Formal Graph Systems and Parallel Graph Algorithm Design

Uchida, Tomoyuki

http://hdl.handle.net/2324/3182
Formal Graph Systems and Parallel Graph Algorithm Design

Tomoyuki Uchida

March 25, 1994

Research Institute of Fundamental Information Science
Kyushu University 33
Fukuoka 812, Japan
E-mail: mukouchi@rifis.sci.kyushu-u.ac.jp    Phone: 092-641-1101 ex. 4484
Formal Graph Systems and Parallel Graph Algorithm Design

December 1993

Tomoyuki Uchida

内 田 智 之
Abstract

Parallel algorithms have been given for solving a number of graph problems in the literatures. We can design a parallel algorithm with many processors solving a graph problem by finding the parallelism of the problem and assigning some parts computable in parallel to each processor. In order to gain the parallelism of a graph problem, it is required to analyze an input graph for the problem. Hence, it is important for designing a parallel algorithm to analyze the structure of a graph given as an instance of a graph problem. In this thesis, we give a new graph rewriting system which is a suitable system to analyze the structure of an input graph for a graph problem and design an efficient parallel algorithm solving the problem.

Chapter 2 makes a brief review of the parallel computation theory. We introduce the parallel random access machine (PRAM) being one of the parallel computation models. The class NC represents the class of problems solvable by polylogarithmic parallel time algorithms with a polynomial number of processors on a PRAM.

In Chapter 3, we define a formal graph system (FGS), as a new framework for graph rewriting, which is useful to design efficient parallel graph algorithms. An FGS is a logic program having hypergraphs instead of terms in first-order logic. We give a regular FGS which is an FGS of a special form and the family of graphs definable by a regular FGS is equal to that of graphs generated by a hyperedge replacement grammar (HRG) being one of the well-known context-free graph grammars. This result implies that regular FGSs generate trees, two-terminal series parallel (TTSP) graphs, the homeomorphisms of the given graphs, outerplanar graphs, all graphs of cyclic bandwidth \( \leq k \) for a constant \( k \geq 2 \) and \( s \)-decomposable graphs for a constant \( s \).

Chapter 4 defines a refutation tree which represents explicitly the structure of a
graph generated by an FGS. We consider the refutation tree problem which is to compute a refutation tree of a graph generated by an FGS. First, we present two regular FGSs generating the families of TTSP graphs and outerplanar graphs for which we devise EREW PRAM algorithms solving the refutation tree problem with $O(n + m)$ processors, where $n$ and $m$ are the numbers of vertices and edges of an input graph, respectively. The algorithm for TTSP graphs computes refutation trees in $O((\log n)^2 + \log m)$ time. The other algorithm solves the refutation tree problem for outerplanar graphs in $O((\log n)^2)$ time by employing the algorithm for TTSP graphs. Many NP-complete graph problems are known to admit polynomial time algorithms, when instances are restricted to the families of graphs generated by context-free graph grammars. These results suggest that efficient parallel algorithms may be designed for a large number of NP-complete problems when input graphs are TTSP graphs and outerplanar graphs. Next, we define a notion of a simple FGS and we prove that the refutation tree problem can be solved in polynomial time for the class of simple FGSs.

The purpose of Chapter 5 is to find two subclasses of simple FGSs, called size-bounded simple FGSs and bounded simple FGSs, for which the refutation tree problem is solvable in NC. An FGS generates graphs by replacing hyperedges labeled with the same variable at a time. Hence, in order to compute a refutation tree of an input graph generated by an FGS, it may be required to solve the graph isomorphism problem deciding whether two graphs are isomorphic. The graph isomorphism problem does not seem to be solvable in polynomial time, while it is in NP. However, it has been shown that the graph isomorphism problem is in $\text{NC}^3$ for connected graphs of constantly bounded valence and having constantly bounded separators, and is in $\text{NC}^2$ for connected trees of constantly bounded valence. We prove that the refutation tree problem for a size-bounded simple FGS is $\text{NC}^2$-reducible to the graph isomorphism problem
for connected graphs of constantly bounded valence. This yields that the refutation
tree problem for a size-bounded simple FGS can be solved in NC if connected graphs
of constantly bounded valence and having constantly bounded separators are given.
Moreover, we show two classes of FGSs which are helpful to design NC algorithms for
many graph problems. One is the class of size-bounded simple FGSs which generate
only undirected trees of constantly bounded valence. The other is the class of bounded
simple FGSs. For size-bounded simple FGSs generating undirected trees of constantly
bounded valence, we present an NC algorithm solving the refutation tree problem.
We show that solving the graph isomorphism problem is not required for computing a
refutation tree of a graph generated by a bounded simple FGS and that the refutation
tree problem for a bounded simple FGS is in NC^2.

The bound of valence of a given graph is an important property of FGSs con-
tributing to designing fast parallel algorithms for many graph problems. For graphs of
constantly bounded valence, Chapter 6 shows that the coloring technique works very
successfully to devise fast parallel algorithms with linear numbers of processors. By
using the vertex coloring technique, we give a fast parallel algorithm for the problem
VIMS(k), which is to find a maximal vertex-induced subgraph of valence at most k,
where k is a given constant. This algorithm runs in \(O(\log^* n)\) time using \(O(n)\) pro-
cessors on an EREW PRAM for an input graph of constantly bounded valence with n
vertices. We also design an \(O(\log^* m)\) time \(O(m)\) processor EREW PRAM algorithm
for the problem EIMS(k), which is to find a maximal edge-induced subgraph of degree
at most k, where m is the number of edges of an input graph. Shoudai and Miyano
[99, 100] and Diks et al. [23] have also given \(O(\log^* n)\) (resp., \(O(\log^* m)\)) time parallel
algorithms for solving VIMS(k) (resp., EIMS(k)) with a linear number of processors by
employing the parallel maximal independent set algorithm in [44]. Since \(\log^* n\) grows
extremely slowly and can be viewed as a constant for all practical purposes, these algorithms are regarded as constant time parallel algorithms for all practical purposes. Hence, the constants hidden in these $O$-notation are very essential. Our algorithms run faster with fewer numbers of processors than their algorithms by counting these hidden constants.
Acknowledgements

First of all, I express my gratitude to Prof. Satoru Miyano. I thank him for his invaluable suggestions and constructive comments. He has led me to the very interesting research world of parallel algorithm design.

I would like to thank Prof. Setsuo Arikawa for his many helps and advice from the beginning of this study. I would also thank Prof. Yasuo Kawahara and Prof. Shunichi Tanaka for their valuable comments. I would express my appreciation to Prof. Takayoshi Shoudai and Prof. Shuji Shiraishi. The seminars with them were the most valuable experiences in this research.

I had good discussions at SIGPPC workshops. I thank to Prof. Hiroto Yasuura and Prof. Kazuo Iwama for useful discussions and suggestions on this research at SIGPPC workshops.

I appreciate the support from the people at Department of Information Systems. Especially, I thank Mr. Takeo Okazaki and Mr. Ayumi Shinohara for their helps and comments.

Finally I thank my parents.
Contents

1 Introduction ................................................................................. 1

2 Efficient Parallel Algorithms .................................................... 8
   2.1 The Class NC ...................................................................... 9
   2.2 NC-Reducibility .................................................................. 12
   2.3 Parallel Random Access Machine (PRAM) .......................... 14

3 Formal Graph System as a New Graph Rewriting System ............ 18
   3.1 Formal Graph System (FGS) .............................................. 19
   3.2 Regular FGSs Characterizing Hyperedge Replacement Grammars ... 27
   3.3 Simple FGSs .................................................................... 34
   3.4 Discussion ....................................................................... 37

4 Refutation Tree Problem ............................................................ 41
   4.1 $O((\log n)^2 + \log m)$ Time Parallel Algorithm for Two-Terminal Series Parallel Graphs .......................................................... 44
   4.2 $O((\log n)^2)$ Time Parallel Algorithm for Outerplanar Graphs ........ 57
   4.3 Polynomial Time Algorithm for Graphs Generated by a Simple FGS .... 62
   4.4 NC Algorithm for Graphs Generated by a Context-Free Graph Grammar 68
   4.5 Discussion ...................................................................... 74
5 Refutation Tree Problem and Its Relation to Graph Isomorphism

5.1 NC Algorithm for a Size-Bounded Simple FGS .......................... 77
5.2 Refutation Tree Problem equivalent to Graph Isomorphism Problem .. 80
5.3 Discussion .................................................................................. 81

6 $O(\log^* n)$ Time Parallel Algorithms for Bounded Degree Maximal Sub-
graph Problems ............................................................................. 83
6.1 Preliminaries ............................................................................... 84
6.2 Bounded Degree Maximal Vertex-Induced Subgraph Problem ......... 85
6.3 Bounded Degree Maximal Edge-Induced Subgraph Problem ........... 89
6.4 Discussion .................................................................................. 91

Bibliography .................................................................................. 92
Chapter 1

Introduction

Designing parallel algorithms for many graph problems has been extensively investigated in last two decades. A parallel algorithm for a graph problem is designed by finding the parallelism of the problem. We find the inherent parallelism in the graph problem by analyzing an input graph and obtain an idea of designing a fast parallel algorithm for the problem. For example, the structure of two-terminal series parallel graphs helps us to get a fast parallel algorithm for the graph 3-colorability problem \([53]\). In this thesis, we define a new graph rewriting system which is useful to find an inherent parallelism in a graph problem.

Many formal models of the parallel computation appear in the literatures. The Parallel Random Access Machine (PRAM) is one of the primary models that we are working with. A PRAM consists of a number of processors each of which has its own local memory and a common shared memory to which all processors have accesses in order to communicate each other. All the processors operate synchronously under the control of a common clock. In the PRAM model, two or more processors try to read from or write into the same memory cell concurrently. Then, PRAMs can be classified according to the possibility of read- and write-conflicts. The strongest PRAM is the Concurrent Read Concurrent Write (CRCW) PRAM which permits simultaneous reading and writing accesses to the same memory cell. The Concurrent
Read Exclusive Write (CREW) PRAM allows simultaneous reads but no simultaneous writes. The weakest one is the Exclusive Read Exclusive Write (EREW) PRAM for which simultaneous accesses are forbidden for both reading and writing. A CRCW PRAM can be simulated by an EREW PRAM with the same number of processors just by increasing the parallel time by only a factor of $O(\log P)$, where $P$ is the number of processors [112]. The class NC can be defined as the collection of problems which can be solved in $O((\log n)^k)$ time with a polynomial number of processors on a PRAM for some fixed $k \geq 0$ and an input size $n$ (see [43, 57, 63, 84]). This class name is an abbreviation for Nick's Class, in honor of Nicholas Pippenger, the first researcher who studied the class. Most theoretical computer scientists have strongly believed $\text{NC} \subsetneq \text{P}$, but it has not been known whether or not $\text{NC}=\text{P}$.

The following graph 3-colorability problem is a well-known NP-complete problem [105]:

**Graph 3-Colorability Problem**

**INSTANCE:** Graph $G = (V, E)$.

**QUESTION:** Is $G$ 3-colorable, i.e., does there exist a function $f : V \to \{1, 2, 3\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?

If an instance is a graph all of whose vertices have degree 2 or less, then the graph 3-colorability problem can be trivially solved in polynomial time. In recent years, Xin He [53] has proved that the graph 3-colorability problem for two-terminal series parallel graphs can be solved in NC. The family of two-terminal series parallel graphs is known to be generated by some graph grammars [49, 88]. For other NP-complete graph problems, e.g., the Hamiltonian circuit problem, the independent set problem and the max cut problem [40], a number of polynomial time algorithms [67, 69, 101] and efficient parallel algorithms [95] have been provided if input graphs are in the family of graphs generated by some graph grammars [50, 101]. Namely, for such problems, graph grammars are very useful for analyzing the structure of an input graph.
In this thesis, we define a formal graph system (FGS), as a new system of graph rewriting, which is a suitable system for analyzing the structure of graphs and designing efficient parallel algorithms. An FGS is a kind of logic programs which deals with graphs just like terms [75]. By regarding terms as trees, conventional logic programs can be directly simulated by FGSs. Moreover, an elementary formal system [9, 10, 102], that is a logic program on strings, is also a special case of FGSs. In Chapter 3, we give an FGS of a special form, called a regular FGS, which has the same ability of generating graphs as a hyperedge replacement grammar (HRG) due to Habel and Kreowski [49, 50, 71]. This result shows that regular FGSs characterize HRGs. By this result, we can see that regular FGSs generate some interesting families of graphs, e.g., trees, two-terminal series parallel graphs, the homeomorphisms of the given graph, outerplanar graphs, all graphs of cyclic bandwidth ≤ k for a constant k ≥ 2 and s-decomposable graphs for a constant s (see [25, 51, 71, 88]). Habel and Kreowski [50] have given a pumping lemma for an HRG which generalizes those pumping lemmas for context-free string languages [55] and context-free graph languages [25, 66, 111]. From this pumping lemma, we can see that HRGs cannot generate the families of balanced binary trees and Sierpinski triangles. Hence, regular FGSs cannot generate those families. However, those families are definable by FGSs (for example, see Figure 3.2). Thus FGSs have a diversity of expressibility.

There are a number of investigations on the syntax analysis of graph languages from the complexity point of view [12, 65, 68, 71, 91, 95, 101, 111]. But, only a few works are known from the viewpoint of the parallel algorithm design. Slisenko [101] and Kriauciuškas [67] have found the family of graphs, which is generated by a context-free graph grammar (CFGG) defined in [101], for which the Hamiltonian circuit problem, the graph k-colorability problem, the independent set problem and the max
cut problem [40] are solvable in polynomial time. Moreover, Rytter and Szymacha [95] have shown that the traveling salesman problem, the induced path problem, the independent set problem, the dominating set problem and the vertex cover problem [40] for the same family of graphs can be solved in NC. These results have also shown that the researches of graph grammars may lead us to solve a number of graph problems drastically faster, if possible.

In Chapter 4, we investigate FGSs from the parallel algorithm design. We present efficient parallel algorithms for solving the refutation tree problem which is to compute a refutation tree associated with the structure of a graph generated by an FGS. FGSs are logic programs which use graphs instead of terms in first-order logic. A refutation tree [87, 97, 103] describes how a term is generated by applying the rules of a logic program. A refutation tree of a graph for an FGS is defined in a similar way. It represents the logical structure of the graph in the FGS and simultaneously describes a decomposition of the graph explicitly. Since an FGS consists of definite clauses which explicitly describes how it generates graphs and since refutation trees give methods for applying the divide-and-conquer technique to the input graphs, it may be useful for solving many graph problems to employ FGSs in designing efficient parallel algorithms.

For an HRG, a parse tree of a graph $G$ is, in a sense, a tree which shows a method of decomposing $G$ into a set of terminal elements by assigning a right-hand side of a production in the HRG to each node in the tree. Therefore, it does not express in a straightforward way how $G$ is decomposed into the set of terminal elements. However, since a subgraph of $G$ is assigned to each node of the refutation tree, the refutation tree expresses the structures of graphs directly and can also cope with a family of graphs for which no CFGG exists. These show that FGSs have a diversity of abilities in generating graphs more than CFGGs, and that refutation trees for an FGS are more
suitable for expressing the structures of graphs than parse trees for a CFGG.

We present two regular FGSs for which we devise efficient parallel algorithms solving the refutation tree problem with $O(n + m)$ processors on an EREW PRAM, where $n$ and $m$ are the numbers of vertices and edges of an input graph, respectively. One is a regular FGS defining the family of two-terminal series parallel (TTSP) graphs and the other defines the family of outerplanar graphs. For the family of TTSP graphs, we present an $O((\log n)^2 + \log m)$ time algorithm. By using this algorithm, for the family of series parallel graphs being the undirected version of TTSP graphs, we also show that the refutation tree problem can be solved in the same amount of time using the same number of processors on an EREW PRAM. For the family of undirected outerplanar graphs, Diks et al. [24] have given an optimal parallel algorithm on a CREW PRAM which solves the problem of testing outerplanarity in $T(n) = O(\log n \log^* n)$ time using $n/T(n)$ processors. With $O(n + m)$ processors on an EREW PRAM, we give an $O((\log n)^2)$ time algorithm for testing outerplanarity by employing the algorithm for TTSP graphs. These results assert that efficient parallel algorithms may be designed for a large number of NP-complete problems when these instances are restricted to the families of TTSP graphs and undirected outerplanar graphs. For example, Xin He [53] has presented efficient parallel algorithms for solving the graph 3-colorability problem and the depth-first spanning tree problem (DFST) for series parallel graphs, where DFST for general graphs is an open problem in parallel algorithm design [4, 5].

Next, we define an idea of a simple FGS and show that we can compute a refutation tree of a graph generated by a simple FGS in polynomial time. In this proof, we give a polynomial time algorithm which constructs and solves a path system. Graphs generated by simple FGSs are of constantly bounded valence. However, this polynomial time algorithm does not seem to be efficiently parallelizable, since the problem of
solving a path system is P-complete [17]. But, in Chapter 5, we give some subclasses of simple FGSs such that the refutation tree problem is solvable in NC. An FGS generates graphs by replacing hyperedges labeled with the same variable at a time. Hence, in order to compute a refutation tree for a graph generated by an FGS, it may be required to solve the graph isomorphism problem defined as follows:

**Graph Isomorphism Problem**

**INSTANCE:** Two graphs $G_1$ and $G_2$.

**QUESTION:** Are $G_1$ and $G_2$ isomorphic?

Lingas [73] has shown that the graph isomorphism problem is in NC$^3$ for connected graphs of constantly bounded valence and having constantly bounded separators. We give a size-bounded simple FGS as a subclass of simple FGSs such that the refutation tree problem is NC$^2$-reducible to the graph isomorphism problem for graphs of constantly bounded valence. This means that the refutation tree problem for a size-bounded simple FGS is in NC, if a connected graph of constantly bounded valence and having a constantly bounded separator is given. Moreover, we give size-bounded simple FGSs generating undirected trees of constantly bounded valence for which the refutation tree problem is in NC by employing the NC$^2$ algorithm in [7] solving the graph isomorphism problem. A bounded simple FGS is a size-bounded simple FGS such that we can compute a refutation tree of a graph without solving the graph isomorphism problem. Hence, the refutation tree problem for a bounded simple FGS can be solved in NC$^2$ when an input graph is connected.

For designing an NC algorithm solving the refutation tree problem for a simple FGS, the connectedness and bound of valence of a given graph are the crucial properties. In Chapter 6, we focus on the bounded valence of a given graph and show that it contributes to drastic speed-ups of parallel graph algorithms. We consider two combinatorial graph problems, denoted by VIMS($k$) and EIMS($k$), which are to find
maximal subsets of vertices and edges whose induced subgraphs are of valence at most \( k \) for a given integer \( k \), respectively. Shoudai and Miyano [99, 100] and Diks et al. [23] have shown that VIMS(\( k \)) and EIMS(\( k \)) are in NC by describing algorithms which employ the parallel maximal independent set (MIS) algorithm [45, 64, 76]. If we apply the fast parallel MIS algorithm in [44] to the algorithms in [23, 99, 100], we can easily see that VIMS(\( k \)) (resp., EIMS(\( k \))) for graphs of constantly bounded valence can be solved in \( O(\log^* n) \) (resp., \( O(\log^* m) \)) time with \( O(n) \) (resp., \( O(m) \)) processors on an EREW PRAM, where \( n \) and \( m \) are the numbers of vertices and edges of an input graph, respectively. However, since \( \log^* n \) grows extremely slowly and can be viewed as a constant for all practical purposes, it is important to focus on the constants \( k \) and the valence \( \Delta \) of an input graph. From this viewpoint, we present parallel algorithms for VIMS(\( k \)) and EIMS(\( k \)) employing the vertex-coloring technique. When the vertex coloring algorithm in [44] is used, for an input graph of valence at most \( \Delta \), our algorithm for VIMS(\( k \)) runs \( k \) times as fast as the algorithm in [23, 99, 100] equipped with the MIS algorithm in [44]. Moreover, the number of processors needed by our algorithm is \( \Delta/k \) times as few as that of their algorithm. If the valence \( \Delta \) of an input graph satisfies \( \Delta = o(\log n) \), our method also provides an algorithm faster than that in [23, 99, 100] even though we apply the \( O((\log n)^2) \) time parallel MIS algorithm by [76] to their algorithm. Moreover, our algorithm for EIMS(\( k \)) runs \( 2k \) times as fast as the algorithm in [23, 99] employing the maximal matching in [44] for graphs of valence at most \( \Delta = o(\log n) \).

This thesis shows that FGSs play important role in designing graph algorithms, in particular, parallel algorithms, and that the coloring technique is very useful for designing fast parallel algorithms for solving some combinatorial graph problems when instances are graphs of constantly bounded valence.
Chapter 2

Efficient Parallel Algorithms

The class NC is defined as the set of uniform circuit families each of which has a polynomial size and a polylogarithmic depth of an input size [89]. A parallel algorithm is called efficient if it runs in polylogarithmic time using a polynomial number of processors. Stockmeyer and Vishkin [104] have shown that NC is equal to the class of problems which have efficient parallel algorithms. Namely, the class NC is generally accepted as a characterization of the class of problems that can be solved with a high degree of parallelism using a feasible amount of hardware.

We have known a number of problems which seem to have only polynomial time algorithms, e.g., P-complete problems [48, 85]. If an efficient parallel algorithm for any P-complete problem could be found then a similar algorithm would exist for any other. There is no proof, but a great deal of circumstantial evidence, that NC $\subseteq$ P. The P-completeness for a problem means that we cannot expect to find an efficient parallel algorithm solving the problem. However, the P-completeness cannot deny that we hope for a drastic speed-up by parallel processing. For example, Shiloach and Vishkin [98] have given an $O(n^2 \log n)$ parallel time algorithm for the maximum flow problem which has been proved in [47] to be P-complete by parallelizing a known $O(n^3)$ sequential time algorithm.

This chapter makes a brief survey of the parallel computation theory. See more
2.1 The Class NC

In this section, we give the definition of NC based on the uniform circuit family.

An \textit{n-input m-output Boolean function} is a function from \(\{0,1\}^n\) to \(\{0,1\}^m\), where 0 and 1 denote boolean values \textit{false} and \textit{true}, respectively. Let \(B_{n,m}\) denote the set of all \(n\)-input \(m\)-output Boolean functions. For example, the Boolean functions \(\lor, \land,\) \textit{NAND} and \textit{NOR} are 2-input 1-output Boolean functions, that is, these are in \(B_{2,1}\). The function \(\neg\) is in \(B_{1,1}\).

\textbf{Definition 2.1} Let \(\Omega\) be a set of \(n\)-input \(m\)-output Boolean functions. We call \(\Omega\) a basis. An \(\Omega\)-circuit \(\alpha\) with inputs \(x_1, \ldots, x_n\), and outputs \(y_1, \ldots, y_m\) is a finite labeled directed acyclic graph satisfying the following conditions (1)-(3): For a vertex \(v\), edges entering into and leaving from \(v\) are called the \textit{input lines} and the \textit{output lines} of \(v\), respectively. The numbers of input lines and output lines of \(v\) are called as the \textit{fan-in} and the \textit{fan-out} of \(v\), respectively.

