2, \ldots, y_{p_{\mathtt{R}}-1}, y_{p_{\mathtt{R}}}, e'_1, e'_2, \ldots, e'_{q'}, x_1, \ldots, x_{p_{\mathtt{L}}-1}, x_{p_{\mathtt{L}}}, e_1, e_2, \ldots, e_q, bv(B)].$$

From the definition of $\mathcal{E}(B)$, we see the next.

Lemma 6.1. *No other element ε than $\mathcal{E}(B)$ admits a vertex code γ applicable to ε such that $G + \gamma(u_{N+1})$ with $\gamma(u_{N+1}) = \gamma$ is left-side-heavy at B.* □

Code set Γ We next consider all vertex codes γ for each element $\varepsilon \in \mathcal{E}(B)$ such that γ is applicable to ε and $G + \gamma(u_{N+1})$ with $\gamma(u_{N+1}) = \gamma$ is left-side-heavy at B and left-sibling-heavy at the new block B' created by γ with `new-block` (if any). Let $\Gamma(\varepsilon)$ denote the set of such vertex codes γ.

(1) For a vertex $v \in \mathcal{E}(B)$, $\Gamma(v)$ is defined as follows. Let B' denote the new block formed by operation `new-block` with v. Then it holds

$$\mathtt{sd}(B'; G') = \mathtt{nil}. \tag{6.1}$$

We have

$$\mathtt{sbl}(B'; G') = \mathtt{stc} \text{ if } \mathcal{B}(v) = \emptyset. \tag{6.2}$$

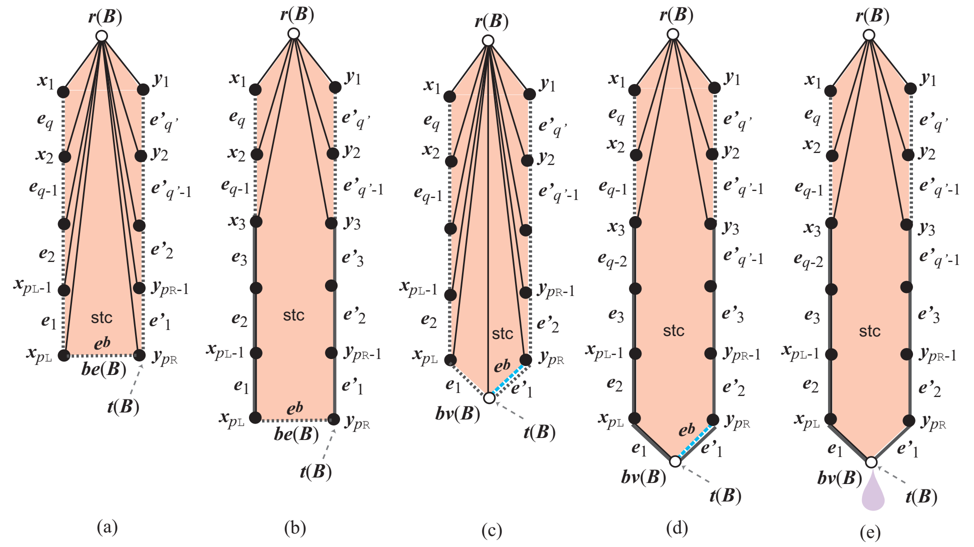

Figure 6.6: Illustrations of Case-5 with $\mathtt{sd}(B;G) = \mathtt{stc}$: (a) $be(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) = \emptyset$; (b) $be(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$; (c) $bv(B) \neq \emptyset$ and $V_{\mathtt{axis}}(B) = \emptyset$; (d) $bv(B) \neq \emptyset$, $\mathcal{B}(bv(B)) = \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$; (e) $\mathcal{B}(bv(B)) \neq \emptyset$ and $V_{\mathtt{axis}}(B) \neq \emptyset$.

1. For $v \in V'(B) - V_{\mathtt{cut}}(B)$ such that v is not y_j in Case-1(d) and 3(d), define

$$\Gamma(v) = \{(d(B), \mathtt{at}, d(v), \mathtt{new\text{-}block}, c) \mid c \in \mathcal{C}\},$$

where $\mathtt{at} = \mathtt{h}^{\mathrm{R}}$ (resp., $\mathtt{at} = \mathtt{h}^{\mathrm{L}}$ and $\mathtt{at} = *$) if v is a right vertex (resp., a left vertex and the bottom vertex) of B. If $\mathtt{sd}(B;G) = \mathtt{nil}$, and $v \in V'(B)$ is not y_j in Case-3(d), then

$$\mathtt{sd}(B;G') = \begin{cases} \mathtt{nil} & \text{if } v \text{ is a left vertex or the bottom vertex of } B; \\ \mathtt{stc} & \text{if } v \text{ is a right vertex of } B. \end{cases} \tag{6.3}$$

2. For $v = y_i$ in Case-1(c), let $h = |\mathcal{B}(y_i)|$, and let B_h be the h-th block in $\mathcal{B}(y_i)$ (i.e., B_h is the rightmost block rooted at y_i), and B'_{h+1} denote the $(h+1)$st block in $\mathcal{B}(x_i)$. Then

$$\Gamma(y_i) = \{(d(B), \mathtt{h}^{\mathrm{R}}, d(y_i), \mathtt{new\text{-}block}, c) \mid c \in \mathcal{C}, c \leq c(\ell v(B'_{h+1}))\},$$

and

$$\mathtt{sbl}(B';G') = \begin{cases} \mathtt{eqv} & \text{if } c = c(\ell v(B_h)) \text{ and } |V(G(B_h))| = 2 \\ \mathtt{pfx} & \text{if } c = c(\ell v(B_h)) \text{ and } |V(G(B_h))| \geq 3 \\ \mathtt{stc} & \text{if } c < c(\ell v(B_h)). \end{cases} \tag{6.4}$$

3. For $v = y_j$ in Case-1(d) or Case-3(d), $\mathcal{B}(y_j) = \emptyset$ holds, and let B''_1 be the leftmost block in $\mathcal{B}(x_j)$. Define

$$\Gamma(v) = \{(d(B), \mathtt{h}^{\mathrm{R}}, d(v), \mathtt{new\text{-}block}, c) \mid c \in \mathcal{C}, c \leq c(\ell v(B''_1))\}.$$

Let $V^{\mathtt{L}}(G(B);G)$ (resp., $V^{\mathtt{R}}(G(B);G)$) denote the set of left (resp., right) wing-vertices of B and the descendants of left (resp., right) vertices of B. If $\mathtt{sd}(B;G) = \mathtt{nil}$, then

$$\mathtt{sd}(B;G') = \begin{cases} \mathtt{eqv} & \text{if } c = c(\ell v(B''_1)) \text{ and } |V^{\mathtt{L}}(G(B);G)| = |V^{\mathtt{R}}(G(B);G)| + 1 \\ \mathtt{pfx} & \text{if } c = c(\ell v(B''_1)) \text{ and } |V^{\mathtt{L}}(G(B);G)| \geq |V^{\mathtt{R}}(G(B);G)| + 2 \\ \mathtt{stc} & \text{if } c < c(\ell v(B''_1)). \end{cases} \tag{6.5}$$

In G, it holds $\mathcal{B}(y_j) = \emptyset$. Hence $\mathtt{sbl}(B';G') = \mathtt{stc}$ holds.

4. For $v \in V_{\mathtt{cut}}(B)$ not in Case-1(c), let B'_1 denote the rightmost block in $\mathcal{B}(v)$ in G. Define

$$\Gamma(v) = \{(d(B), \mathtt{at}, d(v), \mathtt{new\text{-}block}, c) \mid c \in \mathcal{C}, c \leq c(\ell v(B'_1))\},$$

where $\mathtt{at} = \mathtt{h}^{\mathtt{R}}$ (resp., $\mathtt{at} = \mathtt{h}^{\mathtt{L}}$ and $\mathtt{at} = *$) if v is a right vertex (resp., a left vertex and the bottom vertex) of B.

Then

$$\mathtt{sbl}(B';G') = \begin{cases} \mathtt{eqv} & \text{if } c = c(\ell v(B'_1)) \text{ and } |V(G(B'_1))| = 2 \\ \mathtt{pfx} & \text{if } c = c(\ell v(B'_1)) \text{ and } |V(G(B'_1))| \geq 3 \\ \mathtt{stc} & \text{if } c < c(\ell v(B'_1)). \end{cases} \tag{6.6}$$

$$\text{If } \mathtt{sd}(B;G) = \mathtt{nil}, \text{ then } \mathtt{sd}(B;G') = \mathtt{nil}. \tag{6.7}$$

(2) For each right edge $e' \in \mathcal{E}(B)$, $\Gamma(e')$ is defined as follows.

(i) $\mathtt{sd}(B;G) \in \{\mathtt{pfx}, \mathtt{nil}\}$ and $\sigma^{\mathtt{L}}_{\mathtt{wing}}(G(B);G) \sqsupset \overline{\sigma^{\mathtt{R}}_{\mathtt{wing}}}(G(B);G)$ (i.e., in Case-2(b)-(c), Case-3(c) and Case-4(b)): In this case, $\Gamma(e'_{q'})$ for the edge $e'_{q'} = (y', y'')$ is defined as follows. Let $\hat{e} = (x', x'')$ be the corresponding left edge with $x' = \mu(y'; r(B))$ and $x'' = \mu(y''; r(B))$, and $\gamma(x_{p_{\mathtt{R}}+1}) = (d(B), \mathtt{w}^{\mathtt{L}}, d(\hat{e}), \mathtt{op}(x_{p_{\mathtt{R}}+1}), c(x_{p_{\mathtt{R}}+1}))$ be the code applied to $\hat{e}$, where $\mathtt{op}(x_{p_{\mathtt{R}}+1}) \in \{\mathtt{subdivide}, \mathtt{triangle}\}$ and $x_{p_{\mathtt{R}}+1}$ is the $(R+1)$th left wing-vertex for $R = |V^{\mathtt{R}}_{\mathtt{wing}}(B)|$.

If $\mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{triangle}$, then define

$$\Gamma(e'_{q'}) = \{(d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{triangle}, c) \mid c \in \mathcal{C}, c \leq c(x_{p_{\mathtt{R}}+1})\},$$

where if $\mathtt{sd}(B;G) = \mathtt{nil}$, then

$$\mathtt{sd}(B;G') = \begin{cases} \mathtt{eqv} & \text{if } c = c(x_{p_{\mathtt{R}}+1}) \text{ and } |V^{\mathtt{L}}(G(B);G)| = |V^{\mathtt{R}}(G(B);G)| + 1 \\ \mathtt{pfx} & \text{if } c = c(x_{p_{\mathtt{R}}+1}) \text{ and } |V^{\mathtt{L}}(G(B);G)| \geq |V^{\mathtt{R}}(G(B);G)| + 2 \\ \mathtt{stc} & \text{if } c < c(x_{p_{\mathtt{R}}+1}). \end{cases} \tag{6.8}$$

If $\mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{subdivide}$, then

$$\begin{aligned} \Gamma(e'_{q'}) = \{&(d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{subdivide}, c) \mid c \in \mathcal{C}, c \leq c(x_{p_{\mathtt{R}}+1})\} \\ &\cup \{(d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{triangle}, c) \mid c \in \mathcal{C}\}, \end{aligned}$$

where if $\texttt{sd}(B;G)=\texttt{nil}$, then

$$\texttt{sd}(B;G')=\begin{cases}\texttt{eqv} & \text{if } \gamma(u_{N+1})=(d(B),\texttt{w}^{\texttt{R}},d(e'_{q'}),\texttt{subdivide},c(x_{p_{\texttt{R}}+1})) \text{ and} \\ & |V^{\texttt{L}}(G(B);G)|=|V^{\texttt{R}}(G(B);G)|+1 \\ \texttt{pfx} & \text{if } \gamma(u_{N+1})=(d(B),\texttt{w}^{\texttt{R}},d(e'_{q'}),\texttt{subdivide},c(x_{p_{\texttt{R}}+1})) \text{ and} \\ & |V^{\texttt{L}}(G(B);G)|\geq|V^{\texttt{R}}(G(B);G)|+2 \\ \texttt{stc} & \text{for other } \gamma(u_{N+1})\in\Gamma(e'_{q'}).\end{cases} \tag{6.9}$$

(ii) Otherwise, i.e., e' is not the edge $e'_{q'}$ in Case-2(b)-(c), Case-3(c) and Case-4(b):
If $e'\in A(B)\cup\{e'_{q'+1}\}$, then define

$$\Gamma(e')=\{(d(B),\texttt{w}^{\texttt{R}},d(e'),\texttt{triangle},c)\mid c\in\mathcal{C}\}.$$

If $e'\notin A(B)\cup\{e'_{q'+1}\}$, then define

$$\Gamma(e')=\{(d(B),\texttt{w}^{\texttt{R}},d(e'),\texttt{op},c)\mid \texttt{op}\in\{\texttt{subdivide},\texttt{triangle}\},c\in\mathcal{C}\}.$$

In (ii),

$$\texttt{sd}(B;G)=\texttt{nil} \text{ changes into } \texttt{sd}(B;G')=\texttt{stc}. \tag{6.10}$$

(3) For each left edge $e_j\in\mathcal{E}(B)$, $\Gamma(e_j)$ is defined as follows.
If $e_j\in A(B)\cup\{e_{q+1}\}$, then define

$$\Gamma(e_j)=\{(d(B),\texttt{w}^{\texttt{L}},d(e_j),\texttt{triangle},c)\mid c\in\mathcal{C}\}.$$

If $e_j\notin A(B)\cup\{e_{q+1}\}$, then define

$$\Gamma(e_j)=\{(d(B),\texttt{w}^{\texttt{L}},d(e_j),\texttt{op},c)\mid \texttt{op}\in\{\texttt{subdivide},\texttt{triangle}\},c\in\mathcal{C}\}.$$

In (3),

$$\texttt{sd}(B;G)=\texttt{nil} \text{ remains unchanged, i.e., } \texttt{sd}(B;G')=\texttt{nil}. \tag{6.11}$$

(4) For the edge $e^b=(y,z)\in\mathcal{E}(B)$ $(d(z)\geq d(y))$, $\Gamma(e^b)$ is defined as follows.

(a) B has no axial-vertex and $|V(B)|$ is even:
If $\texttt{sd}(B;G)=\texttt{stc}$ (see Figure 6.6(c)), then

$$\Gamma(e^b)=\{(d(B),*,d(z),\texttt{star},c)\mid c\in\mathcal{C}\}.$$

If $\texttt{sd}(B;G)=\texttt{nil}$ and $|V(B)|=2$ (see Figure 6.5(f)), then $z=bv(B)$ holds and

$$\Gamma(e^b)=\{(d(B),*,d(z),\texttt{triangle},c)\mid c\in\mathcal{C},c\leq c(z)\}.$$

If $\mathtt{sd}(B;G) = \mathtt{nil}$ and $|V(B)| \geq 3$ (see Figure 6.5(c)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{star}, c) \mid c \in \mathcal{C}, c \leq c(z)\}.$$

In case (a), if $\mathtt{sd}(B;G) = \mathtt{nil}$, then

$$\mathtt{sd}(B;G') = \begin{cases} \mathtt{nil} & \text{if } c = c(z) \\ \mathtt{stc} & \text{if } c < c(z). \end{cases} \tag{6.12}$$

(b) B has no axial-vertex and $|V(B)|$ is odd:

If $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.6(a)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{op}, c) \mid \mathtt{op} \in \{\mathtt{star}, \mathtt{subdivide}, \mathtt{triangle}\}, c \in \mathcal{C}\}.$$

If $\mathtt{sd}(B;G) = \mathtt{nil}$ (see Figure 6.5(a)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{op}, c) \mid \mathtt{op} \in \{\mathtt{star}, \mathtt{subdivide}, \mathtt{triangle}\}, c \in \mathcal{C}\},$$

and

$$\mathtt{sd}(B;G') = \mathtt{nil}. \tag{6.13}$$

(c) B has an axial-vertex and $|V(B)|$ is even:

If $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.6(d)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{subdivide}, c) \mid c \in \mathcal{C}\}.$$

If $\mathtt{sd}(B;G) = \mathtt{nil}$, B has an axial-vertex and $|V(B)|$ is even (see Figure 6.5(d)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{subdivide}, c) \mid c \in \mathcal{C}, c \leq c(z)\},$$

and

$$\mathtt{sd}(B;G') = \begin{cases} \mathtt{nil} & \text{if } c = c(z) \\ \mathtt{stc} & \text{if } c < c(z). \end{cases} \tag{6.14}$$

(d) B has an axial-vertex and $|V(B)|$ is odd:

If $\mathtt{sd}(B;G) = \mathtt{stc}$ (see Figure 6.6(b)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{op}, c) \mid \mathtt{op} \in \{\mathtt{subdivide}, \mathtt{triangle}\}, c \in \mathcal{C}\}.$$

If $\mathtt{sd}(B;G) = \mathtt{nil}$ (see Figure 6.5(b)), then

$$\Gamma(e^b) = \{(d(B), *, d(z), \mathtt{op}, c) \mid \mathtt{op} \in \{\mathtt{subdivide}, \mathtt{triangle}\}, c \in \mathcal{C}\},$$

and

$$\mathtt{sd}(B;G') = \mathtt{nil}. \tag{6.15}$$

From the above definition, we can observe that the following two properties hold.

Lemma 6.2. *For a block B in a canonical embedding G, and an edge $e \in \mathcal{E}(B)$, let γ be a vertex code applicable to e. Then $G' = G + \gamma(u_{N+1})$ with $\gamma(u_{N+1}) = \gamma$ is left-side-heavy at B if and only if $\gamma \in \Gamma(e)$.* □

Lemma 6.3. *For a block B in a canonical embedding G, and a vertex $v \in \mathcal{E}(B)$, let γ with* `new-block` *be a vertex code applicable to v. Then $G' = G + \gamma(u_{N+1})$ with $\gamma(u_{N+1}) = \gamma$ is left-side-heavy at B and left-sibling-heavy at the new block B' created by γ if and only if $\gamma \in \Gamma(v)$.* □

Lemma 6.4. *For two distinct elements $w', w'' \in \mathcal{E}(B)$ of a block B, let $\gamma' = (d'_1, \mathtt{at}', d'_2, \mathtt{op}', c')$ and $\gamma'' = (d''_1, \mathtt{at}'', d''_2, \mathtt{op}'', c'')$ be codes in $\Gamma(w')$ and $\Gamma(w'')$, respectively, where $d''_1 = d'_1$. If w' precedes w'' in $\mathcal{E}(B)$, then $\gamma' < \gamma''$ holds.*

Proof: First we consider the case where $\mathtt{at}' \neq \mathtt{at}''$. Let e^b be the bottom edge of B or the right edge incident to the bottom vertex $bv(B)$ of B. Note that if w' is the vertex $bv(B)$ and w'' is the edge e^b of B, then $\mathtt{at}' = \mathtt{at}'' = *$. Specifically, there are the following subcases.

1. w' is a right vertex of B: Then $\mathtt{at}' = \mathtt{h}^{\mathtt{R}}$, and
$$\mathtt{at}'' = \begin{cases} \mathtt{w}^{\mathtt{R}} & \text{if } w'' \text{ is a right edge of } B; \\ \mathtt{h}^{\mathtt{L}} & \text{if } w'' \text{ is a left vertex of } B; \\ \mathtt{w}^{\mathtt{L}} & \text{if } w'' \text{ is a left edge of } B; \\ * & \text{if } w'' \text{ is the edge } e^b \text{ or is the} \\ & \quad \text{bottom vertex } bv(B). \end{cases}$$

2. w' is a right edge of B: Then $\mathtt{at}' = \mathtt{w}^{\mathtt{R}}$, and
$$\mathtt{at}'' = \begin{cases} \mathtt{h}^{\mathtt{L}} & \text{if } w'' \text{ is a left vertex of } B; \\ \mathtt{w}^{\mathtt{L}} & \text{if } w'' \text{ is a left edge of } B; \\ * & \text{if } w'' \text{ is the edge } e^b \text{ or the vertex } bv(B). \end{cases}$$

3. w' is a left vertex of B: Then $\mathtt{at}' = \mathtt{h}^{\mathtt{L}}$, and
$$\mathtt{at}'' = \begin{cases} \mathtt{w}^{\mathtt{L}} & \text{if } w'' \text{ is a left edge of } B; \\ * & \text{if } w'' \text{ is the edge } e^b \text{ or the vertex } bv(B). \end{cases}$$

4. w' is a left edge of B: Then $\mathtt{at}' = \mathtt{w}^{\mathtt{L}}$, and $\mathtt{at}'' = *$ if w'' is the edge e^b or the bottom vertex $bv(B)$.

By the ordering $* > \mathtt{w}^{\mathtt{L}} > \mathtt{h}^{\mathtt{L}} > \mathtt{w}^{\mathtt{R}} > \mathtt{h}^{\mathtt{R}}$, it holds $\mathtt{at}' < \mathtt{at}''$, implying $\gamma' < \gamma''$.

Next we consider the case where $\mathtt{at}' = \mathtt{at}''$. In this case, by the definition of $\mathcal{E}(B)$, one of the following two cases occurs:

1. Both w' and w'' are vertices or edges in the same side of B: by the property (1), it holds $d_2' = d(w') < d(w'') = d_2''$ if both w' and w'' are right vertices (resp., right edges, left vertices, and left edges) of B.

2. w' is the bottom vertex $bv(B)$ and w'' is the right edge e^b incident to $bv(B)$: in this case, $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$ and $\mathcal{B}(bv(B)) \neq \emptyset$. By the definition of $\Gamma(w')$ and $\Gamma(w'')$, we have $d_1' = d(B) = d_1''$, $\mathtt{at}' = \mathtt{at}'' = *$, $d_2' = d(w') = d(bv(B)) = d_2''$, $\mathtt{op}' = \mathtt{new\text{-}block}$ and $\mathtt{op}'' \in \{\mathtt{triangle}, \mathtt{subdivide}, \mathtt{star}\}$.

Hence $d_2' \leq d_2''$ holds. Especially when $d_2' = d_2''$, it holds $\mathtt{op}' < \mathtt{op}''$ by the ordering $\mathtt{subdivide} > \mathtt{triangle} > \mathtt{star} > \mathtt{new\text{-}block}$. This implies $\gamma' < \gamma''$. □

6.2 State Change in Entire Embeddings

For a canonical embedding G with $N = |V(G)|$, let $Ch^*(G)$ denote the set of all canonical child-embeddings $G+\gamma(u_{N+1})$ of G. Now we identify the condition for $G+\gamma(u_{N+1})$ to remain canonical, i.e., left-sibling-heavy and left-side-heavy at all blocks.

For the spine $B^1, B^2, \ldots, B^p$ of G, define sequences

$$\mathcal{E}(G) = [r_G, \mathcal{E}(B^1), \mathcal{E}(B^2), \ldots, \mathcal{E}(B^p)]$$

and

$$s(G) = [s_1 = \mathtt{sbl}(B^1; G), s_2 = \mathtt{sd}(B^1; G), s_3 = \mathtt{sbl}(B^2; G), s_4 = \mathtt{sd}(B^2; G), \ldots,$$
$$s_{2p-1} = \mathtt{sbl}(B^p; G), s_{2p} = \mathtt{sd}(B^p; G)].$$

Lemmas 4.1 and 6.1 tell that a canonical child-embedding $G' = G + \gamma(u_{N+1})$ of G is generated by applying a code $\gamma(u_{N+1}) = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$ to an element $\varepsilon \in \mathcal{E}(G)$.

Let B^h be the block in the spine with $\varepsilon \in V'(B^h) \cup E(B^h)$, i.e., the block to which a new vertex u_{N+1} is introduced, where B^h is determined by $d(B^h) = d_1$. Let B' denote the new block created by $\gamma(u_{N+1})$ (if ε is a vertex), where B' is the $(h+1)$st block in the spine of G'. Observe that G' is also left-sibling-heavy and left-side-heavy at any block not in the spine of G or at any block B^i with $i > h$ in the spine of G. Thus, to know when G' is canonical, we only need to examine states $s_1, s_2, \ldots, s_{2h-1}, s_{2h}$ in G' and the new sibling-state $s_{2h+1} = \mathtt{sbl}(B'; G')$ (if ε is a vertex) (recall that the side-state $s_{2h+2} = \mathtt{sbl}(B'; G')$ is always $\mathtt{nil}$). We easily observe the next.

Lemma 6.5. *Let G be a canonical embedding. For an element $\varepsilon \in \mathcal{E}(G)$ and a code $\gamma \in \Gamma(\varepsilon)$, let $G' = G + \gamma(u_{N+1})$ be the child-embedding of G with $\gamma(u_{N+1}) = \gamma$.*

(i) *Any state $s_j = \mathtt{stc}$ $(1 \leq j \leq 2h)$ in $s(G)$ remains unchanged in G'.*

(ii) *Any state* $s_j = \mathtt{nil}$ $(1 \le j < 2h)$ *in* $s(G)$ *remains unchanged in* G'.

(iii) *If* $s_{2h} = \mathtt{sd}(B^h; G)$ *is* $\mathtt{nil}$*, then* $\mathtt{sd}(B^h; G')$ *takes one of* $\mathtt{nil}$*,* $\mathtt{pfx}$*,* $\mathtt{eqv}$ *and* $\mathtt{stc}$ *according to* (6.3), (6.5), (6.7), (6.8)-(6.15).

(iv) *If* ε *is a vertex, then the new block* B' *containing* u_{N+1} *is the tip-block of* G'*, and* s_{2h+1} *in* $s(G')$ *is given by* $\mathtt{sbl}(B'; G')$*, which takes one of* $\mathtt{pfx}$*,* $\mathtt{eqv}$ *and* $\mathtt{stc}$ *according to* (6.2), (6.4) *and* (6.6).

Proof. (i) For $s_j = s_{2j'-1} = \mathtt{sbl}(B^{j'}; G)$ with $1 \le j \le 2h$, it holds $\mathtt{sbl}(B^{j'}; G') = \mathtt{stc}$ even if $B^{j'}$ has a left sibling $\hat{B} \in \mathcal{B}(r(B^{j'}))$, since $\sigma(G'(\hat{B}); G') = \sigma(G(\hat{B}); G) \gg \sigma(G'(B^{j'}); G') = [\sigma(G(B^{j'}); G), \gamma(u_{N+1})]$ holds by $\sigma(G(\hat{B}); G) \gg \sigma(G(B^{j'}); G)$. The case of $s_j = s_{2j'} = \mathtt{sd}(B^{j'}; G)$ with $1 \le j \le 2h$ can be treated analogously.

(ii) For $s_j = s_{2j'} = \mathtt{sd}(B^{j'}; G) = \mathtt{nil}$ with $1 \le j < 2h$, where $\sigma_{\mathtt{R}}(G(B^{j'}); G) = \emptyset$, a new vertex u_{N+1} is attached to a left vertex of $B^{j'}$ or a descendant block of a left vertex of $B^{j'}$. Hence $\sigma_{\mathtt{core}}(G'(B^{j'}); G') = \sigma_{\mathtt{core}}(G(B^{j'}); G)$ and $\sigma_{\mathtt{R}}(G'(B^{j'}); G') = \sigma_{\mathtt{R}}(G(B^{j'}); G) = \emptyset$ hold. From the left-side-heaviness of $B^{j'}$ in G, we have $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^{j'}); G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^{j'}); G)$. This implies $\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^{j'}); G') = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G'(B^{j'}); G')$. By $\sigma_{\mathtt{R}}(G'(B^{j'}); G') = \emptyset$, we have $\mathtt{sd}(B^{j'}; G') = \mathtt{nil}$.

(iii) Immediate from the analysis to obtain (6.3), (6.5), (6.7)-(6.15).

(iv) By the definition of tips, B' becomes the tip block of G'. It is immediate to see that s_{2h+1} in $s(G')$ is given by $\mathtt{sbl}(B'; G')$, which $\mathtt{sbl}(B'; G')$ is determined by (6.2), (6.4) and (6.6). □

The *copy-state* $cs(G)$ of G is defined to be the state $s_{i^*} \in \{\mathtt{eqv}, \mathtt{pfx}\}$ with the minimum index i^* in $s(G)$, and the block B^ℓ attaining $s_{i^*} = \mathtt{sbl}(B^\ell; G)$ or $s_{i^*} = \mathtt{sd}(B^\ell; G)$ is called the *dominating block* of G; let $cs(G) = \mathtt{stc}$ and $i^* = \infty$ otherwise (i.e., each state in $s(G)$ is $\mathtt{stc}$ or $\mathtt{nil}$). We distinguish two cases, $cs(G) \in \{\mathtt{stc}, \mathtt{eqv}\}$ and $cs(G) \in \{\mathtt{pfx}\}$.

