Classifying Networks: When Can Two Anonymous Networks Compute The Same Vector-Valued Functions?

Nancy E. Norris

UCSC-CRL-94-01
March 30, 1994

Baskin Center for
Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064 USA

ABSTRACT

An "anonymous network" is a computer network in which all processors run the same algorithm during a computation. We will consider two classifications of anonymous networks: Networks will be called "f-equivalent" if the set of vector-valued functions each can anonymously compute is the same, and "p-equivalent" if the set of functions each can compute is the same, "up to a permutation". We first give a characterization of the vector-valued functions a given network can anonymously compute. This extends a result in [YK88] characterizing the scalar-valued functions a network can compute. Next, we develop algebraic and graph-theoretic techniques for handling edge-labeled digraphs. We will use these techniques, along with results from algebraic automata theory and permutation group theory, to derive a polynomial-time algorithm for determining whether two networks are f-equivalent. This will yield a polynomial-time algorithm for determining whether two edge-labeled digraphs have the same lattice of quotient-graph isomorphisms, and will let us conclude that classifying networks by what they can compute is easy. Classifying networks by "p-equivalence", on the other hand, is likely to be a much harder problem. We will present a polynomial-time transformation of the group-isomorphism problem to the problem of "p-equivalence". As of this writing, the best known algorithm solves group-isomorphism in $O(n^{\lg n})$ time for a group of order n.

Contents

List of Figures

Acknowledgements

This is a revised version of my dissertation, submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Mathematics, December, 1993.

I want to thank my advisor Manfred Warmuth for the countless engrossing conversations and problem-solving sessions that led to this paper. Manfred is responsible for alerting me to the usefulness of algebraic automata theory in this work; in particular, for pointing out the connection between rooted universal covers and languages accepted by finite-state machines. He is largely responsible for the the algorithm for finding the "coarsest c-partition" of a graph, for parts of other algorithms, and for a good assortment of other ideas in this paper. Working with Manfred has been a true delight.

I want to thank Nick Littlestone for numerous ideas and proof suggestions, now incorporated into this paper. Nick persuaded me of the utility of universal covers in this inquiry, and suggested that I look at edge-label maps and their relationship with graph epimorphisms: It is due to Nick that edge-label maps frolic with isomorphisms in these pages. Nick is also the source of the best counterexamples in the state. Finally, I want to thank Nick for proofreading and for consultation on notation and definitions.

I would like to thank Al Kelley and Gerhard Ringel for their very generous assistance as readers; for the time and interesting conversations and useful feedback that they provided. Thanks to the Santa Fe Institute for their Complex-Systems Summer School, and to the Center for Nonlinear Studies and the T-Division of Los Alamos National Lab for their support during the Fall of 1990. I would also like to thank the following mathematicians, who listened to my story and pointed me to useful books and theorems: Geoff Mason, Ralph Abraham, Susan Addington, Nick Burgoyne, Bruce Cooperstein, Peter Doyle, Sol Freidberg, Ortwin Wohlrab, and others. I want to thank Jo Ann MacFarland, as well as others in the math and computer-science boards, for effecting the impossible on so very many occasions. Finally, I want to thank all and any Norris for all and everything.

1. Introduction

One year at a large philosophy conference, a group of n philosophers in a seminar discovered that they were all named Linda. Since all had the same name, talk turned naturally toward symmetry breaking experiments that they might undertake. The dining philosophers problem (involving symmetry-breaking in a ring) immediately came up, but seemed a bit old hat. Then Linda (one of them) suggested an experiment: They would form themselves into a network and attempt to distributively decide where *to eat. They went outdoors to a field, and there made 3n can telephones by connecting pairs of cans with string. Each philosopher picked up 6 cans. They then arranged themselves around the field, tightened the strings, and attempted to discern the majority preference among two eating places, by talking over their phone lines. The conversation became rather involved. One said to another, "The Linda on my third can said that the Linda on her second can said that the Linda on her fourth can wants pizza. The Linda on my fourth can said that the Linda on her second can also wants pizza. If I am going to tally their votes I need to know if these were the same Linda or two different people". Another philosopher wondered: "If we were arranged in a line or a circle instead of in our current configuration, would this problem be easier to solve?" Later, over lunch, conversation turned to the more general question of the effects of symmetry on distributed function-computation.*

Consider a system of interacting agents; for instance, the molecules in a fluid, the birds in a flock, or the neurons in a neural net. We will call such a system "anonymous" if the agents are functionally identical. This paper examines "anonymous networks"; anonymous systems whose agents are the processors in a connected network. For anonymous networks, in contrast with some other anonymous systems, the coupling between agents is fixed: Which processor is linked to which in the network does not change over time. We can state the problem of anonymous computing briefly as follows: Consider a network of n identical processors, each connected to one or more processors in the network by two-way links. Suppose that all of the processors have the same id, so that none of the processors has a distinct id. by which it can identify itself to other processors. However, each processor can distinguish among its links and has a unique label for each link. In "anonymous computing", an operator, outside of the network, assigns each processor a unique name (not available to the processors) from the set $\{1, \ldots, n\}$. The operator gives the network an *input vector* $\vec{x} = (x_1, \ldots, x_n)$ from a set $\mathbf{I^n}$: Processor 1 gets input x_1, processor 2 gets x_2, and so on. The task of the processors in the network is to compute a function $f(\vec{x}) = \vec{y} = (y_1, \ldots, y_n)$, where $\vec{y}$ is a member of a set $\mathbf{O^n}$, by communicating among themselves over their links. The network will have computed the function when processor 1 has computed y_1, processor 2, y_2, and so on. The network is said to "compute (a function) *anonymously*" if all the processors run the same algorithm during the computation. The

algorithm at processor i receives x_i as input but not the id i. Thus processors do not have access to their ids unless these are given as part of the input.

We will call such a network an *anonymous network*, and the computational problem, *computing a vector-valued function on an anonymous network*. We will use the conventional graphical representation for a network: Networks will be drawn as graphs in which vertices represent processors, and edges; two-way links between processors. An edge between processor v and processor w in a network is labeled with a pair (k, l) of "link-labels", where k is processor v's name for the link and l is processor w's name for the link. (Figure 1.1.) The "graph of a network" is an edge-labeled, directed graph which will be defined more precisely in the next chapter.

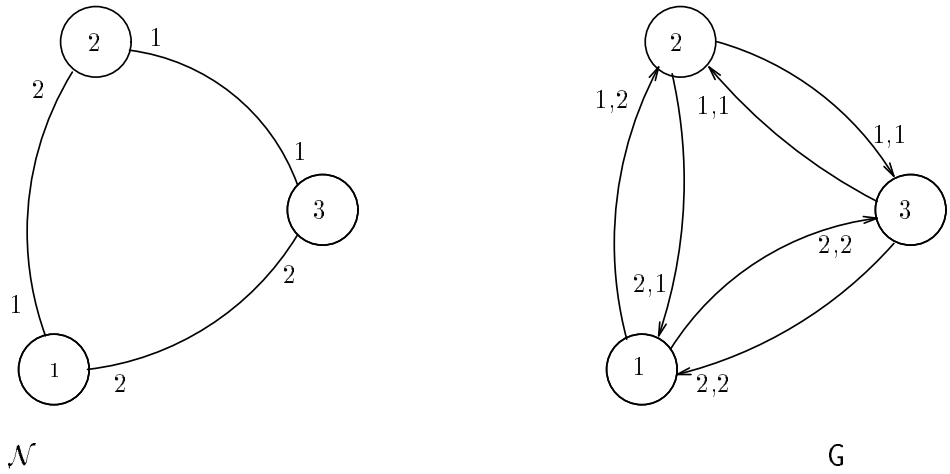


Figure 1.1: A network $\mathcal{N}$ and its graph G

For instance, the network in Figure 1.1 computes the function $f(x_1, x_2, x_3) = (x_2 + x_3, x_3 - x_2, x_3)$ if processor 1 compute $x_2 + x_3$, processor 2 computes $x_3 - x_2$, and processor 3 computes x_3 whenenver processor 1 gets input x_1, processor 2 gets input x_2, and processor 3 gets input x_3.

1.1 Why Anonymous?

There are several reasons for considering anonymous networks. Anonymous networks can be used to investigate distributed, leaderless decision-making, and also provide a convenient vehicle for studying the effects of network-topology on network behavior. One might, for instance, use anonymous networks to determine the class of computations which are impossible for a neural-net, even if its processors are universal computers running the same program instead of the usual finite-state machines. A non-anonymous network, if it is connected and has arbitrarily powerful processors, can compute any function. By contrast, the set of functions an anonymous network can compute depends on the network's topology. We will see that highly "symmetrical" anonymous networks can compute fewer functions than "asymmetrical" anonymous networks.

A potentially useful framework for anonymous networks is in the study of fault-tolerant computing[1]. Here the fault in question is in the transmission of processor ids: One can think of an anonymous network as a distributed system in which some or all of the processor ids are transmitted incorrectly during computation. A network experiencing faulty transmission of ids may not be entirely incapacitated. In some networks, depending on the network's topology, processors may be able to construct unique identities for themselves even if they are not assigned unique ids. A processor accomplishes this by making a local model of the network (by exchanging messages with other processors) and locating itself in the model. If a network is highly symmetrical, two or more processors may locate themselves in the same spot in their models of the network, and thus assign themselves the same id. In this case the processors are functionally indistinguishable and have the same behavior under any algorithm. The extent to which the global structure of a network can be reconstructed by a processor making local queries comes heavily into play in the study of anonymous networks.

1.2 Summary Of The Paper

This paper addresses three problems. They are:
(1) Characterizing the set of functions a given network can compute. In the next chapter we will develop techniques that can be used to show, for instance, that the network G_2 in Figure 1.2 can compute any function from $\mathbf{R}^3$ to $\mathbf{R}^3$, because each processor can distinguish itself from the others in the network. The processors in G_1 cannot distinguish among themselves, and G_1 can, it will turn out, only compute functions which satisfy: $f(x,x,x) = (y,y,y)$, and if $f(x_1,x_2,x_3) = (y_1,y_2,y_3)$, then $f(x_2,x_3,x_1) = (y_2,y_3,y_1)$, and $f(x_3,x_1,x_2) = (y_3,y_1,y_2)$.