(1) For each \(1 \leq i \leq n\), there is exactly one vertex labeled with \(x_i\) whose fan-in and fan-out are zero and one, respectively. These \(n\) vertices are called the \textit{input} vertices.

(2) There are exactly \(m\) vertices whose fan-in and fan-out are one and zero, respectively. These vertices are called the \textit{output} vertices and labeled distinctively with \(y_1, \ldots, y_m\).

(3) Each internal vertex \(v\) which has fan-in \(\lambda\) and fan-out \(\mu\) is labeled with a \(\lambda\)-input \(\mu\)-output Boolean function \(f\) in \(B_{\lambda,\mu} \cap \Omega\). The number of input lines of \(v\) is equal to that of argument of \(f\). These vertices are called \textit{gate} vertices.
Let $\alpha$ be an $\Omega$-circuit with inputs $x_1, \ldots, x_n$ and outputs $y_1, \ldots, y_m$. When the input vertices with labels $x_1, \ldots, x_n$ are assigned values in $\{0, 1\}$, let the value of the output vertex labeled with $y_i$ be a unique value in $\{0, 1\}$ for each $1 \leq i \leq m$. Then, $\alpha$ computes a function $f$ if the function $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ can be associated with $\alpha$.

The size of an $\Omega$-circuit $\alpha$ is the number of vertices in $\alpha$, and the depth of $\alpha$ is the length of the longest path from some input to some output. The size and the depth of $\alpha$ are denoted by $size(\alpha)$ and $depth(\alpha)$, respectively. Let $f$ be in $B_{n,m}$. Then we define $size(f)$ and $depth(f)$ as follows:

$$size(f) = \min \{ size(\alpha) \mid \alpha \text{ computes } f \},$$

$$depth(f) = \min \{ depth(\alpha) \mid \alpha \text{ computes } f \}. $$

A function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ is standard if there is a nonnegative function $s(n)$ such that for each $n \geq 0$ and each $x \in \{0, 1\}^n$, $f(x) \subseteq \{0, 1\}^{s(n)}$. A $\{v, \wedge, \neg\}$-circuit is simply called a circuit.

**Definition 2.2** A circuit family is a sequence $\{\alpha_n\}_{n \geq 0}$ of circuits $\alpha_n$ having $n$ inputs for each $n \geq 0$.

Let $f = \{f_n\}_{n \geq 0}$ be a standard function and let $\{\alpha_n\}_{n \geq 0}$ be a circuit family. We say that $\{\alpha_n\}_{n \geq 0}$ computes $f$ if the circuit $\alpha_n$ computes $f_n$ for each $n \geq 0$.

Let $\{\alpha_n\}_{n \geq 0}$ be a circuit family with $size(\alpha_n) = s(n)$ and the output size $h(n)$. Then the encoding $\hat{\alpha}$ of each $\alpha_n$ with vertices $v_1, \ldots, v_{s(n)}$ is the string $\hat{v}_1 \cdots \hat{v}_{s(n)}$ defined as follows: For each $1 \leq i \leq s(n)$, the vertex $v_i$ of $\alpha_n$ is encoded as a binary integer and $\hat{v}_i$ denotes a 4-tuple

$$[v_i, type(v_i), first(v_i), second(v_i)],$$

where $type(v_i)$ is the label of $v_i$ in $\{v, \wedge, \neg, x_1, \ldots, x_{s(n)}, y_1, \ldots, y_{h(n)}\}$, $first(v_i)$ and $second(v_i)$ are encoded as binary integers representing the endpoints of the first and the second input lines of $v_i$, respectively. For a gate with types $\neg$, we omit its second
Definition 2.3 Let $\Delta = \{\lor, \land, \neg, [ , (, (comma))\} \cup \{0, 1\}^+$. A circuit family $\{\alpha_n\}_{n \geq 0}$ is said to be log-uniform if there exist the function $c : \{1\}^* \rightarrow \Delta^*$ computable in space $O(\log n)$ on a deterministic Turing machine such that $c(1^n)$ is a standard encoding $\hat{\alpha}_n$ of $\alpha_n$ for each $n \geq 0$.

For functions $S(n)$ and $D(n)$, let $U$-$\text{SIZE}$-$\text{DEPTH}(S(n), D(n))$ denote the set of standard functions computable by a log-uniform circuit family of size $O(S(n))$ and depth $O(D(n))$. Then, the class NC is defined as follows:

**Definition 2.4 (Pippenger [89])** Let $k$ be a nonnegative integer.

1. $\text{NC}^k = \bigcup_{k \geq 0} \text{U-SIZE-DEPTH}(n^c, (\log n)^k)$.
2. $\text{NC} = \bigcup_{k \geq 0} \text{NC}^k$.

A function $f$ is said to be $\text{NC}$-computable (resp., $\text{NC}^k$-computable) if $f$ is in NC (resp., $\text{NC}^k$).

Not only the functions but the "search problems" are handled in the parallel computation theory. For example, a maximal independent set (MIS) of an undirected graph $G$ is a maximal set $U$ of vertices in $G$ such that no two vertices in $U$ are adjacent. The MIS problem is to find a MIS of a given graph. This MIS problem is formally defined as a family of binary relations $\text{MIS} = \{\text{MIS}_n\}_{n \geq 0}$ in the following way: A graph with $n$ vertices is represented by an $n \times n$-adjacency matrix and a subset of vertices is represented by an $n$-bit vector. Then $\text{MIS} = \{\text{MIS}_n\}_{n \geq 0}$ is defined only for integers of the form $n = m^2$ as

$$\text{MIS}_{m^2} \subseteq \{0, 1\}^{m^2} \times \{0, 1\}^m,$$

where for each $(x, y) \in \text{MIS}_{m^2}$, $x$ is a symmetric matrix representing an undirected graph with $m$ vertices and $y$ is a bit vector representing a MIS in the graph.
**Definition 2.5 (Cook [18])** A problem (or search problem) $S$ with a size parameter $h(n)$ is a family $\{S_n\}_{n \geq 0}$ of binary relations $S_n \subseteq \{0,1\}^n \times \{0,1\}^{h(n)}$ for $n \geq 0$. For $n \geq 0$, an element $x \in \{0,1\}^n$ is called an instance and an $h(n)$-bit sequence $y \in \{0,1\}^{h(n)}$ satisfying $S_n(x, y)$, if any, is called a solution for $x$. For convenience, we assume that there exists $y$ with $S_n(x, y)$ for each $x \in \{0,1\}^n$.

If a solution $y$ with $S_n(x, y)$ is unique for each $x \in \{0,1\}^n$, then the problem of finding a solution is exactly the same as computing the function defined by $S$. Moreover, if $h(n) = 1$ for all $n \geq 0$, then the problem $S$ is regarded as a decision problem.

**Definition 2.6** Let $S = \{S_n\}_{n \geq 0}$ be a problem with a size parameter $h(n)$. Then $S$ is said to be $NC$-solvable if there is an NC-computable function $f_S : \{0,1\}^* \rightarrow \{0,1\}^*$ such that for every $x \in \{0,1\}^*$ and $n \geq 0$, $S_n(x, f_S(x))$ holds.

In the same way as NC-solvable, we can define $NC^k$-solvable. That is, $NC^k$ is the set of all problems solvable by a uniform circuit family $\{\alpha_n\}_{n \geq 0}$ with $size(\alpha_n) = n^{O(1)}$ and $depth(\alpha_n) = O(\log^k n)$ for all $n \geq 0$.

### 2.2 NC-Reducibility

In previous section, we consider a circuit whose each gate vertex is labeled with a Boolean function in $\{0,1, \lor, \land, \neg\}$. In this section, we consider an oracle circuit.

An oracle circuit is a Boolean circuit with an additional class of oracle gates allowed. An oracle gate can have any number of inputs and outputs. The input for an oracle gate is an input bit sequence of values on its input lines and output is an output bit sequence of values on its output lines. When a problem $S$ is given as an oracle, the output of an oracle gate for $S$ with an input $x$ is any answer for $x$ in $S$.

**Definition 2.7** Let $S = \{S_n\}_{n \geq 0}$ and $T = \{T_n\}_{n \geq 0}$ be problems of size parameters $h(n)$ and $g(n)$, respectively. We say that $S$ is $NC^k$-reducible to $T$, denoted $S \leq_{NC^k} T$, if
if there is a log-space uniform family \( \{ A_n \}_{n \geq 0} \) of oracle circuits \( A_n \) satisfying the following conditions:

1. Each oracle circuits \( A_n \) can involve several oracle gates which can solve \( T \).
2. The depth of \( \{ A_n \} \) is \( O((\log n)^k) \), where the depth of an oracle gate with \( r \) input lines is measured as \( \lceil \log r \rceil \). For each input \( x \in \{0, 1\}^n \), the oracle circuit \( A_n \) outputs \( z \) with \( S_n(x, z) \) so far as the input-output relation satisfies \( T \) at each oracle gate. Namely, even though each oracle gate outputs any solutions of \( T_n \), the oracle circuit \( A_n \) outputs a solution for \( x \) of \( S_n \).

We say that \( S \) is \( NC \)-reducible to \( T \), denoted \( S \leq_{NC} T \), if \( S \leq_{NC^k} T \) for some \( k \).

It is known that if \( S \) is log space reducible to \( T \) then \( S \) is \( NC^2 \)-reducible to \( T \). The relation \( \leq_{NC} \) is transitive, and the class \( NC^k \) and \( NC \) are closed under the relations \( \leq_{NC} \) and \( \leq_{NC^1} \), for all \( k \geq 1 \).

A problem \( T \) is said to be \( P \)-complete if \( T \) is in \( P \) and for each problem \( S \) in \( P \), \( S \leq_{NC} T \). Let \( T \) be a \( P \)-complete problem. Then if \( T \) is in \( NC \) then \( P = NC \). At present it has not been known whether or not \( P = NC \), but it is strongly believed that \( P \neq NC \). Assuming that \( P \neq NC \), a \( P \)-complete problem is not in \( NC \). In other words, a \( P \)-complete problem is a problem which has no efficient parallel algorithms under the assumption of \( P \neq NC \). Goldschlager et al. [47] have proved that the following maximum flow problem is \( P \)-complete:

**Maximum Flow Problem**

**INSTANCE:** A directed graph \( D = (V, A) \) with two distinguished vertices, a source \( s \) and a sink \( t \), a positive integer capacity \( c(u, v) \) on every edge \( (u, v) \in A \), and a positive integer \( i \).

**QUESTION:** Let \( x \) be the \( i \)th bit of the value of the maximum flow from \( s \) to \( t \) in \( D \). Is \( x \) 1?
However, Shiloach and Vishkin [98] have presented an $O(n^2 \log n)$ parallel time algorithm for solving the maximum flow problem by parallelizing a known $O(n^3)$ sequential time algorithm. Hence, the P-completeness cannot deny that we hope for a drastic speed-up by parallel processing.

### 2.3 Parallel Random Access Machine (PRAM)

A parallel random access machine (PRAM) is one of the most useful models of parallel computation [39, 46, 96]. This is an idealized model, and can be viewed as the parallel version of the sequential RAM [6, 19]. A PRAM consists of several independent RAM $P_1, P_2, \ldots$ operating synchronously in parallel and a common memory (see Figure 2.1).

Each RAM has its own local memory and communicates with one another through the common memory. In one unit of time, each processor can read and write into one common or local memory, and execute the following single RAM operation: load, store, addition, subtraction, read, write, conditional branches based on the predicates $=$ and $<$, jump, and halt. PRAMs can be classified according to restrictions on common memory.
memory access as follows:

1. Exclusive-Read Exclusive-Write (EREW) PRAM: Simultaneous accesses to any memory location by different processors are forbidden for both reading and writing.

2. Concurrent-Read Exclusive-Write (CREW) PRAM: Simultaneous reads are allowed but no simultaneous writes.

3. Concurrent-Read Concurrent-Write (CRCW) PRAM: Both simultaneous reads and writes are allowed. In this case we have to specify how to resolve write conflicts. The following three methods of resolving write conflicts are commonly used [112]:

   (a) The COMMON model: All processors writing into the same location must write the same value.

   (b) The ARBITRARY model: Any one of processors participating in a common write may succeed, and the algorithm should work correctly regardless of which one succeeds.

   (c) The PRIORITY model: There is a linear ordering on the processors, and the minimum numbered processor writes its value in a concurrent write.

Any algorithm running on a PRIORITY CRCW PRAM in time $T$ can also simulated on a COMMON CRCW PRAM in $O(T)$ time with $P \log P$ processors, further each step of the PRIORITY CRCW PRAM requires at most $O(\log \log P)$ steps on the ARBITRARY CRCW PRAM with $P$ processors [13]. Namely, the PRIORITY CRCW PRAM is the most powerful in CRCW PRAM models.

The CRCW PRAM is the most powerful model and the CREW PRAM is more powerful than the EREW PRAM. Any algorithm for a PRIORITY CRCW PRAM can be simulated by an EREW PRAM with the same number of processors and with
the parallel time increased by only a factor of $O(\log P)$, where $P$ is the number of processors [112]. Since any EREW PRAM algorithm will run unchanged on the CREW PRAM, the simulation also implies that the CREW PRAM can simulate the PRIOR-ITY CRCW PRAM within the same bounds. We also see that the EREW PRAM can simulate the CREW PRAM within these bounds. These two simulation results cannot be improved in general. For example, we have known that the problem of computing the Boolean OR function of $n$ variables can be done in $O(1)$ time on the COMMON CRCW PRAM, and hence obviously can be done in that time on the PRIORITY CRCW PRAM. However, Cook et al. [16] have proved that the computation requires $\Omega(\log n)$ time on the CREW PRAM, regardless of the number of processors available. This means that the weakest CRCW PRAM is strictly more powerful than the CREW PRAM. In [57], we have known that a search problem requires $\Omega(\log n)$ time on the EREW PRAM, whereas it can be solved in $O(1)$ time on the CREW PRAM. Thus, the CREW PRAM is strictly more powerful than the EREW PRAM.

**Definition 2.8** A PRAM algorithm solving a problem of an input size $n$ is *efficient* if it runs

1. in time $O((\log n)^k)$ for some constant $k \geq 0$,
2. with a polynomial number of processors.

This definition is based on the observation that the time $O((\log n)^k)$ is very fast and a polynomial number of processors is feasible. Let $S$ be a problem whose fastest sequential algorithm runs in time proportional to $T(n)$ where $n$ is an input size. A PRAM algorithm for $S$ running in parallel time $t(n)$ with $p(n)$ processors is *optimal* if $t(n) \cdot p(n) = O(T(n))$.

Let CRCW-PRAM($T(n), P(n)$) be the class of problems computable on a PRIOR-ITY CRCW PRAM in parallel time $T(n)$ with $P(n)$ processors. Then, the following
The theorem relates the class NC and efficient parallel algorithms.

**Theorem 2.1 (Stockmeyer and Vishkin [104])**

1. \( \text{NC} = \bigcup_{k \geq 0} \text{CRCW-PRAM}(\log n, n^{O(1)}) \).
2. \( \text{NC}^k \subseteq \text{CRCW-PRAM}(\log n \log k, n^{O(1)}) \subseteq \text{NC}^{k+1} \).
Chapter 3

Formal Graph System as a New Graph Rewriting System

The area of graph grammars and graph languages has grown quite extensively. The investigations of graph grammars was motivated from a wide spectrum of applications, i.e., pattern recognition, software specification and development, VLSI design, databases, analysis of concurrent systems, and many others [14, 29, 30, 31]. Graph grammars have been defined in various ways. In particular, context-free graph grammars have been known as formal systems derived by expanding the concept of context-free grammars to graphs. For example, Habel and Kreowski [49, 50, 71, 88] and Bauderon and Courcelle [11] have introduced hyperedge replacement grammars (HRGs) where exactly one edge or hyperedge is replaced at a time. Rozenberg and Welzl [91, 92] have defined boundary node label controlled (BNLC) grammars where one vertex is replaced at a time.

The purpose of this chapter is to define a formal graph system (FGS) as a new graph rewriting system and to present some subclasses of FGSs having very interesting properties from the viewpoints of the generating power and the efficient parallel algorithm design. We define an FGS by extending an elementary formal system [9, 10, 102], which is a kind of logic programs which deals with strings, so that it can directly manipulate
graphs. An FGS is a logic program which use graphs instead of terms in first-order logic. In Section 3.2, we prove that the class of graphs is generated by a hyperedge replacement grammar (HRG) due to Habel and Kreowski [49, 50, 71] if and only if it is defined by a regular FGS which is an FGS of a special form. This result shows that regular FGSs generate trees, two-terminal series parallel graphs, the homeomorphisms of the given graph, outerplanar graphs, all graphs of cyclic bandwidth \( \leq k \) for a constant \( k \geq 2 \) and \( s \)-decomposable graphs for a constant \( s \) (see [25, 51, 71, 88]). Unfortunately, the result also shows that regular FGSs cannot generate the families of planar graphs, complete graphs, balanced binary trees and Sierpinski triangles, since Habel and Kreowski [50] have proved that HRGs cannot generate those families.

In Section 3.3, we define two subclasses, called simple FGSs and bounded simple FGSs. We show that graphs generated by a simple FGS are of constantly bounded valence. These classes are very interesting from the viewpoint of designing parallel algorithms. In Chapter 4, we discuss the complexity of deciding whether a given connected graph is generated by a simple FGS and a bounded simple FGS.

This chapter is based on the papers [108, 109].

3.1 Formal Graph System (FGS)

Let \( \Sigma \) and \( \Lambda \) be finite alphabets. A colored-graph \( g = (V, E, \varphi, \psi) \) over \( (\Sigma, \Lambda) \) consists of a vertex set \( V \), an edge set \( E \), a vertex labeling \( \varphi : V \rightarrow \Sigma \) and an edge labeling \( \psi : E \rightarrow \Lambda \). We allow multiple edges in \( g \) and \( g \) is directed or undirected. We use lower case letters for representing colored-graphs.

Let \( X \) be a finite alphabet whose elements are called variables. We denote variables by \( x, y, \ldots \). We assume that each variable \( x \) in \( X \) has the rank, denoted by \( \text{rank}(x) \), that is a nonnegative integer. Assume that \( \Sigma \cap X = \emptyset \) and \( \Lambda \cap X = \emptyset \).
Definition 3.1 A directed term graph \( g = (V, E, \varphi, \psi, H, \lambda, \text{ports}) \) over \( \langle \Sigma, \Lambda, X \rangle \) consists of the following:

1. \( (V, E, \varphi, \psi) \) is a directed colored-graph over \( \langle \Sigma, \Lambda \rangle \).
2. \( H \) is a finite set whose elements are called hyperedges.
3. \( \lambda : H \to X \) is a labeling function. For a hyperedge \( e \in H \), \( \text{rank}(\lambda(e)) \) is called the rank of \( e \).
4. \( \text{ports} : H \to V^* \) is a mapping such that for every \( e \in H \), \( \text{ports}(e) \) is a list of \( \text{rank}(\lambda(e)) \) distinct vertices. These vertices are called the ports of \( e \).

We denote term graphs by \( f, g, \ldots \). A term graph is called ground if \( H = \emptyset \). We denote by \( T(\Sigma, \Lambda, X) \) the family of term graphs over \( \langle \Sigma, \Lambda, X \rangle \) and by \( T(\Sigma, \Lambda) \) the family of all ground term graphs. A term graph \( g = (V, E, \varphi, \psi, H, \lambda, \text{ports}) \) is a star term graph for a variable \( x \) if \( E = \emptyset \) and \( H \) consists of a unique hyperedge \( e \) labeled with \( x \) such that the set of ports of \( e \) is \( V \).

We can define an undirected term graph in the same way.

We identity directed (or undirected) term graphs without hyperedge with colored-graphs.

Example 3.1 We draw directed and undirected term graphs as in Figure 3.1 (a) and (b), respectively.

Let \( g_1, \ldots, g_n \) be term graphs in \( T(\Sigma, \Lambda, X) \). An atom is an expression of the form \( p(g_1, \ldots, g_n) \), where \( p \) is a predicate symbol with arity \( n \). Hereafter we use \( p, q, \ldots \) to denote predicate symbols. Let \( A, B_1, \ldots, B_n \) be atoms, where \( n \geq 0 \). Then, a graph rewriting rule is a clause of the form \( A \leftarrow B_1, \ldots, B_n \). We call the atom \( A \) the head and the part \( B_1, \ldots, B_n \) the body of the graph rewriting rule. Especially, a graph rewriting rule of the form \( A \leftarrow \) is called a fact.
Definition 3.2 A formal graph system (FGS) is a finite set of graph rewriting rules.

Example 3.2 We draw an FGS $\Gamma_{st}$ as Figure 3.2 generating the family $S$ of all Sierpiński triangles.

We give some notions for term graphs and atoms. A term graph $f = (V_f, E_f, \varphi_f, \psi_f, H_f, \lambda_f, ports_f)$ is a subgraph of $g = (V_g, E_g, \varphi_g, \psi_g, H_g, \lambda_g, ports_g)$ if $f' = (V_f, E_f, \varphi_f, \psi_f)$ is a subgraph of $g' = (V_g, E_g, \varphi_g, \psi_g)$, $H_f \subseteq H_g$, and $\lambda_f(e) = \lambda_g(e)$ and $ports_f(e) = ports_g(e)$ for all $e \in H_f$.

Let $g_1 = (V_1, E_1, \varphi_1, \psi_1, H_1, \lambda_1, ports_1)$ and $g_2 = (V_2, E_2, \varphi_2, \psi_2, H_2, \lambda_2, ports_2)$ be term graphs. We say that $g_1$ and $g_2$ are isomorphic, denoted by $g_1 \simeq g_2$, if the following conditions are satisfied:

1. The colored-graphs $g'_1 = (V_1, E_1, \varphi_1, \psi_1)$ and $g'_2 = (V_2, E_2, \varphi_2, \psi_2)$ are isomorphic including vertex and edge labelings.

2. Let $\pi : V_1 \to V_2$ be the bijection giving the isomorphism in (1). There is a bijection $\varpi : H_1 \to H_2$ such that for every $e \in H_1$, $ports_2(\varpi(e)) = \pi'(ports_1(e))$ and
Figure 3.2: FGS $\Gamma_{st}$ generating the family $S$ of all Sierpinski triangles.
\[ \lambda_2(\varpi(e)) = \lambda_1(e), \text{ where } \pi^* \text{ is defined as } \pi^*((v_1, \ldots, v_m)) = (\pi(v_1), \ldots, \pi(v_m)). \]

Let \((v_1, \ldots, v_r)\) and \((u_1, \ldots, u_r)\) be two lists of \(r\) distinct vertices of \(g_1\) and \(g_2\), respectively. We also say that \([g_1, (v_1, \ldots, v_r)]\) and \([g_2, (u_1, \ldots, u_r)]\) are isomorphic if (1) and (2) are satisfied and \(\pi(v_i) = u_i\) for each \(1 \leq i \leq r\). We call a pair \([g, \sigma]\) of a term graph \(g\) over \((\Sigma, \Lambda, X)\) and a list \(\sigma\) of \(r\) distinct vertices in \(g\) an \(r\)-hypergraph over \((\Sigma, \Lambda, X)\). For any two atoms \(p(f_1, \ldots, f_n)\) and \(p(g_1, \ldots, g_n)\), we denote \(p(f_1, \ldots, f_n) \simeq p(g_1, \ldots, g_n)\) if \(f_i \simeq g_i\) for each \(1 \leq i \leq n\).