Case of $cs(G) \in \{\mathtt{stc}, \mathtt{eqv}\}$: If $\mathtt{sbl}(B^\ell; G) = \mathtt{eqv}$ (resp., $\mathtt{sd}(B^\ell; G) = \mathtt{eqv}$) for the dominating block B^ℓ of G, then we see that $G' = G + \gamma(u_{N+1})$ cannot be left-sibling-heavy (resp., left-side-heavy) at B^ℓ for any element $\varepsilon \in V'(B^i) \cup E(B^i)$ with $i \ge \ell$ in the spine of G. If $cs(G) = \mathtt{eqv}$, then let $\mathcal{E}^*(G)$ be the sequence of elements obtained from $\mathcal{E}(G)$ by deleting the elements contained in a block B^i with $i \ge \ell$ in the spine of G. Let $\mathcal{E}^*(G) = \mathcal{E}(G)$ if $cs(G) = \mathtt{stc}$. The next lemma tells when G' is canonical and how to determine the copy-state of G'.

Lemma 6.6. *Let* G *be a canonical embedding with* $cs(G) \in \{\mathtt{stc}, \mathtt{eqv}\}$*. For an element* $\varepsilon \in \mathcal{E}^*(G)$ *and a code* $\gamma \in \Gamma(\varepsilon)$*, let* $G' = G + \gamma(u_{N+1})$ *be the child-embedding of* G *with* $\gamma(u_{N+1}) = \gamma$.

(i) G' *is canonical for any code* $\gamma \in \Gamma(\varepsilon)$.

(ii) *If ε is an edge in $E(B^h)$, then $cs(G') = \texttt{stc}$ if $\texttt{sd}(B^h; G') \in \{\texttt{nil}, \texttt{stc}\}$, and $cs(G') = \texttt{sd}(B^h; G')$ otherwise (i.e., $\texttt{sd}(B^h; G') \in \{\texttt{pfx}, \texttt{eqv}\}$).*

(iii) *If ε is a vertex in $V'(B^h)$, then*

$$cs(G') = \begin{cases} \texttt{sd}(B^h; G') & \textit{if } \texttt{sd}(B^h; G') \in \{\texttt{pfx}, \texttt{eqv}\} \\ \texttt{sbl}(B^{h+1}; G') & \textit{if } \texttt{sd}(B^h; G') \in \{\texttt{nil}, \texttt{stc}\} \textit{ and } \texttt{sbl}(B^{h+1}; G') \in \{\texttt{pfx}, \texttt{eqv}\} \\ \texttt{stc} & \textit{otherwise, i.e., } \texttt{sd}(B^h; G'), \texttt{sbl}(B^{h+1}; G') \in \{\texttt{nil}, \texttt{stc}\}. \end{cases} \tag{6.16}$$

Proof. (i) By definition of $cs(G) \in \{\texttt{stc}, \texttt{eqv}\}$, it holds $s_j \in \{\texttt{stc}, \texttt{nil}\}$ for all $j = 1, 2, \ldots, 2h$. Hence by Lemma 6.5, $s_j = \texttt{stc}$ with $j \leq 2h$ and $s_j = \texttt{nil}$ with $j < 2h$ remain unchanged, $s_{2h} = \texttt{nil}$ takes $\texttt{sd}(B^h; G') \in \{\texttt{nil}, \texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ and $\texttt{sbl}(B'; G') \in \{\texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ holds for the new block B' (if ε is a vertex). Thus, G' is canonical.

(ii) and (iii) are immediate from Lemmas 6.2 and 6.3, the definition of copy-state and the fact that $s_{j'} \in \{\texttt{stc}, \texttt{nil}\}$, $j = 1, 2, \ldots, 2h - 1$ holds in G'. □

Case of $cs(G) \in \{\texttt{pfx}\}$: We define the critical element $\varepsilon^* \in \mathcal{E}(G)$, the critical code $\varepsilon^* \in \Gamma(\varepsilon^*)$, and the critical block of G as follows.

Case-A. The copy-state of G is given by $\texttt{sbl}(B^\ell; G) = \texttt{pfx}$ (see Figure 6.7(a)): The *copy-end-vertex* of G is defined to be the vertex $v' = \mu^-(t^p; t^{\ell-1})$, and the *succeeding-copy-vertex* of G is defined to be the vertex v^* such that $\gamma(v^*)$ appears immediately after $\gamma(v')$ in $\sigma(G)$. The *critical code* γ^* of G is defined to be $\gamma(v^*) = (d_1, \texttt{at}, d_2, \texttt{op}, c^*)$.

Case-B. The copy-state of G is given by $\texttt{sd}(B^\ell; G) = \texttt{pfx}$ (see Figure 6.7(b)): The *copy-end-vertex* of G is defined to be the vertex $v' = \mu(t^p; t^{\ell-1})$, and the *succeeding-copy-vertex* of G is defined to be the vertex v^* such that $\gamma(v^*)$ appears immediately after $\gamma(v')$ in $\sigma(G)$. For the code $\gamma(v^*) = (d_1, \texttt{at}, d_2, \texttt{op}, c^*)$, the *critical code* γ^* of G is defined to be $\gamma^* := \overline{\gamma}(v^*)$ if $\texttt{at} \in \{\texttt{w}^\texttt{L}, \texttt{h}^\texttt{L}\}$ (i.e., v^* is a left wing-vertex of B^ℓ or v^* is a child vertex of a left vertex of B^ℓ), and to be $\gamma^* := (d_1, \texttt{at}, d_2, \texttt{op}, c^*)$ otherwise.

In Case-A and B, we can set the code of a new vertex u_{N+1} by $\gamma(u_{N+1}) := \gamma^*$, i.e., there is a unique element $\varepsilon^* \in \mathcal{E}(G)$ to which $\gamma(u_{N+1})$ is applicable. We call such an element ε^* the *critical element* of G, and call the block B^k such that $\varepsilon^* \in V'(B^k) \cup E(B^k)$ the *critical block* of G. We call the color entry c^* in the critical code γ^* the *critical color* of G. A child-embedding $G' = G + \gamma(u_{N+1})$ of G is called *critical* if $\gamma(u_{N+1})$ is the critical code γ^* of G.

Let $\mathcal{E}^*(G)$ denote the subsequence of $\mathcal{E}(G)$ that consists of all elements from r_G to the critical element ε^* of G. Thus,

$$\mathcal{E}^*(G) = [r_G, \mathcal{E}(B^1), \mathcal{E}(B^2), \ldots, \mathcal{E}^*(B^k)],$$

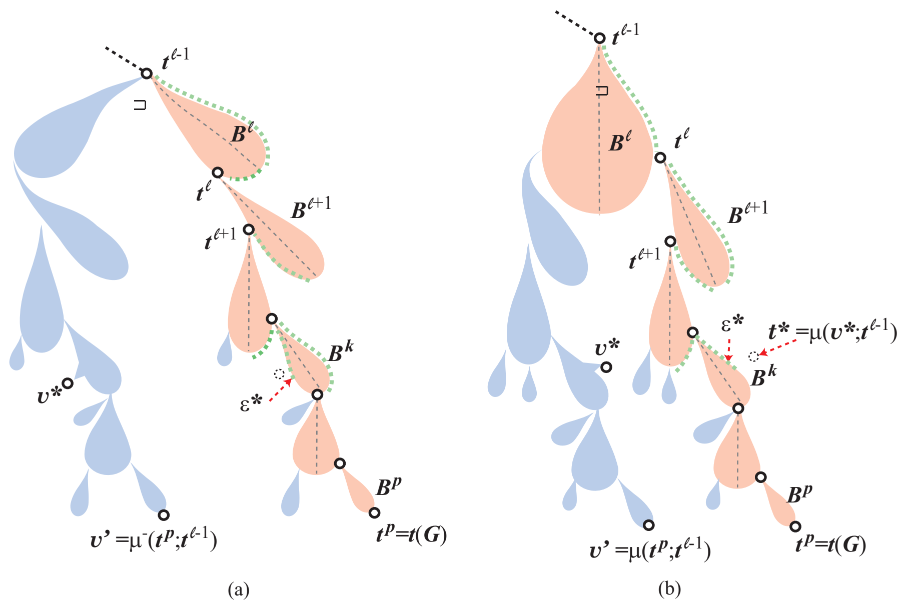

Figure 6.7: (a) Block B^ℓ with copy-state $\mathtt{sbl}(B^\ell; G) = \mathtt{pfx}$; (b) Block B^ℓ with copy-state $\mathtt{sd}(B^\ell; G') = \mathtt{pfx}$.

where $\mathcal{E}^*(B^k)$ denotes the subsequence of $\mathcal{E}(B^k)$ that consists of all elements from preceding the critical element ε^* of G (including ε^*). Note that the critical element ε^* is the last one in $\mathcal{E}^*(G)$.

Lemma 6.7. *Let G be a canonical embedding with $cs(G) = \mathtt{pfx}$, and let $cs(G) = s_{i^*} = \mathtt{pfx}$ be given by $s_{2\ell-1} = \mathtt{sbl}(B^\ell; G) = \mathtt{pfx}$. For an element $\varepsilon \in \mathcal{E}^*(G)$ and a code $\gamma \in \Gamma(\varepsilon)$, let $G' = G + \gamma(u_{N+1})$ be the child-embedding of G with $\gamma(u_{N+1}) = \gamma$.*

(i) *G' is not left-sibling-heavy at B^ℓ if "$\varepsilon \notin \mathcal{E}^*(G)$" or "$\varepsilon = \varepsilon^*$ and $\gamma > \gamma^*$."*

(ii) *It holds $\mathtt{sbl}(B^\ell; G') = \mathtt{stc}$ if "$\varepsilon \in \mathcal{E}^*(G) - \{\varepsilon^*\}$" or "$\varepsilon = \varepsilon^*$ and $\gamma^* > \gamma$."*

(iii) *For the critical G', it holds $\mathtt{sbl}(B^\ell; G') = \mathtt{eqv}$ if the succeeding-copy-vertex v^* of G is followed immediately by the first head-vertex $x_1 = \ell v(B^\ell)$ in $\sigma(G)$; it holds $\mathtt{sbl}(B^\ell; G') = \mathtt{pfx}$ otherwise.*

Proof. Since $cs(G) = s_{i^*} = \mathtt{pfx}$, a new code $\gamma(u_{N+1})$ corresponds to the code of the succeeding-copy-vertex v^* of G in $\sigma(G')$ to determine the state s_{i^*} in G'. Let $\gamma^* = (d_1^*, \mathtt{at}^*, d_2^*, \mathtt{op}^*, c^*)$ be the critical code $\gamma(v^*)$ of G, where $d_1^* = d(B^k)$.

(i) It suffices to show that $\gamma(u_{N+1}) > \gamma(v^*)$ if "$\varepsilon \notin \mathcal{E}^*(G)$" or "$\varepsilon = \varepsilon^*$ and $\gamma > \gamma^*$." For each element $\varepsilon \in V'(B^i) \cup E(B^i)$ with $i > k$, any code $\gamma = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$ is lexicographically smaller than γ^* since $d_1 = d(B^i) > d(B^k) = d_1^*$ holds in the spine of G. This indicates $\gamma(u_{N+1}) > \gamma^*$.

Assume that $\varepsilon \in \mathcal{E}(B^k) - \mathcal{E}^*(B^k)$ and $\gamma(u_{N+1}) = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$, where $d_1 = d(B^k) = d_1^*$. If ε^* precedes ε in $\mathcal{E}(B^k)$, then $\gamma(u_{N+1}) > \gamma^*$ holds by Lemma 6.4. Finally consider the case where $\varepsilon = \varepsilon^*$. In this case, clearly $\gamma(u_{N+1}) > \gamma^*$ if $\gamma > \gamma^*$.

(ii) It suffices to show that $\gamma(v^*) > \gamma(u_{N+1})$ if "$\varepsilon \in \mathcal{E}^*(G) - \{\varepsilon^*\}$" or "$\varepsilon = \varepsilon^*$ and $\gamma^* > \gamma$." For each element $\varepsilon \in V'(B^i) \cup E(B^i)$ with $i < k$, any code $\gamma = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$ is lexicographically smaller than γ^* since $d_1 = d(B^i) < d(B^k) = d_1^*$ holds in the spine of G. This indicates $\gamma(v^*) > \gamma(u_{N+1})$.

Assume that $\varepsilon \in \mathcal{E}^*(B^k)$ and $\gamma(u_{N+1}) = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$, where $d_1 = d(B^k) = d_1^*$. If ε precedes ε^* in $\mathcal{E}(B^k)$, then $\gamma^* > \gamma(u_{N+1})$ holds by Lemma 6.4. Finally consider the case where $\varepsilon = \varepsilon^*$. In this case, clearly $\gamma^* > \gamma(u_{N+1})$ if $\gamma^* > \gamma$.

(iii) Let $\hat{B} \in \mathcal{B}(t^{\ell-1})$ be the left sibling of B^ℓ. If G' is critical, i.e., $\varepsilon = \varepsilon^*$ and $\gamma^* = \gamma(u_{N+1})$, then $\sigma(G'(B^\ell); G') = [\sigma(G(B^\ell); G), \gamma(u_{N+1})]$ is a prefix of $\sigma(G'(\hat{B}); G') = \sigma(G(\hat{B}); G)$. Hence it holds $\mathtt{sbl}(B^\ell; G') = \mathtt{eqv}$ if $|\sigma(G(\hat{B}); G)| = |\sigma(G(B^\ell); G)| + 1$, i.e., the succeeding-copy-vertex v^* of G is followed immediately by the first head-vertex $x_1 = \ell v(B^\ell)$ in $\sigma(G)$. It holds $\mathtt{sbl}(B^\ell; G') = \mathtt{pfx}$ otherwise. □

Lemma 6.8. *Let G be a canonical embedding with $cs(G) = \mathtt{pfx}$, and let $cs(G) = s_{i^*} = \mathtt{pfx}$ be given by $s_{2\ell} = \mathtt{sd}(B^\ell; G) = \mathtt{pfx}$. For an element $\varepsilon \in \mathcal{E}^*(G)$ and a code $\gamma \in \Gamma(\varepsilon)$, let $G' = G + \gamma(u_{N+1})$ be the child-embedding of G with $\gamma(u_{N+1}) = \gamma$.*

(i) *G' is not left-side-heavy at B^ℓ if "$\varepsilon \notin \mathcal{E}^*(G)$" or "$\varepsilon = \varepsilon^*$ and $\gamma > \gamma^*$."*

(ii) *It holds $\mathtt{sd}(B^\ell; G') = \mathtt{stc}$ if "$\varepsilon \in \mathcal{E}^*(G) - \{\varepsilon^*\}$" or "$\varepsilon = \varepsilon^*$ and $\gamma^* > \gamma$."*

(iii) *For the critical G', it holds $\mathtt{sd}(B^\ell; G') = \mathtt{eqv}$ if the succeeding-copy-vertex v^* of G is followed immediately by the vertex u which code $\gamma(u)$ appears as the first code in $\sigma_{\mathtt{R}}(G(B^\ell); G)$; it holds $\mathtt{sd}(B^\ell; G') = \mathtt{pfx}$ otherwise.*

Proof. Since $cs(G) = s_{i^*} = \mathtt{sd}(B^\ell; G) = \mathtt{pfx}$, a new vertex u_{N+1} corresponds to the succeeding-copy-vertex v^* of G, whose code $\gamma(u_{N+1})$ determines the side-state of s_{i^*} in G'. Let $\gamma^* = (d_1^*, \mathtt{at}^*, d_2^*, \mathtt{op}^*, c^*)$ be the critical code $\overline{\gamma}(v^*)$ of G, where $d_1^* = d(B^k)$.

(i) It suffices to show that $\gamma(u_{N+1}) > \gamma^*$ if "$\varepsilon \notin \mathcal{E}^*(G)$", since clearly $\gamma(u_{N+1}) > \gamma^*$ if "$\varepsilon = \varepsilon^*$" and "$\gamma^* < \gamma$."

For each element $\varepsilon \in \mathcal{E}(B^i)$ with $i > k$, any code $\gamma = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$ is lexicographically larger than γ^* since $d_1 = d(B^i) > d(B^k) = d_1^*$ holds in the spine of G. This indicates $\gamma(u_{N+1}) > \gamma^*$.

For each element $\varepsilon \in \mathcal{E}(B^k) - \mathcal{E}^*(B^k)$ and $\gamma(u_{N+1}) = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$, where $d_1 = d(B^k) = d_1^*$. Note that ε precedes ε^* in $\mathcal{E}(B^k)$. By Lemma 6.4, $\gamma(u_{N+1}) > \gamma^*$ holds.

(ii) It suffices to show that $\gamma^* > \gamma(u_{N+1})$ if "$\varepsilon \in \mathcal{E}^*(G) - \{\varepsilon^*\}$," since clearly $\gamma^* > \gamma(u_{N+1})$ holds if "$\varepsilon = \varepsilon^*$" and "$\gamma^* > \gamma$."

For each element $\varepsilon \in \mathcal{E}(B^i)$ with $i < k$, any code $\gamma = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$ is lexicographically smaller than γ^* since $d_1 = d(B^i) < d(B^k) = d_1^*$ holds in the spine of G. This indicates $\gamma^* > \gamma(u_{N+1})$.

For each element $\varepsilon \in \mathcal{E}^*(B^k) - \{\varepsilon^*\}$ and $\gamma(u_{N+1}) = (d_1, \mathtt{at}, d_2, \mathtt{op}, c) \in \Gamma(\varepsilon)$, where $d_1 = d(B^k) = d_1^*$. Note that ε precedes ε^* in $\mathcal{E}(\mathcal{B})$. By Lemma 6.4, $\gamma^* > \gamma(u_{N+1})$ holds.

(iii) If G' is critical, i.e., $\varepsilon = \varepsilon^*$ and $\gamma(u_{N+1}) = \gamma^*$, then $[\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^\ell); G'), \sigma_{\mathtt{L}}(G'(B^\ell); G')]$ is a prefix of $[\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G'(B^\ell); G'), \overline{\sigma_{\mathtt{R}}}(G'(B^\ell); G')] = [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^\ell); G), \overline{\sigma_{\mathtt{R}}}(G(B^\ell); G), \overline{\gamma}(u_{N+1})]$. Hence it holds $\mathtt{sd}(B^\ell; G') = \mathtt{eqv}$ if $|\sigma_{\mathtt{L}}(G(B^\ell); G)| = |\sigma_{\mathtt{R}}(G(B^\ell); G)| + 1$, i.e., the succeeding-copy-vertex v^* of G is followed immediately by the vertex u whose code $\gamma(u)$ appears as the first code in $\sigma_{\mathtt{R}}(G(B^\ell); G)$. Otherwise, it holds $\mathtt{sd}(B^\ell; G') = \mathtt{pfx}$. □

Lemma 6.9. *Let G be a canonical embedding with $cs(G) = \mathtt{pfx}$. For an element $\varepsilon \in \mathcal{E}^*(G)$ and a code $\gamma \in \Gamma(\varepsilon)$, let $G' = G + \gamma(u_{N+1})$ be the child-embedding of G with $\gamma(u_{N+1}) = \gamma$.*

(i) *If G' is critical, then each state $s_{j'} = \mathtt{pfx}$ $(i^* < j' \le 2h)$ in $s(G)$ changes into $s_{j'} \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$ in G'.*

(ii) *If G' is not critical, then each state $s_{j'} = \mathtt{pfx}$ $(i^* < j' \le 2h)$ in $s(G)$ changes into $s_{j'} = \mathtt{stc}$ in G'.*

Proof. (I) Assume that $s_{i^*} = \mathtt{sbl}(B^\ell; G) = \mathtt{pfx}$, i.e., $\sigma(G(\hat{B}^\ell); G) \sqsupset \sigma(G(B^\ell); G)$ holds for the left sibling $\hat{B}^\ell$ of B^ℓ (see Figures 6.8 and 6.9). Let B^j be the block B^j such that $s_{j'} = s_{2j-1} = \mathtt{pfx}$ in the spine of G, and let $t' = \mu^-(t^{j-1}; t^{\ell-1})$ for $t^{j-1} = r(B^j)$ and $t^{\ell-1} = r(B^\ell)$.

First we show that it holds $s_{j'} = \mathtt{sbl}(B^j; G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$ in G', and that if G' is not critical, then $s_{j'} = \mathtt{sbl}(B^j; G')$ is $\mathtt{stc}$. Since $\mathtt{sbl}(B^j; G) = s_{j'} = \mathtt{pfx}$, block B^j has a left sibling $B^j_3 \in \mathcal{B}(t^{j-1})$. Note that $[\sigma(G(B^j_3); G), \sigma(G(B^j); G)]$ appears as the last subsequence of $\sigma(G(B^\ell); G)$. Then there are blocks $B^j_1, B^j_2 \in \mathcal{B}(t')$ such that $[\sigma(G(B^j_1); G), \sigma(G(B^j_2); G)]$ appears as the last subsequence of $\sigma(G(\hat{B}^\ell); G)$. By $\sigma(G(\hat{B}^\ell); G) \sqsupset \sigma(G(B^\ell); G)$, we see that

$$\sigma(G(B^j_1); G) = \sigma(G(B^j_3); G) \text{ and } \sigma(G(B^j_2); G) \sqsupset \sigma(G(B^j); G).$$

See Figure 6.8. Since G is left-sibling-heavy at B^j_2, it holds

$$\sigma(G(B^j_1); G) \ge \sigma(G(B^j_2); G)$$

by Lemma 5.1. In G', a new vertex u_{N+1} is added to the block B^h with $h \ge j$ in the spine, and we have $\sigma(G(B^j_2); G) \ge \sigma(G'(B^j); G')$. More specifically, if G' is critical, then

$$\sigma(G(B^j_2); G) \sqsupseteq \sigma(G'(B^j); G');$$

if G' is not critical, then

$$\sigma(G(B^j_2); G) \gg \sigma(G'(B^j); G').$$

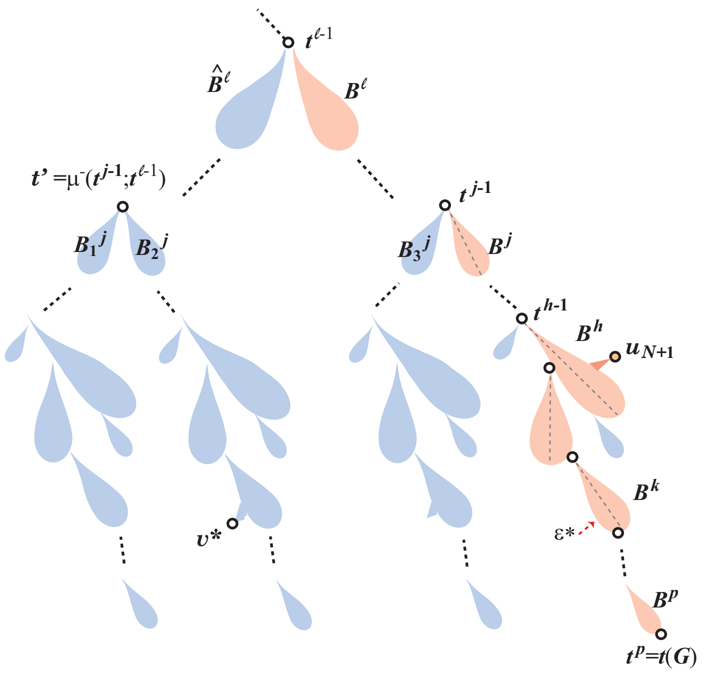

Figure 6.8: Critical block B^ℓ with copy-state $cs(G) = \mathtt{sbl}(B^\ell;G) = \mathtt{pfx}$, and a block B^j $(j > \ell)$ with $s_{2j-1} = \mathtt{sbl}(B^j;G) = \mathtt{pfx}$.

Therefore we have $\sigma(G'(B_3^j);G') = \sigma(G(B_1^j);G) \geq \sigma(G(B_2^j);G) \geq \sigma(G'(B^j);G')$, i.e., $s_{j'} = \mathtt{sbl}(B^j;G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$ in G'. In particular, if G' is not critical, then $\sigma(G'(B_3^j);G') \gg \sigma(G'(B^j);G')$, i.e., $s_{j'} = \mathtt{sbl}(B^j;G') = \mathtt{stc}$ in G'.

Next we show that, for the block B^j such that $s_{j'} = s_{2j} = \mathtt{pfx}$, it holds $s_{j'} = \mathtt{sd}(B^j;G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$ in G', and if G' is not critical, then $s_{j'} = \mathtt{sd}(B^j;G')$ is $\mathtt{stc}$. Note that $\sigma(G(B^j);G)$ appears as the last subsequence of $\sigma(G(B^\ell);G)$. There is a block $B_1^j \in \mathcal{B}(t')$ such that $\sigma(G(B_1^j);G)$ appears as the last subsequence of $\sigma(G(\hat{B}^\ell);G)$. By $\sigma(G(\hat{B}^\ell);G) \sqsupset \sigma(G(B^\ell);G)$, we see that $\sigma(G(B_1^j);G) \sqsupset \sigma(G(B^j);G)$. By Lemma 5.2, $\mathtt{sd}(B^j;G) = \mathtt{pfx}$ implies that

$$\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G) \text{ and } \sigma_{\mathtt{L}}(G(B^j);G) \sqsupset \overline{\sigma_{\mathtt{R}}}(G(B^j);G) \neq \emptyset.$$

See Figure 6.9. From $\sigma_{\mathtt{R}}(G(B^j);G) \neq \emptyset$, we have

$$\sigma_{\mathtt{core}}(G(B_1^j);G) = \sigma_{\mathtt{core}}(G(B^j);G),\ \sigma_{\mathtt{L}}(G(B_1^j);G) = \sigma_{\mathtt{L}}(G(B^j);G),\ \sigma_{\mathtt{R}}(G(B_1^j);G) \sqsupset \sigma_{\mathtt{R}}(G(B^j);G).$$

Hence $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B_1^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B_1^j);G)$ must hold by $\sigma_{\mathtt{core}}(G(B_1^j);G) = \sigma_{\mathtt{core}}(G(B^j);G)$ and $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G)$.

Since G is left-side-heavy at B_1^j, it holds

$$[\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B_1^j);G), \sigma_{\mathtt{L}}(G(B_1^j);G)] \geq [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B_1^j);G), \overline{\sigma_{\mathtt{R}}}(G(B_1^j);G)]$$

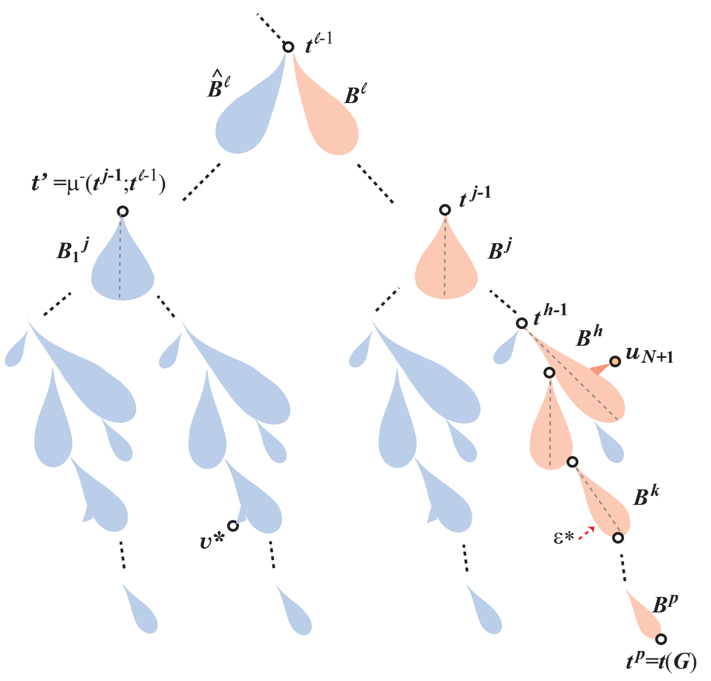

Figure 6.9: Critical block B^ℓ with copy-state $cs(G) = \texttt{sbl}(B^\ell;G) = \texttt{pfx}$, and a block B^j ($j > \ell$) with $s_{2j} = \texttt{sd}(B^j;G) = \texttt{pfx}$.

by Lemma 5.2. From this and $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j_1);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j_1);G)$, we have

$$\sigma_{\mathtt{L}}(G(B^j);G) = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G)$$

In G', a new vertex u_{N+1} is added to the right side of B^h (if $j = h$) or block B^h with $h > j$ in the spine, and we have $[\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^j);G'), \sigma_{\mathtt{L}}(G'(B^j);G')] \geq [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G'), \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')]$, where $\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^j);G') = \sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G'(B^j);G')$. More specifically, if G' is critical, then

$$\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j);G) = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \sqsupseteq \overline{\sigma_{\mathtt{R}}}(G'(B^j);G');$$

if G' is not critical, then

$$\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j);G) = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \gg \overline{\sigma_{\mathtt{R}}}(G'(B^j);G').$$

Therefore we have $\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')$, i.e., $s_{j'} = \texttt{sd}(B^j;G') \in \{\texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ in G'. In particular, if G' is not critical, then $\sigma_{\mathtt{L}}(G'(B^j);G') \gg \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')$, i.e., $s_{j'} = \texttt{sd}(B^j;G') = \texttt{stc}$ in G'.