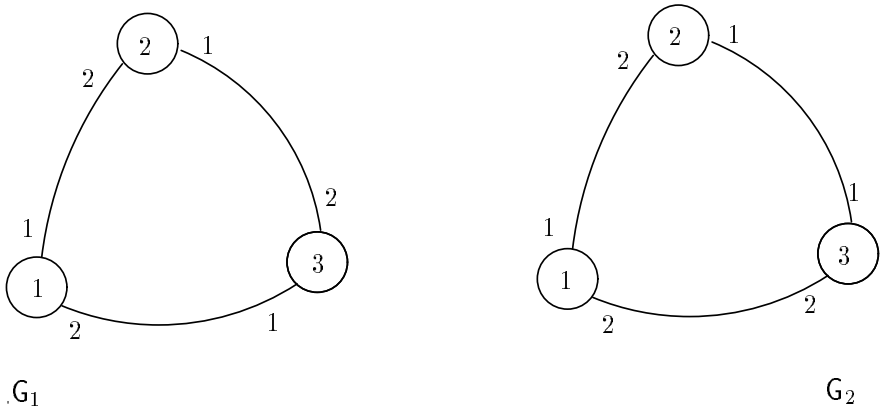


Figure 1.2: G_1 and G_2 compute different functions

(2) Classifying networks by what they can compute. We will consider two classifications: For the first, we will say that two networks are "f-equivalent" if the set of functions each can compute is the same. For the second, say that two networks are "p-equivalent" iff they

[1] See [FLM85,FLP85]

compute the same functions "up to a permutation". (This will be defined in Chapter 3.) The question for each classification is: Are there features of the graphs of networks which characterize the f- and p- equivalence-classes a given graph belongs to? For example, the networks G_1 and G_2 in Figure 1.3 look quite disimilar. However, the methods developed in this paper will show that they compute the same functions. What do their graphs have in common? (We will examine this example in greater detail in Chapter 5, in Example

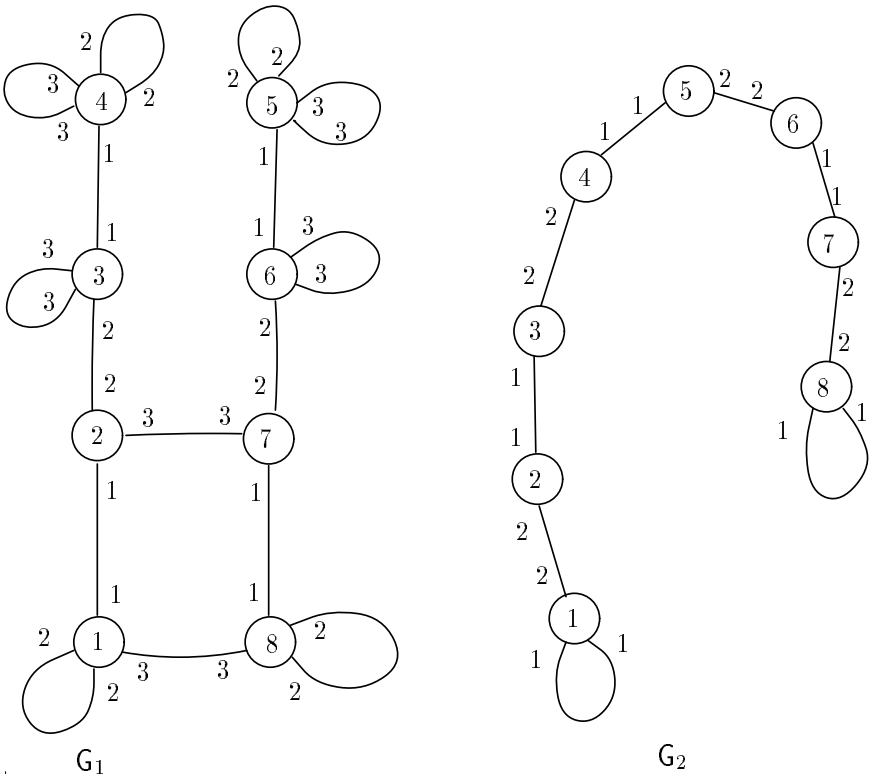


Figure 1.3: G_1 and G_2 compute the same functions

5.5.2.)

(3) Finding a small set of graph features which correctly classify graphs according to the two equivalence-relations above.

Chapter 2 addresses the first problem. In this chapter we will introduce the model and some of the tools to be used later – covering graphs, universal covers, and a monoid; the "edge-label monoid" $\mathcal{E}(\mathsf{G})$ associated with the graph G of a network. We will characterize the set of functions computable by a network in terms of a collection of trees related to the universal cover of the network's graph.

Chapter 3 introduces quotient-graphs and their isomorphisms. We will find partitions π of the set of vertices of a graph G for which a quotient-graph G/π is well-defined. The second problem — that of classifying networks, will be solved in terms of these quotients: The "characteristic graph features" we seek for distinguishing networks will be seen to be the set of all isomorphisms of quotients of the graph of a network. Since isomorphisms

are bijections between partitions, two networks can have the same set of isomorphisms of quotients even if their graphs are not isomorphic. We will see that two networks compute the same set of functions iff they have identical sets of quotient-graph isomorphisms, and that two networks compute the same set of functions "up to a permutation" iff their sets of quotient-graph isomorphisms are the same "up to a permutation".

The third problem given above — that of finding a small set of graph features which correctly classify graphs, can be rephrased as follows: Is there a small set of graph features which two graphs share iff they have the same set of quotient-graph isomorphisms, or, respectively, iff they have the same set of isomorphisms "up to a permutation"? In Chapter 5 we will see that a graph with n vertices can have $O(n^{\lg n})$ quotients and $O(n^{\lg n+1})$ isomorphisms of quotients, so the quotient-graph isomorphisms themselves are not a "small set of characteristic features". Chapters 4 and 5 address the question of finding such features for the case when the edge-label monoid $\mathcal{E}(\mathsf{G})$ of a graph G is a group. In Chapter 4 we will look at the relationship between subgroups of $\mathcal{E}(\mathsf{G})$ and quotient-isomorphisms of G; finding a one-to-one map from the set of isomorphisms of quotients of G into the set of left cosets of subgroups of $\mathcal{E}(\mathsf{G})$. We will use this correspondence in Chapter 5 to show that the set of quotient-isomorphisms of G forms a lattice. We will find a small set of generators for this lattice — these will be the desired "characteristic features" — and show that finding these generators is easy. Thus, the answer to the first classification question: "Is it easy to tell whether two graphs are f-equivalent?" – is "yes". The other classification problem — that of determining whether two graphs are p-equivalent, will be found to be at least as hard as determining whether two finite permutation groups are isomorphic.

Chapter 6 shows that the results found above for graphs in which $\mathcal{E}(\mathsf{G})$ is a group also hold for graphs in which $\mathcal{E}(\mathsf{G})$ is an arbitrary monoid. We will see that a graph in which $\mathcal{E}(\mathsf{G})$ is a monoid can be decomposed into a collection of graphs, all of which have isomorphic edge-label groups. This will let us reduce the problem of classifying arbitrary graphs to the problem discussed above, of classifying 'group graphs'.

1.3 Related Work

Anonymous Networks: The 1972 paper by Rosentiehl, Fiskel and Hollinger ([RFH72]) on "intelligent graphs" investigates networks of identical finite-state machines. At each time-step on one of these networks, each processor (that is, each finite-state machine) updates its state as a function of the states of its neighboring processors. The paper examines various problems that such a network might attempt to solve, including the "firing squad problem", in which a network has one "distinguished processor" and all processors attempt to enter a given state at the same time. Some papers on cellular automata also examine networks of finite-state machines; e.g., "Computation on Finite Networks of Automata" by M. Tchuente ([Tch87]), addresses the question of what functions are computable on such networks.

Angluin's seminal paper of 1980, *Local and Global Properties in Networks of Processors*, ([Ang80]) addresses the question of how well a network can function if the network's

processors do not have global knowledge about the network, e.g., if they do not 'know' the graph of the network or their identities. The processors in Angluin's networks are arbitrarily powerful computers instead of finite-state machines which communicate with each other by passing messages over links. Angluin assumes that all processors of the same degree are identical. The paper considers two questions: One, the question of characterizing networks which can distributively choose a 'distinguished processor' or leader, and two, the question of how well the processors in a network can construct a model of the network's graph, using only the information available through message-passing.

A number of papers on anonymous computing have suceeded Angluin's, with the majority of these considering networks having the ring topology. In *Computing on an Anonymous Ring* ([ASW88]), Attiya, Snir and Warmuth characterize the set of (scalar-valued) functions computable by a ring, and derive lower bounds for the message-complexity of synchronous and asynchronous computation. They also consider network computations other than function computation; for instance, the problem of "network orientation", in which all processors in a ring attempt to agree on a consistent notion of left and right; and "start-synchronization", in which the processors in a ring coordinate their clocks so that all begin a computation at the same time. In *Gap Theorems for Distributed Computation* ([MW93]), Moran and Warmuth show that an anonymous ring with n processors requires $\Omega(n \log n)$ messages to compute any non-constant function, and that a non-anonymous ring has the same lower bound if the processor ids are taken from a suficiently large domain. In [BMW93], Bodlander, Moran and Warmuth show that this lower bound remains even if the set of possible ids is small, i.e., is $n^{1+\epsilon}$ for positive ϵ.

In [BB89], Beame and Bodlander investigate the message complexity of distributed computing The papers by Scheiber and Snir ([SS89]) and by Matias and Afek ([MA89]) show that the processors in an anonymous network can distributively compute distinct ids for themselves by using probabalistic algorithms to break symmetry. Kranakis, Krizanc and van den Berg ([KKvdB90]) study the bit complexity of the problem of computing boolean functions on arbitrary anonymous networks.

Perhaps the most complete investigation of anonymous networks to date appears in the papers of Yamashita and Kameda ([YK87b,YK87a,YK88]). In *Computing on Anonymous Networks*, Yamashita and Kameda describe the classes of networks which can distributively solve the following problems: Choosing a unique processor as a leader, choosing a unique edge in the network, constructing a spanning tree of the graph of the network, and finding the graph of the network. The authors consider these problems for four levels of information which processors in a network might have about the network: A processor can have no information; can know an upper bound on the number of processors in the network, can know the exact number of processors in the network, or can know the graph of the network. The authors show, for instance, that processors in a network whose graph is a tree can distributively compute the graph of the network, if they are given the size of the network. In *Computing Functions on Anonymous Networks*, Yamashita and Kameda characterize the set of scalar valued functions a given anonymous network can compute. We will give a more thorough review of this paper at the end of the next chapter.

Other Related Topics: The edge-label monoid $\mathcal{E}(\mathsf{G})$ mentioned above is used extensively in algebraic automata theory, where it plays a part in the decomposing of finite-state

machines into simpler machines. We will make similar use of the monoid in this paper, when we examine quotient graphs of networks. A more complete description of the relation between our results and results in algebraic automata theory will be given at the ends of Chapters 2 and 3.

The proof of Theorem 5.5.1 in Chapter 5 makes use of a result from computational group theory, and perhaps itself belongs most correctly to that field. Computational group theory considers questions of the form: "How hard is Problem X from group theory?", where Problem X might be, for instance, finding the stabilizer subgroup of a set, or checking a permutation for membership in a permutation group given by a set of generators. Theorem 5.5.1 discusses "isomorphisms" of the block-systems of a permutation group, and shows that it is easy to determine whether two permutation groups have the same set of block-system isomorphisms.

2. Characterizing The Functions A Network Can Compute

2.1 Introduction

The aim of this chapter is to find a characterization of the set of functions computable by any given network. To this end, in Section 2.1 we will describe the computational model used in the paper, explicating what we mean by 'function computation' and 'anonymous network', and what properties we will assume of the graph of a network. In Section 2.2 we will examine a natural monoid associated with the graph of a network and will characterize the graph covering maps in terms of this monoid. In Section 2.3, we will introduce the notion of a covering graph, and show that two "rooted universal covers" are isomorphic iff they are isomorphic to a certain finite depth. Finally, in Section 2.4, we will characterize the set of functions computable by a network in terms of a set of rooted trees related to the universal cover.

Main Results: The main result of this chapter is a characterization of the functions computable by a network in terms of a set of trees associated with the network (Theorem 2.5.1). In subsequent chapters we will make heavy use of Proposition 2.3.1, which defines graph isomorphism in terms of a monoid associated with the graph.

Related Results: The "edge-label monoid", which we introduce in this chapter, is standard in algebraic automata theory; e.g., see [Hol82]. We also define what it means for one graph to "cover" another graph and define the "universal cover" of a graph. These definitions are from algebraic topology; see [Mas67]. We will review these and some results from the anonymous computing literature at the end of the chapter.