Let \(g = (V_g, E_g, \varphi_g, \psi_g, H_g, \lambda_g, \text{ports}_g)\) and \(f = (V_f, E_f, \varphi_f, \psi_f, H_f, \lambda_f, \text{ports}_f)\) be term graphs, \(e\) be a hyperedge in \(H_f\) of rank \(r\) with ports \((u_1, \ldots, u_r)\), and \([g, \sigma]\) be an \(r\)-hypergraph with \(\sigma = (v_1, \ldots, v_r)\). The hyperedge replacement \(e \leftarrow [g, \sigma]\) is the following operation on \(f\): The term graph obtained by the hyperedge replacement \(e \leftarrow [g, \sigma]\) on \(f\), denoted by \(f(e \leftarrow [g, \sigma]) = (V, E, \varphi, \psi, H, \lambda, \text{ports})\) is defined in the following way: Let \(g' = (V'_g, E'_g, \varphi'_g, \psi'_g, H'_g, \lambda'_g, \text{ports}'_g)\) be a copy of \(g\). For a vertex \(v \in V_g\), we denote the corresponding copy vertex by \(v'\). We attach \(g'\) to \(f\) by removing the hyperedge \(e\) from \(H_f\) and by identifying the ports \(u_1, \ldots, u_r\) of \(e\) in \(f\) with \(v'_1, \ldots, v'_r\) in \(g'\), respectively. We set \(\varphi(u_i) = \varphi_f(u_i)\) for each \(1 \leq i \leq r\), i.e., the label of \(u_i\) in \(f\) is used for the new term graph \(f(e \leftarrow [g, \sigma])\). Let \(\Upsilon = \{e_1 \leftarrow [g_1, \sigma_1], \ldots, e_m \leftarrow [g_m, \sigma_m]\}\) be a set of hyperedge replacements on \(f\). We denote by \(f(\Upsilon)\) the term graph obtained by applying all hyperedge replacements in \(\Upsilon\) in parallel.

Let \(x\) be a variable in \(X\) with \(\text{rank}(x) = r\). Let \(g = (V_g, E_g, \varphi_g, \psi_g, H_g, \lambda_g, \text{ports}_g)\) be a term graph in \(T(\Sigma, \Lambda, X)\) and \([g, \sigma]\) be an \(r\)-hypergraph. We call the form \(x := [g, \sigma]\) a binding for \(x\). We say that a binding \(x := [g, \sigma]\) is trivial if \(g\) is a term graph consisting of \(r\) vertices without any edges such that it has a unique hyperedge \(e \in H\) with \(\lambda(e) = x\). A substitution \(\theta\) is a finite collection of bindings \(\{x_1 := [g_1, \sigma_1], \ldots, x_n := [g_n, \sigma_n]\}\), where \(x_i\)'s are mutually distinct variables in \(X\) and each \(g_i\) (\(1 \leq i \leq n\)) has no hyperedge labeled with a variable in \(\{x_1, \ldots, x_n\}\).
Let \( f = (V_f, E_f, \varphi_f, \psi_f, H_f, \lambda_f, \text{port}_f) \) be a term graph and \( \theta = \{ x_1 := [g_1, \sigma_1], \ldots, x_n := [g_n, \sigma_n] \} \) be a substitution. For \( x_i \), let \( H_{\lambda_f}(x_i) = \{ e \in H_f \mid \lambda_f(e) = x_i \} \) and \( \Upsilon_i = \{ e \leftarrow [g_i, \sigma_i] \mid e \in H_{\lambda_f}(x_i) \} \). Then let \( \Upsilon_\theta = \bigcup_{i=1}^n \Upsilon_i \). The term graph \( f\theta \) called the instance of \( f \) by \( \theta \) is defined by \( f(\Upsilon_\theta) \). We remark that the set of the hyperedges in \( f\theta \) consists of the hyperedges in \( H_f \) which are not in \( H_{\lambda_f}(x_1) \cup \cdots \cup H_{\lambda_f}(x_n) \) and the newly added hyperedges of the graphs attached to \( f \) by the substitution \( \theta \).

**Example 3.3** Let \( f, g \) and \( h \) be term graphs given in Figure 3.3, and \( \theta = \{ x := [g, (u_1, u_2)], y := [h, (v_1, v_2)] \} \) be a substitution, where \( u_1 \) and \( u_2 \) (resp., \( v_1 \) and \( v_2 \)) are vertices in \( g \) (resp., \( h \)). Then the instance \( f\theta \) is a term graph shown in Figure 3.3.

We introduce some notions similar to those in logic programming [75]. Let \( \theta = \{ x_1 := [g_1, \sigma_1], \ldots, x_n := [g_n, \sigma_n] \} \) and \( \tau = \{ y_1 := [h_1, \sigma'_1], \ldots, y_m := [h_m, \sigma'_m] \} \) be substitutions. Then the composition \( \theta \tau \) of \( \theta \) and \( \tau \) is the substitution obtained from \( \{ x_1 := [g_1 \tau, \sigma_1], \ldots, x_n := [g_n \tau, \sigma_n], y_1 := [h_1, \sigma'_1], \ldots, y_m := [h_m, \sigma'_m] \} \) by deleting the following bindings:

(i) \( x_i := [g_i \tau, \sigma_i] \) which is trivial.
(ii) \( y_j := [h_j, \sigma'_j] \) with \( y_j = x_k \) for some \( x_k \).

Let \( \theta_1, \ldots, \theta_n \) be substitutions. We denote a composition \( \theta_1 \cdots \theta_n \) by \( \prod_{i=1}^n \theta_i \). The substitution given by the empty set is called the identity substitution and is denoted by \( \varepsilon \). The elementary properties of substitutions are given in the following proposition without proofs.

**Proposition 3.1** Let \( \theta, \tau \) and \( \gamma \) be substitutions.

(a) \( \varepsilon \theta = \varepsilon \theta = \theta \).

(b) \( (g\tau)\gamma = g(\tau\gamma) \) for any term graph \( g \).

(c) \( (\theta\tau)\gamma = \theta(\tau\gamma) \).

**Proof.** (a) This follows immediately from the definition of \( \varepsilon \).

(b) Clearly it suffices to prove the result when \( g \) is a star term graph for a variable \( x \).

Let \( \tau = \{ x_1 := [g_1, \sigma_1], \ldots, x_n := [g_n, \sigma_n] \} \) and \( \gamma = \{ y_1 := [h_1, \sigma'_1], \ldots, y_m := [h_m, \sigma'_m] \} \) be substitutions. If \( x \not\in \{ x_1, \ldots, x_n \} \cup \{ y_1, \ldots, y_m \} \) then \( (g\tau)\gamma = g = g(\tau\gamma) \). If \( x \in \{ x_1, \ldots, x_n \} \), say \( x = x_i \), then \( (g\tau)\gamma = g_i\gamma = g(\tau\gamma) \). If \( x \in \{ y_1, \ldots, y_m \} \) then \( (g\tau)\gamma = g\gamma = h_j = g(\tau\gamma) \).

(c) Clearly if suffices to show that if \( g \) is a star term graph for a variable \( x \) then
\[
g((\theta\tau)\gamma) = g(\theta(\tau\gamma)).
\]
By (b), we show that
\[
g((\theta\tau)\gamma) = (g\theta(\tau))\gamma = (g\theta)(\tau\gamma) = g(\theta(\tau\gamma)).
\]

For a substitution \( \theta \), we define \( p(f_1, \ldots, f_n)\theta = p(f_1\theta, \ldots, f_n\theta) \) and \( (A \leftarrow B_1, \ldots, B_m)\theta = A\theta \leftarrow B_1\theta, \ldots, B_m\theta \).

Let \( g_1 \) and \( g_2 \) are two term graphs or atoms. Then a substitution \( \theta \) is a unifier of \( g_1 \) and \( g_2 \) if \( g_1\theta \simeq g_2\theta \). If \( g_1 \simeq g_2\theta \) and \( g_1\theta' \simeq g_2 \) for some substitutions \( \theta \) and \( \theta' \), \( g_1 \) is called a variant of \( g_2 \). A goal is a graph rewriting rule of the form \( \leftarrow B_1, \ldots, B_m \) \((m \geq 0) \). If \( m = 1 \), "\( \leftarrow B_1 \)" is called a unit goal. If \( m = 0 \), we denote it "\( \Box \)" and call it an empty
goal. We assume a computation rule $Q$ to select an atom from a goal. For a graph rewriting rule $C$, let $\text{var}(C)$ be the set of all variables labeling the hyperedges of the term graphs in $C$.

Let $\Gamma$ be an FGS, and $D$ a goal. A derivation from $D$ is a (finite or infinite) sequence of triples $(D_i, \theta_i, C_i)$ ($i = 0, 1, \ldots$) that satisfies the following conditions:

(3) $D_i$ is a goal, $\theta_i$ is a substitution, $C_i$ is a variant of a graph rewriting rule in $\Gamma$, and $D_0 = D$.

(4) $\text{var}(C_i) \cap \text{var}(C_j) = \emptyset$ ($i \neq j$), and $\text{var}(C_i) \cap \text{var}(D) = \emptyset$ for every $i$.

(5) Let $D_i$ be $\leftarrow A_1, \ldots, A_k$ and $A_m$ be the atom selected by $Q$.

Let $C_i$ be $A \leftarrow B_1, \ldots, B_q$. Then $\theta_i$ is a unifier of $A$ and $A_m$ and $D_{i+1}$ is the goal $\leftarrow A_1\theta_i, \ldots, A_m\theta_i, B_1\theta_i, \ldots, B_q\theta_i, A_{m+1}\theta_i, \ldots, A_k\theta_i$.

A refutation is a finite derivation ending with the empty goal.

Definition 3.3 Let $\Gamma$ be an FGS. We define the relation $\Gamma \vdash C$ for a rule $C$ inductively as follows:

(1) If $\Gamma \ni C$, then $\Gamma \vdash C$.

(2) If $\Gamma \vdash C$, then $\Gamma \vdash C\theta$ for any substitution $\theta$.

(3) If $\Gamma \vdash A \leftarrow B_1, \ldots, B_n$ and $\Gamma \vdash B_i \leftarrow C_1, \ldots, C_m$,

then $\Gamma \vdash A \leftarrow B_1, \ldots, B_{i-1}, C_1, \ldots, C_m, B_{i+1}, \ldots, B_n$.

A rule $C$ is provable from $\Gamma$ if $\Gamma \vdash C$. For an FGS $\Gamma$ and its predicate symbol $p$ with arity $n$, we define $\text{GL}(\Gamma, p) = \{(h_1, \ldots, h_n) \in (T(\Sigma, \Lambda))^n | \Gamma \vdash p(h_1, \ldots, h_n) \leftarrow \}$. In case $n = 1$, $\text{GL}(\Gamma, p)$ defines a subset of $T(\Sigma, \Lambda)$ called a graph language. We say that a graph language $L \subseteq T(\Sigma, \Lambda)$ is definable by FGS or an FGS language if such a pair $(\Gamma, p)$ exists.
Let $SS(\Gamma)$ is the set of all ground atoms such that there exists a refutation from $\leftarrow A$ in $\Gamma$. Let $PS(\Gamma)$ be the set of all ground atoms which are provable from $\Gamma$. Then we can prove the following proposition in the same way as logic programs (see [75]).

**Proposition 3.2** For any FGS $\Gamma$, $SS(\Gamma) = PS(\Gamma)$.

### 3.2 Regular FGSs Characterizing Hyperedge Replacement Grammars

This section introduces a subclass of FGSs called regular FGSs, and characterizes hyperedge replacement grammars [49, 50, 71, 113] by regular FGSs. Hyperedge replacement is one of the most elementary and frequently used concepts of graph transformation with the characteristics of context-free rewriting.

**Definition 3.4** Let $\Delta$ be a subset of $\Sigma$ called a pointer set. A regular term graph with a pointer set $\Delta$ is a term graph $g$ such that there is at most one hyperedge labeled with $x$ for each $x \in X$ and there is at most one vertex labeled with $s$ for each $s \in \Delta$.

**Definition 3.5** Let $\Gamma$ be an FGS and $\Pi$ be the set of predicate symbols in $\Gamma$. Then $\Gamma$ is said to be regular if every predicate symbol in $\Gamma$ is unary and there is a sequence $reg(p)$ of distinct symbols in $\Sigma$ for every $p \in \Pi$ such that each graph rewriting rule $p_0(g_0) \leftarrow p_1(g_1), \ldots, p_n(g_n)$ in $\Gamma$ satisfies the following conditions:

1. Each term graph $g_i = (V_i, E_i, \varphi_i, \psi_i, H_i, \lambda_i, ports_i) \ (0 \leq i \leq n)$ is a regular term graph with respect to $\Delta = \bigcup_{p \in \Pi} \{a \mid a$ is a symbol of $reg(p)\}$.

2. For every label $a$ of vertices in $g_0$, if $a$ is in $\Delta$, then $a$ is a symbol of $reg(p_0)$.

3. For $1 \leq i \leq n$, $g_i$ is a star term graph for a variable of rank $m_i$ such that the $j$th port $(1 \leq j \leq m_i)$ of the hyperedge in $H_i$ is labeled with the $j$th symbol of $reg(p_i)$, where $m_i$ is the length of $reg(p_i)$. 

27
(4) $\lambda_0(H_0) = \bigcup_{1 \leq i \leq n} \lambda_i(H_i)$ and $\lambda_i(H_i) \cap \lambda_j(H_j) = \emptyset$ for $1 \leq i < j \leq n$.

We extend hyperedge replacement grammars in Habel and Kreowski [50] and Lautemann [71] by introducing vertex and edge labelings in such a way that these labels do not interfere with hyperedge replacement mechanism.

Let $N$ be a finite alphabet. We call an element of $N$ a nonterminal. Nonterminals are denoted by capital letters $A, B, C, \cdots$. We assume that each nonterminal $A$ in $N$ has the rank, denoted by $\text{rank}(A)$, that is a nonnegative integer.

**Definition 3.6 (Habel and Kreowski [50] and Lautemann [71])** A hyperedge replacement grammar (HRG) $G = (S, R)$ is defined as follows:

1. $S$ is a nonterminal in $N$ with $\text{rank}(S) = 0$, called the start symbol.

2. $R$ is a finite set of productions of the form $A \rightarrow [g, \sigma]$, where $A$ is a nonterminal in $N$ with $\text{rank}(A) = r$ and $[g, \sigma]$ is an $r$-hypergraph over $(\Sigma, \Lambda, N)$.

**Example 3.4** We draw an HRG $G = (S, R)$ as Figure 3.4, where $R = \{ S \rightarrow T_0, T \rightarrow [T_1, (v_1, v_2, v_3)], T \rightarrow [T_2, (v_1, v_2, v_3)]\}$.

Let $G = (S, R)$ be an HRG. For $A \in N$, an $r$-hypergraph $[g, \sigma]$ over $(\Sigma, \Lambda, N)$, and $i \geq 1$, we define the relation $A \rightarrow^i [g, \sigma]$ inductively as follows:

1. We denote $A \rightarrow^1 [g, \sigma]$ if there is a production $A \rightarrow [g, \sigma]$ is in $R$.

2. For $i \geq 2$, we denote $A \rightarrow^i [g, \sigma]$ if there are $j, l \geq 1$, an $r$-hypergraph $[f, \sigma]$, a hyperedge $e$ in $f$ of rank $s$ with label $B$, and an $s$-hypergraph $[h, \sigma']$ such that $j + l = i$, $A \rightarrow^j [f, \sigma]$, $B \rightarrow^l [h, \sigma']$, and $[g, \sigma] = [f(e \leftarrow [h, \sigma']), \sigma]$.

We write $A \rightarrow^+ [g, \sigma]$ if $A \rightarrow^i [g, \sigma]$ for some $i \geq 1$. The graph language generated by an HRG $G = (S, R)$ is the set $L(G) = \{ g \mid g$ is a term graph in $T(\Sigma, \Lambda) and S \rightarrow^+ g\}$. A set $L$ of term graphs in $T(\Sigma, \Lambda)$ is called an HR language if $L = L(G)$, for some HRG $G$. 28
Figure 3.4: HRG $\mathcal{G} = (S, R)$ with $R = \{ S \rightarrow T_0, T \rightarrow [T_1, (v_1, v_2, v_3)], T \rightarrow [T_2, (v_1, v_2, v_3)] \}$
Theorem 3.1 A graph language $L$ is definable by a regular FGS if and only if $L$ is an HR language.

Proof. Let $G = (S, R)$ be an HRG. Let $N$ be the set of nonterminals of $G$. Let $\Sigma$ and $\Lambda$ be the sets of labels of vertices and edges of the term graphs in $R$, respectively.

We define a regular FGS $\Gamma_G$ in the following way: We regard each nonterminal $A$ in $N$ as a unary predicate symbol. For a unary predicate symbol $A$, let $reg(A)$ be a sequence of $\text{rank}(A)$ distinct symbols not in $\Sigma$. Then let $\Delta = \bigcup_{A \in N} \{ a \mid a \text{ is a symbol in } reg(A) \}$.

For each production $A \rightarrow [g, (v_1, \ldots, v_r)]$ of $G$, let $g = (V, E, \varphi, \psi, H, \lambda, \text{ports})$ and let $H = \{ e_1, \ldots, e_n \}$. Then $\Gamma_G$ contains the graph rewriting rule

$$A(h) \leftarrow \lambda(e_1)(f_1), \ldots, \lambda(e_n)(f_n),$$

where regular term graphs $h$ and $f_1, \ldots, f_n$ with respect to $\Delta$ are defined as follows (see Example 3.5):

(a) $h = (V, E, \varphi', \psi, H', \lambda', \text{ports})$ is defined by modifying the vertex and hyperedge labelings of $g$ as follows:

(i) For hyperedges $e_1, \ldots, e_n$ in $H$, let $\lambda'(e_1), \ldots, \lambda'(e_n)$ be distinct variables of $\text{rank}(\lambda(e_1)), \ldots, \text{rank}(\lambda(e_n))$, respectively.

(ii) For each vertex $v_j$ ($1 \leq j \leq r$), $v_j$ is labeled with the $j$th symbol of $\text{reg}(A)$.

For other vertices in $V$, the same labeling as $\varphi$ is used for $\varphi'$.

Then $h$ is a regular term graph over $(\Sigma \cup \Delta, \Lambda, \{ \lambda'(e_1), \ldots, \lambda'(e_n) \})$ with respect to $\Delta$.

(b) For $1 \leq i \leq n$, $f_i$ is a star term graph for a variable $\lambda'(e_i)$ such that the $j$th port of the hyperedge in $f_i$ is labeled with the $j$th symbol in $\text{reg}(\lambda(e_i))$ for $1 \leq j \leq \text{rank}(\lambda'(e_i))$. 

30
It is easy to see that \( \Gamma_G \) consisting of these graph rewriting rules is regular.

Let \( f_1 \) and \( f_2 \) be \( r_1 \)- and \( r_2 \)-hypergraphs over \( \langle \Sigma, \Lambda, N \rangle \), respectively, and let \( e \) be a hyperedge in \( f_1 \) with label \( A \). Let \( P \) be a production \( A \rightarrow [h, \sigma] \) in \( R \). Then we denote \( f_1 \Rightarrow (P, e) f_2 \), called a directed derivation from \( f_1 \) to \( f_2 \), if there exists \( i \geq 1 \) such that \( S \rightarrow^i f_1, S \rightarrow^{i+1} f_2, A \rightarrow^1 [h, \sigma] \) and \( f_2 = f_1(e \leftarrow [h, \sigma]) \).

A sequence of directed derivations of the form \( g_1 \Rightarrow (P_1, e_1) \cdots \Rightarrow (P_k, e_k) \) \( g_{k+1} \) is called a derivation of length \( k \) from \( g_1 \) to \( g_{k+1} \) and is denoted by \( g_1 \Rightarrow^k g_{k+1} \).

Let \( G = (S, R) \) be an HRG and \( \Gamma_G \) be the regular FGS constructed from \( G \). Then we show that if \( L \) is an HR language \( L(G) \) then \( L \) is definable by the FGS \( \Gamma_G \).

Let \( g = (V_g, E_g, \varphi_g, \psi_g, H_g, \lambda_g, \text{ports}_g) \) be a term graph over \( \langle \Sigma, \Lambda, N \rangle \). We call a term graph \( g' = (V_{g'}, E_{g'}, \varphi_{g'}, \psi_{g'}, H_{g'}, \lambda_{g'}, \text{ports}_{g'}) \) over \( \langle \Sigma, \Lambda, X \rangle \) an \( X \)-term graph of \( g \) where for each hyperedge \( e \in H_g \), \( \lambda_{g'}(e) \) is a distinct variable in \( X \) whose rank is \( \text{rank}(\lambda_g(e)) \). Let \( g_1 \) and \( g_2 \) be term graphs over \( \langle \Sigma, \Lambda, N \rangle \), \( P \) a production \( A \rightarrow [f, \sigma] \) of \( G \) and \( e \) a hyperedge in \( g_1 \). Then we assume that \( g_1 \Rightarrow (P, e) g_2 \) on \( G \). Let \( g'_1 = (V'_1, E'_1, \varphi'_1, \psi'_1, H'_1, \lambda'_1, \text{ports}_{s_1}) \), \( g'_2 = (V'_2, E'_2, \varphi'_2, \psi'_2, H'_2, \lambda'_2, \text{ports}_{s_2}) \), and \( f' = (V'_f, E'_f, \varphi'_f, \psi'_f, H'_f, \lambda'_f, \text{ports}_{f'}) \) be \( X \)-term graphs of \( g_1 \), \( g_2 \), \( f \), respectively, and \( \sigma' \) be a copy of \( \sigma \) on \( f' \), such that \( \{\lambda'_1(e) \mid e \in H'_1\} \cap \{\lambda'_f(e) \mid e \in H'_f\} = \emptyset \) and \( g'_1(e \leftarrow [f', \sigma']) = g'_2 \). Then we call a substitution \( \tau = \{\lambda'_i(e) := [f', \sigma']\} \) a simulated substitution for \( g_1 \Rightarrow (P, e) g_2 \).