(II) Assume that $s_{i^*} = \texttt{sd}(B^\ell;G) = \texttt{pfx}$, i.e., $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^\ell);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^\ell);G)$ and $\sigma_{\mathtt{L}}(G(B^\ell);G)] \sqsupset \overline{\sigma_{\mathtt{R}}}(G(B^\ell);G) \neq \emptyset$. Recall that, for a vertex $u \in V(G(B^\ell)) - \{r(B^\ell)\}$, we define $\overline{\gamma}(u)$ to be the code obtained from $\gamma(u)$ by replacing the second entry $w^{\mathtt{R}}$ (resp., $h^{\mathtt{R}}$)

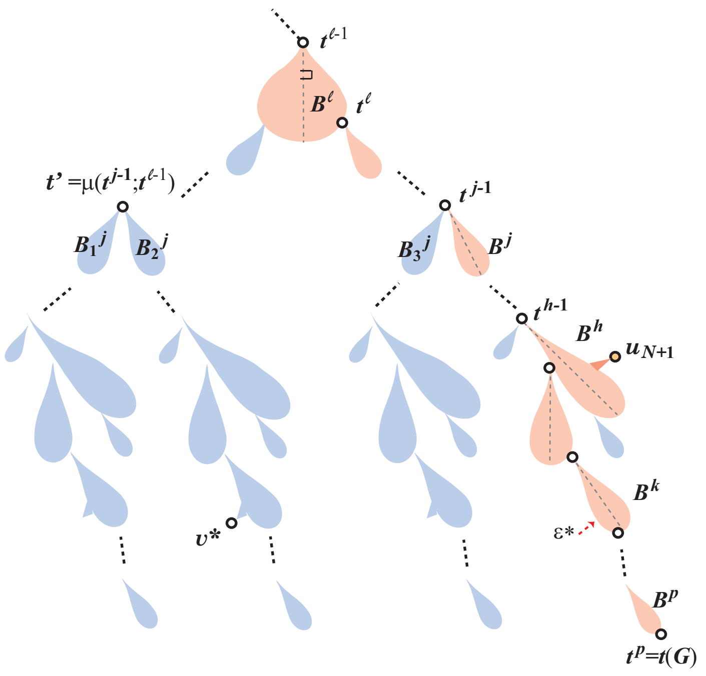

Figure 6.10: Critical block B^ℓ with copy-state $cs(G) = \mathtt{sd}(B^\ell; G) = \mathtt{pfx}$, and a block B^j ($j > \ell$) with $s_{2j-1} = \mathtt{sbl}(B^j; G) = \mathtt{pfx}$.

with w^{L} (resp., h^{L}) if u is a right wing-vertex of B^ℓ (resp., u is a child-vertex of a vertex in the right side of B^ℓ), and we set $\overline{\gamma}(u) = \gamma(u)$ otherwise. Let $t^{\ell-1} = r(B^\ell)$.

First we show that, for a block B^j such that $s_{j'} = s_{2j-1} = \mathtt{sbl}(B^j; G) = \mathtt{pfx}$, it holds $s_{j'} = \mathtt{sbl}(B^j; G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$, and if G' is not critical, then $s_{j'} = \mathtt{sbl}(B^j; G')$ is $\mathtt{stc}$.

Let $t^{j-1} = r(B^j)$. Since $\mathtt{sbl}(B^j; G) = \mathtt{pfx}$, the block B^j has a left sibling $B^j_3 \in \mathcal{B}(t^{j-1})$. Note that $[\sigma(G(B^j_3); G), \sigma(G(B^j); G)]$ appears as the last subsequence of $\sigma^{\mathrm{R}}_{\mathtt{dscd}}(G(B^\ell); G)$, and hence it appears as the last subsequence of $\sigma_{\mathtt{R}}(G(B^\ell); G)$. Accordingly, let $t' = \mu(t^{j-1}; t^{\ell-1})$. There are blocks $B^j_1, B^j_2 \in \mathcal{B}(t')$ such that $[\sigma(G(B^j_1); G), \sigma(G(B^j_2); G)]$ appears as the last subsequence of $\sigma^{\mathrm{L}}_{\mathtt{dscd}}(G(B^\ell); G)$, and hence it appears as the last subsequence of $\sigma_{\mathtt{L}}(G(B^\ell); G)$.

By $\sigma_{\mathtt{L}}(G(B^\ell); G)] \sqsupset \overline{\sigma_{\mathtt{R}}}(G(B^\ell); G) \neq \emptyset$ and the definition of $\overline{\gamma}$, we see that

$$\sigma(G(B^j_1); G) = \sigma(G(B^j_3); G) \text{ and } \sigma(G(B^j_2); G) \sqsupset \sigma(G(B^j); G).$$

See Figure 6.10. Since G is left-sibling-heavy at B^j_2, it holds

$$\sigma(G(B^j_1); G) \geq \sigma(G(B^j_2); G)$$

by Lemma 5.1. In G', a new vertex u_{N+1} is added to the block B^h with $h \geq j$ in the spine, and we have $\sigma(G(B^j_2); G) \geq \sigma(G'(B^j); G')$. More specifically, if G' is critical, then

$$\sigma(G(B^j_2); G) \sqsupseteq \sigma(G'(B^j); G');$$

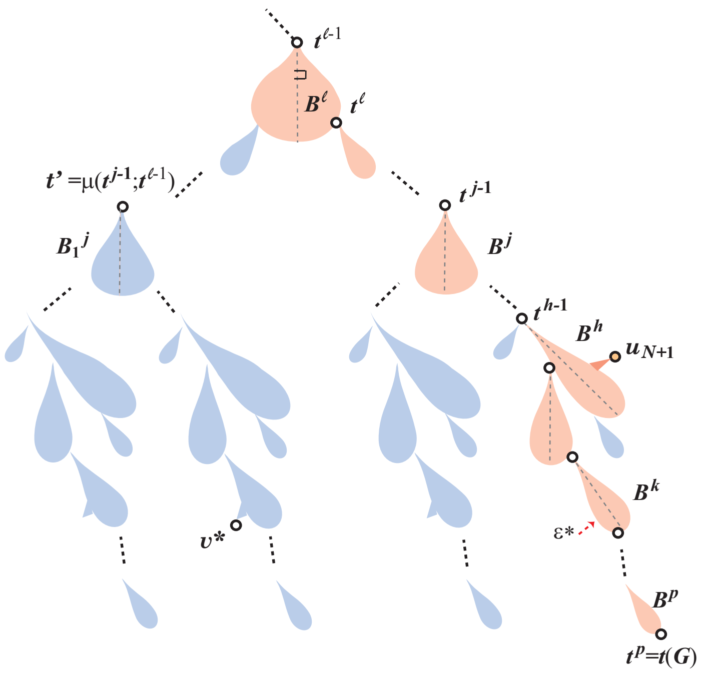

Figure 6.11: Critical block B^ℓ with copy-state $cs(G) = \mathtt{sd}(B^\ell; G) = \mathtt{pfx}$, and a block B^j ($j > \ell$) with $s_{2j} = \mathtt{sd}(B^j; G) = \mathtt{pfx}$.

if G' is not critical, then

$$\sigma(G(B_2^j); G) \gg \sigma(G'(B^j); G').$$

Therefore we have $\sigma(G'(B_3^j); G') = \sigma(G(B_3^j); G) = \sigma(G(B_1^j); G) \geq \sigma(G(B_2^j); G) \geq \sigma(G'(B^j); G')$, i.e., $s_{j'} = \mathtt{sbl}(B^j; G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$. In particular, if G' is not critical, then $\sigma(G'(B_3^j); G') \gg \sigma(G'(B^j); G')$, i.e., $s_{j'} = \mathtt{sbl}(B^j; G') = \mathtt{stc}$.

Next we show that, for a block B^j such that $s_{j'} = s_{2j} = \mathtt{sd}(B^j; G') = \mathtt{pfx}$, it holds $s_{j'} = \mathtt{sd}(B^j; G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$, and if G' is not critical, then $s_{j'} = \mathtt{sd}(B^j; G')$ is $\mathtt{stc}$. Note that $\sigma(G(B^j); G)$ appears as the last subsequence of $\sigma^{\mathtt{R}}_{\mathtt{dscd}}(G(B^\ell); G)$, and hence appears as the last subsequence of $\sigma_{\mathtt{R}}(G(B^\ell); G)$. Let $t' = \mu(t^{j-1}; t^{\ell-1})$ for $t^{j-1} = r(B^j)$. There is a block $B_1^j \in \mathcal{B}(t')$ such that $\sigma(G(B_1^j); G)$ appears as the last subsequence of $\sigma_{\mathtt{L}}(G(B^\ell); G)$. By $\sigma_{\mathtt{L}}(G(B^\ell); G) \sqsupset \overline{\sigma_{\mathtt{R}}}(G(B^\ell); G) \neq \emptyset$ and the definition of $\overline{\gamma}$, we see that $\sigma(G(B_1^j); G) \sqsupset \sigma(G(B^j); G)$. By Lemma 5.2, $\mathtt{sd}(B^j; G) = \mathtt{pfx}$ implies that

$$\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j); G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j); G) \text{ and } \sigma_{\mathtt{L}}(G(B^j); G) \sqsupset \overline{\sigma_{\mathtt{R}}}(G(B^j); G) \neq \emptyset.$$

See Figure 6.11.

From $\sigma_{\mathtt{R}}(G(B^j); G) \neq \emptyset$, we have

$$\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B_1^j); G) = \sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j); G),\ \sigma^{\mathtt{R}}_{\mathtt{core}}(G(B_1^j); G) = \sigma^{\mathtt{R}}_{\mathtt{core}}(G(B^j); G),$$

$$\sigma_{\mathtt{L}}(G(B_1^j); G) = \sigma_{\mathtt{L}}(G(B^j); G), \sigma_{\mathtt{R}}(G(B_1^j); G) \sqsupset \sigma_{\mathtt{R}}(G(B^j); G).$$

Hence $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j_1);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j_1);G)$ must hold by $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j_1);G) = \sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G)$, $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G)$, and $\sigma^{\mathtt{R}}_{\mathtt{core}}(G(B^j);G) = \sigma^{\mathtt{R}}_{\mathtt{core}}(G(B^j_1);G)$.

Since G is left-side-heavy at B^j_1, it holds

$$[\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j_1);G), \sigma_{\mathtt{L}}(G(B^j_1);G)] \geq [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j_1);G), \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G)]$$

by Lemma 5.2. From this and $\sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j_1);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j_1);G)$, we have

$$\sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G).$$

In G', a new vertex u_{N+1} is added to the right side of B^h (if $j = h$) or block B^h with $h > j$ in the spine, and we have $[\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^j);G'), \sigma_{\mathtt{L}}(G'(B^j);G')] \geq [\overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G'(B^j);G'), \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')]$, where $\sigma^{\mathtt{L}}_{\mathtt{core}}(G'(B^j);G') = \sigma^{\mathtt{L}}_{\mathtt{core}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G(B^j);G) = \overline{\sigma^{\mathtt{R}}_{\mathtt{core}}}(G'(B^j);G')$. More specifically, if G' is critical, then

$$\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j);G) = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \sqsupseteq \overline{\sigma_{\mathtt{R}}}(G'(B^j);G');$$

if G' is not critical, then

$$\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j);G) = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \gg \overline{\sigma_{\mathtt{R}}}(G(B^j);G').$$

Therefore we have $\sigma_{\mathtt{L}}(G'(B^j);G') = \sigma_{\mathtt{L}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G(B^j_1);G) \geq \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')$, i.e., $s_{j'} = \mathtt{sd}(B^j;G') \in \{\mathtt{pfx}, \mathtt{eqv}, \mathtt{stc}\}$. In particular, if G' is not critical, then $\sigma_{\mathtt{L}}(G'(B^j);G') \gg \overline{\sigma_{\mathtt{R}}}(G'(B^j);G')$, and hence $s_{j'} = \mathtt{sd}(B^j;G') = \mathtt{stc}$. □

We are ready to characterize when $G + \gamma(u_{N+1})$ is canonical.

Lemma 6.10. *Let G be a canonical embedding with $cs(G) = \mathtt{pfx}$. For an element $\varepsilon \in \mathcal{E}^*(G)$ and a code $\gamma \in \Gamma(\varepsilon)$, let $G' = G + \gamma(u_{N+1})$ be the child-embedding of G with $\gamma(u_{N+1}) = \gamma$.*

(i) *If ε is not the critical element ε^* of G, then G' is canonical for any code $\gamma \in \Gamma(\varepsilon)$.*

(ii) *Let ε be the critical element ε^* of G. Then G' is canonical if and only if $\gamma^* \geq \gamma$.*

(iii) *If G' is critical, then $cs(G') = s_{i^*} \in \{\mathtt{pfx}, \mathtt{eqv}\}$ holds and s_{i^*} in G' takes either* pfx *or* eqv *according to Lemmas 6.7(iii) and 6.8(iii).*

(iv) *If G' is not critical and ε is an edge in $E(B^h)$, then $cs(G') = \mathtt{stc}$ if $\mathtt{sd}(B^h;G') \in \{\mathtt{nil}, \mathtt{stc}\}$ and $cs(G') = \mathtt{sd}(B^h;G')$ otherwise (i.e., $\mathtt{sd}(B^h;G') \in \{\mathtt{pfx}, \mathtt{eqv}\}$).*

(v) *If G' is not critical and ε is a vertex in $V'(B^h)$, then*

$$cs(G') = \begin{cases} \mathtt{sd}(B^h;G') & \text{if } \mathtt{sd}(B^h;G') \in \{\mathtt{pfx}, \mathtt{eqv}\} \\ \mathtt{sbl}(B^{h+1};G') & \text{if } \mathtt{sd}(B^h;G') \in \{\mathtt{nil}, \mathtt{stc}\} \text{ and } \mathtt{sbl}(B^{h+1};G') \in \{\mathtt{pfx}, \mathtt{eqv}\} \\ \mathtt{stc} & \text{otherwise, i.e., } \mathtt{sd}(B^h;G'), \mathtt{sbl}(B^{h+1};G') \in \{\mathtt{nil}, \mathtt{stc}\}. \end{cases} \tag{6.17}$$

Proof. (i) and (ii) are immediate from Lemmas 6.2, 6.3, 6.5, 6.7, 6.8 and 6.9.

(iii) is immediate from Lemmas 6.7(iii) and 6.8(iii).

(iv) and (v) are immediate from Lemmas 6.2, 6.3, 6.5 and 6.9, and the definition of copy-state. □

Chapter 7

Algorithm

7.1 Outline of Algorithm

Based on the results in the previous sections, we can see that all canonical embeddings with at most n vertices to be enumerated are arranged in a specific order indicated by the defined parent-child relationship between canonical embeddings. This order implies that all canonical embeddings can be outputted in a recursive way similar to the scheme of the depth-first search. That is, starting from a graph consisting of a single vertex, we recursively enumerate its first canonical child-embedding by appending a new vertex to the current graph until reaching an embedding that has no child-embedding; and then we backtrack to the most recent embedding which has child-embeddings not being enumerated yet. Note that during the enumeration, we only output the difference between an enumerated embedding with its parent embedding.

The above idea of enumeration is presented as the following Algorithm GENERATE, where "/*. . . */" indicates a comment. How to implement the procedure of the algorithm will be explained with more details in Sections 7.2-7.5.

Algorithm GENERATE($n, \mathcal{C}$)
Input: An integer $n \geq 1$ and a set $\mathcal{C} = (c_1, c_2, \ldots, c_K)$ of K colors.
Output: All canonical embeddings of colored and rooted outerplanar graphs with at most n vertices.

```
1 begin
2     for each $c \in \mathcal{C}$ do
3         Let $G$ be the graph consisting of a single vertex $u_1 (= r_G)$ with $c(u_1) = c$;
4         Let $B_r$ be the imaginary parent-block of the root $r_G$;
5         Output $c(u_1) = c$;
6         $\varepsilon_1 := u_1$;  $\gamma_1 := (0, *, 0, \texttt{new-block}, c_1)$;
7         GEN($G, B_r, \varepsilon_1, \gamma_1$)
```

```
8      endfor
9 end.
```

Given an embedding G with N vertices, a block B, an element $\varepsilon \in \mathcal{E}(B)$ and a code $\gamma \in \Gamma(\varepsilon)$, Procedure GEN$(G, B, \varepsilon, \gamma)$ recursively generates all descendant-embeddings of G with at most n vertices, which is given as follows.

```
Procedure GEN($G, B, \varepsilon, \gamma$)
/* Let $N = |V(G)| \in [1, n-1]$, and $u_{N+1}$ be a new vertex that will be created. */
1 begin
2      $G'$ :=Append($G, B, \varepsilon, \gamma$); /* Compute a child-embedding $G' = G + \gamma$ of $G$. */
3      if $N$ is odd then Output $\gamma(u_{N+1}) = \gamma$ endif;
4      if $N + 1 < n$ then $\varepsilon_1 := r_G$;  $\gamma_1 := (0, *, 0, \texttt{new-block}, c_1)$; GEN($G', B_r, \varepsilon_1, \gamma_1$) endif;
5      if $N$ is even then Output $\gamma(u_{N+1}) = \gamma$ endif;
6      RemoveTip($G'$); /* Compute $G$ obtained from $G'$ by removing the tip of $G'$. */
7      [$B', \varepsilon', \gamma'$] :=NextCode($B, \varepsilon, \gamma; G$); /* Calculate three parameters $B'$, $\varepsilon'$ and $\gamma'$ to
        generate the next child-embedding of $G$ */
8      if [$B', \varepsilon', \gamma'$] $\neq \emptyset$ then GEN($G, B', \varepsilon', \gamma'$) endif
9 end.
```

Section 7.2 will introduce data structures of each canonical embedding maintained in the algorithm. Sections 7.3-7.5 will explain how to realize the three routines—Append in Line 2, RemoveTip in Line 6 and NextCode in Line 7 of **Procedure** GEN, respectively. Note that there are parts of data involved in these three routines but not maintained can be calculated by the maintained data structures. The calculation of these unmaintained data will be presented in Appendix.

7.2 Data Structures

The goal of this section is to define sufficient and compact data structures for each canonical embedding. In this work, each rooted outerplanar embedding consists of blocks, where each block contains left/right head-vertices, axial-vertices, wing-vertices and bottom vertex, and left/right head-edges and axial-edges. Further, each colored and rooted embedding G is encoded by a code sequence $\sigma(G)$, where code $\gamma(u)$ of each vertex u tells the way of the generation of the vertex u. Thus, instead of using the traditional data structures of graph such as adjacent list and adjacent matrix, we will define a new data structure for vertex and a new data structure for block of each colored and rooted embedding, respectively, such that these data are sufficient to calculate codes of all vertices of the embedding. Note that our goal is to enumerate all canonical embeddings with at most n vertices. Also we will define additional data which include the information of canonical embeddings such as states of blocks and copy-states of the whole embeddings.

Specifically, given a colored and rooted outerplanar embedding G, the *index* $\mathrm{idx}(u)$ of a vertex v is defined to be a positive integer i such that the code of u appears as the ith code in $\sigma(G)$. From now on, an integer i for a vertex u with $\mathrm{idx}(u) = i$ will be treated as the vertex u for simplicity. Given a vertex $u \in V(G) - \{r_G\}$, let B be a block, ε be the element (i.e., vertex or edge) of $V(B) \cup E(B)$, and let $\gamma = (d_1, \mathtt{at}, d_2, \mathtt{op}, c)$ be the code which is applied to ε to generate the vertex u, where d_1 is the depth of B, $\mathtt{at}$ depends on the position of ε in B (i.e., left, right or bottom), d_2 is the depth of ε or the depth of one end-vertex of the edge ε, $\mathtt{op}$ is the operation used to generate the vertex u, and c is the color of the vertex u.

To calculate the code γ of such a vertex u, we claim that the following data for vertices and blocks are necessary: the depth of a block, the data showing the position of vertex in its parent-block (i.e., left, right or bottom), the depth of a vertex, the operation used to generate a vertex, and the color of a vertex. Note that the applicable element ε can be an edge. Since we plan to maintain data for vertices and blocks only without maintaining data for edges, we expect that we can compute the type of each applicable edge e (i.e., head-/axial-/bottom-edge) in its parent-block B, the depth $d(e)$, and the position of the edge e in B (i.e., left or right) by the defined data of endvertices of e and the data of its parent-block.

For this purpose, we define the orientation for edge elements of each block as follows. Each left (resp., right) edge incident to vertices in $V^{\mathtt{L}}(B) \cup \{bv(B)\}$ (resp., $V^{\mathtt{R}}(B) \cup \{bv(B)\}$) is defined to be directed from its endvertex with smaller depth to the other endvertex.

Given a wing-vertex u, let $e' = (u', u)$ and $e'' = (u, u'')$ be two edges introduced to edge $e = (u', u'')$ directed from u' to u'' when u is newly created. We have $d(u) > d(u')$ and $d(u) > d(u'')$, since $\gamma(u)$ appears later than both $\gamma(u')$ and $\gamma(u'')$ in the signature $\sigma(G)$. Then we define the edge $e' = (u', u)$ to be directed from u' to u, and define the edge $e'' = (u, u'')$ to be directed from u to u''. See Figure 7.1 for illustrations.

As mentioned before, the above data are not sufficient for canonical embeddings. By definition, an embedding is canonical if and only if it is left-sibling-heavy and left-side-heavy at all blocks of this embedding. Besides, canonical embeddings are classified based on their copy-states. Thus, for each canonical embedding, we will additionally maintain the sibling-state and side-state of all blocks, its copy-state and its dominating block.

Now we present a data structure of each vertex u and a data structure of each block B for a canonical embedding G as follows, respectively (in the following definitions of data, we let the data to be $\emptyset$ if they do not exist).

Data for each vertex u

$$\mathrm{data}(u) = \Big(\mathrm{idx}(u), d(u), \mathrm{pblock}(u), \mathtt{op}(u), c(u), \mathrm{type}(u), \mathrm{blocks}(u), \mathrm{pre}(u), \mathrm{wgedge}(u), \mathrm{cstate}(u)\Big)$$

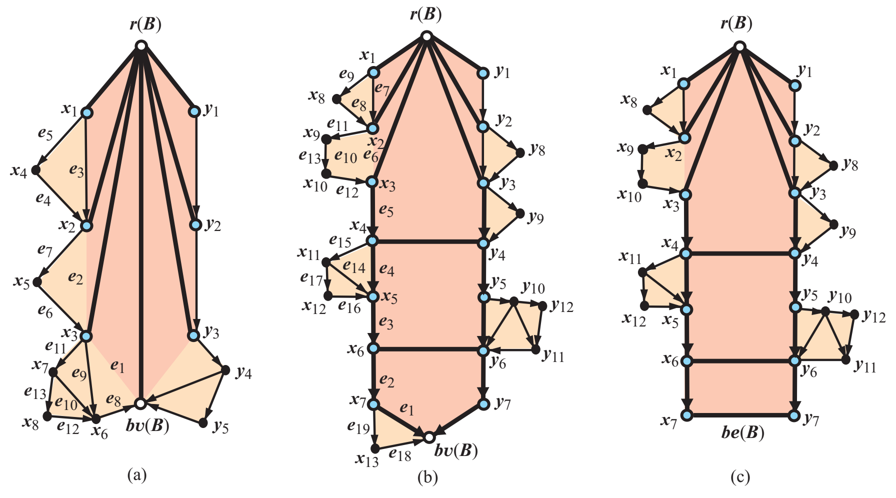

Figure 7.1: The orientation of left edges and right edges of a rooted block B.

- $\mathrm{idx}(u) = i$: a positive integer i (≥ 1) is the index of the vertex u in G such that $\gamma(u)$ appears as the ith code in $\sigma(G)$.
- $d(u)$: the depth of the vertex u.
- $\mathrm{pblock}(u)$: the parent-block B of the vertex u, i.e., $u \in V'(B)$.
- $\mathtt{op}(u)$: the operation used to generate the vertex u.
- $c(u)$: the color of the vertex u.
- $\mathrm{type}(u) = [t^{\mathtt{L}}_{\mathtt{head}}, t^{\mathtt{R}}_{\mathtt{head}}, t^{\mathtt{L}}_{\mathtt{axis}}, t^{\mathtt{R}}_{\mathtt{axis}}, t^{\mathtt{L}}_{\mathtt{wing}}, t^{\mathtt{R}}_{\mathtt{wing}}, t^{\mathtt{L}}_{\mathtt{cut}}, t^{\mathtt{R}}_{\mathtt{cut}}]$: an array with eight entries in $\mathbb{Z}^+_0$ showing the vertex type of u in its parent-block B (i.e., whether u is a left/right head-/axial-/wing-/cut-vertex or the bottom vertex of B) and showing the index of u among the same type of vertices in B. Specifically, u is the $t^{\mathtt{L}}_{\mathtt{head}}$-th left head-vertex of B if $t^{\mathtt{L}}_{\mathtt{head}} \geq 1$, and u is not left head-vertex of B if $t^{\mathtt{L}}_{\mathtt{head}} = 0$. The meanings of $t^{\mathtt{R}}_{\mathtt{head}} \geq 0$, $t^{\mathtt{L}}_{\mathtt{axis}} \geq 0$, $t^{\mathtt{R}}_{\mathtt{axis}} \geq 0$, $t^{\mathtt{L}}_{\mathtt{wing}} \geq 0$, $t^{\mathtt{R}}_{\mathtt{wing}} \geq 0$, $t^{\mathtt{L}}_{\mathtt{cut}} \geq 0$ and $t^{\mathtt{R}}_{\mathtt{cut}} \geq 0$ are similar.

  The data $\mathrm{type}(u)$ can be used to compute the position of each vertex u in its parent-block B (i.e., left, right or bottom) in $O(1)$ time, and then to compute the attachment-label in the code of a vertex in $O(1)$ time. Specifically, u is a left vertex of B if and only if the first, third or fifth entry of $\mathrm{type}(u)$ is larger than 0; and u is a right vertex of B if and only if the second, forth or sixth entry of $\mathrm{type}(u)$ is larger than 0. Since the bottom vertex $bv(B)$ is neither a left vertex nor a right vertex of B, $u = bv(B) \neq \emptyset$ if and only if $t^{\mathtt{L}}_{\mathtt{head}} = t^{\mathtt{R}}_{\mathtt{head}} = t^{\mathtt{L}}_{\mathtt{axis}} = t^{\mathtt{R}}_{\mathtt{axis}} = t^{\mathtt{L}}_{\mathtt{wing}} = t^{\mathtt{R}}_{\mathtt{wing}} = t^{\mathtt{L}}_{\mathtt{cut}} = t^{\mathtt{R}}_{\mathtt{cut}} = 0$.

  Besides, we know whether a left (resp., right) vertex $u \in V'(B)$ is a cut-vertex or not in $O(1)$ time by $\mathrm{type}(u)$. That is, a left (resp., right) vertex $u \in V'(B)$ is a cut-vertex if and only if the seventh (resp., eighth) entry of type(u) is larger than 0.

- blocks(u): all blocks rooted at u arranged from left to right, where we set blocks(u) $= \emptyset$ if u is not a cut-vertex. By blocks(u), we can know any ith block rooted at the vertex u in $O(1)$ time.

- pre(u) $= [v_1, v_2]$: If u is a left or right vertex of B, then the first entry v_1 (resp., the second entry v_2) of pre(u) is the vertex in $V'(B)$ with the smallest (resp., largest) index such that v_1 (resp., v_2) points to the vertex u by a directed edge. Note that $v_1 = v_2$ occurs if and only if there is only one vertex pointing to u in B. For example in Figure 7.1 (a), pre(x_6) $= [x_3, x_8]$ for the left vertex x_6 in B, and pre(y_5) $= [y_4, y_4]$ for the right vertex y_5 in B. If u is the bottom vertex of B, then the first entry v_1 (resp., the second entry v_2) of pre(u) is the vertex in $V'(B)$ with the largest index such that v_1 (resp., v_2) is a left (resp., right) vertex of B that is not the last left (resp., right) core-vertex pointing to the vertex u by a directed edge. Note that $v_1 = v_2$ never occurs if u is the bottom vertex, $v_1 \neq \emptyset$ and $v_2 \neq \emptyset$. For example in Figure 7.1 (a), pre($bv(B)$) $= [x_6, y_5]$.

  By definition, the first entry of pre(u) for a wing-vertex u of B is the tail of the directed edge to which is applied the code $\gamma(u)$ to generate the vertex u. For example in Figure 7.1 (a), the vertex x_6 is generated by applying the code $\gamma(x_6)$ to the edge $e = (x_3, bv(B))$ directed from x_3 to the bottom vertex $bv(B)$, where x_3 is the first entry in pre(x_6) $= [x_3, x_8]$.