2.2 The Model

Intuitively, a network is nothing more than a collection of processors connected by two-way links. Processors in a network can "tell their links apart"; that is, a processor can specify the number of the link through which it will send any given message. Thus, each link between two processors has a pair of numbers associated with it: The first number is the first processor's name for the link, and the second number is the second processor's name for the link. This motivates the following definition:

Definition 2.2.1: A *network* $\mathcal{N}$ is a triple $\langle \mathsf{V}_{\mathcal{N}}, \mathsf{E}_{\mathcal{N}}, \rho_{\mathcal{N}} \rangle$, where:

- $\mathsf{V}_{\mathcal{N}}$, the *processor set*, is a finite set of arbitrarily powerful processors,
- $\rho_{\mathcal{N}}$, the *processor labeling map* is a bijection from $\mathsf{V}_{\mathcal{N}}$ to $\{1, \ldots, |\mathsf{V}_{\mathcal{N}}|\}$, and
- $\mathsf{E}_{\mathcal{N}}$, the set of *links*, is a subset of the set of all pairs (two element multisets) $\mathcal{L} = \{(v,i),(w,j)\}$, where $v, w \in \mathsf{V}_{\mathcal{N}}$ and $i, j \in \mathbf{N}$.

A link $\mathcal{L} = \{(v,i),(w,j)\}$ is called the *ith link of processor* v and the *jth link of processor* w. $\mathcal{L} = \{(v,i),(w,j)\}$ is said to be *incident with* v and w, and two processors are called *adjacent* if they are incident with the same link. Two processors may share multiple links,

but we will require that all all processors in a network satisfy the following "link-label condition":

Property 1: Link-Label Condition:

For each $v \in \mathcal{N}$ there is a nonnegative integer $deg(v)$, where $deg(v) = 0$ if v has no links. If v is incident with one or more links, then for each $1 \leq i \leq deg(v)$, processor v has exactly one ith link. Processor v has no ith links for $i > deg(v)$.

Note: $deg(v)$ is the number of links incident with v, where a loop $\{(v,i),(v,j)\}$ is counted as being incident with v twice, and a loop $\{(v,i),(v,i)\}$ is incident with v once.

EXAMPLE 2.2.1: For instance, in Figure 2.1, $\mathcal{N}_1$ satisfies the link-label condition, and $deg(v) = 4$. $\mathcal{N}_2$ does not satisfy the link-label condition because v has two "first" links $\mathcal{L}_1 = \{(v,1),(w,1)\}$ and $\mathcal{L}_2 = \{(v,1),(v,2)\}$, and two "second" links: $\mathcal{L}_2$, and $\mathcal{L}_3 = \{(v,2),(w,3)\}$.

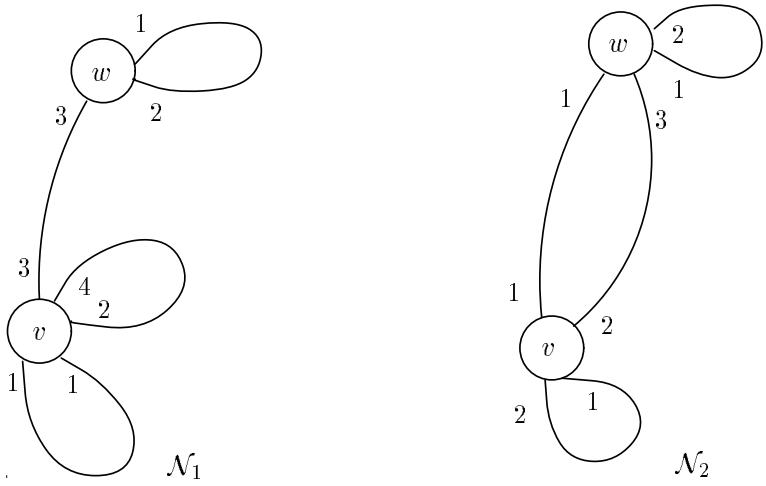


Figure 2.1: The Link-Label Condition

Definition 2.2.2: The number $i = \rho_{\mathcal{N}}(v)$ is called the *id* of processor v. If $\rho_{\mathcal{N}}(v) = i$ for $v \in \mathsf{V}_{\mathcal{N}}$ we will sometimes refer to processor v as "processor i".

2.2.1 Computing On A Network

Computations are performed distributively on a network, with each processor i running a deterministic algorithm A_i from a set $\mathsf{A} = \{A_1, \ldots, A_n\}$, for $n = |\mathsf{V}_{\mathcal{N}}|$. Initially each algorithm A_i is given $deg(i)$ and a letter from an input-alphabet $\mathbf{I}$ as input. Define a *message* to be any finite-length word over an arbitrary message alphabet. A processor begins to execute its algorithm when it either spontaneously "wakes up", or when it receives a message from another processor. Algorithms run in steps. During each step of an algorithm run by a processor v, the processor may:

• Send messages through one or more of its links. Processor v can specify the number $i \in \{1, \ldots, deg(v)\}$ of the link that a given message is to be sent through. During a step a processor can send at most one message out of any given link.

• Receive messages through one or more of its links. A message received through a the jth link of a processor is given the label j. This means that a processor can tell through which of its links a given message has come. During a step a processor receives at most one message through any given link.

• Perform a computation based on messages received, the results of computations in previous steps, and input. The result of a given computation may be specially marked and called the *output* of the algorithm's run on v. Each processor computes only one output, and then terminates.

Synchronous and Asynchronous Executions: In a distributed system there is no central control: No one processor is designated from the beginning as the leader or organizer of the network (although processors can in some cases *choose* a leader[1] and there is no global clock available to the processors for coordinating their message-sending. Processors may make computations at different speeds or be temporarily interrupted, and the time it takes messages to travel between any two processors can vary unpredictably, although we assume that all messages sent eventually do arrive in finite time. Since there is no global clock, two executions of A from the same start-state may well be different: A given step may take place at different times in the two executions and messages may be sent and arrive at different times. This will perhaps motivate the following definition:

Definition 2.2.3: Let $\mathcal{N}$ be a network with n processors, let i be a processor in $\mathcal{N}$, and let A_i be an algorithm. An *execution of A_i by processor i* from input x_i is the sequence of steps of A_i (where a *step* includes message-sending, message-receiving and computation, as outlined above) together with the time that each step begins, each message is sent or received, and each computation takes place, as measured by an external clock. Let $\mathsf{A} = \{A_1, \ldots, A_n\}$ be a collection of algorithms. An *asynchronous execution of* A *by* N from a given input $\vec{x} = (x_1, \ldots, x_n)$ is the set of executions of each $A_i \in \mathsf{A}$ by processor i ($i = 1, \ldots, n$) from input x_i. An execution must satisfy the requirement that messages arrive at a processor via a given link in the order in which they were sent, after an unspecified but finite delay. A special case of asynchronous execution is *synchronous execution*, in which all processors begin executing their algorithms at the same time, as measured by the external clock, and each step of an algorithm occurs at some discrete time $t \in \mathbf{N}$: Messages sent at time t arrive at time $t + 1$, and computations take zero time.[2]

Function Computation: We will now define what it means for a network to compute a function. We will distinguish between three kinds of functions: The "computable functions" (Definition 2.2.4), the functions which are "computable by a network" (Definition 2.2.5), and the functions which are " anonymously computable by a network" (Definition 2.2.6).

Definition 2.2.4: A function is said to be *computable* if it can be computed on a Turing machine.

---

[1] For instance, see [MA89] and [SS89]. The papers [YK87b, YK88] and [Ang80], along with others, also discuss leader election.

[2] There are a number of alternative ways to define the synchronous execution that we could have used here; e.g., a computation begun at time t could be completed at time $t + 1$.

Definition 2.2.5: Let $\mathbf{I}$ and $\mathbf{O}$ be input and output alphabets. We will assume that they are totally ordered sets.[3] Let $\vec{x} = (x_1, \ldots, x_n) \in \mathbf{I^n}$. If $\mathcal{N}$ is a network with n processors, we will say that $\mathcal{N}$ *is given input* $\vec{x}$ if processor i is assigned input x_i for $i = 1, \ldots, n$. The network $\mathcal{N}$ is said to *compute* $\vec{y} \in \mathbf{O^n}$ *given* $\vec{x} \in \mathbf{I^n}$ if there is a collection $\mathsf{A} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$ of algorithms such that for any execution of A by $\mathcal{N}$ given input $\vec{x}$, algorithm A_i produces output y_i, for $i = 1, \ldots, n$, in a finite number of steps. A computable function $f : \mathbf{I^n} \to \mathbf{O^n}$ is said to be *computable by* N if there is a collection A of algorithms such that for each $\vec{x} \in \mathbf{I^n}$ and for any execution of A given input $\vec{x}$, the network $\mathcal{N}$ computes $f(\vec{x})$.

EXAMPLE 2.2.2: A network with three processors will sucessfully compute the function $f(x_1, x_2, x_3) = (x_1 + x_2^2, x_3, x_3/2)$ if processor 1 computes $x_1 + x_2^2$, processor 2 computes x_3, and processor 3 computes $x_3/2$ whenever processor 1 is given x_1, processor 2 is given x_2 and processor 3 is given x_3 as input.

Remark 2.2.1: Suppose that there is a function f and a collection $\mathsf{A} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$ of algorithms such that A can compute f on a network *only* under the synchronous execution. If such an A exists, there is always a collection $\mathsf{A}' = \{\mathsf{A}'_1, \ldots, \mathsf{A}'_n\}$ of algorithms which computes f on the network under any execution. That is, the set of functions computable by a network under synchronous executions equals the set of functions computable by the network. To show this, let A be as above, and define $\mathsf{A}' = \{\mathsf{A}'_1, \ldots, \mathsf{A}'_n\}$ as follows: Initially, all processors send a token through each of their links. Each processor i then repeats the following: After receiving a token through each of its links, processor i simulates a step of A_i and then sends a token through each of its links. The collection A' can now simulate the synchronous execution of A.

Anonymous Computation: In this paper we will investigate computation on "anonymous networks", whose processors have no immediate way of distingushing among themselves.

Definition 2.2.6: A network $\mathcal{N}$ with n processors will be said to *perform an anonymous computation* if it executes a collection $\mathsf{A} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$ of algorithms for which $\mathsf{A}_1 = \mathsf{A}_2 = \cdots = \mathsf{A}_n$. That is, in an anonymous computation, all processors in the network run the same algorithm during any given execution. A network *compute a function* f *anonymously* if f is computed by $\mathcal{N}$ via a collection of algorithms $\mathsf{A} = \{\mathsf{A}_1, \ldots, \mathsf{A}_n\}$ for which $\mathsf{A}_1 = \mathsf{A}_2 = \ldots = \mathsf{A}_n$.

We will informally refer to a network performing anonymous computations as an "anonymous network".

One implication of this defintion is that the algorithm running on each processor in an anonymous computation does not have access to the processor's id, unless this is given as part of the input.[4] This means that the processors have no easy way of determining from where a given message originated in the network. One of the chief computational problems facing an anonymous network is that of constructing unique labels or ids for the processors using whatever information is available locally. We will see later that the

[3] We will actually need a stronger condition on $\mathbf{I}$; that its order be a computable function (Definition 2.2.4).

[4] If the input to each processor i in an anonymous computation *always* consists of an ordered pair(i, x_i) (that is, always includes the processor id) then the network can compute any reasonable function over this domain. This will follow from Theorem 2.5.1 in Section 2.5.

processors in an anonymous network can in fact construct labels for themselves, although these labels are not usually unique. Only in networks with special network topologies will processors be able to construct unique labels for themselves.