Let \( g_1 \Rightarrow (P_1, e_1) \cdots \Rightarrow (P_{n-1}, e_{n-1}) g_n (= g) \) be a derivation of length \( n - 1 \) on \( G \) for a ground term graph \( g \in L \), and for each \( 1 \leq i \leq n - 1 \), \( \tau_i \) be a simulated substitution for \( g_i \Rightarrow (P_i, e_i) g_{i+1} \). Then we show that a sequence \( F' = \{(\leftarrow H_i, \theta_i, C_i)\}_{0 \leq i \leq n-1} \), which is given inductively as follows, is a derivation from the goal \( \leftarrow S(g) \) in the FGS \( \Gamma_G \):

1. \( H_0 = S(g) \), \( C_0 \) is a variant of the graph rewriting rule in \( \Gamma_G \) constructed from the production \( S \rightarrow g_1 \), and \( \theta_0 = \tau_1 \cdots \tau_{n-1} \).
(2) For $i \geq 2$, $C_i = (A^i(h^i) \leftarrow B^i_1(f^i_1), \ldots, B^i_{m^i}(f^i_{m^i}))$ is a variant of the graph rewriting rule in $\Gamma_g$ constructed from the production $P_{i-1}$ in $G$, $\theta_i = \tau_i \cdots \tau_{n-1}$, and $H_i = H_{i-1} \cup \{B^i_1(f^i_1)\theta_i, \ldots, B^i_{m^i}(f^i_{m^i})\theta_i\} - \{A^i(h^i)\theta_i\}$, where $P_0$ is the production $S \rightarrow g_1$.

It can be shown by the definition of the simulated substitution that the above sequence $F$ is a derivation from the goal $\leftarrow S(g)$ on the FGS $\Gamma$.

Conversely, let $(\Gamma, p)$ be a pair of a regular FGS and its predicate symbol $p$. Without loss of generality, we may assume that $p$ is the only predicate symbol in $\Gamma$ that does not appear in the body of any graph rewriting rule in $\Gamma$. Let $\Sigma$ and $\Lambda$ be the sets of labels of vertices and edges of the term graphs in $\Gamma$, respectively.

We define an HRG $G_\Gamma = (p, R_\Gamma)$ in the following way: Let $\Pi$ be the set of predicate symbols of $\Gamma$. Since $\Gamma$ is regular, every predicate symbol $q$ in $\Pi$ is unary and there is a sequence $\text{reg}(q)$ of distinct symbols in $\Sigma$ such that each graph rewriting rule in $\Gamma$ satisfies the conditions of Definition 3.5. Here we note that $\text{reg}(p)$ is the empty sequence. We regard a predicate symbol $q$ in $\Pi$ as a nonterminal. For a nonterminal $q$, let the rank of $q$ be the length of $\text{reg}(q)$. Especially, $p$ is the start symbol of $G_\Gamma$.

For each graph rewriting rule $q_0(g_0) \leftarrow q_1(g_1), \ldots, q_n(g_n)$ of $\Gamma$, let $g_i = (V_i, E_i, \varphi_i, \psi_i, H_i, \lambda_i, \text{ports}_i)$ and $\tau_i$ be the rank of $q_i$ for $0 \leq i \leq n$. Then $R_\Gamma$ contains the production $q_0 \rightarrow [h, (v_1, \ldots, v_{r_0})]$, where a term graph $h$ over $\langle \Sigma, \Lambda, \Pi \rangle$ and the sequence $(v_1, \ldots, v_{r_0})$ of vertices in $h$ are defined by modifying the hyperedge labeling of $g_0$ as follows:

(i) Let $g_i$ ($1 \leq i \leq n$) be the term graph with the hyperedge labeled with $\lambda_0(e)$.

Then let the new label of $e$ in $h$ be $q_i$.

(ii) Let $(v_1, \ldots, v_{r_0})$ be the sequence of vertices in $h$ with $(\varphi_0(v_1), \ldots, \varphi_0(v_{r_0})) = \text{reg}(q_0)$. 

32
Then $h$ is a term graph over $(\Sigma, \Lambda, \Pi)$ and $[h, (v_1, \ldots, v_n)]$ is an $\tau_p$-hypergraph.

Since $\Gamma$ is regular, it is easy to see that $G = (p, R)$ consisting of the start symbol $p$ and these productions is an HRG.

We show that if a graph language is definable by a regular FGS $\Gamma$ and a unary predicate symbol $p$ in $\Gamma$ then it is a graph language generated by an HRG $G$. For a ground term graph $g$ in $GL(\Gamma, p)$, there is a derivation $D$ from the goal $\rightarrow p(g)$. It is easy to construct a derivation on the HRG $G$ from $D$. Then, we can show that $GL(\Gamma, p)$ is definable by the HRG $G = (p, R)$.

**Example 3.5** Let $G = (S, R)$ be an HRG given in Figure 3.4. Then the regular FGS $\Gamma_g$ obtained from $G$ is given in Figure 3.5.

It is known in [51, 71, 88] that the families of trees, two-terminal series parallel graphs, graphs homeomorphic to a given graph, outerplanar graphs, graphs of cyclic bandwidth $\leq 2$ and $k$-decomposable graphs are generated by HRGs. By Theorem 3.1, these families are also definable by regular FGSs. Habel and Kreowski [50] has given a
pumping lemma for hyperedge replacement grammars which generalizes those pumping
lemmas for context-free string languages [55] and context-free graph languages [25, 66, 111]. By using the pumping lemma, we can see that HRGs cannot generate the families
of balanced binary trees and Sierpinski triangles. Hence, regular FGSs cannot generate
those families.

### 3.3 Simple FGSs

This section defines simple FGSs and bounded simple FGSs as a subclass of simple
FGSs. Let $g = (V, E, \varphi, \psi)$ be a ground term graph. For a vertex $v \in V$, the degree
of $v$ is the number of edges having $v$ as an endpoint and is denoted by $\text{deg}(v)$. If $g$ is
directed, the degree of $v$ is the number of edges beginning from $v$ or ending to $v$. Then
the valence of $g$ is $\max\{\text{deg}(v) \mid v \in V\}$.

**Definition 3.7** Let $\Delta$ be a subset of $\Lambda$ called an enclosure set. A term graph $g =
(V, E, \varphi, \psi, H, \lambda, \text{ports})$ is simple with $\Delta$ if $g$ satisfies the following conditions:

1. If a vertex $v$ is a port of two distinct hyperedges in $H$, then any edge in $E$ having
   $v$ as an endpoint is not labeled with any symbol in $\Delta$.

2. For any hyperedge $e \in H$, if a port $v$ of $e$ is an endpoint of an edge labeled with
   a symbol in $\Delta$, then every edge adjoining to $v$ is labeled with a symbol in $\Delta$.

**Definition 3.8** Let $\Delta$ be an enclosure set. An FGS $\Gamma$ is said to be simple with $\Delta$ if,
for each predicate symbol $p$ of $\Gamma$ with the arity $n$, there is a sequence $S(p)$ of $n$ ground
term graphs in $T(\Sigma, \Delta)$ such that each term graph in $S(p)$ has no vertex whose degree
is 0 and that each graph rewriting rule

$$q_0(g_1, \ldots, g_n) \leftarrow q_1(g_1', \ldots, g_{n_1}'), \ldots, q_k(g_k, \ldots, g_{n_k}')$$

in $\Gamma$ satisfies the conditions (1)–(4): For each $0 \leq i \leq k$, let $S(g_i) = (f_{i_1}, \ldots, f_{i_{n_i}})$. 

34
(1) For each \(0 \leq i \leq k\) and \(1 \leq j \leq l_i\), the subgraph of \(g^i_j\) induced by the all edges labeled with a symbol in \(\Delta\) is isomorphic to the ground term graph \(f^i_j\).

(2) For \(1 \leq j \leq l_0\), the term graph \(g^i_0 = (V^i_0, E^i_0, \varphi^i_0, \psi^i_0, H^i_0, \lambda^i_0, \text{ports}^i_0)\) is simple with \(\Delta\).

(3) For \(1 \leq i \leq k\) and \(1 \leq j \leq l_i\), the term graph \(g^i_j = (V^i_j, E^i_j, \varphi^i_j, \psi^i_j, H^i_j, \lambda^i_j, \text{ports}^i_j)\) is a simple term graph with \(\Delta\) satisfying the following conditions:
   (i) Every edge in \(E^i_j\) is labeled with a symbol in \(\Delta\).
   (ii) There exists a term graph \(g^0_l (1 \leq l \leq l_0)\) which has a subgraph isomorphic to \(g^i_j\) where the vertex and edge labelings are ignored.

(4) \(\bigcup_{j=1}^{j=k} \lambda^i_j (H^i_0) \subseteq \bigcup_{i=1}^{i=n} \lambda^i_j (H^i_j)\).

**Example 3.6** We draw a simple FGS \(\Gamma\) as Figure 3.6. An edge labeled with a symbol in the enclosure set is shown by a broken line.

By the definition of simple FGSs, we show the following lemma without proof.

**Lemma 3.1** Let \(\Gamma\) be a simple FGS and \(p\) be a predicate symbol in \(\Gamma\). Then any term graph in \(GL(\Gamma, p)\) is of constantly bounded valence.

Let \(g = (V, E, \varphi, \psi, H, \lambda, \text{ports})\) be a term graph and \(\tilde{V}\) be the set of vertices which are not ports of any hyperedges in \(H\). Then we define the size of \(g\) as a pair \((\#\tilde{V}, \#E)\) of the numbers \(\#\tilde{V}\) and \(\#E\) of vertices in \(\tilde{V}\) and edges in \(E\), respectively. Let \(g_i\) be a term graph and \((\#\tilde{V}_i, \#E_i)\) be the size of \(g_i\) for \(1 \leq i \leq n\). Then, for an atom \(p(g_1, \ldots, g_n)\), we define \(\|p(g_1, \ldots, g_n)\| = (\#\tilde{V}_1 + \cdots + \#\tilde{V}_n, \#E_1 + \cdots + \#E_n)\). Let \(A_1, \ldots, A_n\) be atoms and let \(\|A_i\| = (s_i, t_i)\) for \(1 \leq i \leq n\). Then, we define \(\|A_1\| \geq \|A_2\| + \cdots + \|A_n\|\) if \(s_1 \geq s_2 + \cdots + s_n\) and \(t_1 \geq t_2 + \cdots + t_n\).
Figure 3.6: Simple FGS $\Gamma$. The broken lines represent the edges labeled with elements of an enclosure set.
Definition 3.9 A graph rewriting rule $A \leftarrow B_1, \ldots, B_m$ is said to be size-bounded if $\|A\theta\| \geq \|B_1\theta\| + \cdots + \|B_m\theta\|$ for any substitution $\theta$. An FGS $\Gamma$ is size-bounded if every graph rewriting rule in $\Gamma$ is size-bounded.

For example, the simple FGS $\Gamma$ in Figure 3.6 and the FGS $\Gamma_{st}$ in Figure 3.2 generating the family of all Sierpinski triangles are size-bounded.

Let $g = (V, E, \varphi, \psi, H, \lambda, \text{ports})$ be a term graph. For a variable $x \in X$, the number of hyperedges in $H$ labeled with $x$ is denoted by $o_x(g)$. For an atom $p(g_1, \ldots, g_n)$ and a variable $x \in X$, we define $o_x(p(g_1, \ldots, g_n)) = o_x(g_1) + \cdots + o_x(g_n)$. A graph rewriting rule $A \leftarrow B_1, \ldots, B_m$ is said to be occurrence-bounded if $o_x(A) \leq 1$ and $o_x(B_1) + \cdots + o_x(B_m) \leq 1$ for each variable $x \in X$.

Definition 3.10 An FGS $\Gamma$ is bounded if every graph rewriting rule in $\Gamma$ is size-bounded and occurrence-bounded.

The two classes of FGS, simple FGSs and bounded simple FGSs, are very interesting from the complexity point of view. In Chapter 4, we consider the complexity of deciding whether a given connected graph is generated by a fixed simple FGS and a fixed bounded simple FGS.

### 3.4 Discussion

We defined an FGS as a new graph rewriting system. We showed that the family of graphs generated by an HRG can be also defined by a regular FGS and vice versa. This result implies that regular FGSs generate trees, two-terminal series-parallel graphs, the homeomorphisms of a given graph, outerplanar graphs, graphs of cyclic bandwidth $\leq k$ and $k$-decomposable graphs [51, 71, 88]. However, regular FGSs can not generate planar graphs, complete graphs, complete binary trees and complete bipartite graphs, since HRGs can not generate these families [50, 111].
Many other graph grammars as generalizations of the usual string grammars have been defined in various ways [14, 29, 30, 31]. Janssens and Rozenberg [58, 59, 90] have introduced node label controlled (NLC) grammars as a basic framework for the mathematical investigation of graph grammars. The more general work on graph grammars is well presented in Nagl [86] and Ehrig [28]. NLC grammars rewrite only single nodes and establish connections between the embedded graph and the neighbors of the rewritten node on the basis of the labels of the involved nodes only. On one hand, “context-sensitivity” is hidden in the embedding mechanism of NLC grammars: Instead of checking the presence of certain labels in the neighborhood of the node to be replaced, the embedding mechanism disconnects to all undesired labels. Then, Ehrenfeucht et al. [27] have shown that NLC grammars do not satisfy the Church Rosser property [56] and there are no corresponding normal forms for NLC grammars to Chomsky normal form and Greibach normal form for context-free string grammars [56]. On the other hand, NLC grammars are “context-free” in the sense that a production of an NLC grammar has a label as left hand side and such a production is applicable to a node in a graph if and only if this node is labeled by the label representing the left hand side of the production. That is, there is no context-sensitive condition for the applicability of a production to a node.

The class of boundary NLC grammars (BNLC grammars) has been defined in [91] as the following context-free NLC grammars and has been investigated in [92, 93] from the “graph theoretical” point of view. An NLC grammar is a BNLC grammar if

(i) the left-hand side of each production is a nonterminal label, and

(ii) all the graphs involved (i.e., the axiom and the right-hand sides of productions) are such that two nonterminally labeled nodes are never adjacent.

In [91, 92], Rozenberg and Welzl have shown that BNLC grammars satisfy the Church Rosser property and that quite a number of interesting families of graphs can be gener-
ated by BNLC grammars. For example, BNLC grammars can generate trees, complete bipartite graphs, maximal outerplanar graphs, \( k \)-trees, graphs of cyclic bandwidth \( \leq k \), for a fixed positive integer \( k \).

In [20, 27, 32, 34, 38, 59, 60, 78, 79], other restrictions, extensions, and variations of NLC grammars have been presented and investigated from the viewpoint of generating power. In particular, Main and Rozenberg [78] have defined handle NLC grammars as the extension of NLC grammars where graphs consisting of two nodes labeled with nonterminals and an edge connecting these two nodes are allowed in the left-hand side of each production. Handle NLC grammars have the property that the family of those graph languages is exactly the recursively enumerable graph languages.

In [61], Janssens and Rozenberg have defined neighborhood controlled embedding (NCE) grammars as a generalization of NLC grammars and have investigated in [20, 21, 22, 35] from the viewpoint of the graph theory. In NLC grammars, only node labels may be used in the embedding process, while in NCE grammars the nodes themselves are allowed too. In the same way as NLC grammars, Engelfriet et al. [33, 36, 37] have given restrictions, extensions and variations of NCE grammars. In particular, Engelfriet et al. [36] have defined a boundary NCE grammar in the same way as a boundary NLC grammar.

Janssens and Rozenberg have proved in [61] that the classes of NLC and NCE graph languages are the same. From the viewpoint of generating power, Engelfriet et al. [32, 34] have compared some restricted subclasses of NCEs with some restricted subclasses of NLCs.

HRGs are characterized by vertex labeling boundary NCE grammars (BNCE grammars) of bounded nonterminal degree [37, 70, 113]. From the result in this chapter, the class of graphs generated by a BNCE grammar of bounded nonterminal degree
can be also defined by a regular FGS. However, the relation between FGSs and BNCE grammars (resp., BNLC grammars) is not yet clear.

While regular FGSs satisfy the Church Rosser property, general FGSs do not have this property. Thus, it remains us to analyze FGSs from the viewpoints of the graph theory and the generating power in more detail.
Chapter 4

Refutation Tree Problem

All classes of graph grammars discussed in the literatures seem to render the most attractive theory with a variety of results on structure, decidability and complexity. In particular, it is well-known that many NP-complete graph problems [40] can be solved in polynomial time, when an instance is restricted to the families of graphs generated by graph grammars.

The Hamiltonian circuit problem is NP-complete even for planar graphs [41]. However, Slisenko [101] has shown that the Hamiltonian circuit problem can be solved in polynomial time when a given graph is generated by a context-free graph grammar (CFGG) defined in [101]. The basic idea of his polynomial-time algorithm is as follows:

1. Find a parse tree $PT$ for an input graph $g$, which is a tree representing how a context-free graph grammar $G$ can generate $g$, if $g$ is generated by $G$.

2. Solve the Hamiltonian circuit problem by applying some kind of dynamic programming to $PT$.

This shows how to use a tree-like parse for solving the Hamiltonian circuit problem. Using the same idea, Kriauciunas [67] has proved that many search problems being NP-complete in general, e.g., the Hamiltonian circuit problem, the graph $k$-colorability
problem, the independent set problem and the max cut problem, are solvable in polynomial time for graphs admitting tree-like parsing. Moreover, Rytter and Szymacha [95] have shown that the graphs generated by Slisenko’s CFGGs are "easy" from the viewpoint of parallel computations and proved that the traveling salesman problem, the induced path problem, the independent set problem, the dominating set problem and the vertex cover problem can be solved in NC when input graphs are generated by a Slisenko’s CFGG. A similar extension for an HRG can be found in [70]. Hence, constructing a parse tree of a graph generated by a graph grammar is one of the most useful strategies for designing parallel graph algorithms. The purpose of this chapter is to show that FGSs are suitable graph rewriting systems for the parallel tree-like analysis.

We define a refutation tree in a similar way as a logic program [87, 97, 103]. A refutation tree describes how a term graph is generated by applying the graph rewriting rules of an FGS. Namely, it represents the logical structure of the graph in the FGS and simultaneously describes a decomposition of the graph explicitly. Hence, it may be useful for graph problems to employ FGSs in designing efficient parallel algorithms.

The refutation tree problem is to compute a refutation tree of a graph generated by an FGS. In Sections 4.1 and 4.2, we present two regular FGSs for which we devise efficient parallel algorithms solving the refutation tree problem with \(O(n + m)\) processors on an EREW PRAM, where \(n\) and \(m\) are the numbers of vertices and edges of an input graph, respectively. One is an regular FGS defining the family of two-terminal series parallel (TTSP) graphs and the other defines the family of outerplanar graphs. For TTSP graphs, we present an \(O((\log n)^2 + \log m)\) time algorithm in Section 4.1. By using this algorithm, for series parallel graphs which are the undirected version of TTSP graphs, we present an efficient parallel algorithm running in the same amount of time.
using the same number of processors on an EREW PRAM. For undirected outerplanar graphs, Diks et al. [24] have given an optimal parallel algorithm on a CREW PRAM which solves the problem of testing outerplanarity in $T(n) = O(\log n \log^* n)$ time using $n/T(n)$ processors. In Section 4.2, with $O(n + m)$ processors on an EREW PRAM, we give an $O((\log n)^2)$ time algorithm by employing our algorithm for TTSP graphs. These results assert that efficient parallel algorithms may exist for a large number of NP-complete problems for TTSP graphs and undirected outerplanar graphs.

In Section 4.3, we give a polynomial time algorithm computing a refutation tree of a graph generated by a simple FGS.

The parsing problem for a CFGG is to construct a parse tree of an input graph. As CFGGs, we deal with those defined in [101]. Rytter and Szymacha [95] have proved that the parsing problem for a CFGG is in NC$^2$. In Section 4.4, we prove that the parsing problem for a CFGG is NC$^1$-reducible to the refutation tree problem for a bounded simple FGS. This means that the result for bounded simple FGSs includes their result in [95].

Formally, we define the refutation tree problem. Let $F = \{(D_i, \theta_i, C_i)\}_{0 \leq i \leq k}$ be a refutation from a unit goal $D$. The refutation tree of $F$ is a tree defined as follows:

1. Every vertex is labeled with a unit goal or the empty goal.
2. The label of the root is the unit goal $D_0 = D$.
3. Every leaf is labeled with the empty goal.
4. $T_0$ is a tree consisting of only one vertex labeled with $D$. For $0 \leq i \leq k$, $T_{i+1}$ is a tree obtained by applying $(D_i, \theta_i, C_i)$ to the tree $T_i$ as follows: let $D_i$ be $\leftarrow A_i^1, \ldots, A_i^n$. Assume that $T_i$ has a leaf labeled with $\leftarrow A_i^m$. Let $C_i$ be $A_i \leftarrow B_i^1, \ldots, B_i^{q_i}$ and $\theta_i$ be a unifier of $A_i$ and the atom $A_i^m$. Then $T_{i+1}$ is obtained by adding new $q_i$ vertices labeled with $\leftarrow B_i^1 \theta_i, \ldots, \leftarrow B_i^{q_i} \theta_i$ as the
Definition 4.1 Let $\Gamma$ be an FGS and $p$ be its unary predicate symbol. The \textit{refutation tree problem} for $(\Gamma, p)$, denoted by $RT(\Gamma, p)$, is defined as follows:

\textbf{Refutation Tree Problem for $(\Gamma, p)$}

\textbf{INSTANCE:} A ground term graph $g$.

\textbf{PROBLEM:} If there is a refutation from the goal $\leftarrow p(g)$ in $\Gamma$, construct its refutation tree.

When we need not to point out explicitly the predicate symbol $p$, we omit $p$ from the pair $(\Gamma, p)$. By Proposition 3.2, $RT(\Gamma, p)$ for an input graph $g$ can be regarded as the problem to decide whether $g$ is generated by $\Gamma$.

This chapter is based on the papers [108, 109].

\section*{4.1 $O((\log n)^2 + \log m)$ Time Parallel Algorithm for Two-Terminal Series Parallel Graphs}

The family of two-terminal series parallel (TTSP) graphs finds important applications in the fields such as electric networks and scheduling problems [26, 110]. Moreover, efficient parallel algorithms exist for a large number of NP-complete combinatorial problems when these problems are restricted to the family of TTSP graphs.

A \textit{multidag} is a directed colored-graph $g = (V, E, \varphi, \psi)$ that allows multiple edges and does not contain any cycles. For a vertex $v \in V$, the number of edges entering $v$, called an \textit{indegree} of $v$, is denoted by $\text{indeg}(v)$, and the number of edges leaving $v$, called an \textit{outdegree} of $v$, is denoted by $\text{outdeg}(v)$. A vertex $v$ with $\text{indeg}(v) = 0$ (resp., $\text{outdeg}(v) = 0$) is called a \textit{source} (resp., a \textit{sink}). A \textit{two-terminal multidag} is a multidag with exactly one source and one sink.

Definition 4.2 \textit{Two-terminal series parallel} (TTSP) graphs are two-terminal multidags over $(\{a, s, t\}, \{b\})$ defined as follows:
(1) A directed colored-graph consisting of two vertices $u$ labeled with $s$ and $v$ labeled with $t$, and a single edge $(u, v)$ labeled with $b$ is a TTSP graph. The vertices $u$ and $v$ are the source and sink, respectively.

(2) For $i = 1, 2$, let $g_i$ be a TTSP graph with the source $u_i$ labeled with $s$ and the sink $v_i$ labeled with $t$. Then the graph obtained by either of the following two operations is a TTSP graph:

(a) **Parallel composition:** Identify $u_1$ with $u_2$, and identify $v_1$ with $v_2$. The resulting graph has $u_1 (= u_2)$ as the source and $v_1 (= v_2)$ as the sink.