  Besides, we can calculate the previous edge of a given edge $e' = (v, w)$ directed from v to w in $\mathcal{E}(B)$ by the second entry of pre(v). Specifically, let $e = (u, v)$ directed from u to v and $e' = (v, w)$ directed from v to w be two consecutive edges in $\mathcal{E}(\mathcal{B})$. If we know the edge $e' = (v, w)$, then we can calculate the tail u of the edge $e = (u, v)$ by pre(v). For example in Figure 7.1 (a), suppose that the edges $e = (x_8, x_6)$ and $e' = (x_6, bv(B))$ are two consecutive edges in $\mathcal{E}(B)$. Given the edge $e' = (x_6, bv(B))$, the tail x_8 of its previous edge $e = (x_8, x_6)$ is the second entry of pre(x_6) $= [x_3, x_8]$.

- wgedge(u) $= u''$: u'' is the head of directed edge e to which can be applied a code to generate a wing-vertex u of B, where we set wgedge(u) $= \emptyset$ if u is not a wing-vertex of B.

  For example in Figure 7.1 (a), vertex x_6 is generated by applying the code $\gamma(x_6)$ to the edge $e = (x_3, bv(B))$ directed from v_3 to $bv(B)$. We can see that wgedge(x_6) $= bv(B)$ is the tail of the edge $e = (x_3, bv(B))$.

  Note that for a wing-vertex u of B, two data wgedge(u) and pre(u) can calculate the edge $e = (u', u'')$ to which is applied the code $\gamma(u)$ to generate the vertex u, where u' is the first entry in pre(u), and u'' = wgedge(u).

- cstate$(u) = [cs(G'), B^d]$: the copy-state $cs(G')$ and the dominating block B^d (if any) of the canonical embedding G' constructed when u is introduced (hence $|V(G')| = \mathrm{idx}(u)$).

Data for each block B

$$\mathrm{data}(B) = \Big(d(B), r(B), \ell v(B), V^{\mathtt{L}}_{\mathtt{head}}(B), V^{\mathtt{R}}_{\mathtt{head}}(B), V^{\mathtt{L}}_{\mathtt{axis}}(B), V^{\mathtt{R}}_{\mathtt{axis}}(B), V^{\mathtt{L}}_{\mathtt{wing}}(B), V^{\mathtt{R}}_{\mathtt{wing}}(B), V^{\mathtt{L}}_{\mathtt{cut}}(B), V^{\mathtt{R}}_{\mathtt{cut}}(B), \mathtt{sbl}(B), \mathtt{sd}(B)\Big)$$

- $d(B)$: the depth of the block B.
- $r(B)$: the root of the block B.
- $\ell v(B)$: the first head vertex of the block B.
- $V^{\mathtt{L}}_{\mathtt{head}}(B)$: an array storing all left head-vertices in G arranged in the increasing vertex indices. Then the ith vertex in $V^{\mathtt{L}}_{\mathtt{head}}(B)$ and the size $|V^{\mathtt{L}}_{\mathtt{head}}(B)|$ can be accessed in $O(1)$ time. Similarly for other arrays $V^{\mathtt{R}}_{\mathtt{head}}(B)$, $V^{\mathtt{L}}_{\mathtt{axis}}(B)$, $V^{\mathtt{R}}_{\mathtt{axis}}(B)$, $V^{\mathtt{L}}_{\mathtt{wing}}(B)$, $V^{\mathtt{R}}_{\mathtt{wing}}(B)$, $V^{\mathtt{L}}_{\mathtt{cut}}(B)$ and $V^{\mathtt{R}}_{\mathtt{cut}}(B)$.
- $\mathtt{sbl}(B) = u$: the witness vertex u of $\mathtt{sbl}(B; G)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$ and there are at least two blocks rooted at $r(B)$, where we set $\mathtt{sbl}(B) = \mathtt{sbl}(B; G)$ otherwise.
- $\mathtt{sd}(B) = u$: the witness vertex u of $\mathtt{sd}(B; G)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$, where we set $\mathtt{sd}(B) = \mathtt{sd}(B; G)$ otherwise (i.e., $\mathtt{sd}(B; G) \in \{\mathtt{nil}, \mathtt{pfx}, \mathtt{eqv}\}$).

We have completed to present the data structure for each block in a canonical embedding.

Given a canonical embedding G with $N \in [1, n]$ vertices, there are at most $N - 1$ blocks. By definition, there are ten entries in data(u) of each vertex u of G and there are thirteen entries in data(B) for each block B of G. Then we can easily derive the following result about the computer memory required for maintaining all data of each canonical embedding.

Lemma 7.1. *Given an integer $n \geq 1$, each canonical embedding with $N \in [1, n]$ vertices requires $O(n)$ space for maintaining the data structures for its vertices and for its blocks.* □

7.3 Realization of Routine Append

Now we are ready to explain how to implement Routines Append. Given an embedding G with N ($\in [1, n-1]$) vertices, a block B in $\mathcal{E}^*(G)$, an element $\varepsilon \in \mathcal{E}(B)$ and a code $\gamma \in \Gamma(\varepsilon)$, Routine Append$(G, B, \varepsilon, \gamma)$ computes a child-embedding $G' = G + \gamma$ of G. That is, it updates or creates the data of vertices and blocks in G' that are different from that of G due to a newly introduced vertex u_{N+1} with $\gamma(u_{N+1}) = \gamma$. We present the routine as follows.

Routine Append$(G, B, \varepsilon, \gamma)$

```
/* Let $u_N$ be the vertex of $G$ with idx$(u_N) = N$, and $u_{N+1}$ be a vertex that will be newly
  introduced by applying $\gamma$ to the element $\varepsilon \in \mathcal{E}(B)$. Let $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$. */
A1 begin
A2   if $\varepsilon \in V'(B)$ then
A3     Create data$(B')$ for a new edge-block $B'$ with $V(B') = \{\varepsilon, u_{N+1}\}$;
A4     Update data$(\varepsilon)$;
A5   endif
A6   Update the data of the head of the directed edge $\varepsilon$ if $\texttt{at} \in \{\texttt{w}^{\texttt{L}}, \texttt{w}^{\texttt{R}}\}$;
A7   Update data$(u_N)$ if $|V'(B)| \geq 3$ is odd in $G$, and $\gamma$ satisfies $\texttt{at} = *$ and $\texttt{op} \in \{\texttt{star}, \texttt{subdivide}\}$;
A8   Update data$(B)$;
A9   Compute $cs(G')$ according to Lemmas 12 and 16;
A10  if $cs(G') = \texttt{pfx}$ then
A11    Identify the dominating block $B^d$ of $G'$
A12  endif
A13  Create data$(u_{N+1})$
A14 end.
```

By the above routine, we generate the child-embedding G' from G by appending the new vertex u_{N+1} to a vertex $\varepsilon \in \mathcal{E}(B)$ with the operation `new-block`, or by appending the new vertex u_{N+1} to an edge $\varepsilon \in \mathcal{E}(B)$ with one of operations $\{\texttt{star}, \texttt{subdivide}, \texttt{triangle}\}$. In the following, we will explain how to update or create data structure of vertices and blocks of G' in the above routine line by line. Let $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$ be the last input in the above Routine Append.

- In Line A3: we create data(B') for the new block B' generated by appending u_{N+1} to the vertex ε. Specifically, $d(B') := d(B) + 1$, $r(B') := \varepsilon$, $\ell v(B') := N + 1$, $V^{\texttt{L}}_{\texttt{head}}(B') := [N + 1]$, $V^{\texttt{R}}_{\texttt{head}}(B') := \emptyset$, $V^{\texttt{L}}_{\texttt{axis}}(B') := \emptyset$, $V^{\texttt{R}}_{\texttt{axis}}(B') := \emptyset$, $V^{\texttt{L}}_{\texttt{wing}}(B') := \emptyset$, $V^{\texttt{L}}_{\texttt{cut}}(B') := \emptyset$, $V^{\texttt{R}}_{\texttt{wing}}(B') := \emptyset$, $V^{\texttt{R}}_{\texttt{cut}}(B') := \emptyset$, and $\texttt{sd}(B') := \texttt{nil}$. It remains to create $\texttt{sbl}(B')$. Let B^* be the rightmost block rooted at the vertex ε if ε is a cut-vertex. By Lemma 6.5(iv), we have

$$\texttt{sbl}(B') := \begin{cases} \texttt{eqv} & \text{if } \varepsilon \text{ is a cut-vertex, } c(u_{N+1}) = c(\ell v(B^*)) \text{ and } \ell v(B') - \ell v(B^*) = 1; \\ \texttt{pfx} & \text{if } \varepsilon \text{ is a cut-vertex, } c(u_{N+1}) = c(\ell v(B^*)) \text{ and } \ell v(B') - \ell v(B^*) \geq 2; \\ u_{N+1} & \text{if } \varepsilon \text{ is a cut-vertex and } c(u_{N+1}) < c(\ell v(B^*)); \\ \texttt{stc} & \text{if } \varepsilon \text{ is not a cut-vertex.} \end{cases}$$

Based on the above analysis, data(B') can be updated in $O(1)$ time.

- In Line A4: we update data(ε) for the vertex ε. Recall that ε is a cut-vertex in G if and only if blocks$(\varepsilon) = \emptyset$. Note that ε is a cut-vertex in G' with a newly additional block

B' rooted at it. Then at most two data —type(ε) and blocks(ε) among data(ε) are changed from G to G'. Recall that ε is the bottom vertex of B if and only if all entries of type(ε) are equal to 0. Thus, we only need to update blocks(ε) in G' and need to update type(ε) in G' if ε is a left or right vertex of B that is not a cut-vertex in G as well. Specifically, we update blocks(ε) by adding B' to its end.

It remains to update type(ε) in G'. If ε is a left vertex of B that is not a cut-vertex in G, then ε is the ($|V^{\mathtt{L}}_{\mathtt{cut}}|+1$)st left cut-vertex of B in G', i.e., the seventh entry of type(ε) is set to be $|V^{\mathtt{L}}_{\mathtt{cut}}(B)|+1$, and other entries of type(ε) are unchanged; if ε is a right vertex of B that is not a cut-vertex in G, then ε is the ($|V^{\mathtt{R}}_{\mathtt{cut}}|+1$)st right cut-vertex of B in G', i.e., the last entry of type(ε) is set to be $|V^{\mathtt{R}}_{\mathtt{cut}}(B)|+1$, and other entries of type(ε) are unchanged.

Based on the above analysis, we can update data(ε) for the vertex ε in $O(1)$ time.

- In Line A6: we update the data of the head of the edge ε if $\mathtt{at} \in \{\mathtt{w}^{\mathtt{L}}, \mathtt{w}^{\mathtt{R}}\}$. By definition of code of vertex, u_{N+1} is a new left (resp., right) wing-vertex of B in G' if $\mathtt{at} = \mathtt{w}^{\mathtt{L}}$ (resp., $\mathtt{at} = \mathtt{w}^{\mathtt{R}}$). Before updating the data of the head of ε, we need to determine the orientation of the edge ε. By Lemma 8.3, we can know the head and the tail of the edge ε. Let $\varepsilon = (u', u'')$ be the directed edge with the tail u' and the head u'', and let pre(u'') $= [u, v]$ in G (possibly $u = \emptyset$ and $v = \emptyset$). Then we update pre(u'') for the head u'' of the edge ε in G' as follows: pre(u'') $:= [u_{N+1}, v]$ if u_{N+1} is a left vertex of B in G', and pre(u'') $:= [u, u_{N+1}]$ in G' if u_{N+1} is a right vertex of B in G'. Clearly, we can update the data of the head of the edge ε in $O(1)$ time.

- In Line A7: we update data(u_N) if $|V'(B)| \geq 3$ is odd in G and γ satisfies $\mathtt{at} = *$ and $\mathtt{op} \in \{\mathtt{star}, \mathtt{subdivide}\}$. In this case, ε is the edge $e^b = (u_{N-1}, u_N)$ in G, where idx(u_{N-1}) $= N-1$ and idx(u_N) $= N$, and the bottom vertex u_N of B in G turns to be the last left core-vertex of B in G'. Note that only two data—type(u_N) and pre(u_N) among data(u_N) are changed from G to G'. Then we will update type(u_N) and pre(u_N) in G' as follows.

By definition, $u_N = bv(B) \neq \emptyset$ in G if and only if all entries of type(u_N) are equal to zero. If $|V'(B)| \geq 3$ is odd in G and $\mathtt{op} = \mathtt{star}$, then

$$\text{type}(u_N) := [|V^{\mathtt{L}}_{\mathtt{head}}(B)|+1, 0, 0, 0, 0, 0, 0, 0],$$

i.e., the bottom vertex u_N in G turns to be a left head-vertex of B in G'.

If $|V'(B)| \geq 3$ is odd in G, $\mathtt{at} = *$ and $\mathtt{op} = \mathtt{star}$, then

$$\text{type}(u_N) := [0, 0, |V^{\mathtt{L}}_{\mathtt{axis}}(B)|+1, 0, 0, 0, 0, 0],$$

i.e., the bottom vertex u_N in G turns to be a left axial-vertex of B in G'.

Besides, we update pre(u_N) in G' as follows. Recall that $u_N = bv(B)$ in G, and there are only two vertices u_{N-2} and u_{N-1} pointing to u_N by a directed edge in G, respectively, where

u_{N-2} and u_{N-1} are the last left and right core-vertex of B in G, respectively. By definition, we have $\text{pre}(u_N) = [\emptyset, \emptyset]$ in G. After u_N becomes the last left core-vertex of B in G', u_{N-2} is the only one vertex pointed to u_N by a directed edge in G'. Hence $\text{pre}(u_N) := [u_{N-2}, u_{N-2}]$ in G'.

Based on the above analysis, $\text{data}(u_N)$ can be updated in $O(1)$ time when $|V'(B)| \geq 3$ is odd in G and γ satisfies $\texttt{at} = *$ and $\texttt{op} \in \{\texttt{star}, \texttt{subdivide}\}$.

• In Line A8: we update $\text{data}(B)$ in G'. Since the first three data—the depth $d(B)$, the root $r(B)$ and the first head-vertex $\ell v(B)$ of B among $\text{data}(B)$ are not changed from G to G', we only need to update the rest: $V^{\texttt{L}}_{\texttt{head}}(B)$, $V^{\texttt{R}}_{\texttt{head}}(B)$, $V^{\texttt{L}}_{\texttt{axis}}(B)$, $V^{\texttt{R}}_{\texttt{axis}}(B)$, $V^{\texttt{L}}_{\texttt{wing}}(B)$, $V^{\texttt{R}}_{\texttt{wing}}(B)$, $V^{\texttt{L}}_{\texttt{cut}}(B)$, $V^{\texttt{R}}_{\texttt{cut}}(B)$, $\texttt{sbl}(B)$ and $\texttt{sd}(B)$, respectively. Recall that $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$.

Now we explain how to update the arrays of left/right head-/axial-/wing-/cut-vertex of B for the case that ε is a vertex in $\mathcal{E}(B)$ and for the case that ε is an edge in $\mathcal{E}(B)$, respectively.

- If ε is a left (resp., right) vertex of B that is not a cut-vertex in G, then ε is a cut-vertex in G', and we add $\text{idx}(\varepsilon)$ to the end of $V^{\texttt{L}}_{\texttt{cut}}(B)$ (resp., $V^{\texttt{R}}_{\texttt{cut}}(B)$).

- If ε is an edge in $\mathcal{E}(B)$ that is not $\{e^b\}$, and γ satisfies $\texttt{at} = \texttt{w}^{\texttt{L}}$ (resp., $\texttt{at} = \texttt{w}^{\texttt{R}}$), then we add u_{N+1} to the end of $V^{\texttt{L}}_{\texttt{wing}}(B)$ (resp., $V^{\texttt{R}}_{\texttt{wing}}(B)$).

- If ε is the edge e^b and $|V'(B)| = 1$, then we add u_{N+1} to the end of $V^{\texttt{R}}_{\texttt{head}}(B)$.

- If ε is the edge e^b and $|V'(B)| \geq 2$ in G, then $V^{\texttt{L}}_{\texttt{head}}(B)$, $V^{\texttt{L}}_{\texttt{axis}}(B)$, $V^{\texttt{R}}_{\texttt{head}}(B)$ and $V^{\texttt{R}}_{\texttt{axis}}(B)$ are needed to be updated due to the change for the vertex type of u_N of B from G to G' and the newly created vertex u_{N+1} in G'. Recall that if $|V'(B)| \geq 2$ is odd in G, and γ satisfies $\texttt{at} = *$ and $\texttt{op} \in \{\texttt{star}, \texttt{subdivide}\}$, then the bottom vertex u_N in G turns to be the last left core-vertex of B in G' and the new vertex u_{N+1} is the last right core-vertex of B in G'.

  Specifically, if $\varepsilon = e^b$, $|V'(B)| \geq 2$ is odd and γ satisfies $\texttt{op} = \texttt{star}$ in G, then we add u_N to the end of $V^{\texttt{L}}_{\texttt{head}}(B)$ and add u_{N+1} to the end of $V^{\texttt{R}}_{\texttt{head}}(B)$. If $\varepsilon = e^b$, $|V'(B)| \geq 2$ is odd, and γ satisfies $\texttt{at} = *$ and $\texttt{op} = \texttt{subdivide}$ in G, then we add u_N to the end of $V^{\texttt{L}}_{\texttt{axis}}(B)$ and add u_{N+1} to the end of $V^{\texttt{R}}_{\texttt{axis}}(B)$.

It remains to update $\texttt{sbl}(B)$ and $\texttt{sd}(B)$ in G'. We only need to update $\texttt{sbl}(B)$ in G' if $\texttt{sbl}(B) = \texttt{pfx}$ in G and update $\texttt{sd}(B)$ in G' if $\texttt{sd}(B) \in \{\texttt{nil}, \texttt{pfx}\}$ in G. The reason is presented as follow. Let $B^1, B^2, \ldots, B^k$ be the blocks in $\mathcal{E}^*(G)$. Suppose that $B = B^i$ is the ith ($i \in [1, k]$) block in $\mathcal{E}^*(G)$. By definition of $\mathcal{E}^*(G)$, all blocks B^j for $j = 1, 2, \ldots, k$ satisfy that $\texttt{sbl}(B^j; G) \neq \texttt{eqv}$ and $\texttt{sd}(B^j; G) \neq \texttt{eqv}$. Besides, by Lemma 6.5(i), both $\texttt{sbl}(B; G) = \texttt{stc}$ and $\texttt{sd}(B; G) = \texttt{stc}$ remain unchanged in G'.

Let B^ℓ ($\ell \in [1,k]$) be the dominating block of G, which is maintained by cstate(u_N), and let ε^* and γ^* be the critical element and the critical code of G when $cs(G) = \texttt{pfx}$, both of which can be calculated by Lemma 8.8 in $O(1)$ time.

We will first update $\texttt{sbl}(B)$ in G' in the case of $\texttt{sbl}(B) = \texttt{pfx}$ in G based on the copy-state of G and the criticality of G' as follows. Clearly, each block B^j with smaller superscribe as B (i.e., $1 \leq j < i$) satisfies $\texttt{sbl}(B^j; G) \in \{\texttt{stc}, \texttt{pfx}\}$ and $\texttt{sd}(B^j; G) \in \{\texttt{stc}, \texttt{nil}, \texttt{pfx}\}$. Thus, the copy-state of G is $cs(G) = \texttt{pfx}$, and the dominating block B^ℓ of G satisfies $1 \leq \ell \leq i$.

If G' is the critical child-embedding of G, i.e., $\varepsilon = \varepsilon^*$ and $\gamma = \gamma^*$, then by Lemma 6.9 (1), $\texttt{sbl}(B; G) = \texttt{pfx}$ changes into one of $\{\texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ in G'. In the following, we will give the exact conditions that $\texttt{sbl}(B) = \texttt{pfx}$ in G changes into $\texttt{stc}$, $\texttt{pfx}$ and $\texttt{eqv}$ in G', respectively, after we present new necessary definitions.

We will define element $\hat{\varepsilon}$ and code $\hat{\gamma}$ with respect to the block B in G in a similar way to define the critical element ε^* and the critical code γ^* with respect to the dominating block B^ℓ in G. Let $v' = \mu^-(u_N; r(B))$, which can be computed by Equation 8.1 in Lemma 8.6. Let $\hat{v}$ be the vertex such that $\mathrm{idx}(\hat{v}) = \mathrm{idx}(v') + 1$. Let $\hat{\gamma} = \gamma(\hat{v})$ and $\hat{\varepsilon}$ be the element to which is applied $\hat{\gamma}$ to generate $\hat{v}$. By Lemma 8.5, we can calculate $\hat{\varepsilon}$ and $\hat{\gamma}$ for the vertex $\hat{v}$ in $O(1)$ time. Now we are ready to show the update of $\texttt{sbl}(B)$ in G' as follows:

$$\texttt{sbl}(B) := \begin{cases} \texttt{stc} & \text{if (a) } |\mathrm{blocks}(r(B))| = 1, \text{ and (b) } \varepsilon \neq \hat{\varepsilon}, \text{ or, } \varepsilon = \hat{\varepsilon} \text{ and } \gamma < \hat{\gamma}; \\ u_{N+1} & \text{if (a) } |\mathrm{blocks}(r(B))| \geq 2, \text{ and (b) } \varepsilon \neq \hat{\varepsilon}, \text{ or, } \varepsilon = \hat{\varepsilon} \text{ and } \gamma < \hat{\gamma}; \\ \texttt{eqv} & \text{if } \varepsilon = \hat{\varepsilon},\ \gamma = \hat{\gamma} \text{ and } \mathrm{idx}(\mu^-(u_{N+1}; r(B))) = \mathrm{idx}(\ell v(B)) - 1. \end{cases}$$

If G' is not critical, i.e., $\varepsilon \neq \varepsilon^*$, or, $\varepsilon = \varepsilon^*$ and $\gamma = \gamma^*$, then by Lemma 6.9 (2), it holds $\texttt{sbl}(B; G') = \texttt{stc}$. Hence we update $\texttt{sbl}(B) := \texttt{stc}$ for $|\mathrm{blocks}(r(B))| = 1$, and $\texttt{sbl}(B) := N + 1$ for $|\mathrm{blocks}(r(B))| \geq 1$.

Next we will update $\texttt{sd}(B)$ in G' when $\texttt{sd}(B) \in \{\texttt{pfx}, \texttt{nil}\}$ as follows. If $\texttt{sd}(B) = \texttt{pfx}$ in G, then similarly we can see that $cs(G) = \texttt{pfx}$. Let $u^{\mathtt{R}}$ be the first vertex in $V^{\mathtt{R}}_{\mathtt{wing}}(B)$ if $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$, or $u^{\mathtt{R}} = \ell v(B_1)$ for the leftmost block of $\mathcal{B}(u')$ of the first vertex u' in $V^{\mathtt{R}}_{\mathtt{cut}}(B)$ if $V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$. By Lemma 6.9 (1), $\texttt{sd}(B; G) = \texttt{pfx}$ changes into one of $\{\texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ in G'. Specifically, $\texttt{sd}(B)$ in G' is updated as follows:

$$\texttt{sd}(B) := \begin{cases} u_{N+1} & \text{if } \varepsilon \neq \hat{\varepsilon}, \text{ or, } \varepsilon = \hat{\varepsilon} \text{ and } \gamma < \hat{\gamma}; \\ \texttt{eqv} & \text{if } \varepsilon = \hat{\varepsilon},\ \gamma = \hat{\gamma} \text{ and } \mathrm{idx}(\mu(u_{N+1}; r(B))) = \mathrm{idx}(u^{\mathtt{R}}) - 1. \end{cases}$$

If $\texttt{sd}(B) = \texttt{nil}$ in G, then by Lemma 6.5 (iii), $\texttt{sd}(B; G')$ belongs to $\{\texttt{nil}, \texttt{pfx}, \texttt{eqv}, \texttt{stc}\}$ according to formulas (6.3), (6.5), (6.7), (6.8)-(6.15). By definition, we update $\texttt{sd}(B) := u_{N+1}$ if $\texttt{sd}(B; G') = \texttt{stc}$, and $\texttt{sd}(B) := \texttt{sd}(B; G')$ otherwise.

Based on the above analysis, we can see that data(B) can be updated in $O(1)$ time.

- In Lines A9 and A11, we update the copy-state $cs(G')$ of G' and calculate the dominating block B^d of G', respectively. In the following, we will explain the update of $cs(G')$ and the calculation of B^d based on $cs(G)$, the element type of ε and the criticality of G'. Let B^ℓ be the dominating block, ε^* be the critical element and γ^* be the critical code of G when $cs(G) = \texttt{pfx}$. Let $u^{\texttt{R}}$ be the first vertex in $V^{\texttt{R}}_{\texttt{wing}}(B)$ if $V^{\texttt{R}}_{\texttt{wing}}(B) \neq \emptyset$, or $u^{\texttt{R}} = \ell v(B_1)$ for the leftmost block of $\mathcal{B}(u')$ of the first vertex u' in $V^{\texttt{R}}_{\texttt{cut}}(B)$ if $V^{\texttt{R}}_{\texttt{wing}}(B) = \emptyset$ and $V^{\texttt{R}}_{\texttt{cut}}(B) \neq \emptyset$.

If $cs(G) \in \{\texttt{stc}, \texttt{eqv}\}$ and ε is an edge in $E(B)$, then by Lemma 6.6 (ii), we have

$$cs(G') := \begin{cases} \texttt{stc} & \text{if } \texttt{sd}(B) \text{ is } \texttt{nil} \text{ or equal to } N+1; \\ \texttt{sd}(B) & \text{if } \texttt{sd}(B) \in \{\texttt{pfx}, \texttt{eqv}\}, \end{cases}$$

and the dominating block B^d of G' is

$$B^d := \begin{cases} B & \text{if } cs(G') = \texttt{pfx}; \\ \emptyset & \text{if } cs(G') \in \{\texttt{stc}, \texttt{eqv}\}. \end{cases}$$

If $cs(G) \in \{\texttt{stc}, \texttt{eqv}\}$ and ε is a vertex in $V'(B)$, then by Lemma 6.6 (iii), we have

$$cs(G') := \begin{cases} \texttt{sd}(B) & \text{if } \texttt{sd}(B) \in \{\texttt{pfx}, \texttt{eqv}\}; \\ \texttt{sbl}(B') & \text{if } \texttt{sd}(B) \text{ is } \texttt{nil} \text{ or equal to } N+1, \text{ and } \texttt{sbl}(B') \in \{\texttt{pfx}, \texttt{eqv}\}; \\ \texttt{stc} & \text{both } \texttt{sd}(B) \text{ and } \texttt{sbl}(B') \text{ are } \texttt{nil} \text{ or are equal to } N+1, \end{cases}$$

and the dominating block B^d of G' is

$$B^d := \begin{cases} B & \text{if } cs(G') = \texttt{sd}(B) = \texttt{pfx}; \\ B' & \text{if } cs(G') = \texttt{sbl}(B') = \texttt{pfx}; \\ \emptyset & \text{if } cs(G') \in \{\texttt{stc}, \texttt{eqv}\}. \end{cases}$$

If $cs(G) = \texttt{pfx}$, and G' is critical (i.e., $\varepsilon = \varepsilon^*$ and $\gamma = \gamma^*$), then by Lemma 6.10 (iii), we have

$$cs(G') := \begin{cases} \texttt{eqv} & \text{if } cs(G) = \texttt{sbl}(B^\ell; G) \text{ and } \mathrm{idx}(\mu^-(u_{N+1}; r(B^\ell))) = \mathrm{idx}(\ell v(B^\ell)) - 1; \\ & \text{or if } cs(G) = \texttt{sd}(B^\ell; G) \text{ and } \mathrm{idx}(\mu(u_{N+1}; r(B^\ell))) = \mathrm{idx}(u^{\texttt{R}})) - 1; \\ \texttt{pfx} & \text{if } cs(G) = \texttt{sbl}(B^\ell; G) \text{ and } \mathrm{idx}(\mu^-(u_{N+1}; r(B^\ell))) \leq \mathrm{idx}(\ell v(B^\ell)) - 2; \\ & \text{or if } cs(G) = \texttt{sd}(B^\ell; G) \text{ and } \mathrm{idx}(\mu(u_{N+1}; r(B^\ell))) \leq \mathrm{idx}(u^{\texttt{R}})) - 2, \end{cases}$$

and the dominating block B^d of G' is

$$B^d := \begin{cases} B^\ell & \text{if } cs(G') = \texttt{pfx}; \\ \emptyset & \text{if } cs(G') \in \{\texttt{stc}, \texttt{eqv}\}. \end{cases}$$

If $cs(G) = \texttt{pfx}$, G' is not critical and $\varepsilon \in E(B)$, then by Lemma 6.10 (iv), we have

$$cs(G') := \begin{cases} \texttt{stc} & \text{if } \texttt{sd}(B) \text{ is } \texttt{nil} \text{ or equal to } N+1; \\ \texttt{sd}(B) & \text{if } \texttt{sd}(B) \in \{\texttt{pfx}, \texttt{eqv}\}, \end{cases}$$

and the dominating block B^d of G' is

$$B^d := \begin{cases} B & \text{if } cs(G') = \texttt{pfx}; \\ \emptyset & \text{if } cs(G') \in \{\texttt{stc}, \texttt{eqv}\}. \end{cases}$$

If $cs(G) = \texttt{pfx}$, G' is not critical and $\varepsilon \in V'(B)$, then by Lemma 6.10 (v), we have

$$cs(G') := \begin{cases} \texttt{sd}(B) & \text{if } \texttt{sd}(B) \in \{\texttt{pfx}, \texttt{eqv}\}; \\ \texttt{sbl}(B') & \text{if } \texttt{sd}(B) \in \{\texttt{nil}, \texttt{stc}\} \text{ and } \texttt{sbl}(B') \in \{\texttt{pfx}, \texttt{eqv}\}; \\ \texttt{stc} & \text{if both } \texttt{sd}(B) \text{ and } \texttt{sbl}(B') \text{ are } \texttt{nil} \text{ or are equal to } u_{N+1}, \end{cases}$$

and the dominating block B^d of G' is

$$B^d := \begin{cases} B & \text{if } cs(G') = \texttt{sd}(B) = \texttt{pfx}; \\ B' & \text{if } cs(G') = \texttt{sbl}(B') = \texttt{pfx}; \\ \emptyset & \text{if } cs(G') \in \{\texttt{stc}, \texttt{eqv}\}. \end{cases}$$

Based on the above analysis, we can update the copy-state of G' and calculate its dominating block in $O(1)$ time.