The synchronous execution of an algorithm is in some sense the "hardest case" for an anonymous network. In aynchronous executions, processors may be able to make use of the random arrival-time of messages to break symmetry and to distinguish among themselves. (For instance, see the papers [MA89] and [SS89] in which the authors showed that processors in an anonymous network can choose distinct id's for themselves if they have access to a coin-flip). In this paper we are interested in exploring the effect of a network's topology on its capacity for coordinated action, rather than in investigating symmetry-breaking through randomness. For this reason we require that algorithms for function-computation on a network work under all executions, including synchronous ones. By Remark 2.2.1, any algorithm can simulate a synchronous execution, and so we need only consider synchronous executions in this paper.

2.2.2 The Graph Of A Network

It will be convenient to be able to think of networks as *directed graphs* in the next section, when we define the monoid of a network and begin to explore the structure of networks as mathematical objects. Associating links in a network with directed edges in a graph is not intended to imply that links are one-way, however — the association is strictly for mathematical convenience.

Definition 2.2.7: A *directed, edge-labeled graph* G is a triple $\langle \mathsf{V}(\mathsf{G}), \mathsf{E}(\mathsf{G}), \mathsf{A}(\mathsf{G}) \rangle$, where:

$\mathsf{V}(\mathsf{G})$, the set of *vertices*, and

$\mathsf{A}(\mathsf{G})$, the set of *edge-labels*, are finite sets, and

$\mathsf{E}(\mathsf{G})$, the set of *edges* is a subset of $\mathsf{V}(\mathsf{G}) \times \mathsf{A}(\mathsf{G}) \times \mathsf{V}(\mathsf{G})$.

We will usually refer to directed edge-labeled graphs simply as graphs. If $\mathbf{e} = \langle v\, a\, w \rangle$ is an edge in G, we say that vertices v and w are *adjacent* and that $\mathbf{e}$ is an *edge with label* a, *directed away from* v and *directed towards* w. (We will think of a loop $\langle v a v \rangle$ as being directed towards v and away from v.) An edge $\mathbf{e} = \langle v\, a\, w \rangle$ is *incident with* both v and w.

Pictoral Conventions: We will use the following shorthand in picturing graphs: An edge $\langle v\, a\, w \rangle$ will be drawn as in Figure 2.2 below. We will usually draw the pair $\langle v\, a\, w \rangle$ and

Figure 2.2: The edge $\langle v\, a\, w \rangle$

$\langle w\, a\, v \rangle$ as a single edge with an arrow on each end (Figure 2.2.2) instead of as a pair of edges.

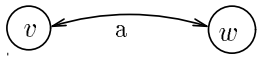


Figure 2.3: The pair $\{\langle v\,a\,w\rangle, \langle w\,a\,v\rangle\}$ of edges

We will use the convention that distinct letters $a, b, c, \ldots$ in a pictured graph represent distinct edge-labels in $\mathsf{A(G)}$.[5]

As mentioned earlier, networks are undirected, but it will be convenient to be able to think of networks as directed objects in the next section, when we define the monoid of a network. For this reason we will associate a network with a *directed* graph.

Definition 2.2.8: Let $\mathcal{N} = \langle \mathsf{V}_\mathcal{N}, \mathsf{E}_\mathcal{N}, \rho_\mathcal{N}\rangle$ be a network with n processors. The *graph of* $\mathcal{N}$ is a graph $\mathsf{G} = \langle \mathsf{V(G)}, \mathsf{E(G)}, \mathsf{A(G)}\rangle$, with:

- $\mathsf{V(G)} = \{1, \ldots, n\}$. The processor-labeling map $\rho_\mathcal{N}$ of $\mathcal{N}$ maps $\mathsf{V}_\mathcal{N}$ bijectively onto $\mathsf{V(G)}$;
- $\mathsf{A(G)}$ is a subset of NXN, and
- $\mathsf{E(G)}$ is the following subset of $\mathsf{V(G)\,X\,A(G)\,X\,V(G)}$: For each link $\mathcal{L} = \{(v,i),(w,j)\} \in \mathsf{E}_\mathcal{N}$ there is a set $\{\langle v\,(i,j)\,w\rangle, \langle w\,(j,i)\,v\rangle\}$ of edges in $\mathsf{E(G)}$. (If $\mathcal{L} = \{(v,i),(v,i)\}$, this set has only one member; $\langle v,(i,i),v\rangle$.)

EXAMPLE 2.2.3: The graph G in Figure 2.4 is the graph of a network having links $\mathcal{L}_1 = \{(1,1),(2,2)\}$, $\mathcal{L}_2 = \{(2,1),(3,1)\}$, $\mathcal{L}_3 = \{(2,3),(2,3)\}$, and $\mathcal{L}_4 = \{(3,2),(1,2)\}$.

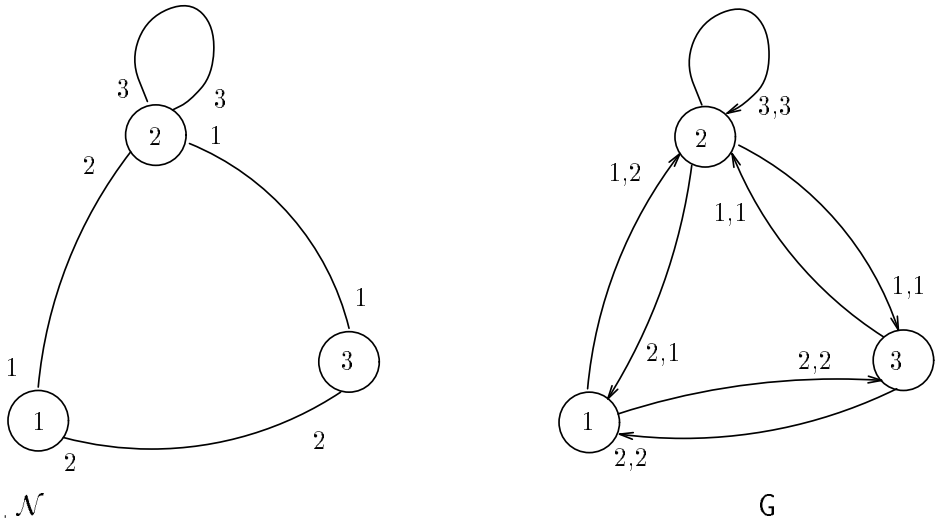


Figure 2.4: The graph of a network

[5]We will use the letters a, b, c, d, e as variables taking values in $\mathsf{A(G)}$ (and later, for words over $\mathsf{A(G)}$). For consistency, the letters labeling the edges of a graph in a figure should also be taken to be variables. However, since we use the convention that distinct letters take distinct values in $\mathsf{A(G)}$, the reader will not be misled by imagining that the letters are actual edge-labels, instead of variables.

A network is called *connected* if its graph is connected. We will assume that any network we consider is connected. We will usually identify a network with its graph when this doesn't cause confusion, and will commonly refer to a processor or vertex as "processor v", "processor i", "vertex v" or "vertex i", and identify links with pairs of edges.

A Condition on Edge-Labels: In this paper we will restrict our attention to graphs having the following edge-label property:

Property 2: *For any vertex v, no two edges directed towards v have the same edge-label, and no two edges directed away from v have the same edge-label.*

Note that this is a generalization of the "link-label condition" (Property 1). All graphs considered in this paper will satisfy Property 2, and so it should not cause confusion to refer to graphs with this property simply as graphs.

In subsequent chapters we will find that the structure of the graph of a network is partly dependent on the input to the network. That is, the graphs we are considering are not only edge-labeled, they are also vertex-labeled; and the vertex-labeling changes as the input to the network changes. This motivates the following notation:

Notation: If $\vec{x}$ is an input-vector for a network $\mathcal{N}$ we will write $(\mathsf{G}, \vec{x})$ for the graph G of $\mathcal{N}$ with vertex i (for $i \in \{1, \ldots, n\}$) labeled with the ith component of $\vec{x}$.

2.3 Monoids and Covering Maps

In this section we will define a monoid, the *edge-label monoid*, which has a natural association with an edge-labeled directed graph. We will make heavy use of this monoid in Chapters 3 and 4 for describing the structure of a graph in terms of its quotient-graphs.

The main results of this section are Propositions 2.3.1, which states that the graph covering maps are surjective maps which commute with the elements of the graph's monoid; and Corollary 2.3.1, which states that a covering map is determined by its action on a single vertex.

2.3.1 The Monoid Of A Graph

An edge-label a of a graph can be thought of as inducing a partial function f_a on the vertices of the graph, where $f_a(v) = w$ whenever there is an edge $\langle v\, a\, w \rangle$ in the graph. The collection of all such functions generates a monoid under function composition. Following is a more complete description of this monoid.

Let $\mathsf{A}(\mathsf{G})$ be the set of edge-labels for a graph G, and let $\mathsf{A}(\mathsf{G})^- = \{a^{-1} : a \in \mathsf{A}(\mathsf{G})\}$, where a^{-1} is a formal symbol. If $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, define $(a^{-1})^{-1}$ to be a. Write $\mathsf{A}(\mathsf{G})^*$ for the set of all words over $\mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, including the empty word, denoted by λ.

We will denote words in $\mathsf{A}(\mathsf{G})^*$, including words of length 1 in $\mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, by the symbols a, b, c, d, e. A word $a = a_1 a_2 \ldots a_k \in \mathsf{A}(\mathsf{G})^*$ will be said to be *reduced* if no two sucessive letters in a are of the form bb^{-1} for $b \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$.

Each letter $a \in \mathsf{A}(\mathsf{G})$ is associated with a partial function $f_a : \mathsf{V}(\mathsf{G}) \to \mathsf{V}(\mathsf{G})$, where $f_a(v) = w$ iff there is an edge $\langle v\, a\, w\rangle \in \mathsf{E}(\mathsf{G})$. If we append a vertex un (for "undefined") to $\mathsf{V}(\mathsf{G})$, we can define f_a to be a total function on $\mathsf{V}(\mathsf{G}) \cup \{un\}$, as follows: $f_a(v)$ equals w if there is an edge $\langle v\, a\, w\rangle \in \mathsf{E}(\mathsf{G})$, and equals un otherwise. We define the "partial inverse" $f_{a^{-1}} : a^{-1} \in \mathsf{A}(\mathsf{G})^-$ similarly: $f_{a^{-1}}(v) = w$ if there is an edge $\langle w\, a\, v\rangle \in \mathsf{E}(\mathsf{G})$, and equals un otherwise. Define $f_a(un) = un$ for all $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$. Let f_λ be the identity function on $\mathsf{V}(\mathsf{G}) \cup \{un\}$, where λ is the empty word. The set $\{f_a : a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^- \cup \{\lambda\}\}$ then generates a monoid, which we denote $\mathcal{E}(\mathsf{G})$, under function composition.

Definition 2.3.1: We will write $\mathcal{E}(\mathsf{G})$ for the *edge-label monoid of* G, and call its elements the *edge-label maps* of G.

We will call G a *group graph* in case $\mathcal{E}(\mathsf{G})$ is a group, and a *monoid graph* otherwise.

Note that Property 2 insures that the maps $f_a \in \mathcal{E}(\mathsf{G})$ are one-to-one.

Notation: Let $a = a_1 a_2 \ldots a_k$ be a nonempty word in $\mathsf{A}(\mathsf{G})^*$. To simplify notation,[6] we will write f_a for the function $f_{a_1} \circ f_{a_2} \circ \cdots \circ f_{a_k}$.

Remark: The relation $f_a f_{a^{-1}} = f_\lambda$ does not hold in general for $\mathcal{E}(\mathsf{G})$ since $f_{a^{-1}}$ may not be defined on all vertices of G. Hence if b is a word in $\mathsf{A}(\mathsf{G})^*$ and b' is its reduced form, it is not true in general that $f_b = f_{b'}$.