(b) **Series composition:** Identify $v_1$ with $u_2$. The source and sink of the resulting graph are $u_1$ and $v_2$, respectively. The identified vertex $v_1 (= u_2)$ is labeled with $a$.

The source and sink of a TTSP graph are labeled with $s$ and $t$, respectively. Other vertices are labeled with $a$ and all edges are labeled with $b$.

In this section, we consider directed term graphs over $\langle \{a, s, t\}, \{b\}, X \rangle$ and colored-graphs over $\langle \{a, s, t\}, \{b\} \rangle$, where $X$ consists of variable symbols of rank at most 2.

The following lemma is obvious.

**Lemma 4.1** Let $\Gamma_{asp}$ be the regular FGS in Figure 4.1. Then $GL(\Gamma_{asp}, p)$ is the family of TTSP graphs.

Let $g = (V, E, \varphi, \psi, H, \lambda, ports)$ be a directed term graph over $\langle \{a, s, t\}, \{b\}, X \rangle$ such that the rank of every hyperedge in $H$ is at most 2. Then we define a directed colored-graph $\hat{g} = (\hat{V}, \hat{E}, \hat{\varphi}, \hat{\psi})$ over $\langle \{a, s, t\}, \{b\} \rangle$ as follows:

(1) $\hat{V} = V$ and $\hat{\varphi} = \varphi$.

(2) Let $E_H$ be the set of new edges defined from $H$ in the following way: For a hyperedge $e \in H$ with $\text{rank}(e) = 2$, let $\text{ports}(e) = (u_e, v_e)$. Then $E_H$ contains
Figure 4.1: Regular FGS $\Gamma_{dSP}$ defining the family of TTSP graphs.

Let $g = (V, E, \varphi, \psi, H, \lambda, \text{ports})$ be a two-terminal multidag with the source $u$ and
the sink $v$, and let $g_i = (V_i, E_i, \varphi_i, \psi_i, H_i, \lambda_i, ports_i)$ be a reducing subgraph of $g$ with the sink $u_i$ and the sink $v_i$ for $1 \leq i \leq k$. Assume that $g_1, \ldots, g_k$ are pairwise disjoint. Then we define a two-terminal term multidag $f = (V_f, E_f, \varphi_f, \psi_f, H_f, \lambda_f, ports_f)$ called the reduction of $g_1, \ldots, g_k$ in $g$ as follows:

1. $V_f = V_g - \bigcup_{i=1}^k (V_i - \{u_i, v_i\})$.
2. $E_f = E_g - \bigcup_{i=1}^k E_i$.
3. For each $1 \leq i \leq k$, let $e_i$ be a new hyperedge with $ports_f(e_i) = (u_i, v_i)$. Let $S_f = \{e_1, \ldots, e_k\}$. Then $H_f = H_g \cup S_f - \bigcup_{i=1}^k H_i$. The hyperedges in $S_f$ are distinctively labeled with new variables of rank 2.

The substitution

$$\theta = \{\lambda_f(e_1) := [g_1, (u_1, v_1)], \ldots, \lambda_f(e_k) := [g_k, (u_k, v_k)]\}$$

is called the reducing substitution for $g_1, \ldots, g_k$.

Let $g = (V, E, \varphi, \psi)$ be a multidag. A path from $v_1$ to $v_k$ is a sequence of vertices $v_1 \cdots v_k$ such that $(v_i, v_{i+1})$ is in $E$ for each $1 \leq i \leq k$. We denote a path from $u$ to $v$ by $P_{uv}$. Two paths $P_{uw}$ and $P_{w'u'}$ are disjoint if $P_{uw}$ and $P_{w'u'}$ are share only vertices in $\{u, v, u', v'\}$.

The forbidden digraph is exclusively the digraph shown in Figure 4.2. A graph is homeomorphic to the forbidden digraph if it consists of four distinct vertices $r, u, v, w$ and disjoint paths $P_{ur}, P_{vr}, P_{wu}$, and $P_{uw}$. Then, the following forbidden subgraph characterization of TTSP graphs is well-known [110].

**Lemma 4.2 (Valdes et al. [110])** Let $g$ be a two-terminal multidag. $g$ is a TTSP graph if and only if $g$ contains no subgraph homeomorphic to the forbidden digraph.

**Lemma 4.3** Let $g$ be a two-terminal term multidag with the source $u$ and the sink $v$, and let $g_i$ be a reducing subgraph of $g$ with the sink $u_i$ and the sink $v_i$ for $1 \leq i \leq k$. 47
Assume that \( g_1, \ldots, g_k \) are pairwise disjoint. Let \( g_0 \) be the reduction of \( g_1, \ldots, g_k \) in \( g \). Then,

1. The reduction \( g_0 \) is a two-terminal term multidag with \( u \) as the source and \( v \) as the sink.

2. The transformed graph of \( g \) is a TTSP graph if and only if, for each \( 0 \leq i \leq k \), the transformed graph of \( g_i \) is a TTSP graph.

**Proof.** We only consider the case \( k = 1 \), since the lemma is proved by induction with respect to \( k \), for \( k > 1 \). Let \( g = (V, E, \varphi, \psi, H, \lambda, ports) \) and \( g_i = (V_i, E_i, \varphi_i, \psi_i, H_i, \lambda_i, ports_i) \) for \( i = 0, 1, 2 \).

1. From the definition of the reduction, the transformed graph of \( g_0 \) is clearly a multidag. Then, we show that \( u \) and \( v \) are the only source and sink of \( g_0 \), respectively. If \( u \) is not in \( V_1 \), then \( u \) is in \( V - (V_1 - \{u_1, v_1\}) = V_0 \). Otherwise, \( u \) is also in \( V_0 \), since \( u \) must be the source \( u_1 \) of \( g_1 \). In the same way, we can show that \( v \) is in \( V_0 \). From the definitions of the reducing subgraph and the reduction, \( u_1 \) is the only vertex in \( g_0 \) such that the outdegree of \( u_1 \) in \( g_0 \) is different from that in \( g \). Since \( g_0 \) has the hyperedge \( e \)
such that $ports_0(e) = (u_1, v_1)$, the vertex $u_1$ is not a sink in $g_0$. Hence, no new sink is introduced into $g_0$. Therefore, $v$ is the only sink in $g_0$. Similarly $u$ is the only source in $g_0$.

(2) We show that the transformed graph of $g$ is a TTSP graph if the transformed graphs of $g_0$ and $g_1$ are TTSP graphs. Assume that $g$ is not a TTSP graph. By Lemma 4.2, the transformed graph $g$ contains a subgraph $\tilde{g}$ homeomorphic to the forbidden digraph. Since the transformed graphs $\tilde{g}_0$ and $\tilde{g}_1$ of $g_0$ and $g_1$ are TTSP graphs, respectively, both $\tilde{g}_0$ and $\tilde{g}_1$ do not contain $\tilde{g}$ as a subgraph. Then, $\tilde{g}$ contains at least one edge in $\tilde{g}_1$. Since $g_1$ is a reducing subgraph of $g$, $\tilde{g}$ contains a path $P$ in $\tilde{g}_1$ from $u_1$ to $v_1$. The graph obtained from $\tilde{g}$ by replacing $P$ by the edge $(u_1, v_1)$ is a subgraph of $\tilde{g}_0$ homeomorphic to the forbidden digraph. This contradicts that $\tilde{g}_0$ is a TTSP graph.

Next, we show that the transformed graphs of $g_0$ and $g_1$ are TTSP graphs if the transformed graph of $g$ is a TTSP graph. Suppose that the transformed graph of $g$ is a TTSP graph. If $\tilde{g}_1$ is not a TTSP graph, then it contains a subgraph $\tilde{g}$ homeomorphic to the forbidden digraph. This contradicts that $\tilde{g}$ is a TTSP graph, since any subgraph of $g_1$ is also a subgraph of $g$. If $\tilde{g}_0$ is not a TTSP graph, let $\tilde{g}$ be a subgraph of $\tilde{g}_0$ homeomorphic to the forbidden digraph. If $(u_1, v_1)$ is not an edge in $\tilde{g}$, then $\tilde{g}$ is also a subgraph of $g$. This contradicts. If $(u_1, v_1)$ is an edge in $\tilde{g}$, then the graph obtained from $\tilde{g}_0$ by replacing $(u_1, v_1)$ with a path in $\tilde{g}_1$ from $u_1$ to $v_1$ is a subgraph of $\tilde{g}$ and homeomorphic to the forbidden digraph. This also contradicts.

Let $g = (V, E, \varphi, \psi)$ be a TTSP graph with the source $u$ and the sink $v$. We denote the number of vertices in $g$ by $|g|$. Let $T$ be any rooted spanning tree of $g$ with $v$ as the root and $T(w)$ be the subtree of $T$ rooted at a vertex $w$ of $g$. The path in $T$ from $u$ to $v$ is called the main path of $T$. Let $w$ be a vertex of $T$ and let $w_1, \ldots, w_k$ be the children
of \( w \) in \( T \). We say \( w_i \) is the heavy child of \( w \) if \( |T(w_i)| = \max\{|T(w_j)| \mid 1 \leq j \leq k\} \).

A path \( v_1 \cdots v_r \) of \( T \) is a heavy path if \( v_i \) is the heavy child of \( v_{i+1} \) for each \( 1 \leq i < r \).

A spanning tree \( T \) of \( g \) is a normal spanning tree if the main path \( MP(T) \) is a heavy path.

Xin He and Yesha [54] have presented the technique which decomposes an input two-terminal multidag \( g \) into reducing subgraphs and its reduction by using a normal spanning tree of \( g \). By using their decomposition technique, we can prove the following lemmas.

**Lemma 4.4 (Xin He and Yesha [54])** Let \( g \) be a chain-type TTSP graph with \( |g| = n \geq 12 \). \( g \) can be decomposed into a reducing component \( g_1 \) and its reduction \( g_0 \) such that \( |g_i| \leq 5n/6 \) for \( i = 0, 1 \).

**Lemma 4.5 (Xin He and Yesha [54])** Let \( g \) be a general TTSP graph with \( |g| = n \geq 6 \). \( g \) can be decomposed into a reducing component \( g_j (j = 1, \ldots, q) \) for some \( q \) and the reduction \( g_0 \) of them such that either:

1. \( |g_j| \leq 5n/6 \) for each \( 1 \leq j \leq q \) and \( g_0 \) is a chain-type TTSP graph, or
2. \( |g_j| \leq 5n/6 \) for each \( 0 \leq j \leq q \).

Let \( \Gamma \) be a regular FGS and \( p \) be its unary predicate symbol. Let \( g \) be a term graph such that its transformed graph \( \hat{g} \) is in \( GL(\Gamma, p) \). Let \( T \) be a refutation tree from the goal \( \leftarrow p(\hat{g}) \) in \( \Gamma \). We define a decomposition tree \( T' \) of \( g \) corresponding to \( T \) in the following way: Since \( \Gamma \) is regular, the label of an internal vertex \( v \) in \( T \) is of the form \( \leftarrow q(\hat{f}) \). By the definitions of a refutation tree and a regular FGS, the colored-graph \( \hat{f} \) is a subgraph of \( \hat{g} \) where the vertex labelings are ignored. Hence there exists a subgraph \( f \) of \( g \) except the vertex labelings such that \( \hat{f} \) is the transformed graph of \( f \). Let \( T_0 \) be a tree obtained from \( T \) by replacing the label \( \leftarrow q(\hat{f}) \) of \( v \) with the label
← q(f) for each internal vertex v in T. Then T' is obtained from T₀ by removing every leaf u from T₀ together with its incident edge if the parent of u is labeled with a goal ← q(h) where h is a star term graph for a variable of rank 2.

**Example 4.1** Let g = (V, E, φ, ψ, H, λ, ports) be a TTSP graph in Figure 4.3. This term graph can be regarded as a colored-graph since it contains no hyperedge. Figure 4.3 shows a reducing subgraph f of g and its reduction h. Then h is the transformed graph of h. Figure 4.3 gives a refutation tree T of f and a decomposition tree T' of h.

**Theorem 4.1** Let Γ<sub>ASP</sub> be a regular FGS in Figure 4.1 defining the family of TTSP graphs. The refutation tree problem for (Γ<sub>ASP</sub>, p) can be solved in O((log n)² + log m) time with O(n m) processors on an EREW PRAM, where n and m are the numbers of vertices and edges of an input graph, respectively.

**Proof.** We give a parallel algorithm Ref.TTSP in Figure 4.4 for solving RT(Γ<sub>ASP</sub>, p) that uses the decomposition technique in [54]. We assume that a graph is given by edge list form.

We first show that Ref.TTSP computes a refutation tree T of a given graph g if any exists. Let f be the two-terminal term multidag of line 9 and assume that the decomposition is successful in line 9.

Let f₁, . . . , f₉ be the decomposition of f and let f₁, . . . , f₉ be their reducing subgraphs of f. Let f₀ be the reduction of f₁, . . . , f₉ in f and let f₀ be the transformed graph of f₀ provided in line 9. From Lemma 4.3, we show that f is a TTSP graph if and only if f₀, f₁, . . . , f₉ are TTSP graphs. Since the computation starts with g = ˆg and since lines 6, 8 and 9 reject graphs other than TTSP graphs, we see that Ref.TTSP recognizes the family of TTSP graphs.
Figure 4.3: The graph $f$ is a reducing subgraph of $g$ and $h$ is its reduction. $T$ is a refutation tree of $f$ and $T'$ is a decomposition tree of $h$. 
Ref.TTSP Algorithm

INPUT: A graph $g$.  
OUTPUT: A refutation tree $T$.

1. $U := \{g\}; i := 0; W := \emptyset$;
2. while $U \neq \emptyset$ do
3. \hspace{1em} $\theta_i := \emptyset$;
4. \hspace{1em} for each $f \in U$ pardo
5. \hspace{2em} $U := U - \{f\}$;
6. \hspace{2em} if $f$ is not a two-terminal term multidag
7. \hspace{3em} then return “$g$ is not a TTSP graph” and exit;
8. \hspace{2em} case
9. \hspace{3em} $|f| < 12$: if $\hat{f}$ is a TTSP graph
10. \hspace{4em} then put a decomposition tree of $f$ into $W$
11. \hspace{4em} else return “$g$ is not a TTSP graph” and exit
12. \hspace{3em} $|f| \geq 12$: apply the decomposition algorithm by [54] to $\hat{f}$;
13. \hspace{4em} if the decomposition is successful
14. \hspace{5em} then let $\hat{f}_1, \ldots, \hat{f}_q$ be the decomposition of $\hat{f}$;
15. \hspace{5em} let $f_1, \ldots, f_q$ be the corresponding reducing subgraphs of $f$;
16. \hspace{5em} let $f_0$ be the reduction of $f_1, \ldots, f_q$;
17. \hspace{5em} $U := U \cup \{f_0, f_1, \ldots, f_q\}$;
18. \hspace{5em} put all bindings in the reducing substitution
19. \hspace{6em} for $f_1, \ldots, f_q$ into $\theta_i$
20. \hspace{3em} else return “$g$ is not a TTSP graph” and exit
21. \hspace{2em} end case
22. end pardo
23. $i := i + 1$;
24. end do
25. compute a substitution $\theta = \prod_{k=0}^{i} \theta_k$;
26. construct a refutation tree $T$ of $g$ from $W$ and $\theta$.

Figure 4.4: Algorithm Ref.TTSP
Now we shall explain how to compute line 15 from $W$ and $\theta$ when a TTSP graph $g$ is given as an input. For a decomposition tree $S$ and a substitution $\tau$, let $S\tau$ be the tree obtained from $S$ by replacing each vertex label $\leftarrow B$ of $S$ with $\leftarrow B\tau$. Then for the set of decomposition trees $W$ and the substitution $\theta$, let $W\theta = \{S\theta \mid S \in W\}$. We construct a refutation tree $T$ of $g$ from $W\theta$ in the following way: Initially, let $T$ be a decomposition tree in $W\theta$ whose root label is $\leftarrow p(g)$. From the definitions of $\Gamma_{dsP}$ and $W\theta$, it is easy to see that $T$ is unique in $W\theta$. For each leaf $v$ of $T$ not labeled with the empty goal, we attach the decomposition tree $T_v$ in $W\theta$ whose root label is equal to that of $v$ by identifying its root with $v$. Again from the definitions of $\Gamma_{dsP}$ and $W\theta$, it is clear that $T_v$ is unique in $W\theta$. We repeat this procedure for $T$ until all leaves of $T$ are labeled with the empty goal. We can see that the resulting decomposition tree $T$ is a refutation tree of $g$.

We illustrate the complexity of $Ref.TTSP$. In order to simplify the analysis of $Ref.TTSP$, we reduce an input multidag $g$ to a directed colored-graph without any multiple edges by performing the following operation as initialization: $Ref.TTSP$ counts the number of edges between each pair of vertices and replaces them by a single edges. The resulting graph $g_0$ is a directed colored-graph without any multiple edges. We can see that $g$ is a TTSP graph if and only if $g_0$ is a TTSP graph. Moreover, the refutation tree of $g$ can be easily obtained from the refutation tree of $g_0$. The conversion from $g$ to $g_0$ takes $O(\log m)$ time with $O(m)$ processors. Thus without loss of generality, we assume that $g$ is a directed colored-graph without any multiple edges.

Line 8 requires constant time. Line 9 takes in $O(\log n)$ time using $O(n + m)$ processors on an EREW PRAM by the decomposition technique in [54]. Furthermore, from Lemmas 4.4 and 4.5, we can show that the while loop (lines 2–13) repeats $O(\log n)$ times.
Let $U_j$ be the content of $U$ just after the $j$th iteration (lines 2–13). Let $d_j$ be the number of elements in $U_j$. Since $\sum_{j=0}^{O(\log n)} d_j = O(n)$, the number of bindings in $\theta = \prod_{k=0}^{O(\log n)} \theta_k$ is $O(n)$. Therefore, by using the balanced binary tree method [43], $\theta$ can be computed in $O(\log n)$ time with $O(n)$ processors. We can construct a refutation tree $T$ of $g$ in $O(\log n)$ time with $O(n)$ processors from $W$ and $\theta$ by executing the above procedure. Therefore, lines 14 and 15 can be computed in $O(\log n)$ time with $O(n)$ processors. Thus Ref.TTSP constructs the refutation tree in $O((\log n)^2 + \log m)$ time with $O(n + m)$ processors.

Let $g = (V_g, E_g, \varphi_g, \psi_g)$ be an undirected ground term graph, and let $u$ and $v$ be two vertices of $g$. An assignment of distinct integers to the vertices of $g$ is called a $uv$-numbering of $g$ if two vertices $u$ and $v$ are the lowest- and highest-numbered, respectively, and every other vertex is adjacent to both a lower-numbered and a higher-numbered vertex. A $uv$-numbering of $g$ induces a direction on the edges of $g$. Let $F_{uv}$ be a $uv$-numbering of $g$. Then, we define a directed ground term graph $\tilde{g} = (\tilde{V}_g, \tilde{E}_g, \tilde{\varphi}_g, \tilde{\psi}_g)$ as follows:

1. $\tilde{V}_g = V_g$ and $\tilde{\varphi}_g = \varphi_g$.
2. Let $w$ and $w'$ be any two vertices in $V_g$ such that $F_{uv}(w) < F_{uv}(w')$. Then $\{w, w'\}$ is in $E_g$ if and only if an ordered edge $(w, w')$ is in $\tilde{E}_g$. For an edge $(w, w')$ in $\tilde{E}_g$, $\tilde{\psi}_g((w, w')) = \psi_g(\{w, w'\})$.

The directed ground term graph $\tilde{g}$ is called the $F_{uv}$-directed graph of $g$.

The graph $g$ is called a series parallel (SP) graph, if there exist two vertices $u, v$ of $g$ and the $uv$-numbering $F_{uv}$ so that the $F_{uv}$-directed graph of $g$ is a TTSP graph. Let $\Gamma_{SP}$ be a regular FGS drawn in Figure 4.5. $\Gamma_{SP}$ obtains from $\Gamma_{dSP}$ by replacing the ordered edge of the directed ground term graph in the fact of $\Gamma_{dSP}$ with the unordered edge. It is easy to see that the family of SP graphs is generated by $\Gamma_{SP}$.
From Theorem 4.1, the following theorem can be proved by using the algorithm in [53].

**Theorem 4.2** Let $\Gamma_{SP}$ be a regular FGS in Figure 4.5 defining the family of SP graphs. The refutation tree problem for $(\Gamma_{SP}, p)$ can be solved in $O((\log n)^2 + \log m)$ time with $O(n + m)$ processors on an EREW PRAM where $n$ and $m$ are the numbers of vertices and edges in an input graph, respectively.

**Proof.** We can prove this theorem by executing the following operations (I) and (II):

(I) Identify two vertices $u$ and $v$ in an input graph $g$ and compute $uv$-numbering $F_{uv}$ of $g$ so that the $F_{uv}$-directed graph $g'$ of $g$ is acyclic and has $u$ as the only source and $v$ as the only sink.

(II) Apply $Ref.TTSP$ algorithm to $g'$.

In [53], Xin He has presented an efficient parallel algorithm computing the part (I) in $O((\log n)^2)$ time with $O(n + m)$ processors on an EREW PRAM. From Theorem 4.1, the part (II) computes in $O((\log n)^2 + \log m)$ time with $O(n + m)$ processors on an EREW PRAM.
4.2 $O((\log n)^2)$ Time Parallel Algorithm for Outerplanar Graphs

A cutpoint of a colored-graph is a vertex whose removal increases the number of components. A nonseparable colored-graph is a connected colored-graph which has at least two vertices and no cutpoints. A block (sometimes called biconnected component) of a colored-graph is a maximal nonseparable subgraph. A colored-graph is outerplanar if it can be embedded in the plane so that all its vertices lie on the same face.

Let $g$ be a connected colored-graph with the cutpoints $u_1, \ldots, u_l$. A block sequence $(b_1, \ldots, b_k)$ of $g$ is a sequence of all the blocks of $g$. A sequence $\langle f_1, v_1 \rangle, \ldots, \langle f_k, v_k \rangle$ is called a dividing sequence of $g$ with respect to $(b_1, \ldots, b_k)$, if each $\langle f_i, v_i \rangle$ satisfies the following conditions for $1 \leq i \leq k$:

1. $v_i$ is a vertex of $b_i$ with $v_i \in \{u_1, \ldots, u_l\}$.
2. $f_i = (V_i, E_i, \varphi_i, \psi_i, H_i, \lambda_i, \text{ports}_i)$ is a term graph such that $f' = (V_i, E_i, \varphi_i, \psi_i)$ is the block $b_i$, where the label of $v_i$ is ignored and $v_i$ is labeled with $s$. The term graph $f_i$ is called a dividing component of $g$ with a head attachment $v_i$.
3. For each vertex $w$ in $V_i \cap \{u_1, \ldots, u_l\}$, if there exists $j < i$ such that $b_j$ and $b_i$ share the vertex $w$, then $H_i$ contains a hyperedge with the port $w$ labeled with a new distinct variable of rank one.