- In Line A13, we create data(u_{N+1}) for the new vertex u_{N+1}. Recall that B' is the new edge-block if ε is a vertex, and $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$.

We first create the following seven data for u_{N+1}: idx(u_{N+1}) := $N+1$; $d(u_{N+1}) := d(\varepsilon)+1$ if ε is a vertex, and $d(u_{N+1}) := d(u_N)+1$ if ε is an edge; pblock(u_{N+1}) := B' if ε is a vertex, and pblock(u_{N+1}) := B if ε is an edge; blocks(u_{N+1}) := $\emptyset$; $\texttt{op}(u_{N+1}) := \texttt{op}$; $c(u_{N+1}) := c$, and cstate(u_{N+1}) := $[cs(G'), B^d]$.

Next we create type(u_{N+1}) as follows. If ε is a vertex, then u_{N+1} is the first left head-vertex of the new block B', and hence

$$\text{type}(u_{N+1}) := [1, 0, 0, 0, 0, 0, 0, 0].$$

Otherwise, ε is an edge of B in G. If $|V'(B)| = 2$ in G', then u_{N+1} is the first right head-vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0, 1, 0, 0, 0, 0, 0, 0].$$

If $|V'(B)| \geq 3$ is even in G' and $\texttt{op} = \texttt{star}$, then u_{N+1} is a right head-vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0, |V^{\texttt{R}}_{\texttt{head}}(B)| + 1, 0, 0, 0, 0, 0, 0].$$

If $|V'(B)| \geq 3$ is even in G', $\texttt{at} = *$ and $\texttt{op} = \texttt{subdivide}$, then u_{N+1} is a right axial-vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0, 0, 0, |V^{\texttt{R}}_{\texttt{axis}}(B)| + 1, 0, 0, 0, 0].$$

If $|V'(B)| \geq 3$ is odd in G', $\texttt{at} = *$ and $\texttt{op} \in \{\texttt{star}, \texttt{subdivide}\}$, then u_{N+1} is the bottom vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0,0,0,0,0,0,0,0].$$

If $\texttt{at} = \texttt{w}^{\texttt{L}}$, then u_{N+1} is a left wing-vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0,0,0,0,|V^{\texttt{L}}_{\texttt{wing}}(B)|+1,0,0,0].$$

If $\texttt{at} = \texttt{w}^{\texttt{R}}$, then u_{N+1} is a right wing-vertex of B in G', and hence

$$\text{type}(u_{N+1}) := [0,0,0,0,0,|V^{\texttt{R}}_{\texttt{wing}}(B)|+1,0,0].$$

It remains to create $\text{pre}(u_{N+1})$ and $\text{wgedge}(u_{N+1})$. Note that $\text{wgedge}(u_{N+1}) \neq \emptyset$ if and only if u_{N+1} is a wing-vertex of B. In the following, we will create $\text{pre}(u_{N+1})$ and $\text{wgedge}(u_{N+1})$ based on the element type of ε, the vertex type of u_{N+1} and $|V'(B)|$. If ε is a vertex, then $\text{pre}(u_{N+1}) := [\emptyset, \emptyset]$ and $\text{wgedge}(u_{N+1}) := \emptyset$. Otherwise, ε is an edge $e = (u', u'') \in E(B)$. Without loss of generality, we assume that $d(u') < d(u'')$. Especially when $\varepsilon = e$ is a left or right edge of B, we can know the orientation of e in $O(1)$ time by Lemma 8.3. Note that the edge e can be directed from u' to u'' or directed from u'' to u'.

If $|V'(B)| = 2$, then u_{N+1} is the first right head-vertex of B, and hence there is no vertex pointed to u_{N+1} by a directed edge. Thus, $\text{pre}(u_{N+1}) := [\emptyset, \emptyset]$ and $\text{wgedge}(u_{N+1}) := \emptyset$.

If $|V'(B)| \geq 3$ is even and u_{N+1} is the last right core-vertex of B, then u' is the last second right core-vertex of B, which is the only vertex pointed to u_{N+1} by a directed edge. Thus, $\text{pre}(u_{N+1}) := [u', u']$ and $\text{wgedge}(u_{N+1}) := \emptyset$.

If u_{N+1} is the bottom vertex of B, then $\text{pre}(u_{N+1}) := [\emptyset, \emptyset]$ and $\text{wgedge}(u_{N+1}) := \emptyset$.

If u_{N+1} is a wing-vertex of B and $e = (u', u'')$ is directed from u' to u'', then $\text{pre}(u_{N+1}) := [u', u']$, and $\text{wgedge}(u_{N+1}) := u''$.

If u_{N+1} is a wing-vertex of B and $e = (u', u'')$ is directed from u'' to u', then $\text{pre}(u_{N+1}) := [u'', u'']$, and $\text{wgedge}(u_{N+1}) := u'$.

Clearly, we can create $\text{data}(u_{N+1})$ in $O(1)$ time.

We have completed the explanation of Routine Append. Based on the above analysis and Lemma 7.1, we can easily derive the running time and the memory for Routine Append with given inputs.

Lemma 7.2. *Given an embedding G with N $(\in [1, n-1])$ vertices, a block B in $\mathcal{E}^*(G)$, an element $\varepsilon \in \mathcal{E}(B)$ and a code $\gamma \in \Gamma(\varepsilon)$, Routine* Append$(G, B, \varepsilon, \gamma)$ *computes a child-embedding $G' = G + \gamma$ of G in $O(1)$ time and $O(n)$ space in the worst case.* □

7.4 Realization of Routine RemoveTip

Given a canonical embedding G' with $N(\geq 1)$ vertices, Routine RemoveTip(G') computes the parent-embedding G obtained from G' by removing the tip $t(G')$ (i.e., by $\texttt{remove}(t(G'))$), which is given as follows:

```
Routine RemoveTip($G'$)
/* Let $N = |V(G')| \geq 1$, and $u_N$ be the vertex of $G'$ with $\mathrm{idx}(u_N) = N$. */
R1 begin
R2   Let $t(G') = u_N$, and $B$ be the block in $G'$ with $u_N \in V'(B)$;
R3   if $|V'(B)| = 1$ then
R4     Let $v = r(B)$ be the root of $B$, and $B'$ be the block in $G'$ with $r(B) \in V'(B')$;
R5     Update data($v$) for $v = r(B)$;
R6     Update data($B'$);
R7     Remove both data($u_N$) and data($B$)
R8   else /* $|V'(B)| \geq 2$ */
R9     Let $e = (u', u'')$ be the edge of $B$ with $\mathrm{idx}(u') \leq \mathrm{idx}(u'')$, to which is applied $\gamma(u_N)$ to
        generate the vertex $u_N$;
R10    Update data($u'$) and data($u''$) for the endvertices $u'$ and $u''$ of the edge $e$ if $|V'(B)| \geq 3$;
R11    Update data($B$);
R12    Remove data($u_N$)
R13  endif
R14 end.
```

Reverse to Routine Append, we update the data for vertices and blocks in G that are changed due to the removal of u_N in Routine RmoveTip. Specifically, if $|V'(B)| = 1$, then the removal of u_N leads to the change of data(v) for $v = r(B)$ and data(B') for the block B' containing v from G' to G. If $|V'(B)| \geq 2$, then by Lemma 8.5, the edge $e = (u', u'')$ in above Line R9 can be calculated in $O(1)$ time, and the removal of u_N leads to the change of data(u') and data(u'') for the endvertices of the edge $e = (u', u'')$ and the change of data(B) from G' to G. In the following, we will explain the update of data in the above Routine RemoveTip line by line.

- In Line R5, we update data(v) in G when B is an edge-block rooted at v in G'. The removal of the vertex u_N can only lead to the change of two data type(v) and blocks(v) among data(v) from G' to its parent-embedding G. Specifically, type(v) is changed from G' to G if there is only one block rooted at v in G' (i.e., $|\mathrm{blocks}(v)| = 1$). In this case, the cut-vertex v in G' is not a cut-vertex in G any more. Let type(v) $= [t^{\mathtt{L}}_{\mathtt{head}}, t^{\mathtt{R}}_{\mathtt{head}}, t^{\mathtt{L}}_{\mathtt{axis}}, t^{\mathtt{R}}_{\mathtt{axis}}, t^{\mathtt{L}}_{\mathtt{wing}}, t^{\mathtt{R}}_{\mathtt{wing}}, t^{\mathtt{L}}_{\mathtt{cut}}, t^{\mathtt{R}}_{\mathtt{cut}}]$ in G. If v is a left (resp., right) cut-vertex such that only one block is rooted at it in G', then v is not left (resp., right) cut-vertex in G, i.e., type(v) $:= [t^{\mathtt{L}}_{\mathtt{head}}, t^{\mathtt{R}}_{\mathtt{head}}, t^{\mathtt{L}}_{\mathtt{axis}}, t^{\mathtt{R}}_{\mathtt{axis}}, t^{\mathtt{L}}_{\mathtt{wing}}, t^{\mathtt{R}}_{\mathtt{wing}}, 0, t^{\mathtt{R}}_{\mathtt{cut}}]$

(resp., type(v) := $[t^{\texttt{L}}_{\texttt{head}}, t^{\texttt{R}}_{\texttt{head}}, t^{\texttt{L}}_{\texttt{axis}}, t^{\texttt{R}}_{\texttt{axis}}, t^{\texttt{L}}_{\texttt{wing}}, t^{\texttt{R}}_{\texttt{wing}}, t^{\texttt{L}}_{\texttt{cut}}, 0]$). Besides, the removal of the tip u_N of G' results in the removal of the edge-block B, i.e., we remove B from blocks(v). Based on the above analysis, data(v) can be updated in $O(1)$ time.

- In Line R6, we update data(B') for the block B' containing the root v of the edge-block B. We can see that the removal of u_N can only result in the change of the four data: $V^{\texttt{L}}_{\texttt{cut}}(B)$, $V^{\texttt{R}}_{\texttt{cut}}(B)$, $\texttt{sbl}(B')$ and $\texttt{sd}(B')$ among data(B') from G' to G. Recall that the root v of the tip-block B is the tip of the block B'. By definition of the tip of a block, v is the last vertex in $V^{\texttt{L}}_{\texttt{cut}}(B')$ (resp., $V^{\texttt{R}}_{\texttt{cut}}(B')$) in G'. Note that if there is only one block rooted at v in G', then the vertex v is not a cut-vertex of B' in G anymore, and hence we remove the last vertex v from $V^{\texttt{L}}_{\texttt{cut}}(B')$ (resp., $V^{\texttt{R}}_{\texttt{cut}}(B')$).

It remains to update $\texttt{sbl}(B')$ and $\texttt{sd}(B')$ in G. We claim that $\texttt{sbl}(B')$ would change from G' to G if $\texttt{sbl}(B') = u_N$ or $\texttt{sbl}(B) = \texttt{eqv}$ in G', and that $\texttt{sbl}(B')$ in G' remains unchanged in G otherwise. We will analyze all cases that $\texttt{sbl}(B')$ in G' remains unchanged in G as follows. If $\texttt{sbl}(B') \in \texttt{stc}$ in G', i.e., there is only one block rooted at $r(B')$ in G', then the removal of u_N will not change the fact that there is only one block rooted at $r(B')$ in G, and hence $\texttt{sbl}(B') \in \texttt{stc}$ in G. If $\texttt{sbl}(B') = u_i$ such that $\mathrm{idx}(r(B')) < \mathrm{idx}(u_i) = i < \mathrm{idx}(u_N) = N$, then $\texttt{sbl}(B'; G') = \texttt{stc}$, and the removal of u_N would not change $\texttt{sbl}(B') = u_i$ in G. If $\texttt{sbl}(B') = \texttt{pfx}$ in G', then $\texttt{sbl}(B') = \texttt{pfx}$ in G. However, if $\texttt{sbl}(B') = \mathrm{idx}(u_N)$ in G' or $\texttt{sbl}(B') = \texttt{eqv}$ in G', then $\texttt{sbl}(B') := \texttt{pfx}$ in G.

Now we update $\texttt{sd}(B')$ in G. Note that $\texttt{sd}(B')$ remains unchanged in G if (a) $\texttt{sd}(B') = u_i$ such that $\mathrm{idx}(r(B')) < \mathrm{idx}(u_i) < \mathrm{idx}(u_N)$ in G', (b) $\texttt{sd}(B') = \texttt{nil}$ in G', or (c) $\texttt{sd}(B') = \texttt{pfx}$ in G' and $V^{\texttt{R}}_{\texttt{wing}}(B') \cup V^{\texttt{R}}_{\texttt{cut}}(B') \neq \emptyset$. However, if $\texttt{sd}(B') = u_N$ or $\texttt{sd}(B') = \texttt{eqv}$, and $V^{\texttt{R}}_{\texttt{wing}}(B') \cup V^{\texttt{R}}_{\texttt{cut}}(B') = \emptyset$ in G', then $\texttt{sd}(B') := \texttt{nil}$ in G. If $\texttt{sd}(B') = u_N$ or $\texttt{sd}(B') = \texttt{eqv}$, and $V^{\texttt{R}}_{\texttt{wing}}(B') \cup V^{\texttt{R}}_{\texttt{cut}}(B') \neq \emptyset$ in G', then $\texttt{sd}(B') := \texttt{pfx}$ in G. If $\texttt{sd}(B') = \texttt{pfx}$ in G' and $V^{\texttt{R}}_{\texttt{wing}}(B') \cup V^{\texttt{R}}_{\texttt{cut}}(B') = \emptyset$, then $\texttt{sd}(B') := \texttt{nil}$ in G.

Based on the above analysis, we can update data(B') for the block B' with $v \in V'(B')$ in $O(1)$ time when B is the edge-block rooted at v.

- In Line R10: we update data(u') and data(u'') for the endvertices u' and u'' of the edge $e = (u', u'')$ with $\mathrm{idx}(u') < \mathrm{idx}(u'')$. Note that $d(u') \leq d(u'')$. If $|V'(B)| \geq 3$ and u_N is the last right core-vertex of B in G', then $e = (u_{N-2}, u_{N-1})$, i.e., u' is the last second right core-vertex u_{N-2} of B in G' and u'' is the last left core-vertex u_{N-1} of B in G'. In this case, the removal of u_N will lead to the change in the vertex type of $u'' = u_{N-1}$, i.e., the last left core-vertex u_{N-1} of B in G' turns to be the bottom vertex of B in G. We can see that only two data— type(u'') and pre(u'') among data(u'') can be changed from G' to G. Specifically, type(u'') := $[0, 0, \ldots, 0]$ in G, and pre(u'') := $[\emptyset, \emptyset]$.

If u_N is a wing-vertex of B in G', then by Lemma 8.3, we can know the orientation of the edge $e = (u', u'')$ with $\mathrm{idx}(u') < \mathrm{idx}(u'')$ in $O(1)$ time. Specifically, the edge e is directed from u' to u'' if u' is the first entry of $\mathrm{pre}(u_N)$ and $u'' = \mathrm{wgedge}(u_N)$, and the edge e is directed from u'' to u' if u'' is the first entry of $\mathrm{pre}(u_N)$ and $u' = \mathrm{wgedge}(u_N)$. In the following, we only analyze the case that the edge e is directed from u' to u'', since we can similarly analyze the case that the edge e is directed from u'' to u'.

The removal of u_N only leads to the change of $\mathrm{pre}(u'')$ for the head u'' of the edge e in G, since u_N is the last vertex of B pointing to u'' by a directed edge in G'. Specifically, let $\mathrm{pre}(u'') = [v_1, v_2]$ in G', where $v_1 = u_N$ if u_N is a left vertex of B, and $v_2 = u_N$ if u_N is a right vertex of B in G'. Then we update $\mathrm{pre}(u'')$ in G based on the positions of the vertices u'' and u_N in its parent-block B in G' as follows. If u'' is not the bottom vertex of B, then $\mathrm{pre}(u'') := [v_1, u']$. If $u'' = bv(B) \neq \emptyset$ and u_N is a left wing-vertex in G', then u' turns to be the last left vertex of B pointing to u'' by a directed edge in G. Thus, $\mathrm{pre}(u'') := [\emptyset, v_2]$ if u' is the last left core-vertex of B in G, and $\mathrm{pre}(u'') := [u', v_2]$ if $u' \neq v_1$ is not the last left core-vertex of B in G. If $u'' = bv(B) \neq \emptyset$ and u_N is a right wing-vertex in G', then u' turns to be the last right vertex of B pointing to u'' by a directed edge in G. Thus, $\mathrm{pre}(u'') := [v_1, \emptyset]$ if u' is the last right core-vertex of B in G, and $\mathrm{pre}(u'') := [v_1, u']$ if $u' \neq v_2$ is not the last right core-vertex of B in G.

Based on the above analysis, we can update the data for the endvertices of the edge $e = (u', u'')$ in $O(1)$ time if $|V'(B)| \geq 3$.

- In Line R11: we update data(B) for the cyclic block B in G. According to the update of data(B) in Routine Append, we only need to update the last ten data: $V^{\mathtt{L}}_{\mathtt{head}}(B)$, $V^{\mathtt{R}}_{\mathtt{head}}(B)$, $V^{\mathtt{L}}_{\mathtt{axis}}(B)$, $V^{\mathtt{R}}_{\mathtt{axis}}(B)$, $V^{\mathtt{L}}_{\mathtt{wing}}(B)$, $V^{\mathtt{R}}_{\mathtt{wing}}(B)$, $V^{\mathtt{L}}_{\mathtt{cut}}(B)$, $V^{\mathtt{R}}_{\mathtt{cut}}(B)$, $\mathtt{sbl}(B)$ and $\mathtt{sd}(B)$, respectively.

We first update the arrays for the left/right head-/axial/wing-vertices of B in G as follow. If $|V'(B)| = 2$, then u_N is the last right head-vertex of B in G', and hence we update $V^{\mathtt{R}}_{\mathtt{head}}(B)$ by remove u_N from it. If $|V'(B)| \geq 3$ is even and u_N is the last right head-vertex (resp., axial-vertex) of B in G', then u_{N-1} is the last left head-vertex (resp., axial-vertex) of B in G', and we update $V^{\mathtt{L}}_{\mathtt{head}}(B)$ (resp., $V^{\mathtt{L}}_{\mathtt{axis}}(B)$) by removing u_{N-1} from it, and update $V^{\mathtt{R}}_{\mathtt{head}}(B)$ (resp., $V^{\mathtt{R}}_{\mathtt{axis}}(B)$) by removing u_N from it. If u_N is a left (resp., right) wing-vertex, then we update $V^{\mathtt{L}}_{\mathtt{wing}}(B)$ (resp., $V^{\mathtt{L}}_{\mathtt{wing}}(B)$) by removing u_N from it.

Next we update $\mathtt{sbl}(B)$ and $\mathtt{sd}(B)$ in G similar to the update of $\mathtt{sbl}(B')$ and $\mathtt{sd}(B')$ in the case of $|V'(B)| = 1$. If $\mathtt{sbl}(B) = u_N$ or $\mathtt{sbl}(B) = \mathtt{eqv}$, then $\mathtt{sbl}(B) := \mathtt{pfx}$ in G. If $\mathtt{sd}(B) = u_N$ or $\mathtt{sd}(B) = \mathtt{eqv}$, and $V^{\mathtt{R}}_{\mathtt{wing}} \cup V^{\mathtt{R}}_{\mathtt{cut}} = \emptyset$, then $\mathtt{sd}(B) := \mathtt{nil}$ in G. If $\mathtt{sd}(B) = u_N$ or $\mathtt{sd}(B) = \mathtt{eqv}$, and $V^{\mathtt{R}}_{\mathtt{wing}} \cup V^{\mathtt{R}}_{\mathtt{cut}} \neq \emptyset$, then $\mathtt{sd}(B) := \mathtt{pfx}$ in G. If $\mathtt{sd}(B) = \mathtt{pfx}$ and $V^{\mathtt{R}}_{\mathtt{wing}} \cup V^{\mathtt{R}}_{\mathtt{cut}} = \emptyset$, then $\mathtt{sd}(B) := \mathtt{nil}$.

We have completed the analysis of Routine RemoveTip. Based on the above analysis and Lemma 7.1, we can easily obtain the following result.

Lemma 7.3. *Given a canonical embedding G' with $N(\geq 1)$ vertices, Routine* RemoveTip(G') *is executed in $O(1)$ time and in $O(n)$ space in the worst case.* □

7.5 Realization of Routine NextCode

Given a given canonical embedding, Routine NextCode helps to generate its child-embeddings that has not been generated. Let G be the current canonical embedding. Given a block $B \in \mathcal{E}^*(G)$, an element $\varepsilon \in \mathcal{E}(B)$ and a code $\gamma \in \Gamma(\varepsilon)$, Routine NextCode$(B, \varepsilon, \gamma; G)$ returns an array with three entries: the code γ' succeeding γ if any, the element ε' with $\gamma' \in \Gamma(\varepsilon')$ and the block B' with $\varepsilon' \in \mathcal{E}(B')$, and returns $\emptyset$ if no such γ' exists. The general structure for Routine NextCode is given as follows.

```
Routine NextCode($B, \varepsilon, \gamma; G$)
/* Let $G$ be current canonical embedding, and $\mathcal{E}^*(G)$ be the sequence consisting of all
  applicable elements of $G$. */
N1 begin
N2   if $\varepsilon$ is not the last element in $\mathcal{E}^*(G)$ and $\gamma$ is the largest code in $\Gamma(\varepsilon)$ then
N3       Let $\varepsilon' \in \mathcal{E}^*(G)$ be the element succeeding $\varepsilon$, $B'$ be the block such that $\varepsilon' \in \mathcal{E}(B')$,
          and $\gamma'$ be the smallest code in $\Gamma(\varepsilon')$;
N4       Return $[\gamma', \varepsilon', B']$
N5   else if $\varepsilon$ is not the last element in $\mathcal{E}^*(G)$ and $\gamma$ is not the largest code in $\Gamma(\varepsilon)$ then
N6       Let $\gamma'$ is the code in $\Gamma(\varepsilon)$ succeeding $\gamma$;
N7       Return $[\gamma', \varepsilon, B]$
N8   else /* $\varepsilon$ is the last element in $\mathcal{E}^*(G)$ and $\gamma$ is the largest code in $\Gamma(\varepsilon)$ */
N9       Return $\emptyset$
N10  endif
N11 end.
```

• In Line N2: we check whether ε is the last element in $\mathcal{E}^*(G)$ or not and check whether γ is the largest code in $\Gamma(\varepsilon)$ or not. By definition of $\mathcal{E}^*(G)$, we can check whether ε is the last element in $\mathcal{E}^*(G)$ in $O(1)$ time. In the following, we first review the definition of $\mathcal{E}^*(G)$, and then give the sufficient and necessary condition that ε is the last element in $\mathcal{E}^*(G)$.

Let $B^1, B^2, \ldots, B^p$ be the blocks in the spine of G, and $\mathcal{E}(G) = [r_G, \mathcal{E}(B^1), \mathcal{E}(B^2), \ldots, \mathcal{E}(B^p)]$. Recall that if $cs(G) = \texttt{stc}$, then $\mathcal{E}^*(G) = \mathcal{E}(G)$. Otherwise, i.e., $cs(G) \in \{\texttt{eqv}, \texttt{pfx}\}$. Let B^ℓ ($\ell \in [1, p]$) be the dominating block of G. If $cs(G) = \texttt{eqv}$, then $\mathcal{E}^*(G) = [r_G, \mathcal{E}(B^1), \mathcal{E}(B^2), \ldots, \mathcal{E}(B^{\ell-1})]$; and if $cs(G) = \texttt{pfx}$, then $\mathcal{E}^*(G) = [r_G, \mathcal{E}(B^1), \mathcal{E}(B^2), \ldots, \mathcal{E}(B^\ell)]$, where the critical element ε^* of G is the last one in $\mathcal{E}(B^\ell)$. Based on the definition of $\mathcal{E}^*(G)$, we can see that an element ε is the last one in $\mathcal{E}^*(G)$ if and only if (1) $cs(G) = \texttt{pfx}$ and ε is the critical element of G, (2) $cs(G) = \texttt{stc}$, B is the tip-block of G (i.e., the tip B is not a cut-vertex), and ε is the last element in $\mathcal{E}(B)$ (by Lemma 8.9), or (3) $cs(G) = \texttt{eqv}$, $B = B^{\ell-1}$, and ε is the last element in $\mathcal{E}(B)$ (by Lemma 8.9). By Lemma 8.10, we can calculate the largest code of $\Gamma(\varepsilon)$ for a given element ε in $\mathcal{E}(B)$ in $O(1)$ time, and hence we can know whether a given code γ is the largest code or not in $O(1)$ time.

- In Line N3: we compute the element ε' succeeding ε and the smallest code γ' in $\Gamma(\varepsilon')$ when ε is not the last element in $\mathcal{E}^*(G)$ in $O(1)$ time. The calculation of ε' depends on whether ε is the last element in $\mathcal{E}(B)$. Note that we can check whether ε is the last element of $\mathcal{E}(B)$ in $O(1)$ time by Lemma 8.9.

Recall that $\varepsilon^i_{\text{fst}}(B)$ and $\varepsilon^i_{\text{lst}}(B)$ are the first element and the last element of $\mathcal{E}(B) \cap X_i$ for

$$X_1 = V_{\texttt{R}}(G(B)),\ X_2 = E_{\texttt{R}}(y_1; B),\ X_3 = V_{\texttt{L}}(G(B)),\ X_4 = E_{\texttt{L}}(x_1; B),\ X_5 = \{e^b, bv(B)\},$$

where x_1 and y_1 are the first left and right head-vertex of B. Let $t(B)$ be the tip of B, which can be calculated in $O(1)$ time by Lemma 8.4. Let B' be the rightmost block rooted at $t(B)$.

If ε is the last element of $\mathcal{E}(B)$, then ε' is the first element in $\mathcal{E}(B')$, which can be calculated in $O(1)$ time by Lemma 8.9. Otherwise, ε is not the last element of $\mathcal{E}(B)$. Then the element ε' next to ε also belongs to $\mathcal{E}(B)$. If $\varepsilon = \varepsilon^i_{\text{lst}}(B) \neq \emptyset$ for some $1 \leq i \leq 4$, then ε' is the first non-empty element in the sequence $\varepsilon^{i+1}_{\text{fst}}(B)$, $\varepsilon^{i+1}_{\text{lst}}(B)$, $\varepsilon^{i+2}_{\text{fst}}(B)$, $\varepsilon^{i+2}_{\text{lst}}(B)$,..., $\varepsilon^5_{\text{fst}}(B)$ and $\varepsilon^5_{\text{lst}}(B)$. If $\varepsilon^i_{\text{fst}}(B) \neq \emptyset$ and $\text{idx}(\varepsilon^i_{\text{fst}}(B)) \leq \text{idx}(\varepsilon) < \text{idx}(\varepsilon^i_{\text{lst}}(B))$ for $i = 1$ (resp., $i = 3$), then ε' is the right (resp., left) vertex with $\text{idx}(\varepsilon') = \text{idx}(\varepsilon) + 1$. If $\varepsilon^i_{\text{fst}}(B) \neq \emptyset$ and $\varepsilon = (v, u)$ is a right (resp., left) edge directed from v to u such that $d(\varepsilon^i_{\text{fst}}(B)) \leq d(\varepsilon) < d(\varepsilon^i_{\text{lst}}(B))$ for $i = 2$ (resp., $i = 4$), then ε' is the right (resp., left) edge $e = (w, v)$, where w is equal to the second entry of $\text{pre}(v)$. If $\varepsilon = \varepsilon^5_{\text{fst}}(B) = bv(B) \neq \emptyset$ and $\mathcal{B}(bv(B)) = \emptyset$, then $\varepsilon' = \varepsilon^5_{\text{lst}}(B) = e^b$.