Definition 2.3.2:

1. Let $a \neq \lambda \in \mathsf{A}(\mathsf{G})^*$. We will say that $f_a \in \mathcal{E}(\mathsf{G})$ is *defined on* $v \in \mathsf{V}(\mathsf{G})$ if $f_a(v) \neq un$. We will take f_λ to be defined on all $v \in \mathsf{V}(\mathsf{G})$.
2. If $v_0, v_k \in \mathsf{V}(\mathsf{G})$, a *path of length* k *from* v_0 *to* v_k is a string $\mathcal{P} = v_0 a_1 v_1 a_2 \ldots v_{k-1} a_k v_k$, with $v_i \in \mathsf{V}(\mathsf{G})$ and $a_i \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, such that $\langle v_{i-1} a_i v_i\rangle \in \mathsf{E}(\mathsf{G})$ if $a_i \in \mathsf{A}(\mathsf{G})$, and $\langle v_i a_i^{-1} v_{i-1}\rangle \in \mathsf{E}(\mathsf{G})$ if $a_i \in \mathsf{A}(\mathsf{G})^-$, for $i = 1, \ldots, k$. A single vertex is a path of length 0. (See Example 2.3.1 below. Note that this definition is nonstandard: A path may contain edges directed towards or away from each other.)

If $\mathcal{P} = v_0 a_1 v_1 a_2 \ldots v_{k-1} a_k v_k$ is a path, the *word of* $\mathcal{P}$ is the string $a = a_k a_{k-1} \ldots a_1$. (See Remark 2.3.1 below.) A path $\mathcal{P} = v_0 a_1 v_1 \ldots v_k$ is called a *simple path* if if the word of $\mathcal{P}$ is reduced and nonempty, and a *closed path* if $v_0 = v_k$. A *tree* is a graph with no simple closed paths.

EXAMPLE 2.3.1: In Figure 2.5, the path $\mathcal{P} = 0\, a^{-1}\, 1\, b^{-1}\, 2\, a\, 3$ is a path of length 3 from vertex 0 to vertex 3. Its word is $ab^{-1}a^{-1}$, a reduced word.

Remark 2.3.1: A path $\mathcal{P} = v_0 a_1 v_1 a_2 \ldots v_k$ has word $a_k a_{k-1} \ldots a_1$ iff $f_{a_k a_{k-1} \ldots a_1}(v_0) = v_k$. For instance, in Example 2.3.1 above, $f_{ab^{-1}a^{-1}}$ is defined on v_0 and $f_{ab^{-1}a^{-1}}(v_0) = v_3$. In other words, if $\mathcal{P}$ is a path from v_0 to v_k having word a, then f_a is defined on v_0 and $f_a(v_0) = v_k$. Conversely, if $f_a(v_0) = v_k$ for a word $a \in \mathsf{A}(\mathsf{G})^*$, then there is a path from v_0 to v_k having a as its word. That is, there is a one-to-one correspondence between the set of paths from a vertex $v \in \mathsf{G}$ and the set of words $a \in \mathsf{A}(\mathsf{G})^*$ such that f_a is defined on v. The sequence of letters in the word of a path is reversed over its sequence in the path because we compose functions on the left.

---

[6] In actuality, we are putting a relation $\sim$ on the free semigroup over $\mathsf{A}(\mathsf{G})^*$, where $a \sim b$ for words a and $b \in \mathsf{A}(\mathsf{G})^*$ iff $f_a = f_b$. The monoid $\mathcal{E}(\mathsf{G})$ is isomorphic to the quotient of the free semigroup mod this relation. See [Hol82]

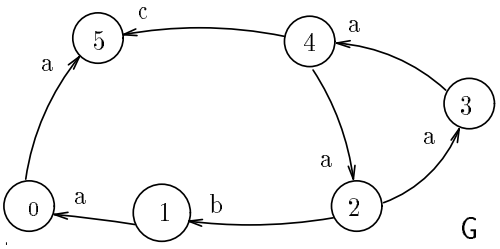


Figure 2.5: Paths and words

2.3.2 Covering Maps

If G_1 and G_2 are graphs, a surjective map from the vertices of G_1 to the vertices of G_2 will be called a covering map if it preserves edges, edge-labels and orientations. More formally, we have:

Definition 2.3.3: Let $\mathsf{G}_1 = \langle \mathsf{V}(\mathsf{G}_1), \mathsf{E}(\mathsf{G}_1), \mathsf{A}(\mathsf{G}_1) \rangle$ and $\mathsf{G}_2 = \langle \mathsf{V}(\mathsf{G}_2), \mathsf{E}(\mathsf{G}_2), \mathsf{A}(\mathsf{G}_2) \rangle$ be graphs. A *covering map* is a surjective map $\delta : \mathsf{V}(\mathsf{G}_1) \to \mathsf{V}(\mathsf{G}_2)$ satisfying:

1. If $\langle v\, a\, w \rangle \in \mathsf{E}(\mathsf{G}_1)$ then $\langle \delta(v) a \delta(w) \rangle \in \mathsf{E}(\mathsf{G}_2)$, and
2. If $\langle v\, a\, w \rangle \in \mathsf{E}(\mathsf{G}_2)$ then each vertex in $\delta^{-1}(v)$ has an edge with label 'a' directed away from it and each vertex in $\delta^{-1}(w)$ has an edge labeled 'a' directed towards it. That is, for each vertex $v' \in \delta^{-1}(v)$ there is an edge $\langle v' a r' \rangle \in \mathsf{E}(\mathsf{G}_1)$ and for each vertex $w' \in \delta^{-1}(w)$ there is an edge $\langle r' a w' \rangle \in \mathsf{E}(\mathsf{G}_1)$.

(See Example 2.3.2.) We will call a covering map δ an *isomorphism* if it is a bijection. An *automorphism* is an isomorphism from a graph to itself. (Note that by Proposition 2.3.1 below, isomorphisms have the expected property of commuting with the edge-label maps.)

We will say that a graph G *covers* a graph H if there is a covering map $\beta : \mathsf{G} \to \mathsf{H}$.

Below are some examples of covering maps.

EXAMPLE 2.3.2: In Figure 2.6, the map $\rho_1 : \mathsf{G}_1 \to \mathsf{G}_2$ which takes v to r and w to s is not a covering map because $\delta^{-1}(r)$ does not have an edge labeled 'b' directed away from it.

The map $\rho_2 : \mathsf{G}_3 \to \mathsf{G}_4$ which takes u', v' to v and w', r' to w, is a covering map.

Remark 2.3.2: Note that there can be a covering map from G_1 to G_2 only if $\mathsf{A}(\mathsf{G}_1) = \mathsf{A}(\mathsf{G}_2)$.

Lemma 2.3.1: *Let $\delta : \mathsf{G}_1 \to \mathsf{G}_2$ be a covering map. Then for all $a \in \mathsf{A}(\mathsf{G}_1)^*$ and for all $v \in \mathsf{V}(\mathsf{G}_1)$, $f_a \in \mathcal{E}(\mathsf{G}_1)$ is defined on v iff $g_a \in \mathcal{E}(\mathsf{G}_2)$ is defined on $\delta(v)$. In other words, there is a path with word a from $v \in \mathsf{V}(\mathsf{G}_1)$ iff there is a path with word a from $\delta(v) \in \mathsf{V}(\mathsf{G}_2)$.*

Proof Let $a = a_1 a_2 \ldots a_k \in \mathsf{A}(\mathsf{G}_1)^*$.

(1) Let f_a be defined on $v \in \mathsf{V}(\mathsf{G}_1)$. By Remark 2.3.1 there is a path $\mathcal{P} = v a_k v_{k-1} a_{k-1} \ldots a_1 v_0$ with word a from v to some vertex $v_0 \in \mathsf{V}(\mathsf{G}_1)$. Then $\delta(\mathcal{P}) = \delta(v) a_k \delta(v_{k-1}) \ldots a_1 \delta(v_0)$ is a path from $\delta(v)$ to $\delta(v_0)$ in G_2, because if $a_i \in a$ is in $\mathsf{A}(\mathsf{G}_1)$ then $\langle v_i a_i v_{i-1} \rangle \in \mathsf{E}(\mathsf{G}_1)$ and so $\langle \delta(v_i) a_i \delta(v_{i-1}) \rangle \in \mathsf{E}(\mathsf{G}_2)$, and if $a_i \in \mathsf{A}(\mathsf{G}_1)^-$ then

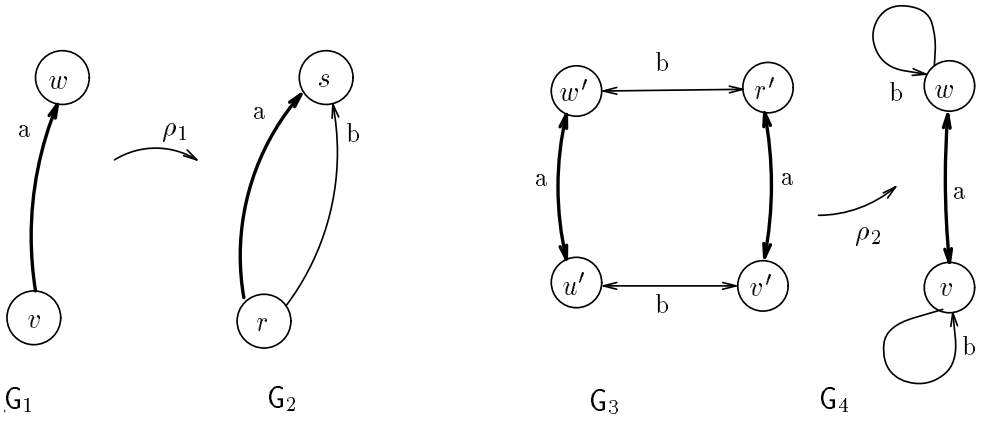


Figure 2.6: Graph Covering

$\langle v_{i-1}a_i^{-1}v_i\rangle \in \mathsf{E}(\mathsf{G}_1)$ and so $\langle \delta(v_{i-1})a_i^{-1}\delta(v_i)\rangle \in \mathsf{E}(\mathsf{G}_2)$. Hence a is the word of a path $\mathcal{P}'$ from $\delta(v)$ in G_2, and so g_a is defined on $\delta(v)$.

(2) Let $v' \in \mathsf{V}(\mathsf{G}_1)$ and suppose that g_a is defined on $\delta(v') \in \mathsf{V}(\mathsf{G}_2)$. Then there is a path $\mathcal{P} = \delta(v')a_k v_{k-1}a_{k-1}\ldots a_1 v_0$ from $\delta(v')$ to some $v_0 \in \mathsf{V}(\mathsf{G}_2)$. Since δ is a covering map, by condition (2) of the definition there is a vertex $v'_{k-1} \in \mathsf{V}(\mathsf{G}_1)$ such that $\delta(v'_{k-1}) = v_{k-1}$ and $\langle v'\, a_k\, v'_{k-1}\rangle \in \mathsf{E}(\mathsf{G}_1)$ if $\langle\ \delta(v')a_k v_{k-1}\rangle \in \mathsf{E}(\mathsf{G}_2)$, and $\langle v'_{k-1}\, a_k^{-1}\, v'\rangle \in \mathsf{E}(\mathsf{G}_1)$ if $\langle v_{k-1}\, a_k^{-1}\, \delta(v')\rangle \in \mathsf{E}(\mathsf{G}_2)$. That is, there is a path $\mathcal{P}_1 = v'a_k v'_{k-1}$ of length 1 in G_1 such that $\delta(\mathcal{P}_1) = \delta(v')a_k\delta(v'_{k-1}) = \delta(v')a_k v_{k-1}$ is a subpath of $\mathcal{P}$ from $\delta(v')$ to v_{k-1}. Arguing by induction over path length we see that there is a path $\mathcal{P}' = v'a_k v'_{k-1}a_{k-1}\ldots a_1 v'_0$ in G_1 such that $\delta(\mathcal{P}') = \delta(v')a_k\delta(v'_{k-1})a_{k-1}\ldots a_1\delta(v'_0) = \mathcal{P}$. Then f_a is defined on v'_0, since $\mathcal{P}'$ has word a. □

Let G_1 and G_2 be graphs. We will see next that the graph covering maps from G_1 to G_2 are the maps which commute with elements of $\mathcal{E}(\mathsf{G}_1)$ and $\mathcal{E}(\mathsf{G}_2)$.