For each $1 \leq i \leq k$ and each $e \in H_i$, let $i_e$ be the maximal number less than $i$ such that $f_{i_e}$ and $f_i$ share only the port of $e$. Then let $\theta_i = \{\lambda_i(e) := [f_{i_e}, \text{ports}(e)] \mid e \in H_i\}$.

The substitution $\theta = \prod_{1 \leq i \leq k} \theta_i$ is called a dividing substitution for $g$.

We can see from Theorem 3.1 that the following lemma holds.

**Lemma 4.6** Let $\Gamma_{OP}$ be an FGS in Figure 4.6. Then $GL(\Gamma_{OP}, p)$ is the set of undirected outerplanar colored-graphs.
Theorem 4.3 Let $\Gamma_{op}$ be an FGS in Figure 4.6 defining the class of outerplanar graphs. The refutation tree problem for $(\Gamma_{op}, p)$ can be solved in $O((\log n)^2)$ time with $O(n + m)$ processors on an EREW PRAM, where $n$ and $m$ are the numbers of vertices and edges of an input connected colored-graph.

Proof. We give a parallel algorithm for solving $RT(\Gamma_{op}, p)$ in Figure 4.7 using the Ref.TTSP algorithm. We assume that a graph is given by edge list form.

By the definition of a dividing component of $g$, we can see that $g$ is an outerplanar graph.
Ref. OUTER Algorithm

INPUT: A connected colored-graph $g$.
OUTPUT: A refutation tree $T$.

1. let $(b_1, \ldots, b_k)$ be a block sequence of $g$;
2. let $(f_1, v_1), \ldots, (f_k, v_k)$ be a dividing sequence of $g$ with respect to $(b_1, \ldots, b_k)$;
3. let $\theta$ be the dividing substitution for $g$;
4. $W := \emptyset$; $\tau := \emptyset$;
5. for each $(f_i, v_i), 1 \leq i \leq k$ pardo
6. Compute $w_i v_i$-numbering $F_{w_i v_i}$ for an edge $\{w_i, v_i\}$ of $f_i$;
7. case
8. $|f_i| > 2$: apply the decomposition algorithm DECOM to $f_i$;
   if the decomposition is successful then
   let $f_1, f_2, f_3$ be the resulting reducing subgraphs of $f_i$;
   let $f_0$ be the reduction of $f_1, f_2, f_3$;
   put all bindings in the reducing substitution for $f_1, f_2, f_3$
   into $\tau$;
   put a decomposition tree of $f_0$ into $W$;
   apply Ref. TTSP to each $F_{w_i v_i}$-directed graph $f_i^j$ $(1 \leq j \leq 3)$
   of $f_i$;
   let $H_1^i, H_2^i, H_3^i$ be the resulting decomposition trees of
   $f_1^i, f_2^i, f_3^i$, respectively;
   let $T_1^i, T_2^i, T_3^i$ be the corresponding decomposition tree of
   $f_1^i, f_2^i, f_3^i$ to $H_1^i, H_2^i, H_3^i$, respectively;
   $W := W \cup \{T_1^i, T_2^i, T_3^i\}$;
   else return "$g$ is not an outerplanar graph" and exit;
9. $|f_i| \leq 2$: put a decomposition tree of $f_i$ to $W$
10. end case
11. end pardo
12. Construct the refutation tree $T$ of $g$ from $W$ and $\tau$.

Figure 4.7: Algorithm Ref. OUTER
colored-graph if and only if the transformed graph of every dividing component of \( g \) is an outerplanar colored-graph.

Let \( f \) be a dividing component of \( g \) with a head attachment \( v \) and \( \{w,v\} \) be an edge of \( f \). Since \( f \) is biconnected, it is easy to see that \( f \) has an \( \text{wv-numbering} \) of \( f \).

**Claim 4.1** Let \( f \) be a dividing component of \( g \) with a head attachment \( v \) such that \( |f| > 2 \) and \( \{w,v\} \) be an edge of \( f \). Let \( F_{wv} \) be an \( \text{wv-numbering} \) of \( f \). Assume that \( w' \) is the smallest vertex with \( w' \neq w \) with respect to \( F_{wv} \). If \( g \) is outerplanar, then we can decompose \( f \) into the reducing subgraphs \( f^1, f^2, f^3 \) of \( f \) with the sources \( w, w, w' \) and the sinks \( v, w', v \), respectively.

**Proof of Claim 4.1.** From the definition of a dividing component of \( g \), the transformed graph \( \hat{f} \) of \( f \) is a block of \( g \) where the label of \( v \) is ignored. Hence if \( g \) is an outerplanar colored-graph, then \( \hat{f} \) is biconnected. From the selection of \( w \) and \( w' \), there exist reducing subgraphs \( f^1, f^2, f^3 \) of \( f \) with the sources \( w, w, w' \) and the sinks \( v, w', v \), respectively, such that any edge in \( f \) is in either \( f^1, f^2 \) or \( f^3 \). If \( g \) is outerplanar, then we can decompose \( f \) into the reducing subgraphs \( f^1, f^2, f^3 \) of \( f \) with the sources \( w, w, w' \) and the sinks \( v, w', v \), respectively.

**Claim 4.2** Let \( f \) be a dividing component of \( g \) with a head attachment \( v \) such that \( |f| > 2 \) and \( \{w,v\} \) be an edge of \( f \). Let \( F_{wv} \) be an \( \text{wv-numbering} \) of \( f \). Assume that \( f^1, f^2, f^3 \) are the reducing subgraphs of \( f \) obtained in Claim 4.1. Then \( f \) is an outerplanar colored-graph if and only if the \( F_{wv}\)-directed graphs \( \hat{f}^1, \hat{f}^2, \hat{f}^3 \) of \( f^1, f^2, f^3 \) are chain-type TTSP graphs, respectively.

**Proof of Claim 4.2.** We can see that if \( \hat{f}^1, \hat{f}^2, \hat{f}^3 \) are chain-type TTSP graphs then \( \hat{f} \) is outerplanar. We show that if \( \hat{f} \) is outerplanar then \( \hat{f}^1, \hat{f}^2, \hat{f}^3 \) are chain-type TTSP graphs. By Claim 4.1, if \( g \) is an outerplanar colored-graph, then we can decompose
f into the reducing subgraphs \( f^1, f^2, f^3 \) of \( f \) with the sources \( w, w, w' \) and the sinks \( v, w', v \), respectively. From the definitions of \( F_w \) and \( F_{w'} \)-directed graph \( \tilde{f} \) of \( f \), \( \tilde{f} \) is a directed colored-graph with no cycle. Hence \( \tilde{f}^1, \tilde{f}^2, \tilde{f}^3 \) of \( f \) are two-terminal directed colored-graphs. Since \( \tilde{f} \) is an outerplanar colored-graph, there exists a path containing all vertices in \( f^j \) for each reducing subgraph \( f^j \) (\( 1 \leq j \leq 3 \)). Hence, if \( f \) is outerplanar then \( \tilde{f}^1, \tilde{f}^2, \tilde{f}^3 \) is chain-type TTSP graphs.

**DECOM** is a procedure of selecting the smallest vertex \( w' \) of \( f \) with \( w' \neq w \) with respect to \( F_w \) and decomposing \( f \) into reducing subgraphs \( f^1, f^2, f^3 \) with the sources \( w, w, w' \) and the sinks \( v, w', v \), respectively, such that \( \tilde{f}^1, \tilde{f}^2, \tilde{f}^3 \) are chain-type TTSP graphs, if possible.

Next, we illustrate some implementation details. The input to the Ref.OUTER algorithm is an undirected connected colored-graph \( g = (V, E, \varphi, \psi) \) given by edge list form where \( n \) and \( m \) are the numbers of vertices in \( V \) and edges in \( E \), respectively.

The all blocks \( b_1, \ldots, b_k \) of \( g \) are computed in \( O((\log n)^2) \) time with \( O(n + m) \) processors by the block-finding algorithm in [106]. The block-finding algorithm in [106] uses any spanning tree of \( g \) in order to find the all blocks of \( g \). Then line 1 can be computed in \( O((\log n)^2) \) time with \( O(n + m) \) processors. Line 2 can be implemented as follows: Let \( S \) be an undirected spanning tree constructed in order to find the blocks of \( g \) in line 1. By selecting a cutpoint of \( g \) in \( S \), we make \( S \) a rooted tree. Let \( u_1, \ldots, u_l \) be the cutpoints of \( g \). Let each block \( b_i \) (\( 1 \leq i \leq k \)) contain cutpoints \( v^i_1, \ldots, v^i_{p_i} \) of \( g \). By applying the pointer jump technique in [106] to \( S \), we can identify the vertex \( v^i_1 \) such that a subtree of \( S \) rooted at \( v^i_1 \) contains \( v^i_2, \ldots, v^i_{p_i} \) in \( O(\log n) \) time with \( O(n) \) processors. Hence for the block sequence \( (b_1, \ldots, b_k) \) of \( g \), we can construct a dividing sequence \( (f_1, v^1_1), \ldots, (f_k, v^k_1) \) of \( g \) in \( O(\log n) \) time with \( O(n) \) processors. Then line 2 takes \( O(\log n) \) time with \( O(n + m) \) processors.
From the dividing sequence of $g$ computed in line 2, we can compute line 3 in $O(\log n)$ time with $O(n + m)$ processors.

Next, we show that lines 5–11 can be computed in $O((\log n)^2)$ time with $O(n + m)$ processors. It takes clearly constant time in line 9. Hence we show that lines 6 and 8 can be computed in $O((\log n)^2)$ time with $O(n + m)$ processors. Let $(f_1, v_1), \ldots, (f_k, v_k)$ be the block sequence of $g$ computed in line 2. We assume that, for each dividing component $f_i$ of $g$ with a head attachment $v_i$, we denote the numbers of vertices and edges of $f_i$ by $n_i$ and $m_i$, respectively. The $w_i v_i$-numbering $F_{w_i v_i}$ for an edge $\{w_i, v_i\}$ of $f_i$ can be computed in $O((\log n_i)^2)$ time with $O(n_i + m_i)$ processors by the algorithm in [80]. \textit{DECOM} can be implemented in constant time if $g$ is an outerplanar graph. By using the \textit{Ref.TTSP} algorithm, we can construct the decomposition trees of the resulting reducing components $f_i^1, f_i^2, f_i^3$ of $f_i$ in $O((\log n_i)^2)$ time with $O(n_i + m_i)$ processors.

Since $\sum_{1 \leq i \leq k} n_i = O(n)$ and $\sum_{1 \leq i \leq k} m_i = O(m)$, lines 5–11 can be computed in $O((\log n)^2)$ time with $O(n + m)$ processors. In line 12, we can compute in $O(\log n)$ time with $O(n)$ processors by using the same procedure in line 15 of the \textit{Ref.TTSP} algorithm. Thus we can solve the refutation tree problem for outerplanar graphs in $O((\log n)^2)$ time with $O(n + m)$ processors on an EREW PRAM.

4.3 Polynomial Time Algorithm for Graphs Generated by a Simple FGS

This section gives a polynomial time algorithm for solving the refutation tree problem for a simple FGS.

\textbf{Theorem 4.4} Let $\Gamma$ be a simple FGS and $p$ be a unary predicate symbol in $\Gamma$. When an input graph is connected, $RT(\Gamma, p)$ is solvable in polynomial time.
Proof. We consider a connected ground term graph $g = (V, E, \phi, \psi)$ with $V = \{1, \ldots, n\}$ as an input graph. Let $V_{E'}$ be the set of endpoints of the edges in $E'$ for a subset $E' \subseteq E$. We denote the subgraph of $g$ formed from $E'$ by $\ll E' \gg = (V_{E'}, E', \phi', \psi')$ and call it the edge-induced subgraph of $g$. For a sequence $\alpha$, set($\alpha$) denotes the set of all elements in $\alpha$.

In this proof, an algebraic approach of path systems[94] is used. A path system is given by a 4-tuple $Q = (N, T, GEN, S)$ [43, 94], where $N$ is a finite set of vertices, $T$ is a subset of $N$ whose vertices are called terminal vertices, $GEN$ is a function whose arguments are vertices in $N$ and whose values are subsets of $N$, and $S$ is a vertex of $N$ (called the goal). A derivation of $S$ is a sequence of pairs $(D_i, C_i)$ ($i = 0, 1 \ldots$) satisfying the following conditions:

1. $D_i$ and $C_i$ are sequences of vertices in $N$, and $D_0 = S$.
2. Let $D_i = (a_1, \ldots, a_n)$ and $a_m$ be the left-most vertex in $D_i$ not in $T$. Let $C_i = (b_1, \ldots, b_k)$. Then $D_{i+1} = (a_1, \ldots, a_{m-1}, b_1, \ldots, b_k, a_{m+1}, \ldots, a_n)$.

Let $F = \{(D_i, C_i)\}_{0 \leq i \leq l}$ be a finite derivation of $S$ such that set($D_l$) $\subseteq T$. The generation tree of $F$ is a tree $T_i$ defined as follows:

3. Every vertex is in $N$.
4. The root is the goal $S$.
5. $T_0$ is a tree consisting of the only vertex $S$. For $0 \leq i < l$, $T_{i+1}$ is a tree obtained by applying $(D_i, C_i)$ to the tree $T_i$ as follows: Let $a_m$ be the left-most vertex in $D_i$ not in $T$ and $C_i = (b_1, \ldots, b_k)$. Then $T_{i+1}$ is obtained by adding new $k$ vertices $b_1, \ldots, b_k$ as the children of $a_m$.

The system $Q$ is solvable if $S$ is generated from $T$ using the function $GEN$, that is, there is a finite derivation $\{(D_i, C_i)\}_{0 \leq i \leq l}$ of $S$ on $Q$ such that $set(D_l) \subseteq T$. 

63
First, we number the edges of $g$. For a graph rewriting rule

$$ r = q_0(g_1^1, \ldots, g_n^1) \leftarrow q_1(g_1^1, \ldots, g_k^1), \ldots, q_k(g_k^1, \ldots, g_k^k) $$

in $\Gamma$, we give a procedure $\text{MakeAtom}$ in Figure 4.8 which computes the set of sequences $[q, a_1, \ldots, a_l]$ satisfying the following conditions:

(6) If $r$ is a fact then $q$ is the predicate symbol $q_0$ in $r$, otherwise $q$ is a predicate symbol in the body of $r$, and $l$ is the arity of $q$.

(7) We assume that $q$ is a predicate symbol $q_i$ ($0 \leq i \leq k$) in $r$. For $1 \leq j \leq l (=l_i)$, let $e_j^1, \ldots, e_j^{h_j}$ be all hyperedges in $g_j^1$. For each $1 \leq j \leq l$, $a_j$ is a pair $(F_j, (\alpha_j^1, \ldots, \alpha_j^{h_j}))$ defined as follows:

(a) $F_j$ is the set of $m_j$ distinct edges of $g$, where $m_j$ is the number of edges of $g_j^1$.

(b) For each $1 \leq s \leq h$, $\alpha_{js}^j$ is a sequence of $t_s$ distinct endpoints of edges in $F_j$, where $t_s$ is the number of ports of the hyperedge $e_j^s$ in $g_j^1$.

Hence, if $g_j^1$ is ground then $a_j = (F_j, (\cdot))$.

Since $\Gamma$ is fixed, $\text{MakeAtom}$ runs in constant time.

Then we give a path system $Q = (N, T, GEN, S)$ as follows: Let $\#$ be the symbol representing the input graph $g$. Let $S = \{p, \#\}$ and let $N = \{\text{MakeAtom}(r) \mid r \in \Gamma\} \cup \{\{p, \#\}\}$.

For an element $a_i$ ($1 \leq i \leq l$) of $[q, a_1, \ldots, a_l] \neq [p, \#]$ in $N$, $g[a_i] = (V_{a_i}, E_{a_i}, \varphi_{a_i}, \psi_{a_i})$ denotes a subgraph of $g$ induced by $a_i$ defined as follows: Let $a_i = (F_i, (\alpha_i^1, \ldots, \alpha_i^{h_i}))$.

Let $f_1, \ldots, f_m$ be the connected components of the edge-induced subgraph $\ll E - F_i \gg$ of $g$ such that each $f_i$ ($1 \leq i \leq m$) has a vertex in $\cup_{1 \leq j \leq l_i} \text{set}(\alpha_j^i)$. Then the vertex set $V_{a_i}$ and the edge set $E_{a_i}$ are the sets of all vertices and edges in $f_1, \ldots, f_m$, and $\ll F_i \gg$, respectively. For the symbol $\#$ of $[p, \#]$ in $N$, we define $g[\#] = g$. Let $T$ be the set of every element $[q, a_1, \ldots, a_l]$ of $N$ such that, for a fact $q(g_1, \ldots, g_l) \leftarrow$ in $\Gamma$ and
Procedure MakeAtom(r);

/* r = q_0(g_0^1, \ldots, g_0^k) \leftarrow q_1(g_1^1, \ldots, g_1^l), \ldots, q_k(g_k^1, \ldots, g_k^h). */
1. \( N := \emptyset; \ K_1 := 1; \ K_2 := k; \)
2. if \( r \) is a fact then \( K_1 := 0; \ K_2 := 0; \)
3. for each \( K_1 \leq i \leq K_2 \) pardo
4. for each \( 1 \leq j \leq l_i \) pardo
5. let \( s \) be the number of edges in \( g_i^j \).
6. let \( F \) be the family of sets of \( s \) distinct edges of \( g; \)
7. if \( g_i^j \) has no hyperedges then \( M\text{-Term}(g_i^j) := \{(f,()) | f \in F\} \)
8. else
9. for each \( \{h_1, \ldots, h_s\} \in F \) pardo
10. let \( e_1, \ldots, e_m \) be the all hyperedges of \( g_i^j \);
11. for each \( 1 \leq t \leq m \) pardo
12. let \( n_t \) be the number of ports of \( e_t \);
13. let \( D_{e_t} := \{(v_1, \ldots, v_{n_t}) | v_1, \ldots, v_{n_t} \in h_1 \cup \cdots \cup h_s\}; \)
14. end pardo
15. let \( M\text{-Port}(\{h_1, \ldots, h_s\}) := \{(d_1, \ldots, d_m) | d_1 \in D_{e_1}, \ldots, d_m \in D_{e_m}\}; \)
16. end pardo
17. \( M\text{-Term}(g_i^j) := \{(f,\alpha) | f \in F, \alpha \in M\text{-Port}(j)\}; \)
18. end pardo
19. \( N := N \cup \{[q_i, a_1, \ldots, a_l] | a_j \in M\text{-Term}(g_i^j), 1 \leq j \leq l_i\}; \)
20. end pardo

Figure 4.8: Procedure MakeAtom
each $1 \leq i \leq l$, $g[a_i]$ is isomorphic to $g_i$ except the edge and vertex labelings for edges in $g_i$ labeled with symbols in $\Delta$ and its endpoints, respectively. We can compute the sets $N$, $T$ and $S$ in constant time, since $\text{MakeAtom}$ runs in constant time.

We define a function $\text{GEN} : N \cup N^2 \cup \cdots \cup N^{k-1} \rightarrow 2^N$ as follows where $k$ is the maximum number of atoms of graph rewriting rules in $\Gamma$:

$$\text{GEN}([q_1, a_1^1, \ldots, a_l^1], \ldots, [q_m, a_m^1, \ldots, a_m^{l_m}])$$

contains an element $[q, a_0^1, \ldots, a_0^l]$ of $N$ if, for a graph rewriting rule $r$ in $\Gamma$, a substitution $\theta$ obtained by the procedure

$$\text{MakeSub}([q, a_0^1, \ldots, a_0^l], ([q_1, a_1^1, \ldots, a_l^1], \ldots, [q_m, a_m^1, \ldots, a_m^{l_m}]), r)$$

in Figure 4.9 satisfies the following condition: Let $r = q(g_i^1, \ldots, g_l^1) \leftarrow q_1(g_1^1, \ldots, g_1^{l_1}), \ldots, q_m(g_m^1, \ldots, g_m^{l_m})$. For $0 \leq i \leq m$ and $1 \leq j \leq l_i$, the instance $g_i^j$ of $g_j^i$ by $\theta$ is isomorphic to $g[a_i^j]$, except the edge and vertex labelings for edges in $g_i^j$ labeled with symbols in $\Delta$ and its endpoints, respectively. $\text{MakeSub}$ can be executed in $\text{NC}^2$ by using the $\text{NC}^2$ algorithm [43, 106] computing connected components. It requires to solve the graph isomorphism problem in order to decide whether the substitution $\theta$ constructed in $\text{MakeSub}$ satisfies the above condition or not. Thus the problem of computing $\text{GEN}$ is $\text{NC}^2$-reducible to the graph isomorphism problem. By Lemma 3.1, graphs definable by $\Gamma$ are of constantly bounded valence. Therefore, we can compute $\text{GEN}$ in polynomial time by using the polynomial-time algorithm [77] solving the graph isomorphism problem whose input graphs are restricted to a family of graphs with constantly bounded valence. Thus, we can construct the path system $Q$ in polynomial time.

It can be easily seen that the path system $Q$ is solvable if and only if the ground term graph $g$ is defined by the FGS $\Gamma$. Since the problem of solving a path system is $\text{P}$-complete [17] in general, we can construct a generation tree $GT$ of $[p, \#]$ on the path system $Q$ in polynomial time. Then we can compute a refutation tree for $g$ in $\Gamma$ from $GT$ by replacing a label $[q, a_1, \ldots, a_l]$ of each vertex in $GT$ with the unit goal
Procedure MakeSub($L, M, r$);

/* \( L = [q, a_0^1, \ldots, a_0^l] */

/* \( M = ([q_1, a_1^1, \ldots, a_1^l], \ldots, [q_m, a_m^1, \ldots, a_m^l]) */

/* \( a_j^j = (F_{ij}, (\alpha_{ij}^1, \ldots, \alpha_{ij}^l)) \) for \( 1 \leq i \leq m, 1 \leq j \leq l_i */

/* \( r = q(q_0^1, \ldots, q_0^l) \leftarrow q_1(q_1^1, \ldots, q_1^l), \ldots, q_m(q_m^1, \ldots, q_m^l). */

/* \( g_i^j = (V_i^j, E_i^j, \varphi_i^j, \psi_i^j, H_i^j, \lambda_i^j, ports_i^j) \) for \( 0 \leq i \leq m, 1 \leq j \leq l_i */

1. \( \theta := \emptyset; U := \emptyset; \)

2. for \( i := 1 \) to \( m \) do

3. for \( j := 1 \) to \( l_i \) do

4. let \( e_1, \ldots, e_t \) be the all hyperedges in \( H_i^j \);

5. if \( t = t_{ij} \) then

6. for \( s := 1 \) to \( t \) do

7. if \( \lambda_i^j(e_s) \notin U \) then

8. \( U := U \cup \{ \lambda_i^j(e_s) \}; \)

9. let \( f_1, \ldots, f_{k_s} \) be the connected components of \( \ll E - F_{ij} \gg \)

10. such that each \( f_k (1 \leq k \leq k_s) \) has a vertex in \( set(\alpha_{ij}^l) \);

11. let \( V_s \) and \( E_s \) be the sets of vertices and edges in \( f_1, \ldots, \) and \( f_{k_s} \);

12. let \( g_s = (V_s, E_s, \varphi_s, \psi_s) \) be the subgraph of \( g_i^j \) induced by \( V_s \) and \( E_s \);

13. \( \theta := \theta \cup \{ \lambda_i^j(e_s) := [g_s, \alpha_{ij}^l] \} \)

14. end do

15. end do

Figure 4.9: Procedure MakeSub
\[ q(g_{a_1}, \ldots, g_{a_l}) \] and by attaching a new vertex labeled with the empty goal to each leaf. Thus, the refutation tree problem for \((\Gamma, p)\) is solvable in polynomial time.