Besides, we can compute the smallest code $\gamma' \in \Gamma(\varepsilon')$ in $O(1)$ time by Lemma 8.10. Based on the above analysis, we can compute the element ε' succeeding ε and the smallest code γ' in $\Gamma(\varepsilon')$ in $O(1)$ time.

- In Line N6: we check whether ε is the last element in $\mathcal{E}^*(G)$, and check whether γ is the largest code in $\Gamma(\varepsilon)$. As mentioned in Line N3, we can check whether ε is the last element in $\mathcal{E}^*(G)$ in $O(1)$ time. By Lemma 8.10, we can compute the largest code in $\Gamma(\varepsilon)$ in $O(1)$ time, and hence we can check whether γ is the largest code in $\Gamma(\varepsilon)$ in $O(1)$ time.

- In Line N7: we calculate the code γ' succeeding γ when ε is not the last element in $\mathcal{E}^*(G)$ and γ is not the largest code in $\Gamma(\varepsilon)$. In the following, we explain the calculation of the code γ' based on the cases in the definition of code set in Section 6.1. Let $\gamma = (d_1, \texttt{at}, d_2, \texttt{op}, c)$.

(1) For $\varepsilon = v \in V'(B) - V_{\texttt{cut}}(B)$ such that v is not y_j in Case-1(d) (Figure 6.1(d)) and 3(d) (Figure 6.3(d)), we have $\gamma' := (d_1, \texttt{at}, d_2, \texttt{op}, c + 1)$.

(2)-(i) For $\varepsilon = e'_{q'}$, let $x_{p_{\mathtt{R}}+1}$ be the $(|V^{\mathtt{R}}_{\mathtt{wing}}(B)|+1)$st left wing-vertex of B, we have

$$\gamma' := \begin{cases} (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1) & \text{if } \mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{triangle}; \\ (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1) & \text{if } \mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{subdivide},\ \mathtt{op} = \mathtt{triangle} \text{ and } c < c_K;; \\ (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c_1) & \text{if } \mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{subdivide},\ \mathtt{op} = \mathtt{triangle} \text{ and } c = c_K; \\ (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c+1) & \text{if } \mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{subdivide},\ \mathtt{op} = \mathtt{subdivide} \text{ and } c < c(x_{p_{\mathtt{R}}+1}). \end{cases}$$

(2)-(ii) For $\varepsilon = e'$ that is not the edge $e'_{q'}$ in Case-2(b)-(c) (Figure 6.2(b)-(c)), Case-3(c) (Figure 6.3(c)) and Case-4(b) (Figure 6.4(b)), recall that $A(B)$ is the set of axial-edges in B. We have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c_1)$ if $e' \notin A(B) \cup \{e'_{q'+1}\}$, $\mathtt{op} = \mathtt{triangle}$ and $c = c_K$; and $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ otherwise.

(3) For a left edge $\varepsilon = e_j \in \mathcal{E}(B)$, we have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c_1)$ if $e_j \notin A(B) \cup \{e_{q+1}\}$, $\mathtt{op} = \mathtt{triangle}$ and $c = c_K$; and $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ otherwise.

(4)-(a) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has no axial-vertex and $|V(B)|$ is even, we have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$ (Figure 6.6(c)) and $c < c_K$, or, $\mathtt{sd}(B; G) = \mathtt{nil}$, $|V(B)| \geq 2$ and $c < c(z)$ (Figure 6.5(c) and Figure 6.5(f)).

(4)-(b) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has no axial-vertex and $|V(B)|$ is odd (Figures 6.5(a)-6.6(a)), we have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{triangle}, c_1)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$, $\mathtt{op} = \mathtt{star}$ and $c = c_K$; $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c_1)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$, $\mathtt{op} = \mathtt{triangle}$ and $c = c_K$; and $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ otherwise.

(4)-(c) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has axial-vertices and $|V(B)|$ is even (Figures 6.5(d)-6.6(d)), we have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$ and $c < c_K$, or, $\mathtt{sd}(B; G) = \mathtt{nil}$, $V^L_{\mathtt{axis}}(B) \cup V^R_{\mathtt{axis}}(B) \neq \emptyset$, $|V(B)|$ is even and $c < c(z)$.

(4)-(d) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has axial-vertices and $|V(B)|$ is odd (Figures 6.5(b)-6.6(b)), we have $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{subdivide}, c_1)$ if $\mathtt{op} = \mathtt{triangle}$ and $c = c_K$; $\gamma' := (d_1, \mathtt{at}, d_2, \mathtt{op}, c+1)$ otherwise (i.e., if $c < c_K$).

We have completed the analysis of Routine NextCode. Based on the above analysis and Lemma 7.1, we can easily derive the following result.

Lemma 7.4. *Given a block B in the spine of G, an element $\varepsilon \in \mathcal{E}(B)$ and a code $\gamma \in \Gamma(\varepsilon)$, Routine* NextCode$(B, \varepsilon, \gamma; G)$ *runs in $O(1)$ time and in $O(n)$ space in the worst case.* □

By Lemmas 7.1 and 7.2-7.4, we derive the time and space complexities of Algorithm GENERATE as follows.

Theorem 7.1. *Given an integer $n \geq 1$ and a set $\mathcal{C} = (c_1, c_2, \ldots, c_K)$ of K colors, Algorithm* GENERATE$(n, \mathcal{C})$ *enumerates all colored and rooted non-isomorphic outerplanar graphs with at most n vertices in $O(1)$ time per each graph in the worst case by using $O(n)$ space.* $\square$

Chapter 8

Conclusion

This thesis is concerned with the problem of generating all colored and rooted outerplanar graphs with at most given number $n(\geq 1)$ of vertices in a systematical way. We have proposed an $O(n)$ space CAT algorithm for this problem, i.e., an algorithm that generates all required outerplanar graphs without repetition in constant time per each and in $O(n)$ space. The design of the algorithm relies on the careful choices of canonical representation and family tree. The canonical representation of a colored and rooted outerplanar graph is a unique embedding of the graph having good properties, which facilitates avoiding isomorphic duplications due to the symmetric structures of graphs. The family tree is a data structure consisting of canonical representations of all outerplanar graphs required to be generated: each node of the family tree corresponds to a canonical outerplanar embedding, and each edge between two nodes corresponds to the parent-child relationship between two canonical outerplanar embeddings which have constant-size differences.

The algorithm proposed in this thesis may have potential applications in various areas such as mathematics, chemoinformatics and computer science. For example, our exhaustive generation algorithm may be used to solve the random generation problem of colored and rooted outerplanar graphs. Recall that the family tree defined in this thesis consists of all canonical outerplanar embeddings with at most given number $n(\geq 1)$ of vertices, where a node at the i-th depth ($i \in [1, n]$) corresponds to a canonical outerplanar embeddings with i vertices, and an edge correspond to the parent-child relationship between two canonical embeddings with constant-size changes. Based on the family tree, for any given canonical embedding G with $i \in [1, n]$ vertices, we can trace all ancestor-embeddings $P(G), P^2(G), \ldots P^{i-1}(G)$ of G by a path $P = v_{i-1} \ldots v_2 v_1$ in the family tree, where for $j = i-1, i-2, \ldots, 1$, the node v_j corresponds to the ancestor-embedding $P^{i-j}(G)$ of G. Then we may randomly generate such a canonical outerplanar embedding G by randomly generating such a path P in the family tree.

Besides, in chemoinformatics, the list of colored and rooted outerplanar graphs may be used to obtain chemical compounds with desired properties. Recall that the majority of chemical graphs in the NCI database are outerplanar graphs [73]. The problem of inferring chemical graphs under given constraints is one of the fundamental and important problems in chemoinformatics. In the literature, Akutsu and Fukagawa [3] considered the general graph inference problem from the numbers of occurrences of vertex-labeled paths with length at most K. In particular, they designed a dynamic programming algorithm for inferring the graphs in a restricted class of outerplanar graphs [4]. However, the time complexity of their algorithm is exponential in K and the number of labels. In recent years, various approaches have been proposed to screen chemical compounds libraries (i.e., a series of stored chemical compounds) for identifying the compounds with the desired properties. We believe that the exhaustive list of colored and rooted outerplanar graphs in this thesis may speed up solving the inference problem of chemical outerplanar graphs. We may design a branch-and bound algorithm based on the exhaustive generation approach proposed in this thesis, which is similar with the tree case [60, 75]. That is, starting from a colored and rooted outerplanar graph consisting of a single vertex, we may obtain all required chemical outerplanar graphs by recursively generating colored and rooted outerplanar graphs while discarding the graphs which violate the given constraints.

The work in this thesis also simulates some interesting problems. For example, it would be interesting to characterize what kinds of graphs or other combinatorial objects which admit CAT algorithms. The characterization of such combinatorial objects may have the following advantages: 1) it may allow us to find patterns hidden in the structure of combinatorial objects which may lead to a more thorough understanding of these objects; and 2) it may produce a new and systematic framework of CAT algorithms for exhaustively generating these objects. The generation algorithm of colored and rooted outerplanar graphs in this thesis is generalized from that of colored and rooted trees [121]. The common key idea in the algorithms of rooted outerplanar graphs in this thesis and that of rooted trees in [121] is to find good canonical representations for them such that the canonical representation of any given graph can be obtained from the canonical representation of another graph by making constant-size changes without generating duplications. The discovery of such canonical representation of objects under study depends on the structures of these objects. An interesting but already very complicated class of graphs is the class of colored and *unrooted* outerplanar graphs. Comparing with colored and rooted outerplanar graphs studied in this thesis, it is much more difficult to design a CAT algorithm for unrooted outerplanar graphs because of much more complicated symmetric structures.

Appendix

During the enumeration, we only maintain the data structures for each block and each vertex for a canonical embedding defined in Section 7.2. Note that these defined data are parts of data involved in the algorithm. There are other data for the embedding involved in the algorithm but are not defined in Section 7.2. Appendix will show how these undefined data are computed by the defined data. Let G be the current canonical embedding.

The following lemma is used to determine whether the bottom vertex $bv(B)$ of a cyclic block B exists or not and compute $bv(B)$ if exists.

Lemma 8.1. *Given a cyclic block B, its bottom vertex $bv(B)$ can be calculated in $O(1)$ time.*

Proof: By definition of the signature of G, the code of the bottom vertex $bv(B)$ is next to the code of the last right core-vertex y^* if $bv(B) \neq \emptyset$, that is, $\mathrm{idx}(bv(B)) = \mathrm{idx}(y^*) + 1$. Note that y^* is the vertex with the largest index in $V^{\mathtt{R}}_{\mathtt{head}}(B) \cup V^{\mathtt{R}}_{\mathtt{axis}}(B)$. Besides, we can check whether any given vertex $u \in V'(B)$ is the bottom vertex of B or not in $O(1)$ time. From the definition of $\mathrm{type}(u)$, u is the bottom vertex of B if and only if all entries of $\mathrm{type}(u)$ are equal to zero. Based the above analysis, B has the bottom vertex $bv(B) = u$ if and only if u is the vertex of B such that $\mathrm{idx}(u) = \mathrm{idx}(y^*) + 1$ and such that the first six entries of $\mathrm{type}(u)$ are equal to zero. Thus, $bv(B)$ can be calculated by $\mathrm{type}(u)$ and $\mathrm{data}(B)$ in $O(1)$ time. □

Clearly, an edge $e = (u, v)$ belongs to a block B if and only if $u, v \in V(B)$. No data for edges of an embedding will be maintained in the algorithm. In the following Lemmas 8.2 and 8.3, we will show that we can compute the information of such an edge e including the edge type, orientation and depth can be calculated by the data of its two end-vertices u and v and the data of the block B.

Lemma 8.2. *Given a block B and an edge $e = (u, v) \in E(B)$, whether e is a head-edge or axial-edge of B can be inferred in $O(1)$ time.*

Proof: Recall that $\mathrm{type}(u')$ for each vertex $u' \in V'(B)$ contains the information whether u' is left/right head-/axial-/wing-/cut-vertex of B, and B contains the index of the vertex u among the same type of vertices in B. Then we can check whether the given edge e is a head-edge, axial-edge or the bottom edge of B as follows. Without loss of generality, assume that $d(u) < d(v)$. If $u = r(B)$, then e is a head-edge of B. If $u \neq r(B)$, and either (1) u is the last head-vertex and v is a axial-vertex on the same side of B, or (2) both u and v are axial-vertices of B, then e is an axial-edge of B. In particular, if $u \neq r(B)$, $bv(B) = \emptyset$, and u and v are the last left and right core-vertices of B, respectively, then e is the bottom edge of B. □

Lemma 8.3. *Given a block B with $\tilde{E}(B) \neq \emptyset$ and a left (resp., right) edge $e = (u, v)$ of B, both the orientation and the depth of the edge e can be determined in $O(1)$ time.*

Proof: By definition, $e = (u, v)$ is a left (resp., right) edge of B if and only if its endvertices $u, v \in V^{\mathtt{L}}_{\mathtt{core}}(B) \cup \{bv(B)\} \cup V^{\mathtt{L}}_{\mathtt{wing}}(B)$ (resp., $u, v \in V^{\mathtt{R}}_{\mathtt{core}}(B) \cup \{bv(B)\} \cup V^{\mathtt{R}}_{\mathtt{wing}}(B)$). Without loss of generality, assume that $\mathrm{idx}(u) < \mathrm{idx}(v)$.

We can compute the orientation and the depth of the edge $e = (u, v)$ by the data type(u), type(v) and pre(v). Specifically, if both u and v are core-vertices on the same side of B, i.e., $u, v \in V^{\mathtt{L}}_{\mathtt{core}}(B) \cup \{bv(B)\}$ (resp., $u, v \in V^{\mathtt{R}}_{\mathtt{core}}(B) \cup \{bv(B)\}$), then the edge $e = (u, v)$ is directed from u to v. By definition, the depth of the edge e is calculated as follows. If u is the last vertex in $V^{\mathtt{L}}_{\mathtt{core}}(B)$ (resp., $V^{\mathtt{R}}_{\mathtt{core}}(B)$) and $v = bv(B)$, then $d(e) = 1$; if u and v are the ith and $(i+1)$st vertex in $V^{\mathtt{L}}_{\mathtt{core}}(B)$ (resp., $V^{\mathtt{R}}_{\mathtt{core}}(B)$), respectively, then

$$d(e) = |V^{\mathtt{L}}_{\mathtt{core}}(B) \cup \{bv(B)\}| - i = |V^{\mathtt{L}}_{\mathtt{head}}(B) \cup V^{\mathtt{L}}_{\mathtt{axis}}(B) \cup \{bv(B)\}| - i$$

$$(\text{resp., } \; d(e) = |V^{\mathtt{R}}_{\mathtt{core}}(B) \cup \{bv(B)\}| - i = |V^{\mathtt{R}}_{\mathtt{head}}(B) \cup V^{\mathtt{R}}_{\mathtt{axis}}(B) \cup \{bv(B)\}| - i).$$

Otherwise, at least one of u and v is a wing-vertex of B. Due to $\mathrm{idx}(u) < \mathrm{idx}(v)$, the vertex v must be a wing-vertex of B. Let $e' = (v', v'')$ be the edge directed from v' to v'', to which is applied code $\gamma(v)$ to generate the wing-vertex v, where v' is the first entry of pre(v) and $v'' = \mathrm{wgedge}(v)$. Due to $\mathrm{idx}(u) < \mathrm{idx}(v)$, we have $u = v'$ or $u = v''$. Then $e = (u, v)$ is directed from u to v if $u = v'$ is the first entry of pre(v), and $e = (u, v)$ is directed from v to u if $u = v'' = \mathrm{wgedge}(v)$. By definition, the depth of the edge e is calculated as follows. Let v be the ith vertex in $V^{\mathtt{L}}_{\mathtt{wing}}(B)$ (resp., $V^{\mathtt{R}}_{\mathtt{wing}}(B)$), and $L_1 = |V^{\mathtt{L}}_{\mathtt{core}}(B) \cup \{bv(B)\}|$ (resp., $L_1 = |V^{\mathtt{R}}_{\mathtt{core}}(B) \cup \{bv(B)\}|$). Then

$$d(e) = \begin{cases} 2i + L_1 - 1 & \text{if } u \text{ is equal to the first entry of pre}(v); \\ 2i + L_1 - 2 & \text{if } u = \mathrm{wgedge}(v). \end{cases}$$

In summary, we can compute the orientation and the depth of a left (resp., right) edge of given block in $O(1)$ time. □

Lemma 8.4. *Given a block B of the spine of G, the tip $t(B)$ of B can be calculated in $O(1)$ time.*

Proof: By definition, $t(B)$ is the vertex with the largest index in array A, where $A = V^{\mathtt{R}}_{\mathtt{cut}}(B)$ if $V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$, $A = V^{\mathtt{R}}_{\mathtt{wing}}(B)$ if $V^{\mathtt{R}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$, $A = V^{\mathtt{L}}_{\mathtt{cut}}(B)$ if $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$, $A = V^{\mathtt{L}}_{\mathtt{wing}}(B)$ if $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$, and $A = V_{\mathtt{core}}(B)$ if $V^{\mathtt{R}}_{\mathtt{cut}}(B) = V^{\mathtt{R}}_{\mathtt{wing}}(B) = V^{\mathtt{L}}_{\mathtt{cut}}(B) = V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$. Clearly, we can compute the tip of any given block in $O(1)$ time. □

Lemma 8.5. *Given a block B and a vertex $u \in V'(B)$, let γ be a code and let ε be the element to which is applied the code γ to generate the vertex u. Then both the code γ and the element ε can be calculated in $O(1)$ time.*

Proof: The number $|V'(B)|$ of non-root vertices of B is calculated by

$$|V'(B)| = |V^{\mathtt{L}}_{\mathtt{head}}(B)| + |V^{\mathtt{R}}_{\mathtt{head}}(B)| + |V^{\mathtt{L}}_{\mathtt{axis}}(B)| + |V^{\mathtt{R}}_{\mathtt{axis}}(B)| + |\{bv(B)\}| + |V^{\mathtt{L}}_{\mathtt{wing}}(B)| + |V^{\mathtt{R}}_{\mathtt{wing}}(B)|.$$

We compute the code γ and the applicable element ε based on $|V'(B)|$ and the vertex type of u as follows.

If $|V'(B)| = 1$, then ε is the root $r(B)$ of B, and $\gamma = (d(B)-1, \mathtt{at}, d(r(B)), \mathtt{new\text{-}block}, c(u))$, where $d(B)$, $d(r(B))$ and $c(u)$ have been maintained into data(B), data$(r(B))$ and data(u); and $\mathtt{at}$ can be computed by type(ε). Specifically, $\mathtt{at} = *$ if $\varepsilon = r_G$, or, $\varepsilon \neq r_G$ and all entries of type(ε) are equal to 0; $\mathtt{at} = \mathtt{h}^{\mathtt{L}}$ if the first, third or fifth entry of type(ε) is larger than 0; and $\mathtt{at} = \mathtt{h}^{\mathtt{L}}$ if the second, fourth or sixth entry of type(ε) is larger than 0.

If $|V'(B)| = 2$, then ε is the edge $e = (r(B), u')$ with idx(u') = idx$(u) - 1$, and the code $\gamma = (d(B), *, d(u'), \mathtt{triangle}, c(u))$, where $d(B)$, $d(u')$ and $c(u)$ have been maintained into data(B), data(u') and data(u).

If $|V'(B)| \geq 3$ and u is a core-vertex of B, then ε is the edge $e = (u', u'')$ with idx(u') = idx$(u) - 1$ and idx(u'') = idx$(u) - 2$, and the code $\gamma = (d(B), *, d(e), \mathtt{op}(u), c(u))$, where $d(B)$, $\mathtt{op}(u)$ and $c(u)$ have been maintained into data(B) and data(u), and $d(e)$ can be calculated by Lemma 8.3.

If $|V'(B)| \geq 3$ and u is a left (resp., right) wing-vertex of B, then ε is the edge $e = (u', v')$, where u' is the first vertex in pre(u) and v' = wgedge(u), and $\gamma = (d(B), \mathtt{w}^{\mathtt{L}}, d(e), \mathtt{op}(u), c(u))$ (resp., $\gamma = (d(B), \mathtt{w}^{\mathtt{R}}, d(e), \mathtt{op}(u), c(u))$). Similarly, the three entries $d(B)$, $\mathtt{op}(u)$ and $c(u)$ of $\gamma(u)$ have been maintained into data(B) and data(u), and $d(e)$ can be calculated by Lemma 8.3.

Based on the above analysis, we can compute code and element that are used to generate a vertex of a block in $O(1)$ time. □

Recall that corresponding vertices with respect to a block of G with $cs(G) = \mathtt{pfx}$ are particularly useful for efficiently generating the critical child-embedding of G in the algorithm. The following Lemmas 8.6 and 8.7 will show the calculation of corresponding vertices based on the left-sibling-heaviness and left-side-heaviness, respectively.

Lemma 8.6. *Given a block B with a sibling $\hat{B}$ preceding it, the corresponding vertex $\mu^+(u; r(B))$ for $u \in V(G(\hat{B})) - \{r(B)\}$ and $\mu^-(u; r(B))$ for $u \in V(G(B)) - \{r(B)\}$ are calculated, respectively, by*

$$\begin{aligned} \mu^+(u; r(B)) &= \mathrm{idx}(u) + \mathrm{idx}(\ell v(B)) - \mathrm{idx}(\ell v(\hat{B})) \quad \text{for } u \in V(G(\hat{B})) - \{r(B)\}, \\ \mu^-(u; r(B)) &= \mathrm{idx}(u) - \mathrm{idx}(\ell v(B)) + \mathrm{idx}(\ell v(\hat{B})) \quad \text{for } u \in V(G(B)) - \{r(B)\}. \end{aligned} \tag{8.1}$$

Proof: By definition, we can derive the above formulas of $\mu^+(u; r(B))$ for $u \in V(G(\hat{B})) - \{r(B)\}$ and $\mu^-(u; r(B))$ for $u \in V(G(B)) - \{r(B)\}$. □

By Equation (8.1) in above Lemma 8.6, we can compute the corresponding vertices with respect to a block having a sibling block in $O(1)$ time.

Lemma 8.7. *Given a cyclic block B with $V^{\mathtt{L}}_{\mathtt{wing}}(B) \cup V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \cup V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$, let $u^{\mathtt{L}}$ be the first vertex in $V^{\mathtt{L}}_{\mathtt{wing}}(B)$ if $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$, or let $u^{\mathtt{L}} = \ell v(B'_1)$ for the leftmost block $B'_1 \in \mathcal{B}(u')$ of the first vertex u' in $V^{\mathtt{L}}_{\mathtt{cut}}(B)$ if $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$. Let $V_{\mathtt{L}}(G(B))$ denote the set of left vertices of B and their descendants if $V^{\mathtt{L}}_{\mathtt{wing}}(B) \cup V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$. Similarly, let $u^{\mathtt{R}}$ be the first vertex in $V^{\mathtt{R}}_{\mathtt{wing}}(B)$ if $V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$, or let $u^{\mathtt{R}} = \ell v(B''_1)$ for the leftmost block $B''_1 \in \mathcal{B}(u'')$ of the first vertex u'' in $V^{\mathtt{R}}_{\mathtt{cut}}(B)$ if $V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$. Let $V_{\mathtt{R}}(G(B))$ denote the set of right vertices of B and their descendants if $V^{\mathtt{R}}_{\mathtt{wing}}(B) \cup V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$. Then the corresponding vertex $\mu(u; r(B))$ for a vertex $u \in V(G(B)) - \{r(B)\}$ is calculated by*

$$\mu(u; r(B)) = \begin{cases} \mathrm{idx}(u) + 1 & \text{for } u \in V^{\mathtt{L}}_{\mathtt{core}}(B) \\ \mathrm{idx}(u) - 1 & \text{for } u \in V^{\mathtt{R}}_{\mathtt{core}}(B) \\ \mathrm{idx}(u) & \text{for } u \in V(G(bv(B))) \\ \mathrm{idx}(u) + \mathrm{idx}(u^{\mathtt{R}}) - \mathrm{idx}(u^{\mathtt{L}}) & \text{for } u \in V_{\mathtt{L}}(G(B)) \\ \mathrm{idx}(u) - \mathrm{idx}(u^{\mathtt{R}}) + \mathrm{idx}(u^{\mathtt{L}}) & \text{for } u \in V_{\mathtt{R}}(G(B)). \end{cases} \tag{8.2}$$

Proof: Recall that given a block B of a canonical embedding G, the code sequence $\sigma(G(B); G)$ consists of $\sigma_{\mathtt{core}}(G(B); G)$, $\sigma_{\mathtt{b}}(G(B); G)$, $\sigma_{\mathtt{L}}(G(B); G)$ and $\sigma_{\mathtt{R}}(G(B); G)$, respectively.

By definition, a pair of left and right core-vertices x and y having the same depth in B are corresponding vertices, where $\mathrm{idx}(y) = \mathrm{idx}(x) + 1$; besides, the corresponding vertex of a vertex $u \in V(G(bv(B)))$ is itself. Thus, the corresponding vertex $\mu(u; r(B))$ for $u \in V_{\mathtt{core}}(B) \cup V(G(bv(B)))$ is calculated by

$$\mu(u; r(B)) = \begin{cases} \mathrm{idx}(u) + 1 & \text{for } u \in V^{\mathtt{L}}_{\mathtt{core}}(B) \\ \mathrm{idx}(u) - 1 & \text{for } u \in V^{\mathtt{R}}_{\mathtt{core}}(B) \\ \mathrm{idx}(u) & \text{for } u \in V(G(bv(B))). \end{cases}$$

For $u \in V(G(B)) - V_{\mathtt{core}}(B) - V(G(bv(B)))$, the corresponding vertex $\mu(u; r(B))$ can be computed if $V^{\mathtt{L}}_{\mathtt{wing}}(B) \cup V^{\mathtt{L}}_{\mathtt{cut}}(B) \neq \emptyset$ and $V^{\mathtt{R}}_{\mathtt{wing}}(B) \cup V^{\mathtt{R}}_{\mathtt{cut}}(B) \neq \emptyset$ as follows. By definition, the code of $u^{\mathtt{L}}$ (resp., $u^{\mathtt{R}}$) is the first one in $\sigma_{\mathtt{L}}(G(B);G)$ (resp., $\sigma_{\mathtt{R}}(G(B);G)$). Then $|V_{\mathtt{L}}(G(B))| = |\sigma_{\mathtt{L}}(G(B);G)| = \mathrm{idx}(u^{\mathtt{R}}) - \mathrm{idx}(u^{\mathtt{L}})$. Hence

$$\mu(u;r(B)) = \begin{cases} \mathrm{idx}(u) + \mathrm{idx}(u^{\mathtt{R}}) - \mathrm{idx}(u^{\mathtt{L}}) & \text{for } u \in V_{\mathtt{L}}(G(B)) \\ \mathrm{idx}(u) - \mathrm{idx}(u^{\mathtt{R}}) + \mathrm{idx}(u^{\mathtt{L}}) & \text{for } u \in V_{\mathtt{R}}(G(B)). \end{cases}$$

□

By Equation (8.2) in above Lemma 8.7, we can compute the corresponding vertex of a vertex in $V(G(B)) - \{r(B)\}$ in $O(1)$ time.

Recall that only a canonical embedding G with $cs(G) = \mathtt{pfx}$ has a unique critical child-embedding; and the critical child-embedding of such an embedding G is the child-embedding obtained from G by applying the critical code γ^* to the critical element ε^* of G. The following lemma will show how to compute the critical element ε^* and the critical code γ^* of G by the maintained data for G.

Lemma 8.8. *Given a canonical embedding G with $cs(G) = \mathtt{pfx}$ and the dominating block B^ℓ of G, both the critical element ε^* and the critical code γ^* of G can be calculated in $O(1)$ time.*

Proof: Suppose that G has $N \geq 1$ vertices, and u_N is the vertex with $\mathrm{idx}(u_N) = N$. Let v' be the succeeding-copy-vertex of G and ε' be the element to which is applied the code $\gamma(v')$ to generate v'.

By definition, the critical element ε^* corresponds to the element ε' with respect to B^ℓ, i.e., both ε^* and ε' are corresponding vertices with respect to B^ℓ, or both ε^* and ε' are edges such that their endvertices are corresponding vertices with respect to B^ℓ; besides, the critical code γ^* is defined based on $\gamma(v')$.