Proposition 2.3.1: *Let* $\mathsf{G}_1 = \langle \mathsf{V}(\mathsf{G}_1), \mathsf{E}(\mathsf{G}_1), \mathsf{A}(\mathsf{G}_1)\rangle$ *and* $\mathsf{G}_2 = \langle \mathsf{V}(\mathsf{G}_2), \mathsf{E}(\mathsf{G}_2), \mathsf{A}(\mathsf{G}_2)\rangle$ *be graphs such that* $\mathsf{A}(\mathsf{G}_1) = \mathsf{A}(\mathsf{G}_2)$. *Then a surjective function* δ *from* $\mathsf{V}(\mathsf{G}_1)$ *onto* $\mathsf{V}(\mathsf{G}_2)$ *is a graph covering map iff* G_1, G_2 *and* δ *satisfy:*

1. *For any* $a \in \mathsf{A}(\mathsf{G}_1)^*$ *and for any* $v \in \mathsf{V}(\mathsf{G}_1)$, $f_a \in \mathcal{E}(\mathsf{G}_1)$ *is defined on* v *iff* $g_a \in \mathcal{E}(\mathsf{G}_2)$ *is defined on* $\delta(v)$.
2. *For all* $v \in \mathsf{V}(\mathsf{G}_1)$, *if* $f_a \in \mathcal{E}(\mathsf{G}_1)$ *then* $\delta f_a(v) = g_a\delta(v)$.

Proof Suppose first that $\delta : \mathsf{G}_1 \to \mathsf{G}_2$ is a covering map, and let $a \in \mathsf{A}(\mathsf{G}_1)^*$ and $v \in \mathsf{V}(\mathsf{G}_1)$. By Lemma 2.3.1, f_a is defined on v iff g_a is defined on $\delta(v)$. By the same lemma, $f_a(v) = v_0$ iff there is a path with word a from v to v_0 in G_1, and there is a path with word a from v to v_0 in G_1 iff there is a path with word a from $\delta(v)$ to $\delta(v_0)$ in G_2. That is, $\delta f_a(v) = \delta(v_0) = g_a\delta(v)$.

Conversely, suppose that $\mathsf{G}_1, \mathsf{G}_2$ and δ satisfy (1) and (2) above. Let $\langle v\, a\, w\rangle \in \mathsf{E}(\mathsf{G}_1)$. Then $f_a(v) = w$ and so by hypothesis, g_a is defined on $\delta(v)$ and $\delta f_a(v) = g_a\delta(v)$. That is, $g_a\delta(v) = \delta(w)$, so $\langle \delta(v)a\delta(w)\rangle \in \mathsf{E}(\mathsf{G}_2)$. Hence condition 1 of the definition of covering map is satisfied.

Let $\langle v\, a\, w\rangle \in \mathsf{E}(\mathsf{G}_2)$. Then $g_a(v) = w$ and $g_{a^{-1}}(w) = v$. Let $v' \in \delta^{-1}(v)$ and $w' \in \delta^{-1}(w)$. Then by hypothesis, f_a is defined on v' and $f_{a^{-1}}$ is defined on w'. That is, there is an edge directed away from v' having label a and an edge directed towards w' having label a, and so δ is a covering map . □

Corollary 2.3.1: *Let δ_1 and δ_2 be covering maps from G_1 to G_2. If $\delta_1(v) = \delta_2(v)$ for some $v \in \mathsf{V}(\mathsf{G}_1)$ then $\delta_1 = \delta_2$. That is, a covering map is determined by its action on a single vertex.*

Proof Let $\delta_1(v) = \delta_2(v)$ and let w be any vertex in G_1. We will show that $\delta_1(w) = \delta_2(w)$ also. Take $f_a \in \mathcal{E}(\mathsf{G}_1)$ such that $f_a(v) = w$. Note that f_a exists since G_1 is connected. Since there is a covering map from G_1 to G_2, there is also an element $g_a \in \mathcal{E}(\mathsf{G}_2)$, by Remark 2.3.2. Since δ_1 and δ_2 are covering maps, we have $\delta_1(w) = \delta_1 f_a(v) = g_a\delta_1(v)$ $= g_a\delta_2(v) = \delta_2 f_a(v) = \delta_2(w)$. □

2.4 Universal Covers

A "universal cover" of a graph G is a (usually infinite) tree covering G. In this section we will explore some properties of universal covers; in particular, of "rooted universal covers", in which one vertex of the tree is chosen to be its root. We will make extensive use of a set of rooted universal covers in characterizing the set of functions a network can compute.

The main result of this section is Proposition 2.4.1, which states that two rooted universal covers are isomorphic if they are isomorphic out to finite (small) depth. This result is proved in greater generality in [Nor93]. Here we give a shortened proof for the class of graphs considered in this paper.

Definition 2.4.1: The *universal cover* U of a graph G is a tree covering G. That is, there is a covering map from U to G. U is infinite unless G is a finite tree.

The following results are standard in algebraic topology, for instance, see [Mas67] and the papers [Ang80] and [Lei82].

We have:

Lemma 2.4.1:

- *The universal cover of a network is unique up to isomorphism.*
- *If β_1 and β_2 are covering maps from* U *to* G *and $\beta_1(\tilde{v}) = \beta_2(\tilde{w})$ for $\tilde{v}$ and $\tilde{w} \in \mathsf{V}(\mathrm{U})$ then there is an automorphism $\alpha : \mathrm{U} \to \mathrm{U}$ such that $\alpha(\tilde{v}) = \tilde{w}$ and $\beta_1 = \beta_2\alpha$.*

Proof Let U_1 and U_2 be universal covers for G. Let $\beta_1 : \mathrm{U}_1 \to \mathsf{G}_1$ and $\beta_2 : \mathrm{U}_2 \to \mathsf{G}_2$ be covering maps and let $\tilde{v} \in \mathsf{V}(\mathrm{U}_1)$ and $\tilde{w} \in \mathsf{V}(\mathrm{U}_2)$ be such that $\beta_1(\tilde{v}) = \beta_2(\tilde{w})$. We will find an isomorphism $\alpha : \mathrm{U}_1 \to \mathrm{U}_2$ such that $\beta_1 = \beta_2\alpha$, and conclude that U_1 and U_2 are isomorphic. If $\mathrm{U}_1 = \mathrm{U}_2$, the same construction yields $\alpha : \mathrm{U} \to \mathrm{U}$ such that $\beta_1 = \beta_2\alpha$.

Observe first that $\tilde{f}_a \in \mathcal{E}(\mathrm{U}_1)$ is defined on $\tilde{v}$ iff $\tilde{g}_a \in \mathcal{E}(\mathrm{U}_2)$ is defined on $\tilde{w}$: Since β_1 and β_2 are covering maps, $\tilde{f}_a$ is defined on $\tilde{v}$ iff $f_a \in \mathcal{E}(\mathsf{G})$ is defined on $v = \beta_1(\tilde{v})$, and f_a is defined on v iff $\tilde{g}_a$ is defined on any $\tilde{w} \in {\beta_2}^{-1}(v)$. We now construct a relation $\alpha : \mathsf{V}(\mathrm{U}_1) \to \mathsf{V}(\mathrm{U}_2)$ as follows: $\alpha(\tilde{v}) = \tilde{w}$. For any $\tilde{f}_a \in \mathcal{E}(\mathrm{U}_1)$ which is defined on $\tilde{v}$

and for which a is a reduced word, define $\alpha\tilde{f}_a(\tilde{v})$ to be $\tilde{g}_a\alpha(\tilde{v}) = \tilde{g}_a(\tilde{w})$. If $\tilde{r} \in \mathsf{V}(\mathrm{U}_1)$, let $\tilde{f}_a$ be the unique element of $\mathcal{E}(\mathrm{U}_1)$ such that $\tilde{f}_a(\tilde{v}) = \tilde{r}$ and a is reduced, and define $\alpha(\tilde{r}) \equiv \alpha\tilde{f}_a(\tilde{v}) = \tilde{g}_a(\tilde{w})$. Then α is an isomorphism:

(1) *α is a one-to-one function:* Let $\alpha(\tilde{u}) = \alpha(\tilde{r})$ for vertices $\tilde{u}$ and $\tilde{r} \in \mathsf{V}(\mathrm{U}_1)$. Let $\tilde{f}_a$ and $\tilde{f}_b \in \mathcal{E}(\mathrm{U}_1)$ map $\tilde{v}$ to $\tilde{u}$ and $\tilde{v}$ to $\tilde{r}$, respectively, for a and b reduced words. Then $\alpha(\tilde{u}) = \tilde{g}_a(\tilde{w})$ and $\alpha(\tilde{r}) = \tilde{g}_b(\tilde{w})$, so $\tilde{g}_a(\tilde{w}) = \tilde{g}_b(\tilde{w})$, and so $\tilde{g}_a = \tilde{g}_b$ since U_2 has no closed paths. Then $f_a = f_b$ in $\mathcal{E}(\mathsf{G})$ since U_2 covers G, and so $\tilde{f}_a = \tilde{f}_b$ since U_1 covers G. Thus $\tilde{u} = \tilde{r}$ and α is one-to-one. A similar argument shows that α is a function.

(2) *α is onto:* If $\tilde{u} \in \mathsf{V}(\mathrm{U}_2)$, let $\tilde{g}_a \in \mathcal{E}(\mathrm{U}_2)$ map $\tilde{w}$ to $\tilde{u}$. Then $\tilde{f}_a$ maps v to some $\tilde{u}'$ and $\alpha(\tilde{u}') = \tilde{u}$.

(3) *α is a covering map:* Let $\tilde{r} \in \mathsf{V}(\mathrm{U}_1)$ and let $\tilde{f}_c \in \mathcal{E}(\mathrm{U}_1)$ be defined on $\tilde{r}$, and $\tilde{f}_a \in \mathcal{E}(\mathrm{U}_1)$ map $\tilde{v}$ to $\tilde{r}$, where c and a are reduced words. Then $\tilde{f}_c(\tilde{r}) = \tilde{f}_c\tilde{f}_a(\tilde{v})$ and $\tilde{g}_c\alpha(\tilde{r}) = \tilde{g}_c\tilde{g}_a(\tilde{w})$. Then $\tilde{g}_c$ is defined on $\alpha(\tilde{r})$ if $\tilde{f}_c$ is defined on $\tilde{r}$, because $\tilde{f}_c\tilde{f}_a$ being defined on $\tilde{v}$ implies that f_af_b is defined on v in G, which in turn implies that $\tilde{g}_c\tilde{g}_a$ is defined on $\tilde{w} = \alpha(\tilde{v})$, since $\beta_2(\tilde{w}) = \beta_1(\tilde{v})$. The same argument shows that if $\tilde{g}_c$ is defined on $\alpha(\tilde{r})$ then $\tilde{f}_c$ is defined on $\tilde{r}$. Furthermore, $\alpha\tilde{f}_c(\tilde{r}) = \alpha\tilde{f}_c\tilde{f}_a(\tilde{v}) = \tilde{g}_c\tilde{g}_a\alpha(\tilde{v}) = \tilde{g}_c\tilde{g}_a(\tilde{w}) = \tilde{g}_c\alpha(\tilde{r})$. Thus α is an isomorphism by Proposition 2.3.1.