In this proof, we give the polynomial-time algorithm computing a refutation tree of an input connected graph by constructing and solving a path system. That is, the refutation tree problem for simple FGSs is in P, when an input graph is connected. This polynomial-time algorithm does not seem to be efficiently parallelizable, since the problem of solving a path system is P-complete [17]. But, it is not known whether the problem is P-complete.

### 4.4 NC Algorithm for Graphs Generated by a Context-Free Graph Grammar

This section shows that the parsing problem for a CFGG is NC\(^1\)-reducible to the refutation tree problem for a bounded simple FGS.

A vertex-colored graph \( G = (V, E, \varphi) \) over \( \Sigma \) consists of a vertex set \( V \), an edge set \( E \), and a vertex labeling \( \varphi : V \to \Sigma \). A star graph is a pair \( S = (G, R) \), where \( G \) is a vertex-colored graph and \( R \) is the set of edges disjoint with the set of edges of \( G \). We require that each edge in \( R \) has exactly one common vertex with \( G \). The graph \( G \) is called a kernel of \( S \). The vertices of \( G \) are labeled by terminal or nonterminal symbols. We denote terminal symbols by \( a, b, \ldots \) and use capital letters for representing nonterminal symbols. The edge in \( R \) is called the leg of \( G \). A simple-star is a star graph with its kernel being a single vertex without any edges. We assume that each nonterminal symbol \( A \) has the rank, denoted by \( \text{rank}(A) \), that is a nonnegative integer.

**Definition 4.3** (Slisenko [101]) A context-free graph grammar (CFGG) \( G = (\sigma, R) \) is defined as follows:

1. \( \sigma \) is a simple-star without any legs, called the axiom.
(2) \( R \) is a finite set of productions of the form \( Y \rightarrow Z \), where \( Y \) is a simple-star with \( r \) legs such that its kernel is labeled by a nonterminal symbol \( A \) with \( \text{rank}(A) = r \), and \( Z \) is a star graph with same \( r \) legs as \( Y \).

We draw a CFGG \( \mathcal{G} \) in Figure 4.10 as an example.

Let \( \mathcal{G} \) be a CFGG. For a simple-star \( Y \), a star graph \( Z \), and \( i \geq 1 \), we define the relation \( Y \rightarrow^i Z \) inductively as follows:

1. We denote \( Y \rightarrow^1 Z \) if there is a production \( Y \rightarrow Z \) in \( \mathcal{G} \).

2. For \( i \geq 2 \), we denote \( Y \rightarrow^i Z \) if there are \( j, l \geq 1 \), a star graph \( Z_1 \) having a simple-star \( Y_0 \) as a subgraph, and a star graph \( Z_2 \) such that \( j + l = i \), \( Y \rightarrow^j Z_1 \), \( Y_0 \rightarrow^l Z_2 \), and \( Z \) is a star graph by replacing the kernel of the subgraph \( Y_0 \) of \( Z_1 \) by the kernel of \( Z_2 \).
We write $Y \rightarrow^+ Z$ if $Y \rightarrow^i Z$ for some $i \geq 1$. A CFGG $G$ generates a vertex-colored graph $g$ if $\sigma \rightarrow^+ g$.

**Theorem 4.5** The parsing problem for a CFGG is $\text{NC}^1$-reducible to the refutation tree problem for a bounded simple FGS.

**Proof.** Let $G$ be a CFGG. Without loss of generality, we assume that each star graph on the right-hand side of a production in $G$ has at most two vertices labeled by nonterminal symbols, and that $p$ is a nonterminal symbol not appearing in $G$.

A *linear term graph* over $(\Sigma, A)$ with $m$ edges is a ground term graph $f = (V_f, E_f, \varphi_f, \psi_f)$ defined as follows: The vertex set $V_f$ consists of $m + 1$ vertices $v_1, \ldots, v_{m+1}$ labeled with symbols in $\Sigma$. The edge set $E_f$ contains an edge labeled with a symbol in $A$ which has vertices $v_i$ and $v_{i+1}$ as endpoints for each $1 \leq i \leq m$. The vertices $v_1$ and $v_{m+1}$ are called the *end-vertices*. We say that $f' = (V', E', \varphi', \psi', H', \lambda', \text{ports'})$ is a *star term graph* for a variable $x$ if $E' = \emptyset$ and $H'$ consists of a unique hyperedge $e$ labeled with $x$ such that the set of ports of $e$ is $V'$.

We define a bounded simple FGS $\Gamma_G$ in the following way: Let $N$ be the set of nonterminal symbols in $G$ and the nonterminal symbol $p$, and $\Sigma$ be the set of terminal symbols in $G$. Let $\Sigma_1$ and $\Sigma_2$ be non-empty disjoint sets of symbols such that $(N \cup \Sigma) \cap (\Sigma_1 \cup \Sigma_2) = \emptyset$, and let $\Lambda_1$ and $\Lambda_2$ be non-empty disjoint sets. We regard each nonterminal symbol $A$ in $N$ as a unary predicate symbol. For a unary predicate symbol $A$, let $\text{reg}(A)$ be a sequence of $\text{rank}(A)$ distinct symbols in $\Sigma$. That is, $\Sigma_2 = \bigcup_{A \in N} \text{set}(\text{reg}(A))$. For a simple-star $Y$ in $G$, let $\text{labels}(Y)$ be a sequence of $m_Y$ symbols in $\Lambda_2$ where $m_Y$ is the number of legs of $Y$. For each production $Y \rightarrow Z$ in $G$, let $A$ and $B_1, B_2$ be nonterminal symbols in $Y$ and $Z$, respectively. If $Y = \sigma$ then let $A$ be the nonterminal symbol $p$. Then, $\Gamma_G$ contains the graph rewriting rule.
$A(h) \leftarrow B_1(g_1), B_2(g_2)$, where simple term graphs $h, g_1, g_2$ with the enclosure set $\Lambda_2$ in $T(\Sigma \cup \Sigma_1 \cup \Sigma_2; \Lambda_1 \cup \Lambda_2; X)$ are defined as follows (see Example 4.2):

(1) We assume that the star graph $Z = (G, R)$ has two simple-stars $Y_1 = (G_1, R_1)$ and $Y_2 = (G_2, R_2)$ as its subgraphs. Let $h = (V_h, E_h, \varphi_h, \psi_h, H_h, \lambda_h, \text{ports}_h)$ be a term graph obtained from $Z$ in the following way: Let $f$ be a ground term graph over $\langle N \cup \Sigma; \Lambda_1 \rangle$ defined as the vertex-colored graph $G$ having an edge labeling from the edge set of $G$ to $\Lambda_1$. For each edge $e$ in $f$ with endpoints $u_e$ and $v_e$, we attach a new linear term graph $f_e$ over $\langle \Sigma_1, \Lambda_1 \rangle$ with 3 edges to $f$ by removing $e$ from $f$ and by identifying $u_e$ and $v_e$ with the end-vertices $u_f$ and $v_f$ of $f_e$, respectively. Then, the labels of $u_e$ and $v_e$ are $\varphi(u_e)$ and $\varphi(v_e)$, respectively. If $u_e$ (resp., $v_e$) is the kernel of the simple-star $Y_i$ ($i = 1, 2$) then we remove $e$ from $R_i$ and add an edge having $u_f$ (resp., $v_f$) of $f_e$ to $R_i$ as a leg of $Y_i$. Let $f_1$ be the resulting graph over $\langle N \cup \Sigma \cup \Sigma_1; \Lambda_1 \rangle$.

For each leg $r$ of $Z$, let $\text{com}(r)$ be the vertex that $r$ and the vertex-colored graph $G$ share in common. Let $r_1, \ldots, r_l$ be the legs of $Z$. For each $1 \leq i \leq l$, we join $f_1$ and a new linear term graph $h^i$ with 2 edges by identifying $\text{com}(r_i)$ of $f_1$ with an end-vertex $u$ of $h^i$, where $h^i$ satisfies the following conditions:

(a) Let $v$ be another end-vertex of $h^i$. Then the label of $v$ is the $i$th symbol in $\text{reg}(A)$, and the other vertices are labeled with symbols in $\Sigma_1$.

(b) The label of the edge having $v$ is the $i$th symbol of $\text{labels}(Y)$ in $\Lambda_2$, and the other edge is labeled with a symbol in $\Lambda_1$.

Here, the label of $\text{com}(r_i)$ in the resulting term graph is that of $\text{com}(r_i)$ in $f_1$. If $\text{com}(r_i)$ is the kernel of $Y_j$ ($j = 1, 2$) then we remove $r_i$ from $R_j$ and add the edge having $u$ of $h^i$ to $R_j$. Let $f_2$ be the resulting ground term graph over $\langle N \cup \Sigma_1 \cup \Sigma_2; \Lambda_1 \cup \Lambda_2 \rangle$. We remark that for each symbol $a$ in $\text{reg}(A)$, the number of vertices in $f_2$ labeled with $a$ is at most one.
For each \( i = 1, 2 \), we construct a star term graph \( g_{Y_i} \) for a variable \( x_i \) from \( f_2 \) such that the rank of \( x_i \) is equal to the number of legs of \( Y_i \). For each leg \( r \) of \( Y_i \), let \( \text{end}(r) \) be an endpoint of \( r \) not the kernel of \( Y_i \). The term graph \( h \) is obtained from \( f_2 \) in the following way: For each \( i = 1, 2 \), let \( r_1^i, \ldots, r_{l_i}^i \) be the legs of \( Y_i \). For each \( i = 1, 2 \), we attach a star term graph \( g_{Y_i} \) to \( f_2 \) by removing the kernel of \( Y_i \) and its all adjoining edges from \( f_2 \) and by identifying the \( j \)th port of the unique hyperedge of \( g_{Y_i} \) with \( \text{end}(r_j^i) \) for each \( 1 \leq j \leq l_i \).

(2) Let \( e_1 \) and \( e_2 \) be the corresponding hyperedges in \( h \) to the simple-star \( Y_1 \) and \( Y_2 \) in \( Z \). For each \( i = 1, 2 \), the term graph \( g_i \) is obtained by applying the same operation as (1) to \( Y_i \) such that the label of its unique hyperedge is of \( e_i \) in \( h \).

It is easy to see that \( \Gamma_G \) consisting of these graph rewriting rules is simple and bounded. Since \( G \) is fixed, we can compute \( \Gamma_G \) in constant time.

Let \( F = (V, E, \varphi) \) be a vertex-colored graph over \( \Sigma \) and let \( g = (V, E, \varphi, \psi) \) be a ground term graph over \( \langle \Sigma, \Lambda_1 \rangle \) obtained from \( F \) by adding an edge labeling \( \varphi : E \to \Lambda_1 \). We construct a ground term graph \( g' \) from \( g \) in the following way: For each edge \( e \) of \( g \) which has endpoints \( u \) and \( v \), we attach a new linear term graph with 3 edges to \( g \) by removing \( e \) from \( E \) and by identifying its two end-vertices \( w_1 \) and \( w_2 \) with \( u \) and \( v \), respectively. This transformation from \( g \) to \( g' \) is computable in constant time. Then, when a vertex-colored graph \( F \) is given, the parsing problem for a CFGG \( G \) is \( \text{NC}^1 \)-reducible to the refutation tree problem for a corresponding bounded simple FGS \( \Gamma_G \) whose instance is a colored-graph \( g' \).

Example 4.2 Let \( G \) be the CFGG in Figure 4.10. Then we draw the bounded simple FGS \( \Gamma_G \) in Figure 4.11 such that the parsing problem for \( G \) is \( \text{NC}^3 \)-reducible to the refutation tree problem for \( (\Gamma_G, p) \).
Figure 4.11: Bounded simple FGS $\Gamma_G$. The labels of thick lines are in the enclosure set.
4.5 Discussion

The refutation tree problem for an FGS can be regarded as the membership problem for the FGS, since it simultaneously decides whether an input graph is generated by the FGS.

Graph grammars are generally considered as rewriting systems with a hard membership problem [1, 2, 33, 51, 59, 78]. For example, Janssens and Rozenberg [59] have proved that NLC grammars can generate PSPACE-complete graph languages. Main and Rozenberg [78] have shown that the family of graphs generated by handle NLC grammars is exactly the recursively enumerable graph languages. Moreover, HRGs can generate NP-complete graph languages [51], e.g., all graphs of cyclic bandwidth \( \leq k \) for a constant \( k \geq 2 \) [72], where the cycle-bandwidth problem, which is to arrange the vertices of a graph on a circle so that any pair of adjacent vertices have a distance at most \( k \), is NP-complete [72]. From Theorem 3.1, regular FGSs can also generate NP-complete languages. The membership problem for a regular FGS is in NP, since the membership problem for a BNLC grammar is in NP [91]. Therefore, the membership problem for a regular FGS is NP-complete. We conjecture that the refutation tree problem for a regular FGS is NP-complete.

On the other hand, it is shown in [71] that the following two conditions s-separability and componentwise derivations admit a polynomial-time algorithm for solving the membership problem for an HRG \( G \):

1. **s-separability:** Removal of any \( s \) vertices from a graph \( G \) generated by \( G \) never creates more than \( O(\log n) \) connected components, where \( s \) is a constant depending on \( G \) and \( n \) is a number of vertices of \( G \).
2. **Componentwise derivations:** For every graph \( G \) and every nonterminal \( A \), \( G \) can be derived from \( A \).
Hence, the membership problem for a regular FGS satisfying the above two conditions (1) and (2) is solvable in polynomial time.

In [12, 68, 91], it is shown that the connectedness and maximal degree of the considered graphs are really crucial for the membership problem for an NLC grammar with the finite Church Rosser property (fCR). Brandenburg [12] and Lange and Welzl [68] have given the following three results:

(1) There is an NLC grammar $G$ which does not have fCR such that graphs generated by $G$ are connected and has degree at most 4, and $G$ can generate PSPACE-complete graph languages.

(2) There is an NLC grammar $G$ with fCR such that graphs generated by $G$ are connected and have unbounded degree, and $G$ can generate NP-complete graph languages.

(3) There is an NLC grammar $G$ with fCR such that graphs generated by $G$ are not connected and have degree at most 2, and $G$ can generate NP-complete graph languages.

In Theorem 4.4, the connectedness of a given graph is very important from the same argument. Theorem 4.4 implies that the membership problem for a simple FGS is in P. However, it is not known whether the refutation tree problem for a simple FGS is P-complete.

Moreover, it remains for us to show the complexities of the refutation tree problems for a size-bounded FGS and a size-bounded simple FGS.
Chapter 5

Refutation Tree Problem and Its Relation to Graph Isomorphism Problem

FGSs generate graphs by replacing hyperedges labeled with the same variable at a time. Then, in order to compute a refutation tree of an input graph, it may be required to solve the graph isomorphism problem defined as follows:

**Graph Isomorphism Problem**

**INSTANCE:** Two graphs $g_1$ and $g_2$.

**QUESTION:** Are $g_1$ and $g_2$ isomorphic?

In this chapter, we investigate the relationship between the refutation tree problem and the graph isomorphism problem, and show that the complexity of the refutation tree problem for a size-bounded simple FGS centers around that of the graph isomorphism problem for graphs of constantly bounded valence. The purpose of this chapter is to give subclasses of size-bounded simple FGSs for which the refutation tree problem is in NC.

In Section 4.3, we presented a polynomial time algorithm for solving the refutation tree problem for a simple FGS by constructing and solving a path system, when an input graph is connected. A generation tree of a path system depicts how an instance is derived from the path system. Rytter [94] has shown that a path system $Q$ can
be solved in NC², when a generation tree of Q is of polynomial size. For a sizebounded simple FGS, we show that generation trees of the path systems constructed by our polynomial time algorithm are of polynomial size. Therefore, the refutation tree problem for a size-bounded simple FGS is NC²-reducible to the graph isomorphism problem for graphs of constantly bounded valence. In Section 5.1, for a size-bounded simple FGS generating undirected trees of constantly bounded valence, we show that the refutation tree problem is in NC by employing the NC² algorithm in [7] for the graph isomorphism problem. Lingas [73] has shown that the graph isomorphism problem is in NC³, when an input graph is a connected graph of constantly bounded valence and having a constantly bounded separator. We can show that the refutation tree problem for a size-bounded simple FGS is in NC, when an input graph is a connected graph of constantly bounded valence and having a constantly bounded separator. A bounded simple FGS is a size-bounded simple FGS such that it is not required to solve the graph isomorphism problem in order to compute a refutation tree. Moreover, we show that the refutation tree problem for a bounded simple FGS is in NC².

On the other hand, in Section 5.2, we present a size-bounded FGS ΓGI such that the graph isomorphism problem is log-space reducible to the decision version of the refutation tree problem for ΓGI.

This chapter is based on the paper [109].

5.1 NC Algorithm for a Size-Bounded Simple FGS

In this section, we give some subclasses of size-bounded simple FGSs for which the refutation tree problem is in NC.

Theorem 5.1 Let Γ be a size-bounded simple FGS and p be a predicate symbol in Γ. When an input graph is connected, RT(Γ, p) is NC²-reducible to the graph isomorphism
problem for graphs of constantly bounded valence.

Proof. From Lemma 3.1, graphs definable by \( \Gamma \) are of constantly bounded valence. By using the algorithm in Theorem 4.4, the problem of constructing a path system \( Q = (N, T, GEN, S) \) simulating how \( \Gamma \) generates graphs is \( \text{NC}^2 \)-reducible to the graph isomorphism problem for graphs of constantly bounded valence.

An element \( [q, a_1, \ldots, a_l] \) of \( N \) is realizable if there is a refutation from the goal \( q(g[a_1], \ldots, g[a_l]) \) on \( \Gamma \). By the definitions of the simple FGS \( \Gamma \), the set \( N \) has a polynomial number of elements. Since \( \Gamma \) is size-bounded, the generation tree of each realizable element in the path system \( Q \) is of polynomial size. Therefore, the set of realizable elements is computable by an \( \text{NC}^2 \) algorithm [94]. Thus, we can solve the problem \( Q \) in \( \text{NC}^2 \). We can compute a refutation tree of an input graph from a generation tree of \( Q \) in the same way as Theorem 4.4.

Moreover, let \( \Gamma \) be a size-bounded simple FGS and \( p \) be a unary predicate symbol in \( \Gamma \) such that each graph rewriting rule \( q_0(g_{01}, \ldots, g_{0k}) \leftarrow q_1(g_{11}, \ldots, g_{1l}), \ldots, q_k(g_{k1}, \ldots, g_{kl}) \) in \( \Gamma \) satisfies the following conditions: Let \( g_i^j = (V_i^j, E_i^j, \varphi_i^j, \psi_i^j, H_i^j, \text{ports}_i^j) \) for \( 0 \leq i \leq k \) and \( 1 \leq j \leq l_i \).

1. For each \( 0 \leq i \leq k \) and \( 1 \leq j \leq l_i \), \( g_i^j = (V_i^j, E_i^j, \varphi_i^j, \psi_i^j) \) is an undirected colored-tree.

2. For each \( 0 \leq i \leq k \) and \( 1 \leq j \leq l_i \), the number of ports of every hyperedge in \( H_i^j \) is only one.

Then it is easy to see that graphs generated by \( \Gamma \) are undirected colored-trees of constantly bounded valence. The graph isomorphism problem is in \( \text{NC}^2 \) [7] when an input graph is restricted to a tree of constantly bounded valence. Hence, from Theorem 5.1, the refutation tree problem \( RT(\Gamma, p) \) is in NC.
Let \( m \) be a positive integer. Let \( g = (V, E, \varphi, \psi) \) be a ground term graph and let \( W_1, \ldots, W_k \) be subsets of \( V \). We assume that \( W \) is a subset of \( V \) such that the number of vertices in \( W \) is not greater than \( m \). The sequence \((W_1, \ldots, W_k, W)\) is an \( m\)-separation of \( g \) if the removal of \( W \) from \( g \) disconnects \( g \) into connected components \( f_1, \ldots, f_k \) such that the vertex set of \( f_i \) is \( W_i \) for each \( 1 \leq i \leq k \). A ground term graph \( g = (V, E, \varphi, \psi) \) is said to have an \( m\)-separator if, for any subset \( U \) of \( V \) with \( l \) vertices, there is an \( m\)-separation \((W_1, \ldots, W_k, W)\) of \( g \) such that for each \( 1 \leq i \leq k \), the intersection \( W_i \cap U \) has less than \((2/3)l\) vertices. We remark that, if \( g \) has an \( m\)-separator then any subgraph of \( g \) has also an \( m\)-separator. Let \( h \) and \( g \) be connected ground term graphs of constantly bounded valence. We assume that \( h \) has an \( m\)-separator. Then Lingas [73] has presented the \( \text{NC}^3 \) algorithm deciding whether \( h \) and \( g \) are isomorphic. Let \( \Gamma \) be a size-bounded simple FGS and \( p \) be a unary predicate symbol in \( \Gamma \). When an input graph is restricted to the family of connected graphs of constantly bounded valence and having constantly bounded separators, the refutation tree problem \( \text{RT}(\Gamma, p) \) is in \( \text{NC} \) from Theorem 5.1.

**Corollary 5.1** Let \( \Gamma \) be a bounded simple FGS and \( p \) be a unary predicate symbol in \( \Gamma \). Then \( \text{RT}(\Gamma, p) \) is in \( \text{NC}^2 \).

**Proof.** From the definition of \( \Gamma \), we don't need to solve the graph isomorphism problem for graphs of constantly bounded valence in order to compute a refutation tree of an input graph for \((\Gamma, p)\). Hence, from Theorem 5.1, \( \text{RT}(\Gamma, p) \) is in \( \text{NC}^2 \).

From Theorem 4.5 and Corollary 5.1, we can see that this corollary includes the result in [95]. Rytter and Szymacha [95] have shown that the following problems are in \( \text{NC}^1 \) if the parse tree is provided: the graph \( k\)-colorability problem, the Hamiltonian cycle problem, the traveling salesman problem, the induced path problem, the independent set problem, the dominating set problem and the vertex cover problem [40]. All these
problems are NP-complete. Hence Theorem 4.5 and Corollary 5.1 assert that efficient parallel algorithms may exist for a large number of NP-complete problems when these problems are restricted to the families of graphs definable by bounded simple FGSs.

5.2 Refutation Tree Problem equivalent to Graph Isomorphism Problem

We present a size-bounded FGS $\Gamma_{GI}$ in Figure 5.1 such that the graph isomorphism problem is reduced to the decision version of the refutation tree problem for $\Gamma_{GI}$.