Specifically, if the copy-state of G is given by $\mathtt{sbl}(B^\ell;G) = \mathtt{pfx}$, then $v' = \mu^-(u_N; r(B^\ell))$ can be calculated by Equation (8.1) in Lemma 8.6, and ε' and $\gamma(v')$ can be calculated by Lemma 8.5. If ε' is a vertex, then ε^* is the corresponding vertex of ε' calculated by Equation (8.1) in Lemma 8.6. If $\varepsilon' = (u', v')$ is an edge, then $\varepsilon^* = (u^*, v^*)$ is the edge such that u^* and u' are corresponding vertices, and v^* and v' are corresponding vertices with respect to B^ℓ, where u^* and v^* can be calculated by Equation (8.1) in Lemma 8.6. Besides, by definition, the critical code γ^* of G is equal to $\gamma(v')$.

If the copy-state of G is given by $\mathtt{sd}(B^\ell;G) = \mathtt{pfx}$, then $v' = \mu(u_N; r(B^\ell))$ can be calculated by Equation (8.2) in Lemma 8.7, and similarly ε' and $\gamma(v')$ can be calculated by Lemma 8.5. Similarly, the corresponding element ε^* of ε' can be calculated by Equation (8.2) in Lemma 8.7. Besides, by definition, $\gamma^* = \overline{\gamma}(v')$ if (1) v' is a left wing-vertex of B^ℓ, or (2) v' is a child vertex of a left vertex of B^ℓ, and $\gamma^* = \gamma(v')$ otherwise.

From the above analysis, we can compute the critical element ε^* and the critical code γ^* of canonical embedding G with $cs(G) = \mathtt{pfx}$ in $O(1)$ time. □

By Sections 6.1 and 6.2, we can generate child-embeddings of G by applying codes in $\Gamma(\varepsilon)$ of elements ε in $\mathcal{E}^*(G)$. To systematically generate child-embeddings of G, we expect that we can automatically gain the next element for a given element in $\mathcal{E}^*(G)$ and the next code of a given code with respect to lexicographical order. The following Lemmas 8.9 and 8.10 serve for this purpose.

Lemma 8.9. *Given a cyclic block B in the spine of G, let $\varepsilon^i_{\mathrm{fst}}(B)$ and $\varepsilon^i_{\mathrm{lst}}(B)$ be the first element and the last element of $\mathcal{E}(B) \cap X_i$ for*

$$X_1 = V_{\mathtt{R}}(G(B)),\ X_2 = E_{\mathtt{R}}(y_1; B),\ X_3 = V_{\mathtt{L}}(G(B)),\ X_4 = E_{\mathtt{L}}(x_1; B),\ X_5 = \{e^b, bv(B)\},$$

where x_1 and y_1 are the first left and right head-vertex of B. Then $\varepsilon^i_{\mathrm{fst}}(B)$ and $\varepsilon^i_{\mathrm{lst}}(B)$ for $i = 1, 2, \ldots, 5$ can be computed in $O(1)$ time, respectively.

Moreover, the first and last element in $\mathcal{E}(B)$ can be computed in $O(1)$ time, respectively.

Proof: In the following, based on cases in Section 6.1, we first present the formulas for $\varepsilon^i_{\mathrm{fst}}(B)$ and $\varepsilon^i_{\mathrm{lst}}(B)$ for $i = 1, 2, \ldots, 5$, and then show how to calculate them, respectively.

Let $t(B)$ be the tip of the block B, which can be calculated by Lemma 8.4. Let x_2 and y_2 be the second left and right core-vertex of B, respectively. Let $u^{\mathtt{L}}$ (resp., $u^{\mathtt{R}}$) be the first vertex in $V^{\mathtt{L}}_{\mathtt{wing}}(B)$ (resp., $V^{\mathtt{R}}_{\mathtt{wing}}(B)$) if $V^{\mathtt{L}}_{\mathtt{wing}}(B) \neq \emptyset$ (resp.,$V^{\mathtt{R}}_{\mathtt{wing}}(B) \neq \emptyset$), or $u^{\mathtt{L}} = \ell v(B'_1)$ (resp., $u^{\mathtt{R}} = \ell v(B''_1)$) for the leftmost block $B'_1 \in \mathcal{B}(u')$ (resp., $B''_1 \in \mathcal{B}(u'')$) of the first vertex $u' \in V^{\mathtt{L}}_{\mathtt{cut}}(B)$ (resp., $u'' \in V^{\mathtt{R}}_{\mathtt{cut}}(B)$) if $V^{\mathtt{L}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{L}}_{\mathtt{cut}}(B) = \emptyset$ (resp., $V^{\mathtt{R}}_{\mathtt{wing}}(B) = \emptyset$ and $V^{\mathtt{R}}_{\mathtt{cut}}(B) = \emptyset$).

1. The first vertex $\varepsilon^1_{\mathrm{fst}}(B)$ and the last vertex $\varepsilon^1_{\mathrm{lst}}(B)$ for $\mathcal{E}(B) \cap X_1 = V_{\mathtt{R}}(G(B))$ are given by

$$\varepsilon^1_{\mathrm{fst}}(B) = \begin{cases} t(B) & \text{for Case-1} \\ y_1 & \text{for Case-2-4 or Case-5(c)} \\ \emptyset & \text{for Case-5(a)-(b),} \end{cases} \tag{8.3}$$

$$\varepsilon^1_{\mathrm{lst}}(B) = \begin{cases} y_{p_{\mathtt{R}}} & \text{for Case-1(a)-(b), Case-2(a)-(c), Case-3(a)-(c), Case-4 or Case-5(c)} \\ t(B) & \text{for Case-1(c)} \\ y_j & \text{for Case-1(d)} \\ y_h & \text{for Case-2(d)} \\ y_j & \text{for Case-3(d)} \\ \emptyset & \text{for Case-5(a)-(b).} \end{cases} \tag{8.4}$$

Clearly, $\varepsilon^1_{\mathrm{fst}}(B)$ can be computed in $O(1)$ time. Now we show how to compute $\varepsilon^1_{\mathrm{lst}}(B)$ in the above cases, respectively. For Case-1(a)-(b), we have $\varepsilon^1_{\mathrm{lst}}(B) = y_{p_{\mathtt{R}}}$ is the vertex with $\mathrm{idx}(y_{p_{\mathtt{R}}}) = \mathrm{idx}(u^{\mathtt{R}}) - 1$.

For Case-2(a)-(c) (Figure 6.2(a)-(c)), we have $\varepsilon^1_{\mathrm{lst}}(B) = y_{p_{\mathtt{R}}}$ is the last vertex in $V^{\mathtt{R}}_{\mathtt{wing}}(B)$.

For Case-3(a)-(c) (Figure 6.3(a)-(c)), Case-4 (Figure 6.4) or Case-5 (Figures 6.5-6.6), we have $\varepsilon^1_{\rm lst}(B) = y_{p_{\tt R}}$ is the last right core-vertex of B.

For Case-1(d) (Figure 6.1(d)), we have $\varepsilon^1_{\rm lst}(B) = y_j$, where $x_i = \mu(t(B); r(B))$, which can be calculated by Equation (8.2) in Lemma 8.7, x_j is the next vertex with the same vertex type as x_i, which can be calculated by type(x_i) and data(B), and $y_j = \mu(x_j; r(B))$ can be calculated by Equation (8.2) in Lemma 8.7.

For Case-2(d) (Figure 6.2(d)), we have $\varepsilon^1_{\rm lst}(B) = y_h$, where x_h is the first vertex in $V^{\tt L}_{\tt cut}(B)$, and $y_h = \mu(x_h; r(B))$, which can be calculated by Equation (8.2) in Lemma 8.7.

For Case-3(d) (Figure 6.3(d)), we have $\varepsilon^1_{\rm lst}(B) = y_j$, where x_j is the first vertex in $V^{\tt L}_{\tt cut}(B)$, and $y_j = \mu(x_j; r(B))$, which can be calculated by Equation (8.2) in Lemma 8.7.

2. The first edge $\varepsilon^2_{\rm fst}(B)$ and the last edge $\varepsilon^2_{\rm lst}(B)$ for $\mathcal{E}(B) \cap X_2 = E_{\tt R}(y_1; B)$ are given by

$$\varepsilon^2_{\rm fst}(B) = \begin{cases} e'_1 & \text{for Case-2(a)-(c), Case-3(a)-(c), Case-4 or Case-5(c)} \\ \emptyset & \text{for Case-1, Case-2(d), Case-3(d) or Case-5(a)-(b),} \end{cases} \tag{8.5}$$

$$\varepsilon^2_{\rm lst}(B) = \begin{cases} e'_{q'+1} & \text{for Case-2(a);} \\ e'_{q'} = (y', y'') & \text{for Case-2(b)-(c), Case-3(c) or Case-4(b)} \\ e'_{q'} = (y_1, y_2) & \text{for Case-3(a)-(b), Case-4(a) or Case-5(c)} \\ \emptyset & \text{for Case-1, Case-2(d), Case-3(d) or Case-5(a)-(b).} \end{cases} \tag{8.6}$$

Now we show how to calculate the edges $\varepsilon^2_{\rm fst}(B)$ and $\varepsilon^2_{\rm lst}(B)$, respectively. For Case-2(a)-(c) (Figure 6.2(a)-(c)), Case-3(a)-(c) (Figure 6.3(a)-(c)), Case-4 (Figure 6.4) or Case-5(c) (Figures 6.5(c)-6.6(c)), the edge $\varepsilon^2_{\rm fst}(B)$ is the right edge $e'_1 = (u, v)$ of B directed from u to v, which can be calculated by data(B) and pre(v). Specifically, the bottom vertex $bv(B)$ of the block B can be calculated by Lemma 8.1. If $bv(B) \neq \emptyset$, then the head v of the edge e'_1 is $bv(B)$, and the tail u of the edge e'_1 is equal to the second entry of pre(v) if the last entry of pre(v) is not $\emptyset$, and the tail u of the edge e'_1 is the last right core-vertex of B otherwise. For example, in Figure 7.1(a), we have pre$(bv(B)) = [x_6, y_5]$, and hence $e'_1 = (y_5, bv(B))$; and in Figure 7.1(b), we have pre$(bv(B)) = [x_6, \emptyset]$, and hence $e'_1 = (y_7, bv(B))$, where y_7 is the last right core-vertex of B. If $bv(B) = \emptyset$, then the head v of the edge e'_1 is the last right core-vertex of B, and the tail u of the edge e'_1 is the last second of pre(v). For example in Figure 7.1(c), x_7 and y_7 are the last left and right core-vertex of B, respectively. We have pre$(y_7) = [y_6, y_6]$, and hence $e'_1 = [y_6, y_7]$.

Besides, for Case-2(a) (Figure 6.2(a)), the edge $\varepsilon^2_{\rm lst}(B)$ is the edge $e'_{q'+1} = (y^*, t(B))$, where y^* is the first entry in pre$(t(B))$. For Case-2(b)-(c) (Figure 6.2(b)-(c)), Case-3(c) (Figure 6.3(c)) or Case-4(b) (Figure 6.4(b)), we have $\varepsilon^2_{\rm lst}(B) = e'_{q'}$, where $\hat{e} = (x', x'')$ is the edge such that x' is the first entry in pre$(x_{p_{\tt R}+1})$ and $x'' = $ wgedge$(x_{p_{\tt R}+1})$; and $e'_{q'} = (y', y'')$ is the corresponding edge of $\hat{e}$, i.e., $y' = \mu(x'; r(B))$ and $y'' = \mu(x''; r(B))$, both of which can be calculated by Equation (8.2) in Lemma 8.7.

3. The first vertex $\varepsilon^3_{\text{fst}}(B)$ and the last vertex $\varepsilon^3_{\text{lst}}(B)$ for $\mathcal{E}(B) \cap X_3 = V_{\mathtt{L}}(G(B))$ are given by

$$\varepsilon^3_{\text{fst}}(B) = \begin{cases} t(B) & \text{for Case-3} \\ x_1 & \text{for Case-4 or Case-5(b)-(c)} \\ \emptyset & \text{for Case-1, Case-2 or Case-5(a),} \end{cases} \tag{8.7}$$

$$\varepsilon^3_{\text{lst}}(B) = \begin{cases} x_{p_{\mathtt{L}}} & \text{for Case-3, Case-4 or Case-5(b)-(c)} \\ \emptyset & \text{for Case-1, Case-2 or Case-5(a).} \end{cases} \tag{8.8}$$

Clearly, the first vertex $\varepsilon^3_{\text{fst}}(B)$ of $\mathcal{E}(B)$ can be computed in $O(1)$ time. It remains to show the calculation of $\varepsilon^3_{\text{lst}}(B) = x_{p_{\mathtt{L}}}$ for Case-3 (Figure 6.3), Case-4 (Fig. 6.4) or Case-5(b)-(c) (Figures 6.5(b)-(c)-6.6(b)-(c)). For Case-3 or Case-4, we have $x_{p_{\mathtt{L}}}$ is the vertex with $\text{idx}(x_{p_{\mathtt{L}}}) = \text{idx}(u^{\mathtt{L}}) - 1$, and for Case-5(b)-(c), we have $x_{p_{\mathtt{L}}}$ is the last vertex in $V^{\mathtt{L}}_{\mathtt{head}}(B) \cup V^{\mathtt{L}}_{\mathtt{axis}}(B)$.

4. The first edge $\varepsilon^4_{\text{fst}}(B)$ and the last edge $\varepsilon^4_{\text{lst}}(B)$ for $\mathcal{E}(B) \cap X_4 = E_{\mathtt{L}}(x_1; B)$ are given by

$$\varepsilon^4_{\text{fst}}(B) = \begin{cases} e_1 & \text{for Case-4 or Case-5(b)-(c)} \\ \emptyset & \text{for Case-1, Case-2, Case-3 or Case-5(a),} \end{cases} \tag{8.9}$$

$$\varepsilon^4_{\text{lst}}(B) = \begin{cases} e_{q+1} & \text{for Case-4} \\ e_q = (x_1, x_2) & \text{for Case-5(b)-(c)} \\ \emptyset & \text{for Case-1, Case-2, Case-3 or Case-5(a).} \end{cases} \tag{8.10}$$

We calculate the edges $\varepsilon^4_{\text{fst}}(B)$ and $\varepsilon^4_{\text{lst}}(B)$ as follows. For Case-4 (Figure 6.4) or Case-5(b)-(c) (Figures 6.5(b)-(c)-6.6(b)-(c)), the edge $\varepsilon^4_{\text{fst}}(B)$ is the left edge $e_1 = (u, v)$ of B directed from u to v, which can be calculated by data(B) and data(v). Specifically, if $bv(B) \neq \emptyset$, then the head v of the edge e_1 is the bottom vertex $bv(B)$, and the tail u of the edge e_1 is the last left core-vertex of B if the first entry of pre(v) is $\emptyset$, and u is equal to the first entry of pre(v) otherwise. For example in Figure 7.1(a), we have pre$(bv(B)) = [x_6, y_5]$, and hence $e_1 = [x_6, bv(B)]$. If $bv(B) = \emptyset$, then the head v of the edge e_1 is the last left core-vertex of B, and the tail u of the edge e_1 is the last entry of pre(v). In Figure 7.1(c), the vertex x_7 is the last left core-vertex of B with pre$(x_7) = [x_6, x_6]$. Hence we have $e_1 = [x_6, x_7]$.

Besides, for Case-4 (Figure 6.4), we have $e_{q+1} = (u, x_{p_{\mathtt{L}}})$, where $x_{p_{\mathtt{L}}}$ is the vertex with $\text{idx}(x_{p_{\mathtt{L}}}) = |V(G)|$, and u is the first entry in pre$(x_{p_{\mathtt{L}}})$. For example in Figure 6.4 (a), suppose that pre$(x_{p_{\mathtt{L}}}) = [x^*, x^*]$. Then we have $e_{q+1} = (x^*, x_{p_{\mathtt{L}}})$.

5. The first element $\varepsilon^5_{\text{fst}}(B)$ and the last element $\varepsilon^5_{\text{lst}}(B)$ for $\mathcal{E}(B) \cup X_5 = \{e^b, bv(B)\}$ are

given by

$$\varepsilon^5_{\rm fst}(B) \;=\; \begin{cases} bv(B) & \text{if Case-5 occurs and } bv(B) \neq \emptyset; \\ \emptyset & \text{if Case-1-4 occurs, or, Case-5 occurs and } bv(B) = \emptyset. \end{cases} \tag{8.11}$$

$$\varepsilon^5_{\rm lst}(B) \;=\; \begin{cases} e^b & \text{if Case-5 occurs and } \mathcal{B}(bv(B)) = \emptyset; \\ \emptyset & \text{if Case-1-4 occurs, or, Case-5 occurs and } \mathcal{B}(bv(B)) \neq \emptyset. \end{cases} \tag{8.12}$$

By Lemma 8.1, we can calculate the bottom vertex $bv(B)$ in $O(1)$ time. Besides, let x^* (resp., y^*) be the last left (resp., right) core-vertex of B. By definition, $e^b = (y^*, bv(B))$ if $bv(B) \neq \emptyset$, and $e^b = (x^*, y^*)$ otherwise. Thus, we can calculate the edge e^b in $O(1)$ time. Based on these, we can calculate both $\varepsilon^5_{\rm fst}(B)$ and $\varepsilon^5_{\rm fst}(B)$ in $O(1)$ time.

Moreover, we can compute the first and last element in $\mathcal{E}(B)$ in O(1) time. Specifically, ε is the first (resp., last) element in $\mathcal{E}(B)$ if and only if ε is the first non-empty element in the sequence $\varepsilon^1_{\rm fst}(B)$, $\varepsilon^1_{\rm lst}(B)$, $\varepsilon^2_{\rm fst}(B)$, $\varepsilon^2_{\rm lst}(B)$, $\varepsilon^3_{\rm fst}(B)$, $\varepsilon^3_{\rm lst}(B)$, $\varepsilon^4_{\rm fst}(B)$, $\varepsilon^4_{\rm lst}(B)$, $\varepsilon^5_{\rm fst}(B)$ and $\varepsilon^5_{\rm lst}(B)$ (resp., the sequence $\varepsilon^5_{\rm lst}(B)$, $\varepsilon^5_{\rm fst}(B)$, $\varepsilon^4_{\rm lst}(B)$, $\varepsilon^4_{\rm fst}(B)$, $\varepsilon^3_{\rm lst}(B)$, $\varepsilon^3_{\rm fst}(B)$, $\varepsilon^2_{\rm lst}(B)$, $\varepsilon^2_{\rm fst}(B)$, $\varepsilon^1_{\rm lst}(B)$ and $\varepsilon^1_{\rm fst}(B)$). □

Lemma 8.10. *Given a block B in the spine of G and an element $\varepsilon \in \mathcal{E}(B)$, let $\gamma_{\rm s}$ and $\gamma_{\rm l}$ be the smallest and the largest code in $\Gamma(\varepsilon)$ with respect to lexicographical order, respectively. Then both $\gamma_{\rm s}$ and $\gamma_{\rm l}$ can be computed in $O(1)$ time.*

Proof: In the following, we will calculate these two codes $\gamma_{\rm s}$ and $\gamma_{\rm l}$ in $\Gamma(\varepsilon)$ based on cases in the definition of code set in Section 6.1.

(1)-1. For $\varepsilon = v \in V'(B) - V_{\tt cut}(B)$ such that v is not y_j in Case-1(d) (Figure 6.1(d)) and 3(d) (Figure 6.3(d)), we have $\gamma_{\rm s} = (d(B), {\tt at}, d(v), {\tt new\text{-}block}, c_1)$ and $\gamma_{\rm l} = (d(B), {\tt at}, d(v), {\tt new\text{-}block}, c_K)$. Note that all entries excluding the second entry ${\tt at}$ of $\gamma_{\rm s}$ and $\gamma_{\rm l}$ can be known by the maintained data, but ${\tt at}$ can be calculated by type(ε) in $O(1)$ time. In this case, both $\gamma_{\rm s}$ and $\gamma_{\rm l}$ can be computed in $O(1)$ time.

(1)-2. For $\varepsilon = y_i$ in Case-1(c) (Figure 6.1(c)), let $h = |\text{blocks}(y_i)|$, x_i be the corresponding vertex of y_i with respect to the block B, which can be calculated by Equation (8.2) in Lemma 8.7, and B'_{h+1} be the $(h+1)$st block in the array blocks(x_i). We have $\gamma_{\rm s} = (d(B), {\tt h}^{\tt R}, d(y_i), {\tt new\text{-}block}, c_1)$ and $\gamma_{\rm l} = (d(B), {\tt h}^{\tt R}, d(y_i), {\tt new\text{-}block}, c(\ell v(B'_{h+1}))$.

(1)-3. For $\varepsilon = y_j$ in Case-1(d) (Figure 6.1(d)) or Case-3(d) (Figure 6.3(d)), let x_j be the corresponding vertex of y_j with respect to the block B, which can be calculated by Equation (8.2) in Lemma 8.7, and let B''_1 be the first block in the array blocks(x_j). We have $\gamma_{\rm s} = (d(B), {\tt h}^{\tt R}, d(y_j), {\tt new\text{-}block}, c_1)$ and $\gamma_{\rm l} = (d(B), {\tt h}^{\tt R}, d(y_j), {\tt new\text{-}block}, c(\ell v(B''_1))$.

(1)-4. For $\varepsilon \in V_{\mathtt{cut}}(B)$ not in Case-1(c), let B'_1 denote the last block in the array blocks(ε) in G. We have $\gamma_{\mathrm{s}} = (d(B), \mathtt{at}, d(\varepsilon), \mathtt{new\text{-}block}, c_1)$ and $\gamma_{\mathrm{l}} = (d(B), \mathtt{at}, d(\varepsilon), \mathtt{new\text{-}block}, c(\ell v(B'_1)))$, where $\mathtt{at}$ is calculated by type(ε) in $O(1)$ time.

(2)-(i) For $\varepsilon = e'_{q'}$ in Figure 6.2(b)-(c), Figure 6.3(c) and Figure 6.4(b), let $x_{p_{\mathtt{R}}+1}$ be the $(|V^{\mathtt{R}}_{\mathtt{wing}}(B)|+1)$st left wing-vertex of B. Note that $e'_{q'}$ is the corresponding edge of the edge $\hat{e}$, to which is applied the code $\gamma(x_{p_{\mathtt{R}}+1})$ to generate $x_{p_{\mathtt{R}}+1}$. We have $\gamma_{\mathrm{s}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{triangle}, c_1)$, and $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{triangle}, c(x_{p_{\mathtt{R}}+1}))$ if $\mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{triangle}$, or $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'_{q'}), \mathtt{subdivide}, c(x_{p_{\mathtt{R}}+1}))$ if $\mathtt{op}(x_{p_{\mathtt{R}}+1}) = \mathtt{subdivide}$. Note that all entries excluding the third entry $d(e'_{q'})$ of γ_{s} and γ_{l} can be known by the maintained data, and that by Lemma 8.3, $d(e'_{q'})$ can be calculated in $O(1)$ time.

(2)-(ii) For $\varepsilon = e'$ that is not the edge $e'_{q'}$ in Case-2(b)-(c) (Figure 6.2(b)-(c)), Case-3(c) (Figure 6.3(c)) and Case-4(b) (Figure 6.4(b)), recall that $A(B)$ is the set of axial-edges in B. We have $\gamma_{\mathrm{s}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'), \mathtt{triangle}, c_1)$, and $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'), \mathtt{triangle}, c_K)$ if $e' \in A(B) \cup \{e'_{q'+1}\}$, or $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{R}}, d(e'), \mathtt{subdivide}, c_K)$ if $e' \notin A(B) \cup \{e'_{q'+1}\}$, where by Lemma 8.2, we can check whether $e' \in A(B)$ in $O(1)$ time.

(3) For a left edge $\varepsilon = e_j \in \mathcal{E}(B)$, we have $\gamma_{\mathrm{s}} = (d(B), \mathtt{w}^{\mathtt{L}}, d(e_j), \mathtt{triangle}, c_1)$, and $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{L}}, d(e_j), \mathtt{triangle}, c_K)$ if $e_j \in A(B) \cup \{e_{q+1}\}$, or $\gamma_{\mathrm{l}} = (d(B), \mathtt{w}^{\mathtt{L}}, d(e_j), \mathtt{subdivide}, c_K)$ if $e_j \notin A(B) \cup \{e_{q+1}\}$, where by Lemma 8.2, we can check whether $e_j \in A(B)$ in $O(1)$ time.

(4)-(a) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has no axial-vertex and $|V(B)|$ is even, if $\mathtt{sd}(B; G) = \mathtt{stc}$ (Figure 6.6(c)), then $\gamma_{\mathrm{s}} = (d(B), *, d(z), \mathtt{star}, c_1)$ and $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{star}, c_K)$; if $\mathtt{sd}(B; G) = \mathtt{nil}$ and $|V(B)| = 2$ (Figure 6.5(f)), then $\gamma_{\mathrm{s}} = (d(B), *, d(z), \mathtt{triangle}, c_1)$ and $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{triangle}, c(z))$; if $\mathtt{sd}(B; G) = \mathtt{nil}$ and $|V(B)| \geq 3$ (Figure 6.5(c)), then $\gamma_{\mathrm{s}} = (d(B), *, d(z), \mathtt{star}, c_1)$ and $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{star}, c(z))$.

(4)-(b) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has no axial-vertex and $|V(B)|$ is odd, we have $\gamma_{\mathrm{s}} = (d(B), *, d(z), \mathtt{star}, c_1)$, and $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{subdivide}, c_K)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$ (Figure 6.6(a)), or $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{subdivide}, c_K)$ if $\mathtt{sd}(B; G) = \mathtt{nil}$ (Figure 6.5(a)).

(4)-(c) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has axial-vertices and $|V(B)|$ is even, $\gamma_{\mathrm{s}} = (d(B), *, d(z), \mathtt{subdivide}, c_1)$, and $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{subdivide}, c_K)$ if $\mathtt{sd}(B; G) = \mathtt{stc}$ (Figure 6.6(d)), or $\gamma_{\mathrm{l}} = (d(B), *, d(z), \mathtt{subdivide}, c(z))$ if $\mathtt{sd}(B; G) = \mathtt{nil}$, $V^{L}_{\mathtt{axis}}(B) \cup V^{R}_{\mathtt{axis}}(B) \neq \emptyset$, and $|V(B)|$ is even (Figure 6.5(d)).

(4)-(d) For $\varepsilon = e^b = (y, z)$ with $d(z) > d(y)$ in the case that B has axial-vertices and $|V(B)|$ is odd (Figures 6.5(b)-6.6(b)), we have $\gamma_{\rm s} = (d(B), *, d(z), \texttt{subdivide}, c_1)$ and $\gamma_{\rm l} =$
$(d(B), *, d(z),$
$\texttt{subdivide}, c_K)$.

By the above analysis, given a block $B \in \mathcal{E}^*(G)$ and an element $\varepsilon \in \mathcal{E}(B)$, we can compute the smallest code $\gamma_{\rm s}$ and the largest code $\gamma_{\rm l}$ in $\Gamma(\varepsilon)$ in $O(1)$ time, respectively. $\Box$

Bibliography

[1] M. A. Adnan. Efficient enumeration of combinatorial objects. Bachelor thesis. Bangladesh University of Engineering and Technology. 2006.

[2] E. A. Akkoyunlu. The enumeration of maximal cliques of large graphs. *SIAM Journal on Computing*, vol. 2, no. 1, pp. 1-6, 1973.

[3] T. Akutsu and D. Fukagawa. Inferring a graph from path frequency. In *Proceedings of 16th Annual Symposium on Combinatorial Pattern Matching*, *Lecture Notes in Computer Science*, vol. 3537, pp. 371-382, 2005.

[4] T. Akutsu and D. Fukagawa. Inferring a chemical structure from a feature vector based on frequency of labeled paths and small fragments. In Series on Advances in Bioinformatics and Computational Biology, *Proceedings of 5th Asia-Pacific Bioinformatics Conference*, Hong Kong, China, January 15-17, 2007; D. Sankoff, L. Wang, F. Chin, Eds.; Imperial College Press, pp. 165-174, 2007.

[5] I. Alonso, J. I. Rémy, and R. Sehott. A linear-time algorithm for the generation of trees. *Algorithmica*, vol. 17, no. 2, pp. 162-182, 1997.

[6] G. E. Andrews. q-Series: Their development and application in analysis, number theory, combinatorics, physics, and computer algebra. *CBMS Regional Conference Series in Mathematics*, No. 66, American Mathematical Society, Providence, 1986.

[7] D. J. Amalraj, N. Sundararajan, and G. Dhar. A data structure based on gray code encoding for graphics and image processing. In *Proceedings of the SPIE: International Society for Optical Engineering*, pp. 65-76, 1990.