Since $\beta_1(\tilde{v}) = \beta_2\alpha(\tilde{v})$, we can use Corollary 2.3.1 to conclude that $\beta_1 = \beta_2\alpha$. The same argument can be used to prove the second part of the propostion. □

Next we define the rooted universal cover of a graph:

Definition 2.4.2: Let $v \in \mathsf{V}(\mathsf{G})$ and let U be the universal cover for G. We define the *rooted universal cover* U_v to be a triple $\mathrm{U}_v = (\mathrm{U}, \tilde{v}, \beta)$; where $\tilde{v} \in \mathrm{U}$ and $\beta : \mathrm{U} \to \mathsf{G}$ is a covering map such that $\beta(\tilde{v}) = v$.

U_v can be thought of as the tree[7] U rooted at $\tilde{v}$.

By Lemma 2.4.1, if $\beta_1(\tilde{v}_1) = \beta_2(\tilde{v}_2) = v$ then there is an automorphism α of U mapping $\tilde{v}_1$ to $\tilde{v}_2$ such that $\beta_1 = \beta_2\alpha$. In this sense U_v is unique up to isomorphism. The map β is called the *canonical covering map* for U_v; it is the unique covering map taking $\tilde{v}$ to v.

If v and w are vertices in G, we will say that U_v and U_w are *isomorphic via an isomorphism* α, written $\mathrm{U}_v \simeq \mathrm{U}_w$, if there is an automorphism $\alpha : \mathrm{U} \to \mathrm{U}$ which maps the root $\tilde{v}$ of U_v to the root $\tilde{w}$ of U_w. Note that by Corollary 2.3.1 there is at most one automorphism of U which maps $\tilde{v}$ to $\tilde{w}$.

We will need the following notation:

Notation: Let G be a network and let $\mathsf{V}(\mathsf{G}) = \{1, \ldots, n\}$. If $\vec{x} = (x_1, \ldots, x_n)$ is an input for G, write $\mathrm{U}_v\vec{x}$ for the graph U_v with vertices labeled with the components of $\vec{x}$ as follows: Each vertex $\tilde{u}$ in U_v is labeled $x_{\beta(\tilde{u})}$, where β is the canonical covering map for

[7] For definiteness we could make the following formal construction of U_v: The vertex set for U_v is the set of all simple paths (that is, having reduced words) from v in G. The edges are triples $\langle \mathcal{P}_1 a \mathcal{P}_2 \rangle$ where if $\mathcal{P}_i = va_1v_2 \ldots a_kv_k$ then $\mathcal{P}_j = va_1v_2 \ldots a_kv_kav_{k+1}$ for $i, j \in \{1, 2\}$. The root of U_v is the path $\mathcal{P} = v$ and the canonical covering map β takes each vertex $\mathcal{P} = va_1v_2 \ldots a_kv_k \in \mathsf{V}(\mathrm{U}_v)$ to $v_k \in \mathsf{V}(\mathsf{G})$. This consruction is given in [Ang80] and [Lei82].

U_v. That is, the vertices in the set $\{\beta^{-1}(i) \in \mathsf{V}(\mathrm{U})\}$ are labeled with the ith component of $\vec{x}$.

Definition 2.4.3: Let $\beta_1 : \mathrm{U}_v \to \mathsf{G}$ and $\beta_2 : \mathrm{U}_w \to \mathsf{G}$ be the canonical covering maps and let $\vec{x}$ and $\vec{y} \in \mathbf{I^n}$. We will say that $\mathrm{U}_v\vec{x}$ and $\mathrm{U}_w\vec{y}$ are *isomorphic* and write $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$ if $\mathrm{U}_v \simeq \mathrm{U}_w$ via an isomorphism α and if for every vertex $\tilde{r} \in \mathrm{U}_v$, the vertex-label of $\tilde{r}$ equals the vertex-label of $\alpha(\tilde{r})$. That is, $x_{\beta_1(\tilde{r})} = y_{\beta_2\alpha(\tilde{r})}$ for all $\tilde{r} \in \mathsf{V}(\mathrm{U}_v)$. (See example 2.4.1.)

The next lemma states that if $\mathrm{U}_v\vec{x}$ and $\mathrm{U}_w\vec{y}$ are isomorphic, then so are are the trees obtained by 'translating' $\mathrm{U}_v\vec{x}$ and $\mathrm{U}_w\vec{y}$ by a path with word w. (See Example 2.4.1 below.) This result is standard; see Proposition 3.1 in [YK88].

Lemma 2.4.2: *Let* G *be a graph with universal cover* U. *If* $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$ *for some* $\vec{x}$ *and* $\vec{y} \in \mathbf{I^n}$, *then* $\mathrm{U}_{f_a(v)}\vec{x} \simeq \mathrm{U}_{f_a(w)}\vec{y}$ *for any* $f_a \in \mathcal{E}(\mathsf{G})$ *which is defined on* v *and* w. *In particular, this means that if* $\mathrm{U}_v \simeq \mathrm{U}_w$ *then* $\mathrm{U}_{f_a(v)} \simeq \mathrm{U}_{f_a(w)}$.

Proof Let $\beta_1 : \mathrm{U}_v \to \mathsf{G}$ and $\beta_2 : \mathrm{U}_w \to \mathsf{G}$ be the canonical covering maps and let $f_a \in \mathcal{E}(\mathsf{G})$ be defined on v and w. Then $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ is defined on the roots $\tilde{v}$ and $\tilde{w}$ of U_v and U_w since β_1 and β_2 are covering maps . Let $\mathrm{U}_{f_a(v)}$ have root $\tilde{r}$ and canonical covering map $\beta_1{}'$. Then $\beta_1\tilde{f}_a(\tilde{v}) = f_a\beta_1(\tilde{v}) = f_a(v) = \beta_1{}'(\tilde{r})$, and so by Lemma 2.4.1, there is an automorphism $\alpha_1 : \mathrm{U} \to \mathrm{U}$ such that $\alpha_1(\tilde{r}) = \tilde{f}_a(\tilde{v})$, and $\beta_1{}' = \beta_1\alpha_1$. If $\mathrm{U}_{f_a(w)}$ has root $\tilde{u}$ and canonical covering map $\beta_2{}'$, then the same argument shows that there is an automorphism $\alpha_2 : \mathrm{U} \to \mathrm{U}$ such that $\alpha_2(\tilde{u}) = \tilde{f}_a(\tilde{w})$ and such that $\beta_2{}' = \beta_2\alpha_2$. Let α_3 be the unique automorphism of U which maps $\tilde{v}$ to $\tilde{w}$. Then we have $\alpha_3(\tilde{f}_a(\tilde{v})) = \tilde{f}_a\alpha_3(\tilde{v}) = \tilde{f}_a(\tilde{w})$. Let $\alpha = \alpha_2^{-1}\alpha_3\alpha_1 : \mathrm{U} \to \mathrm{U}$. Then α is an automorphism mapping $\tilde{r}$ to $\tilde{u}$, and so $\mathrm{U}_{\tilde{f}_a(\tilde{v})} = \mathrm{U}_{\tilde{r}} \simeq \mathrm{U}_{\tilde{u}} = \mathrm{U}_{\tilde{f}_a(\tilde{w})}$ via α. Since $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$ via α_3, we have $x_{\beta_1(\tilde{i})} = y_{\beta_2\alpha_3(\tilde{i})}$ for any $\tilde{i} \in \mathsf{V}(\mathrm{U})$. Hence $x_{\beta_1{}'(\tilde{i})} = x_{\beta_1(\alpha_1(\tilde{i}))} = y_{\beta_2\alpha_3(\alpha_1(\tilde{i}))}$ $= y_{\beta_2\alpha_2\alpha(\tilde{i})} = y_{\beta_2{}'\alpha(\tilde{i})}$ for any $\tilde{i}$ in $\mathsf{V}(\mathrm{U})$, and so, by definition, $\mathrm{U}_{f_a(v)}\vec{x} \simeq \mathrm{U}_{f_a(w)}\vec{y}$. □

Definition 2.4.4: (1) The *distance* between two vertices v and w in a graph G is defined to be the length of the shortest path between v and w.

(2) We will write U_v^k for the subgraph of U_v induced by the set of vertices of U_v of distance at most k from its root $\tilde{v}$.

EXAMPLE 2.4.1: Figure 2.7 pictures $(\mathsf{G}, \vec{x})$ and $\mathrm{U}_v^2\vec{x}$ for $\vec{x} = (x, y, x, y)$. It can be checked that $\mathrm{U}_1\vec{x} \simeq \mathrm{U}_3\vec{x}$ and $\mathrm{U}_2\vec{x} \simeq \mathrm{U}_4\vec{x}$. Since $\mathrm{U}_2\vec{x}$ and $\mathrm{U}_4\vec{x}$ are isomorphic, Lemma 2.4.2 says, for instance, that $\mathrm{U}_{f_a(2)}\vec{x} = \mathrm{U}_3\vec{x}$ is isomorphic to $\mathrm{U}_{f_a(4)}\vec{x} = \mathrm{U}_1\vec{x}$.

The following is a technical lemma:

Lemma 2.4.3: *Let* $v, w \in \mathsf{V}(\mathsf{G})$ *and let* $k > 0$. *Then* $\mathrm{U}_v^k\vec{x} \simeq \mathrm{U}_w^k\vec{y}$ *iff* $\mathrm{U}_v^1\vec{x} \simeq \mathrm{U}_w^1\vec{y}$ *and for all* $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$ *such that* f_a *is defined on* v, *we have* $\mathrm{U}_{f_a(v)}^{k-1}\vec{x} \simeq \mathrm{U}_{f_a(w)}^{k-1}\vec{y}$.

Proof One direction is immediate: If $\delta : \mathrm{U}_v^k\vec{x} \to \mathrm{U}_w^k\vec{y}$ is an isomorphism, then δ restricted to $\mathrm{U}_v^1\vec{x}$ is an isomorphism between $\mathrm{U}_v^1\vec{x}$ and $\mathrm{U}_w^1\vec{y}$, and δ restricted to $\mathrm{U}_{f_a(v)}^{k-1}\vec{x}$ is an isomorphism between $\mathrm{U}_{f_a(v)}^{k-1}\vec{x}$ and $\mathrm{U}_{f_a(w)}^{k-1}\vec{y}$.