Let $\Gamma$ be an FGS and $p$ be its unary predicate symbol. The decision version of the refutation tree problem for $(\Gamma, p)$, denoted by $\text{DRT}(\Gamma, p)$, is defined as follows:

\[
\text{DRT}(T, P) \\
\text{INSTANCE: } A \text{ graph } g \\
\text{QUESTION: } \text{Is there a refutation from the goal } \leftarrow p(g) \text{ in } \Gamma? 
\]

\[\text{Theorem 5.2} \quad \text{Let } \Gamma_{GI} \text{ be a size-bounded FGS in Figure 5.1. Then the graph isomorphism problem is log-space reducible to } \text{DRT}(\Gamma_{GI}, p).\]

\[\text{Proof.} \quad \text{Let } g_1 = (V_1, E_1, \varphi_1, \psi_1) \text{ and } g_2 = (V_2, E_2, \varphi_2, \psi_2) \text{ be ground term graphs over } \langle \Sigma, \Lambda \rangle. \text{ Let } \alpha \text{ is a new symbol not in } \Lambda. \text{ Then, we construct a ground term graph } g = (V, E, \varphi, \psi) \text{ over } \langle \Sigma, \Lambda \cup \{\alpha\} \rangle \text{ as follows:}\\n\]

\[\text{(1) } V = V_1 \cup V_2.\]

\[\text{(2) Let } c \text{ be a new edge having a vertex in } V_1 \text{ and a vertex in } V_2 \text{ as endpoints. Then } \]

\[E = E_1 \cup E_2 \cup \{c\}\]
(3) For each vertex \( v \) in \( V \), if \( v \in V_1 \) then \( \varphi(v) = \varphi_1(v) \). Otherwise \( \varphi(v) = \varphi_2(v) \).

(4) For each edge \( e \) in \( E_1 \cup E_2 \), if \( e \in E_1 \) then \( \psi(e) = \psi_1(e) \). Otherwise \( \psi(e) = \psi_2(e) \).

For the new edge \( c \), \( \psi(c) = \alpha \).

The ground term \( g \) is computable from \( g_1 \) and \( g_2 \) in log-space. We can decide whether \( g_1 \) and \( g_2 \) are isomorphic by solving DRT(\( \Gamma_{\text{Gr}} \), \( p \)) for \( g \).

The graph isomorphism problem is not known whether it is NP-complete or not. Then, the refutation tree problem for (\( \Gamma_{\text{Gr}} \), \( p \)) is also open.

5.3 Discussion

An FGS is said to be occurrence-bounded if each graph rewriting rule in the FGS is occurrence-bounded. For example, regular FGSs are occurrence-bounded. From the definition of occurrence-bounded, we can see easily that it is not required to solve the graph isomorphism problem in order to compute a refutation tree of a graph generated by an occurrence-bounded FGS. However, for an FGS not being occurrence-bounded, e.g., \( \Gamma_{\text{at}} \) in Figure 3.2, we must solve the graph isomorphism problem. Therefore, we investigate the graph isomorphism problem in order to find subclasses of FGSs for which the refutation tree problem is in NC.

The graph isomorphism problem is a special case of the subgraph isomorphism problem.

Subgraph Isomorphism Problem

INSTANCE: Graphs \( g \) and \( h \).

QUESTION: Does \( g \) contain a subgraph isomorphic to \( h \)?

A graph \( g \) is 2-connected if there is no vertex \( v \) such that we can disconnect the graph by removing \( v \). In [74], Lingas and Syslo have given a polynomial time algorithm and an NC algorithm for solving the subgraph isomorphism problem of 2-connected series parallel graphs and these graphs of constantly bounded valence, respectively. By
employing the polynomial time algorithm and the NC algorithm, we may be able to
find subclasses of FGSs which generate 2-connected series parallel graphs and for which
the refutation tree problem admits a polynomial time algorithm and an NC algorithm.

Moreover, Gazit [42] have presented a parallel algorithm (not an NC algorithm)
to determine whether two planar graphs are isomorphic. We may be able to design a
parallel algorithm to solve the refutation tree problem for an FGS generating planar
tables which is not occurrence-bounded. But, even an FGS generating all planar
tables has not been known.
Chapter 6

$O(\log^* n)$ Time Parallel Algorithms for Bounded Degree Maximal Subgraph Problems

The bound of valence of a given graph is one of the essential properties contributing to a drastic speed-up of a parallel graph algorithm, e.g., the refutation tree problem for a simple FGS in previous chapter. In this chapter, we show that the vertex-coloring technique is very useful for designing fast parallel graph algorithms when a given graph is of constantly bounded valence.

For a given integer $k$, we consider the problem of finding a maximal subset of vertices (resp., edges) whose induced subgraph is of valence at most $k$. We denote the problem by $\text{VIMS}(k)$ (resp., $\text{EIMS}(k)$). Shoudai and Miyano [99, 100] have shown that $\text{VIMS}(k)$ and $\text{EIMS}(k)$ are in NC by describing algorithms which employ the parallel maximal independent set (MIS) algorithm[64, 76]. In their algorithms, maximal independent sets are repeatedly computed $k^2$ times for $\text{VIMS}(k)$ and $2k$ times for $\text{EIMS}(k)$, respectively. Later, Diks et al. [23] have independently given the same results as Shoudai and Miyano [99] with the same argument. If we apply the fast parallel MIS algorithm in [44] to the algorithms in [23, 99], we can easily see that $\text{VIMS}(k)$ (resp., $\text{EIMS}(k)$) for graphs of constantly bounded valence can be solved in
$O(\log^* n)$ (resp., $O(\log^* m)$) time with $O(n)$ (resp., $O(m)$) processors on an EREW PRAM, where $n$ (resp., $m$) is the number of vertices (resp., edges) of an input graph. The function $H(n) = \log^* n$ is defined as follows: Let

\[
F(0) = 1,
F(i) = 2^{F(i-1)}, \quad \text{for } i > 0.
\]

The function $H(n) = \log^* n$ is the smallest integer $j$ such that $F(j) \geq n$. $H(n)(= \log^* n)$ grows extremely slowly and can be viewed as a constant for all practical purposes. For example, $H(2^{65536}) = \log^* 2^{65536} = 5$. Hence, it is important to focus on the constants $k$ and the valence $\Delta$ of an input graph. In this chapter, we apply the coloring technique to VIMS($k$) and EIMS($k$), and obtain faster parallel algorithms for these problems from this point of view. When the vertex coloring algorithm by [44] is used, for an input graph with valence at most $\Delta$, our algorithm runs $k$ times as fast as the algorithm in [23, 99] equipped with the MIS algorithm in [44]. Moreover, the number of processors needed by our algorithm is $\Delta/k$ times as few as that of their algorithm. If the valence $\Delta$ of an input graph satisfies $\Delta = o(\log n)$, our method also provides an algorithm faster than that in [23, 99] even though we apply the $O((\log n)^2)$ time parallel MIS algorithm by [76] to their algorithm.

Furthermore, the edge coloring technique works efficiently to solve EIMS($k$). While their algorithms [23, 99] for EIMS($k$) need to solve MIS $2k$ times, it is sufficient for our algorithm to compute the edge coloring only once. Therefore, our algorithm runs $2k$ times as fast for graphs with valence at most $\Delta = o(\log n)$.

This chapter is based on the paper [107].

6.1 Preliminaries

We consider a graph $G = (V, E)$ as an undirected graph without any multiple edges and self-loops. Let $n$ and $m$ be the numbers of vertices in $V$ and edges in $E$, respectively.
For a subset $U \subseteq V$, we define $E[U] = \{u, v \in E \mid u, v \in U\}$. The graph $G[U] = (U, E[U])$ is called the vertex-induced subgraph of $U$. We define $V[F]$ to be the set of endpoints of the edges in $F$ for a subset $F \subseteq E$. We denote by $\langle F \rangle = (V[F], F)$ the graph formed from $F$ and call it the edge-induced subgraph of $F$. For a graph $G$, the valence of $G$ is denoted by $\text{val}(G)$.

A vertex coloring $C$ of $G$ is a mapping $C : V \rightarrow \mathbb{N}$ from the vertices to positive integers (colors), and it is valid if no two adjacent vertices have the same color.

**Definition 6.1** Let $G = (V, E)$ be a graph and let $k \geq 0$ be any integer. The maximum degree $k$ vertex-induced maximal subgraph problem (VIMS($k$)) is to find a maximal subset $U \subseteq V$ such that $G[U]$ is of valence at most $k$.

**Definition 6.2** Let $G = (V, E)$ be a graph and let $k \geq 1$ be any integer. The maximum degree $k$ edge-induced maximal subgraph problem (EIMS($k$)) is to find a maximal subset $F \subseteq E$ such that $\langle F \rangle$ is of valence at most $k$.

We assume an EREW PRAM model of computation where each processor is capable of executing a special operation which counts the number of bit 1's in a word together with conventional simple word and bit operations [15]. The word length is assumed to be $O(\log n)$. Goldberg et al. [44] have presented a vertex coloring algorithm that yields the following lemma under the above conditions of the PRAM model:

**Lemma 6.1** (Goldberg et al. [44]) Let $\Delta$ be an integer. Given a graph $G = (V, E)$ with valence at most $\Delta$, a valid vertex coloring of $G$ with $\Delta + 1$ colors can be computed in $O(\Delta(\Delta + \log^* n) \log \Delta)$ time on an EREW PRAM using $\Delta n$ processors.

### 6.2 Bounded Degree Maximal Vertex-Induced Subgraph Problem

In this section we show an algorithm which solves VIMS($k$) efficiently.
Theorem 6.1 Let \( k \) and \( \Delta \) be nonnegative integers with \( 0 \leq k \leq \Delta \). For a graph \( G = (V, E) \) with valence at most \( \Delta \), VIMS\((k)\) can be solved in \( O(\log^* n) \) time using \( O(n) \) processors on an EREW PRAM.

Proof. Our VIMS algorithm takes a graph \( G = (V, E) \) of valence at most \( \Delta \) as an input and outputs a maximal subset \( S \subseteq V \) such that \( G[S] \) is of valence at most \( k \).

We need to prepare some notations in order to describe the algorithm precisely.

Let \( C : V \rightarrow \mathbb{N} \) be a \((\Delta + 1)\)-vertex coloring of \( G \) with valence at most \( \Delta \). For each \( i = 0, \ldots, \Delta \), let \( C_i(V) = \{ v \in V \mid C(v) = i \} \). For a subset \( S \subseteq V \) and a vertex \( v \in V \), let \( U[v][S] \) be the set of vertices in \( S \) that are adjacent to \( v \). For subsets \( W \) and \( U \) of vertices with \( W \cap U = \emptyset \), let \( E^W_U = \{ \{ v, w \} \mid v, w \in W, w \neq v \text{ and there is } u \in U \text{ such that } \{ v, u \} \in E \} \).

The algorithm is described in Figure 6.1.

We show that the algorithm correctly computes a required maximal subset \( S \). First it colors the input graph \( G \) with colors \( 0, \ldots, \Delta \) at line 2. Then, for each color \( i \), the algorithm determines which vertices colored \( i \) are added to \( S \). Let \( S_i, X_i, Y_i \) and \( Y'_i \) be the contents of \( S, X, Y \) and \( Y' \) at the end of the \( i \)th iteration of lines 3-20, respectively.

We assume that \( S_{i-1} \) is a maximal subset of \( C_0(V) \cup \cdots \cup C_{i-1}(V) \) which induces a subgraph of valence at most \( k \). For \( i = 0 \), the assumption holds obviously since \( S_0 \) is an independent set \( C_0(V) \). We show that the induced subgraph \( G[S_i] \) in \( G[S_{i-1} \cup C_i(V)] \) is of valence at most \( k \) and maximal.

Clearly, after executing lines 4-8, the graph \( G[S] \) is of valence at most \( k \), but may not be maximal. We now prove that the induced subgraph \( G[S_i] \) becomes of valence at most \( k \) and is maximal after lines 9-19. Let \( D^i : Y' \rightarrow \mathbb{N} \) be the \((k\Delta + 1)\)-vertex coloring of the graph \( G_i = (Y'_i, E^i_{Y'_i}) \) computed at line 10. For any two vertices \( v, w \) in \( Y' \) which are adjacent to a vertex in \( Y \), we can see that \( D^i(v) \neq D^i(w) \), since an edge
VIMS Algorithm:

1. $S \leftarrow \emptyset; V' \leftarrow V; i \leftarrow 0;$
2. Compute a $(\Delta + 1)$-vertex coloring $C$ of $G = (V, E);$  
3. while $V' \neq \emptyset$ do
   4. $X \leftarrow C_i(V');$
   5. $V' \leftarrow V' - X;$
   6. $Y \leftarrow \{v \in S \mid d_{G[S \cup X]}(v) > k\};$
   7. $Y' \leftarrow \{v \in X \mid U_v[Y] \neq \emptyset\};$
   8. $S \leftarrow S \cup (X - Y');$
   9. if $Y \neq \emptyset$ then
      10. Compute a $(k\Delta + 1)$-vertex coloring $D^i$ of $G_i = (Y', E_{Y'});$  
      11. $W \leftarrow Y'; j \leftarrow 0;$  
      12. while $W \neq \emptyset$ do
         13. $S \leftarrow S \cup D^i_j(W);$  
         14. $W \leftarrow W - D^i_j(W);$  
         15. $W \leftarrow W - \{w \in W \mid \text{val}(G[S \cup \{w\}] > k\};$
         16. $j \leftarrow j + 1$
      17. end do
   18. $V' \leftarrow V' - \{w \in V' \mid \text{val}(G[S \cup \{w\}] > k\};$
   19. $i \leftarrow i + 1$
   20. end do

Figure 6.1: Algorithm VIMS
\{v, w\} is in $E^{Y'}_Y$ by the definitions of $Y, Y'$ and $E^{Y'}_Y$. Since the vertices in $V'$ at line 18, $d_{G[S_{i-1}\cup X_i]}(v) \leq k$ for a vertex $v$ in $X_i$. For a vertex $w$ in $S_{i-1}$, $d_{G[S_{i-1}\cup X_i]}(w) \leq \Delta$. Hence, from the definitions of $Y, Y'$ and $E^{Y'}_Y$, the valence of $G_i = (Y'_i, E^{Y'}_Y)$ is at most $k\Delta$. Therefore, by using the $(k\Delta + 1)$-vertex coloring $D^i$ of the graph $G_i$, the induced subgraph $G[S]$ can be made maximal, keeping the condition that the valence of the graph $G[S]$ is at most $k$. Hence, we can see that the induced subgraph $G[S_i]$ is of valence at most $k$ and is maximal. Therefore, our algorithm can solve VIMS($k$), correctly.

Finally, we show that our algorithm can compute the VIMS($k$) in $O(\log^* n)$ time using $O(n)$ processors on an EREW PRAM when a graph of constantly bounded valence is given as input. Let $T(G, \Delta)$ be the time needed to compute a valid $(\Delta + 1)$-vertex coloring of the input graph $G$ with valence at most $\Delta$ using $O(\Delta n)$ processors on an EREW PRAM. Hence, line 2 requires $T(G, \Delta)$ time using $O(\Delta n)$ processors on an EREW PRAM. We show that the time needed in the $i$th iteration of lines 3–20 as follows ($0 \leq i \leq \Delta$): Since the valence of the input graph $G$ is at most $\Delta$, lines 4–8 can be processed in $O(\log \Delta)$ time using $\Delta n$ processors. Since $\text{val}(G_i) \leq k\Delta$, line 10 needs the time $T(G_i, k\Delta)$ using $O(k\Delta n)$ processors on an EREW PRAM. It is easy to see that $T(G_i, k\Delta) \leq T(G, k\Delta)$ since the number of vertices of $Y'$ is less than or equal to that of $V$. Since the graph $G_i$ is colored with at most $k\Delta + 1$ colors, the while loop (lines 12 – 17) repeats at most $k\Delta + 1$ times. Therefore, the time needed in the $i$th iteration is $O(\log \Delta) + T(G, k\Delta) + O(k\Delta \log \Delta)$ time using $O(k\Delta n)$ processors on an EREW PRAM. Hence, the $i$th iteration runs $T(G, k\Delta)$ time using $O(k\Delta n)$ processors on an EREW PRAM. Moreover, since the input graph $G$ is colored with at most $\Delta + 1$ colors, the while loop (lines 3–20) repeats at most $\Delta + 1$ times. Line 3-20 requires $O(\Delta T(G, k\Delta))$ time using $O(k\Delta n)$ processors. Therefore, our algorithm runs
in $O(\Delta T(G, k\Delta))$ time using $O(k\Delta n)$ processors.

When we apply Lemma 6.1 to our algorithm, it runs in $O(k\Delta^2(\Delta^2 + \log^* n) \log \Delta)$ time using $O(k\Delta n)$ processors. Hence, for the graphs of constantly bounded valence, our algorithm runs in $O(\log^* n)$ time on an EREW PRAM using $O(n)$ processors.

Using the MIS algorithm in [44], the algorithm in [23, 99] can also solve $\text{VIMS}(k)$ in $O(\log^* n)$ time on an EREW PRAM using $O(n)$ processors for graphs of constantly bounded valence. The MIS algorithm given by [44] uses a $(\Delta + 1)$-coloring of an input graph $G$. Therefore, for an input graph $G$ with valence at most $\Delta$, their VIMS algorithm runs in time $O(k^2\Delta^2(\Delta^2 + \log^* n) \log \Delta)$ with $O(\Delta^2 n)$ processors. Hence, our algorithm is faster than their algorithm with less number of processors.

### 6.3 Bounded Degree Maximal Edge-Induced Subgraph Problem

In this section we apply the vertex coloring technique to $\text{EIMS}(k)$.

**Theorem 6.2** Let $k$ and $\Delta$ be positive integers with $0 \leq k \leq \Delta$. For graphs of constantly bounded valence, $\text{EIMS}(k)$ can be solved in $O(\log^* m)$ time on an EREW PRAM using $O(m)$ processors where $m$ is the number of edges of the input graph.

**Proof.** The algorithm takes a graph $G = (V, E)$ with valence at most $\Delta$ as input, and outputs a maximal subset $F \subseteq E$ such that $\langle F \rangle$ is a graph of valence at most $k$.

Let $D : E \rightarrow \mathbb{N}$ be a $(2\Delta - 1)$-edge coloring of $G$ with valence at most $\Delta$. For each $i = 0, \ldots, 2\Delta - 2$, let $D_i(E) = \{e \in E \mid D(e) = i\}$.

Formally the algorithm is described in Figure 6.2.

We show the correctness of the algorithm. Let $F_0 = \emptyset$ and $Z_0 = E$. For $0 \leq i \leq 2\Delta - 2$, let $F_i$ and $Z_i$ be the contents of $F$ and $Z$ just after the $i$th iteration. We assume
EIMS Algorithm:
1. $F \leftarrow \emptyset$; $Z \leftarrow E$;
2. Compute a $(2\Delta - 1)$ edge coloring $D$ of $G = (V, E)$;
3. $i \leftarrow 0$;
4. while $Z \neq \emptyset$ do
   5. $F \leftarrow F \cup D_i(Z)$;
   6. $Z \leftarrow Z - D_i(Z)$;
   7. $Z \leftarrow Z - \{e \in Z \mid \text{val}(\ll F \cup \{e\} \gg) > k\}$;
   8. $i \leftarrow i + 1$
5. end do

Figure 6.2: Algorithm EIMS

that $F_{i-1}$ is a maximal subset of $D_0(Z_0) \cup \cdots \cup D_{i-1}(Z_{i-1})$ such that $\ll F_{i-1} \gg$ is a maximal subgraph with the valence at most $k$ of the graph $\ll D_0(Z_0) \cup \cdots \cup D_{i-1}(Z_{i-1}) \gg$.

It is easy to see that $\text{val}(\ll F_i \gg) \leq k$ since $D_i(Z_i)$ is a matching of $\ll Z_{i-1} \gg$ and since each edge $e$ in $D_i(Z_i)$ satisfies $\text{val}(\ll F_{i-1} \cup \{e\} \gg) \leq k$. We can also see that $F_i$ is maximal subset of $F_{i-1} \cup D_i(Z_{i-1})$. Therefore, after $2\Delta - 1$ iterations, we see that the resulting $F$ is a maximal set of edges such that $\text{val}(\ll F \gg) \leq k$.

Next, we show that the algorithm runs in $O(T(G) + \Delta \log \Delta)$ time on an EREW PRAM with $p$ processors for $p \geq \Delta m$ where $T(G)$ is the time which our algorithm takes at line 2 on an EREW PRAM using $p$ processors. In lines 3-9, since the number of colors of the edge coloring of the graph $G$ is $2\Delta - 1$, it takes $O(\Delta \log \Delta)$ time on an EREW PRAM with $m$ processors. Therefore, we can see that the algorithm runs in $O(T(G) + \Delta \log \Delta)$ time on an EREW PRAM with $p$ processors for $p \geq \Delta m$.

For an input graph $G$ with valence at most $\Delta$, line 2 can be implemented in time $T(G) = O(\Delta(\Delta + \log^* m) \log \Delta)$ by constructing a line graph $G'$ of $G$ and computing
a valid vertex coloring of $G'$ with $2\Delta - 1$ colors. Hence, for the graphs of constantly bounded valence, our algorithm runs in $O(\log^* m)$ time on an EREW PRAM using $O(m)$ processors.

By using the MIS algorithm in [44], EIMS($k$) can be solved by the algorithm in [23, 99] in $O(\log^* n)$ time on an EREW PRAM with $O(n)$ processors for graphs of constantly bounded valence. However, for a graph $G$ with valence at most $\Delta$, the algorithm in [23, 99] must compute the $(2\Delta - 1)$-edge coloring $2k$ times to solve EIMS($k$), since the MIS algorithm in [44] uses the $(\Delta + 1)$-vertex coloring algorithm. Therefore, the algorithm in [23, 99] requires $O(k\Delta(\Delta + \log^* m) \log \Delta)$ time using $O(\Delta m)$ processors. On the other hand, since our algorithm computes the $(2\Delta - 1)$-edge coloring just once, the running time of our algorithm reduces to $O(\Delta(\Delta + \log^* m) \log \Delta)$ time using the same number of processors to solve EIMS($k$).

### 6.4 Discussion

We have shown that the coloring technique is very useful to devise faster parallel algorithms with less number of processors for VIMS($k$) and EIMS($k$), when instances are graphs of constantly bounded valence. This asserts that the idea of Cole and Vishkin [15] helps to solve these problems drastically faster. Another such cases are known for the maximal independent set problem [44, 62], the $(\Delta + 1)$-vertex coloring problem [44, 62], the list ranking problem [8, 15], the tree contraction problem [3] and the 5-coloring problem for planar graphs [52].

Our approach for EIMS($k$) does not seem to work for graphs without any valence constraint, since it uses the edge coloring. However, our approach to VIMS($k$) seems to work for graphs which allow NC-vertex coloring algorithms with constant colors, for example, planar graphs [52], bipartite graphs, etc.
Bibliography