[8] H. Arimura and T. Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. *SIAM International Conference on Data Mining*, pp. 1087-1098, 2009.

[9] R. Aringhieri, P. Hansen, and F. Malucelli. Chemical trees enumeration algorithms. *4OR*, vol. 1, pp. 67-83, 2003.

[10] D. B. Arnold and M. R. Sleep. Uniform random generation of balanced parenthesis strings. *ACM Transactions on Programming Languages and Systems*, vol. 2, no. 1, pp. 122-128, 1980.

[11] M. D. Atkinson. Uniform generation of rooted ordered trees with prescribed degrees. *Computer Journal*, vol. 36, pp. 593-594, 1993.

[12] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, vol. 6, pp. 21-46, 1996.

[13] D. Avis. Generating rooted triangulations without repetitions. *Algorithmica*, vol. 16, pp. 618-632, 1996.

[14] S. Bacchelli, E. Barcucci, E. Grazzini, and E. Pergola. Exhaustive generation of combinatorial objects by ECO. *Acta Informatica*, vol. 40, pp. 585-602, 2004.

[15] K. Balasubramanian. Exhaustive generation and analytical expressions of matching polynomials of fullerenes C_{20}-C_{50}. *Journal of Chemical Information and Computer Science*, vol. 34, pp. 421-427, 1994.

[16] E. Balas. Machine sequence via disjunctive graphs: an implicit enumeration algorithm. *Operation research*, vol. 17, pp. 941-957, 1968.

[17] E. Barcucci, A. Del Lungo, and E. Pergola. Random generation of trees and other combinatorial objects. *Theoretical Computer Science*, vol. 218, no. 2, pp. 219-232, 1999.

[18] B. Bauslaugh and F. Ruskey. Generating alternating permutations lexicographically. *BIT*, vol. 30, no. 1, pp. 17-26, 1990.

[19] S. Bereg. Enumerating pseudo-triangulations in the plane. *Computational Geometry: Theory and Applications*, vol. 30, no. 3, pp. 207-222, 2005.

[20] E. A. Bender, Z. Gao, and N. C. Wormald. The number of labeled 2-connected planar graphs. *The Electronic Journal of Combinatorics*, vol. 9, R43, 2002.

[21] T. Beyer and S. M. Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, vol. 9, pp. 706-712, 1980.

[22] J. R. Bitner, G. Ehrlich, and E. M. Reingold. Efficient generation of the binary reflected Gray code and and its applications. *Communications of the Association for Computing Machinery*, vol. 19, pp. 517-521, 1976.

[23] O. Bodini and A. Jacquot. Boltzmann samplers for colored combinatorial objects. *CoRR* abs/0911.2801: 2009.

[24] M. Bodirsky, E. Fusy, M. Kang, and S. Vigerske. Enumeration and asymptotic properties of unlabeled outerplanar graphs. *The Electronic Journal of Combinatorics*, vol. 14, no. 1, R66, 2007.

[25] M. Bodirsky, O. Giménez, M. Kang, and M. Noy. On the number of series parallel and outerplanar graphs. In *Proceedings of 2005 European Conference on Combinatorics, Graph Theory and Applications* (EuroComb '05); Stefan Felsner Ed.; *Discrete Mathematics and Theoretical Computer Science*, vol. AE, pp. 383-388, 2005.

[26] M. Bodirsky, C. Gröpl, and M. Kang. Sampling unlabeled biconnected planar graphs. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation* (ISAAC05), *Lecture Notes in Computer Science*, vol. 3827, pp. 593-603, 2005.

[27] M. Bodirsky, C. Gröpl, and M. Kang. Generating labeled planar graphs uniformly at random. *Theoretical Computer Science*, vol. 379, no. 3, pp. 377-386, 2007.

[28] M. Bodirsky, C. Gröpl, and M. Kang. Generating unlabeled connected cubic planar graphs uniformly at random. *Random Structures and Algorithms*, vol. 34, pp. 157-180, 2008.

[29] N. Bonichon, C. Gavoille, and N. Hanusse. Canonical decomposition of outerplanar maps and application to enumeration, coding and generation. *Journal of Graph Algorithms and Applications*, vol. 9, no. 2, pp. 185-204, 2005.

[30] M. Bodirsky and M. Kang. Generating outerplanar graphs uniformly at random. *Combinatorics, Probability and Computing*, vol. 15, pp. 333-343, 2006.

[31] I. Bomze, M. Budinich, P. Pardalos, and M. Pelillo. The maximum clique problem. *Handbook of Combinatorial Optimization*, D. Du and P. Pardalos Eds., vol. 4, pp. 1-74, 1999.

[32] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan. Algorithms for enumerating circuits in matroids. The 14th Annual International Symposium on Algorithms and Computation (ISAAC), December 15-17, 2003, Kyoto; *Lecture Notes in Computer Science*, vol. 2906, pp. 485-494, 2003.

[33] G. Brinkmann. G. Caporossi, and P. Hansen. A constructive enumeration of fusenes and benzenoids. *Journal of Algorithms*, vol. 45, pp. 155-166, 2002.

[34] A. Broder. Generating random spanning trees. In *Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer Science*, pp. 442-447, 1989.

[35] M. Buck and D. Wiedemann. Gray codes with restricted density. *Discrete Mathematics*, vol. 48, pp. 163-171, 1984.

[36] F.C. Bussemaker, S. Cobeljic, D. M. Cvetkovic, and J. J. Seidel. Computer investigation of cubic graphs. T.H.-Report 76-WSK-01, Technological University, Eindhoven, Department of Mathematics, 1976.

[37] J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, vol. 32, no. 6, pp. 547-556, 2004.

[38] L. Bytautas and D. J. Klein. Chemical combinatorics for alkane-isomer enumeration and more. *Journal of Chemical Information and Computer Science*, vol. 38, pp. 1063-1078, 1998.

[39] A. Cayley. A theorem on trees. *The Quarterly Journal of Mathematics*, vol. 23, pp. 376-378, 1889.

[40] G. Caporossi, P. Hansen, and M. Zheng. Enumeration of Fusenes to $h = 20$. *Discrete Mathematical Chemistry*, pp. 63-78, 2000.

[41] C. J. Colbourn and R. C. Read. Orderly algorithms for graph generation. *International Journal of Computer Mathematics. Section A: Computer Systems: Theory*, vol. 7, 167-172, 1979.

[42] R. L. Cummings. Hamilton circuits in tree graphs. *IEEE Transactions on Circuit Theory*, vol. 13, pp.82-90, 1966.

[43] M.L. Contreras, R. Rozas, and R. Valdivia. Exhaustive generation of organic isomers. 3. acyclic, cyclic, and mixed compounds. *Journal of Chemical Information and Computer Science*, vol. 34, pp. 610-616, 1994.

[44] A. Denise and P. Zimmermann. Uniform random generation of decomposable structures using floating-point arithmetic. *Theoretical Computer Science*, vol. 218, no. 2, pp. 219-232, 1999.

[45] P. Diaconis and S. Holmes. Gray codes for randomization procedures. *Statistics and Computing*, vol. 4, pp. 287-302, 1994.

[46] R. Diestel. Graph Theory. Third Edition, Springer-Verlag, Heidelberg, 2005.

[47] B. Djokic, M. Miyakawa, S. Sekiguchi, I. Semba, and I. Stojmenovic. A fast iterative algorithm for generating set partitions. *The Computer Journal*, vol. 32, pp. 281-282, 1989.

[48] N. Du, B. Wu, X. Pei, B. Wang, and L. Xu. Community detection in large-scale social networks. In *Proceedings of WebKDD/SNAKDD 2007: KDD Workshop on Web Mining and Social Network Analysis, in conjunction with the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD 2007), California, pp. 16-26, 2007.

[49] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Random sampling from Boltzmann principles. In *Proceedings of the 29th International Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science*, vol. 2380, pp. 501-513, 2002.

[50] P. Duchon, P. Flajolet, G. Louchard, and G. Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, vol. 13, no. 4-5, pp. 577-625, 2004.

[51] P. Eades, M. Hickey, and R. C. Read. Some Hamilton paths and a minimal change algorithm. *Journal of the ACM*, vol. 31, no. 1, pp. 19-29, 1984.

[52] P. Eades and B. McKay. An algorithm for generating subsets of fixed size with a strong minimal change property. *Information Processing Letters*, vol. 19, pp. 131-133, 1984.

[53] M. C. Er. A fast algorithm for generating set partitions. *The Computer Journal*, vol. 31, pp. 283-284, 1988.

[54] P. Flajolet, E. Fusy, and C. Pivoteau. Boltzmann sampling of unlabelled structures. In *Proceedings of the 4th Workshop on Analytic Algorithms and Combinatorics* (ANALCO'07), New Orleans, January 2007, SIAM Press, pp. 201-211, 2007.

[55] P. Flajolet, P. Zimmermann, and B. V. Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, vol. 132, no. 2, pp. 1-35, 1994.

[56] S. Fujita. Graphs to chemical structures 1. Sphericity indices of cycles for stereochemical extension of Pólya's theorem. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling*, vol. 113, no. 2, pp. 73-79, 2005.

[57] S. Fujita. Graphs to chemical structures 2. Extended sphericity indices of cycles for stereochemical extension of Pólya's coronas. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling*, vol. 113, no. 2, pp. 80-86, 2005.

[58] S. Fujita. Graphs to chemical structures 3. General theorems with the use of different sets of sphericity indices for combinatorial enumeration of nonrigid stereoisomers. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling*, vol. 115, no. 1, pp. 37-53, 2006.

[59] S. Fujita. Graphs to chemical structures 5. Combinatorial enumeration of centroidal and bicentroidal three-dimensional trees as stereochemical models of alkanes. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling*, vol. 117, no. 3, pp. 339-351, 2007.

[60] H. Fujiwara, J. Wang, L. Zhao, H. Nagamochi, and T. Akutsu. Enumerating treelike chemical graphs with given path frequency. *Journal of Chemical Information and Modeling*, vol. 48, pp. 1345-1357, 2008.

[61] G. W. Furnas. The generation of random, binary unordered trees. *Journal of Classification*, vol. 1, pp. 187-233, 1984.

[62] H. N. Gabow and E. W. Myers. Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, vol. 7, pp. 280-287, 1978.

[63] Z. C. Gao and N. C. Wormald. Enumeration of rooted cubic planar maps. *Annals of Combinatorics*, vol. 6, pp. 313-325, 2002.

[64] E. Georgii, S. Dietmann, T. Uno, P. Pagel, and K. Tsuda. Enumeration of condition-dependent dense modules in protein interaction networks. *Bioinformatics*, vol. 25, no. 7, pp. 933-940, 2009.

[65] O. Giménez and M. Noy. Asymptotic enumeration and limit laws of planar graphs. arXiv:math.CO/0501269, 2005, available online at http://arxiv.org/abs/math/0501269.

[66] F. Gray. Pulse code communication, United States Patent Number 2632058, March 17, 1953.

[67] L. A. Goldberg. Efficient algorithms for listing combinatorial structures. Ph.D thesis. Cambridge University Press, New York, 1993.

[68] L. A. Goldberg. Polynomial space polynomial delay algorithms for listing families of graphs. In *proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, San Diego, California, United States, pp. 218-225, 1993.

[69] M. Goldwurm. Random generation of words in an algebraic language in linear binary space. *Information Processing Letters*, vol. 54, no. 4, pp. 229-233, 1995.

[70] F. Harary and E. M. Palmer. Graphical Enumeration. Academic Press, 1973.

[71] M. D. Hirschhorn. Another short proof of Ramanujan's Mod 5 partition congruences, and more. *The American Mathematical Monthly*, vol. 106, pp. 580-583, 1999.

[72] C. A. Holzmann and F. Haray. On the tree graph of a matroid. *SIAM Journal on Applied Mathematics*, vol. 22, pp. 187-193, 1972.

[73] T. Horváth, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, PA, USA, pp. 197-206, 2006.

[74] H. Imai, T. Masada, and F. Takeuchi. Enumerating triangulations in general dimensions. *International Journal of Computational Geometry & Applications*, vol. 12, no. 6, pp. 455-480, 2002.

[75] Y. Ishida, L. Zhao, H. Nagamochi, and T. Akutsu. Improved algorithms for enumerating tree-like chemical graphs with given path frequency. *Genome Informatics*, vol. 21, pp. 53-64, 2008.

[76] F. M. Ives. Permutation enumeration: Four new permutation algorithms. *Communications of the ACM*, vol. 19, 68-72, 1976.

[77] M. Jerrum. Random generation of combinatorial structures from a uniform distribution. *Lecture Notes in Computer Science*, vol. 194, pp. 290-299, 1985.

[78] S. M. Johnson. Generation of permutations by adjacent transpositions. *Mathematics of Computation*, vol. 17, pp. 282-285, 1963.

[79] D. S. Johnson, M. Yanakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, vol. 27, pp. 119-123, 1988.

[80] J. T. Joichi, D. E. White, and S. G. Williamson. Combinatorial Gray codes. *SIAM Journal on Computing*, vol. 9, no. 1, pp. 130-141, 1980.

[81] R. Kannan, P. Tetali, and S. Vempala. Simple Markov chain algorithms for generating bipartite graphs and tournaments. In *Proceedings of the eighth annual ACM Symposium on Discrete Algorithms*, New Orleans, Louisiana, United States, pp. 193-200, 1997.

[82] S. Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM Journal on Computing*, vol. 24, pp. 247-265, 1995.

[83] T. Kashiwabara and S. Masuda. Generation of maximum independent sets of a bipartite graph and maximum cliques of a circular-arc graph. *Journal of algorithms*, vol. 13, pp. 161-174, 1992.

[84] S. Kawano and S. Nakano. Constant time generation of set partition, *IEICE Transactions*, vol. 88-A, no. 4, pp. 930-934, 2005.

[85] S. Kawano and S. Nakano. Generating all series-parallel graphs. *IEICE Transactions*, vol. 88-A, no. 5, pp. 1129-1135, 2005.

[86] R. A. Kaye. A Gray code for set partitions. *Information Processing Letters*, vol. 5, pp. 171-173, 1976.

[87] L. Khachiyan, E. Boros, K. Borys, K. Elbassioni, V. Gurvich, and K. Makino. Enumerating spanning and connected subsets in graphs and matroids. *Lecture Notes in Computer Science*, vol. 4168, pp. 444-455, 2006.

[88] J. V. Knop, W. R. Muller, Z. Jericevi, and N. Trinajstic. Computer enumeration and generation of trees and rooted trees. *Journal of Chemical Information and Computer Science*, vol. 21, pp. 91-99, 1981.

[89] J. V. Knop, W. R. Muller, and K. Szymanski. Computer enumeration and generation of physical trees. *Journal of Computational Chemistry*, vol. 8, no. 4, pp. 549-554, 1987.

[90] J. Knopfmacher and R. Warlimont. Asymptotic isomer enumeration in chemistry. II. *Journal of Mathematical Chemistry*, vol. 26, pp. 95-99, 1999.

[91] D. E. Knuth. The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations. Addison-Wesley Professional; First Edition, 2005.

[92] D. E. Knuth. Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees–History of Combinatorial Generation. Addison-Wesley Professional; First Edition, 2006.

[93] B. Korte and J. Vygen. Combinatorial Optimization: Theory and Algorithms. Springer, 4th Edition, 2007.

[94] V. G. Kulkarni. Generating random combinatorial objects. *Journal of Algorithms*, vol. 11, no. 2, pp. 185-207, 1990.

[95] E. L. Lawlers, J. K. Lenstra, and A. H. G. Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, vol. 9, no. 3, pp. 558-565, 1980.

[96] Z. Li and S. Nakano. Efficient generation of plane triangulations without repetitions. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming* (ICALP'01), *Lecture Notes in Computer Science*, vol. 2076, pp. 433-443, 2001.

[97] G. Li and F. Ruskey. The advantage of forward thinking in generating rooted and free trees. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 939-940, 1999.

[98] V. A. Liskovets and T. R. Walsh. Ten steps to counting planar graphs. *Congressus Numerantium*, vol. 60, pp. 269-277, 1987.

[99] H. Liu and J. Wang. A new way to enumerate cycles in graph. In *Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services* (AICT/ICIW'06), pp. 57, 2006.

[100] J. Lucas. The rotation graph 01 binary trees is hamiltonian. *Journal of Algorithms*, vo. 8, pp. 503-535, 1987.

[101] J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *Journal of Algorithms*, vol. 15, pp. 343-366, 1993.

[102] J. E. Ludman. Gray code generation for MPSK signals. *IEEE Transactions on. Communications*, vol. COM-29, pp. 1519-1522, 1981.

[103] N. Lygeros, P. Marchand, and M. Massot. Enumeration and 3D representation of the stereo-isomers of alkane molecules. *Journal of Symbolic Computation*, vol. 40, pp. 1225-1241, 2005.

[104] K. Makino and T. Uno. New algorithms for enumerating all maximal cliques. *Lecture notes in computer science*, vol. 3111, pp. 260-272, 2004.

[105] P. M. Marcus. Derivation of maximal compatibles using Boolean algebra. *IBM Journal of Research and Development*, vol. 8, pp. 537-538, 1964.

[106] L. M. Masinter, N. S. Sridharan, J. Lederberg, and D. H. Smith. Applications of artificial intelligence for chemical inference. XII. Exhaustive generation of cyclic and acyclic isomers. *Journal of the American Chemical Society*, vol. 96, no. 25, pp. 7702-7714, 1974.

[107] G. Melançon, I. Dutour, and M. Bousquet-Mélou. Random generation of directed acyclic graphs. *Electronic Notes in Discrete Mathematics*, vol. 10, pp. 202-207, 2001.

[108] G. Melançon and F. Philippe. Generating connected acyclic digraphs uniformly at random. *Information Processing Letters*, vol. 90, no. 4, pp. 209-213, 2004.

[109] P. Mateti and N. Deo. On algorithms for enumerating all circuits of a graph. *SIAM Journal of Computing*, vol. 5, no. 1, 90-99, 1976.

[110] B. D. McKay. Isomorphic-free exhaustive generation. *Journal of Algorithms*, vol. 26, pp. 306-324, 1998.

[111] B. D. McKay and N. C. Wormald. Uniform generation of random regular graphs of moderate degree. *Journal of Algorithms*, vol. 11, no. 1, pp. 52-67, 1990.

[112] C.J. Mifsud. Algorithm 154: Combination in lexicographic order. *Communications of the ACM*, vo. 6, no. 3, pp. 103, 1963.

[113] R. Milo, N. Kashtan, S. Itzkovitz, M. E. J. Newman, and U. Alon. Uniform generation of random graphs with arbitrary degree sequences. e-print cond-mat/0312028, 2003.

[114] G. J. Minty: A simple algorithm for listing all the trees of a graph. *IEEE Transactions on Circuit Theory*, vol. CTC12, pp. 120, 1965.

[115] S. Mossige. Generation of permutations in lexicographical order. *BIT Numerical Mathematics*, vol. 10, no. 1, pp. 74-75, 1970.

[116] G. D. Mulligan and D. G. Corneil. Corrections to Bierstones algorithm for generating cliques. *Journal of the ACM*, vol. 19, no. 2, pp. 244-247, 1972.

[117] R. C. Mullin and P. J. Schellenberg. The enumeration of c-nets via quadrangulations. *Journal of Combinatorial Theory*, vol. 4, pp. 259-276, 1968.

[118] H. Nagamochi. A detachment algorithm for inferring a graph from path frequency. *Lecture Notes in Computer Science*, vol. 4112, pp. 274-283, 2006.

[119] S. Nakano. Enumerating floorplans with n rooms. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, *Lecture Notes in Computer Science*, vol. 2223, pp. 107-115, 2001.

[120] S. Nakano. Efficient generation of plane trees. *Information Processing Letters*, vol. 84, pp. 167-172, 2002.

[121] S. Nakano and T. Uno. Efficient generation of rooted trees. NII Technical Report (NII-2003-005), 2003, available on http://research.nii.ac.jp/TechReports/03-005E.html.

[122] S. Nakano and T. Uno. Constant time generation of trees with specified diameter. *Lecture Notes in Computer Science*, vol. 3353, pp. 33-45, 2005.

[123] S. Nakano and T. Uno. Generating colored trees. *Lecture Notes in Computer Science*, vol. 3787, pp. 249-260, 2005.

[124] A. Nijenhuis and H. S. Wilf. Combinatorial Algorithms, Academic Press, New York, 1975.

[125] R. J. Ord-Smith. Generation of permutations in lexicographic order (Algorithm 323). *Communications of the ACM*, vol. 2, pp. 117, 1968.

[126] R. Otter. The number of trees. *Annals of Mathematics*, vol. 49, no. 2, pp. 583-599, 1948.

[127] L. Pan and E. E. Santos. An anytime-anywhere approach for maximal clique enumeration in social network analysis. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* (SMC'08), Singapore, October 12-15, 2008, pp. 3529-3535, 2008.

[128] M. C. Paull and S. H. Unger. Minimizing the number of states in incompletely specified sequential switching junctions. *IRE transactions on Electronic Computers*, vol. EC-8, pp. 356-367, 1959.

[129] G. Pólya. Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen. *Acta Mathematica*, vol. 68, pp. 145-254, 1937.

[130] J. G. Propp and D. B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms*, vol. 9, no. 1-2, pp. 223-252, 1996.

[131] A. Proskurowski and F. Ruskey. Binary tree Gray codes. *Journal of Algorithms*, vol. 6, pp. 225-238, 1985.

[132] A. J. Quiroz. Fast random generation of binary, t-ary and other types of trees. *Journal of Classification*, vol. 6, pp. 223-231, 1989.

[133] J. Ramon and S. Nijssen. General graph refinement with polynomial delay. In *Proceedings of the Fifth International Workshop on Mining and Learning with Graphs* (MLG'07), Firenze, Italy, August 1-3, 2007, pp. 1-4, 2007.

[134] J. Ramon and S. Nijssen. Polynomial-delay enumeration of monotonic graph classes. *Journal of Machine Learning Research*, vol. 10, pp. 907-929, 2009.

[135] A. R. Rao, R. Jana, and S. Bandyopadhya. A Markov chain Monte Carlo method for generating random (0, 1)-matrices with given marginals. *Sankhyā: The Indian Journal of Statistics*, vol. 58, Series A, Pt. 2, pp. 225-242, 1996.

[136] R. C. Read and R. E. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, vol.5, pp. 237-252, 1975.

[137] R. C. Read. Every one a winner, or how to avoid isomorphism search when cataloguing combinatorial configurations. *Annals of Discrete Mathematics*, vol. 2, pp. 107-120, 1978.

[138] R. C. Read. A survey of graph generation techniques. *Lecture Notes in Mathematics*, vol. 884, pp. 77-89, 1981.

[139] R. C. Read. On the enumeration of a class of plane multigraphs. *Aequationes mathematicae*, vol. 31, pp. 47-63, 1986.

[140] J. H. Redfield. The theory of group-reduced distributions. *American Journal of Mathematics*, vol. 49, pp. 433-455, 1927.

[141] D. Richards. Data compression and Gray-code sorting. *Information Processing Letters*, vol. 22, no. 4, pp. 210-205, 1986.

[142] J. Robinson and M. Cohn. Counting sequences. *IEEE Transactions on Computers*, vol. C-30, pp. 17-23, 1981.

[143] R. W. Robinson, F. Harary, and A. T. Balaban. The numbers of chiral and achiral alkanes and mono substituted alkanes. *Tetrahedron*, vol. 32, pp. 355-361, 1976.

[144] F. Ruskey and T. C. Hu. Generating binary trees lexicographically. *SIAM Journal of Computing*, vol. 6, pp. 745-758, 1977.

[145] F. Ruskey. Generating t-ary trees lexicographically. *SIAM Journal of Computing*, vol. 7, pp. 424-439, 1978.

[146] F. Ruskey. Adjacent interchange generation of combinations. *Journal of Algorithms*, vol. 9, pp. 162-180, 1988.

[147] F. Ruskey. Simple combinatorial Gray codes constructed by reversing sublist. *Lecture Notes in Computer Science*, vol. 762, pp. 201-208, 1993.

[148] F. Ruskey and A. Proskurowski. Generating binary trees by transpositions. *Journal of Algorithms*, vol. 11, 68-84, 1990.

[149] F. Ruskey and D. Roelants van Baronaigien. Fast recursive algorithms for generating combinatorial objects. *Congressus Numerantium*, pp. 53-62, 1984.

[150] C. Savage. Gray code sequences of partitions. *Journal of Algorithms*, vol. 10, pp. 577-595, 1989.

[151] C. Savage. A survey of combinatorial Gray codes, *SIAM Review*, vol. 39, pp. 605-629, 1997.

[152] H. I. Scions. Placing trees in lexicographic order. *Machine Intelligence*, vol. 3, pp. 43-60, 1969.

[153] G. Schaeffer. Random sampling of larger planar maps and convex polyhedra. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, Atlanta, Georgia, United States, pp. 760-769, 1999.

[154] L. Schietgat, J. Ramon, and M. Bruynooghe. A polynomial-time metric for outerplanar graphs. The Sixteenth Annual Machine Learning Conference of Belgium, Amsterdam, Netherlands, May 14-15, 2007.

[155] I. Semba. An efficient algorithm for generating all partitions of the set $\{1, 2, \ldots, n\}$. *Journal of Information Processing*, vol. 7, pp. 41-42, 1984.

[156] A. Shioura and A. Tamura. Efficiently scanning all spanning trees of an undirected graph. *Journal of the Operations Research Society of Japan*, vol. 38, pp. 331-344, 1995.

[157] M. Shen. On the generation of permutations and combinations. *BIT Numerical Mathematics*, vol. 2, pp. 228-231, 1962.

[158] A. Sinclair. Algorithms for random generation and counting: a Markov chain approach. Birkhäuser Boston; First Edition, 1993.

[159] N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, available at www.research.att.com/ njas/sequences/.

[160] R. P. Stanley. Enumerative Combinatorics: Volume 1. Cambridge University Press, 1997.

[161] M. M. Syslo. On some generalizations of outerplanar graphs: results and open problems. *Lecture Notes in Computer Science*, vol. 246, pp. 146-164, 1987.

[162] T. Takaoka. $O(1)$ time algorithms for combinatorial generation by tree traversal. *Computer Journal*, vol. 42, No. 5, pp. 400-408, 1999.

[163] T. Takaoka. An $O(1)$ time algorithm for generating multiset permutations. *Lecture Notes in Computer Science*, vol. 1741, pp. 237-246, 1999.

[164] K. Tsuda and E. Georgii. Dense module enumeration in biological networks. *Journal of Physics: Conference Series*, vol. 197, 2009.

[165] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa. A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing*, vol. 6, pp. 505-517, 1977.

[166] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, vol. 15, pp. 249-271, 1963.

[167] H. F. Trotter. PERM (Algorithm115). *Communications of the ACM*, vol. 5, pp. 434-435, 1962.

[168] T. Uno, T. Asai, Y. Uchida, and H. Arimura. LCM: an efficient algorithm for enumerating frequent closed item sets. In *Proceedings of Workshop on Frequent Itemset Mining Implementations* (FIMI'03), 2003.

[169] V. Vajnovszki. Constant time algorithm for generating binary tree gray codes. *Studies in Informatics and Control*, vol. 5, pp. 15-21, 1996.

[170] T. Walsh and V. A. Liskovets. Ten steps to counting planar graphs. In *Proceedings of Eighteenth Southeastern International Conference on Combinatoris, Graph Theory, and Computing*, *Congressus Numerantium*, vol. 60, pp. 269-277, 1987.

[171] T. R. Walsh. Generation of well-formed parenthesis strings in constant worst-case time. *Journal of Algorithms*, vol. 29, pp. 165-173, 1998.

[172] L. Wan, B. Wu, N. Du, Q. Ye, and P. Chen. A new algorithm for enumerating all maximal cliques in complex network. *Lecture Notes in Computer Science*, vol. 4093, pp. 606-617, 2006.

[173] J. Wang, L. Zhao, H. Nagamochi, and T. Akutsu. An efficient enumeration of colored outerplanar graphs. *Lecture Notes in Computer Science*, vol. 4484, pp. 573-583, 2007.

[174] H. Whitney. Congruent graphs and the connectivity of graphs. *American Journal of Mathematics*, vol. 54, pp. 150-168, 1932.

[175] H. S. Wilf. Combinatorial algorithms: an update. Philadelphia, PA, SIAM, 1989.

[176] D. B. Wilson. Generating random spanning trees more quickly than the cover time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, Philadelphia, Pennsylvania, United States, pp. 296-303, 1996.

[177] R. A. Wright, B. Richmond, A. Odlyzko, and B. D. McKay. Constant time generation of free trees. *SIAM Journal on Computing*, vol. 15, pp. 540-548, 1986.

[178] P. Zimmermann. Gaïa: a package for the random generation of combinatorial structures. *Maple Technical Newsletter*, vol. 1, no. 1, pp. 1-9, 1994.