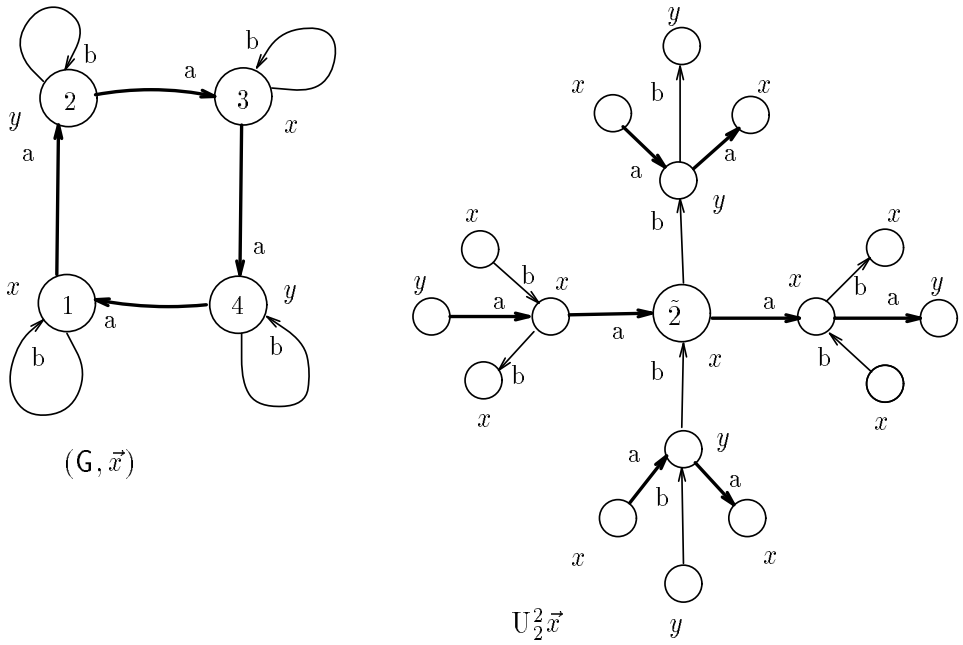


Figure 2.7: $(\mathsf{G}, \vec{x})$ and $\mathrm{U}_2^2\vec{x}$

Conversely, if there is an isomorphism $\delta_1 : \mathrm{U}_v^1\vec{x} \to \mathrm{U}_w^1\vec{y}$, then for any $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, the element f_a is defined on v iff it is defined on w. Suppose that for each $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$ such that f_a is defined on v there is an isomorphism δ_a from $\mathrm{U}_{f_a(v)}^{k-1}\vec{x}$ to $\mathrm{U}_{f_a(w)}^{k-1}\vec{y}$. Define a map δ from $\mathrm{U}_v^k\vec{x}$ to $\mathrm{U}_w^k\vec{y}$ as follows: $\delta = \delta_1$ on $\mathrm{U}_v^1\vec{x}$. For each $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, define $\delta = \delta_a$ on each subtree of $\mathrm{U}_v^k\vec{x}$ rooted at $f_a(v)$ (that is, on each subgraph of $\mathrm{U}_v^k\vec{x}$ induced by $f_a(v)$ and its descendants.) It is easy to verify that δ is an isomorphism. □

Finally we can show:

Proposition 2.4.1: *If* G *has* n *vertices and* $v, w \in \mathsf{V}(\mathsf{G})$, *then* $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$ *iff* $\mathrm{U}_v^{2n-1}\vec{x} \simeq \mathrm{U}_w^{2n-1}\vec{y}$

Proof If $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$ then $\mathrm{U}_v^{2n-1}\vec{x} \simeq \mathrm{U}_w^{2n-1}\vec{y}$ by restriction of the isomorphism. Conversely, suppose that $\mathrm{U}_v^{2n-1}\vec{x} \simeq \mathrm{U}_w^{2n-1}\vec{y}$. We will show that $\mathrm{U}_v^k\vec{x} \simeq \mathrm{U}_w^k\vec{y}$ for all $k \geq 0$, and the conclusion will follow from this.

For each $k \geq 0$, define an equivalence relation $\sim_k$ on the set $\mathsf{V}(\mathsf{G}) \times \{\vec{x}, \vec{y}\}$ by: $\langle v, \vec{z}^1\rangle \sim_k \langle w, \vec{z}^2\rangle$ for $\vec{z}^1$ and $\vec{z}^2 \in \{\vec{x}, \vec{y}\}$ iff $\mathrm{U}_v^k\vec{z}^1 \simeq \mathrm{U}_w^k\vec{z}^2$. Write π_k for the partition of $\mathsf{V}(\mathsf{G})$ corresponding to $\sim_k$. Then π_k satisfies the following two properties:

(1) π_{k+1} *is a refinement of* π_k. That is, if the pairs $\langle v, \vec{z}^1\rangle$ and $\langle w, \vec{z}^2\rangle$ are in distinct blocks of π_k then they are in distinct blocks of π_{k+1}.

(2) If $\pi_{k-1} = \pi_k$ *for some* $k > 0$ *then* $\pi_k = \pi_{k+1}$, *and hence* $\pi_k = \pi_{k+j}$ *for all* $j \geq 0$.

(1) is immediate. (2) follows from Lemma 2.4.3 above; for suppose that for all pairs $\langle v, \vec{z}^1\rangle$ and $\langle w, \vec{z}^2\rangle$ in $\mathsf{V}(\mathsf{G}) \times \{\vec{x}, \vec{y}\}$, we have $\langle v, \vec{z}^1\rangle \sim_{k-1} \langle w, \vec{z}^2\rangle$ implies that

$\langle v, \vec{z}^1 \rangle \sim_k \langle w, \vec{z}^2 \rangle$; that is, $U_v^{k-1}\vec{z}^1 \simeq U_w^{k-1}\vec{z}^2$ implies that $U_v^k\vec{z}^1 \simeq U_w^k\vec{z}^2$. Pick pairs $\langle v, \vec{z}^1 \rangle$ and $\langle w, \vec{z}^2 \rangle$ such that $v \sim_{k-1} w$ and let $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$ be such that f_a is defined on v. Then $U_v^k\vec{z}^1 \simeq U_w^k\vec{z}^2$, and so by Lemma 2.4.3 above, we have $U_{f_a(v)}^{k-1}\vec{z}^1 \simeq U_{f_a(w)}^{k-1}\vec{z}^2$. By hypothesis, $U_{f_a(v)}^k\vec{z}^1 \simeq U_{f_a(w)}^k\vec{z}^2$ and so, again by Lemma 2.4.3, $U_v^{k+1}\vec{z}^1 \simeq U_w^{k+1}\vec{z}^2$. This proves (2).

To prove the proposition, observe first that by (1) and (2) above, π_k becomes strictly finer as k increases, up to a point. π_0 has at least one block. If π_1 has only one block then π_k has only one block for all $k \geq 1$. Suppose that π_1 has at least two blocks. Then if $\pi_2 \neq \pi_1$ then π_2 has at least three blocks, and if $\pi_3 \neq \pi_2$ then π_3 has at least four blocks, and so on. Since π_k can have at most $2n$ blocks, we must have $\pi_{2n-1} = \pi_2 n = \pi_k$ for all $k \geq 2n$. That is, $U_v^{2n-1}\vec{x} \simeq U_w^{2n-1}\vec{y}$ implies that $U_v^k\vec{x} \simeq U_w^k\vec{y}$ for all $k \geq 0$, and so $U_v\vec{x} \simeq U_w\vec{y}$. □

A similar argument gives us the following:

Proposition 2.4.2: *Let a graph* G *have* n *vertices. If* $U_v^{n-1}\vec{x} \simeq U_w^{n-1}\vec{x}$ *for some* $\vec{x} \in \mathbf{I^n}$ *and for* $v, w \in \mathsf{V}(\mathsf{G})$ *then* $U_v^k\vec{x} \simeq U_w^k\vec{x}$ *for all* $k \geq 0$.

Proof Define an equivalence relation $\sim_k$ on $\mathsf{V}(\mathsf{G})$ by: $v \sim_k w$ iff $U_v^k\vec{x} \simeq U_w^k\vec{x}$. The argument used in the proof of Proposition 2.4.1 then yields the desired conclusion. □

2.5 Computing On Anonymous Networks

As mentioned earlier, the processors in a network performing an anonymous computation do not have acess to their ids unless these are given as input. Suppose for a moment that each processor in such a network could *compute* a unique label for itself, and that this label was independent of the input to the network. Theorem 2.5.1 in this section will show that such a network can compute any computable function $f : \mathbf{I^n} \to \mathbf{O^n}$. In this section we will see that the processors in a network performing an anonymous computation *can* compute labels for themselves, where the 'label' each processor i computes for itself is its tree $U_i^{2n-1}\vec{x}$. Depending on the network's topology, this label will not be unique to a processor, and of course will vary with the input to the network. For any input-vector $\vec{x}$ we can put an equivalence-relation $\sim_{\vec{x}}$ on the processors of a network G, where $i \sim_{\vec{x}} j$ iff processors i and j compute the same label for themselves under input $\vec{x}$. Theorem 2.5.1 then says in part that G can compute a function only if it is invariant under this relation, in a sense that will be defined in this section.

The main result of this section is a characterization of the set of functions computable by a network (Theorem 2.5.1). The next three lemmas will be used in this characterization. These lemmas explore the computational capacity of anonymous networks.

Lemma 2.5.1: *Consider two synchronous executions of an algorithm* $\mathcal{A}$ *in anonymous computations on a network* G*; the first with input* $\vec{x}$ *and the second with input* $\vec{y}$*. If* $U_v\vec{x} \simeq U_w\vec{y}$ *for processors* v *and* w *in* G*, then at each step* k *of the first execution, processor* v *sends and receives the same messages and computes the same values as processor* w *does during the* k*th step of the second execution of* $\mathcal{A}$.

Proof (by induction on the steps k of $\mathcal{A}$):

Suppose first that $\mathrm{U}_v^1\vec{x} \simeq \mathrm{U}_w^1\vec{y}$. Then at step 1 of $\mathcal{A}$'s run on v under input $\vec{x}$ processor v must receive the same messages from adjacent processors as processor w receives during step 1 of $\mathcal{A}$'s run on w, under input $\vec{y}$. Since processors v and w run the same algorithm $\mathcal{A}$, if they receive the same messages during step 1 then each must compute the same values and send the same messages as the other. Thus the first step of $\mathcal{A}$ on v with input $\vec{x}$ is the same as the first step of $\mathcal{A}$ on w under input $\vec{y}$.

Suppose now that the result holds for the kth step of $\mathcal{A}$. By Lemma 2.4.2, there is a bijection α between the set of vertices adjacent to v and those adjacent to w in G such that for each vertex u adjacent to v we have $\mathrm{U}_u\vec{x} \simeq \mathrm{U}_{\alpha(u)}\vec{y}$. By hypothesis, by the kth step of $\mathcal{A}$, processors u and $\alpha(u)$ (adjacent to v and w) have sent the same messages to v and w, respectively, in the two executions. Thus in the $k+1$st step of the two executions processors v and w receive identical message-sets from adjacent processors, and so v and w compute the same values and send the same messages to neighboring processors. □

Remark 2.5.1: By Proposition 2.4.1, if $\mathrm{U}_v^{2n-1}\vec{x} \simeq \mathrm{U}_w^{2n-1}\vec{y}$ then $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{y}$, and the above lemma holds.

The following lemma was proved by Yamashita and Kameda ([YK87b,YK88]). Their proof applies to our model with few modifications.

Lemma 2.5.2: *Let G be a network and let G have input $\vec{x}$. A processor v in G can anonymously compute a graph isomorphic to $\mathrm{U}_v^k\vec{x}$ for any $k \geq 0$.*

Proof See Lemma 3.4 in [YK88]. Here we sketch an algorithm for reference.

Algorithm 2.5.1: *Step 1:* Each processor v sends its input-value to adjacent processors. A message (an input value) sent through the ith link of a processor has the value "i" appended to it on the right, and a message received through a processor's jth link has the value "j" appended to it on the right. Each processor v waits until it has received a message through each link, and then uses these messages to construct $\mathrm{U}_v^1\vec{x}$.

Steps 2,...,k: Each processor u sends a pair $(\mathrm{U}_u^{l-1}\vec{x}, i)$ through its ith link, for $i = 1, \ldots, deg(u)$. If a processor receives such a message through its jth link, it appends a j to the message. Each processor v waits until it has received a message through each of its links, and then uses these messages (with the appended link labels), together with its input-value, to construct $\mathrm{U}_v^l\vec{x}$. □

There are several possible procedures for constructing $\mathrm{U}_v^l\vec{x}$. One possibility is to append leaves to $\mathrm{U}_v^{l-1}\vec{x}$. Another possiblility, which we describe here, is to choose the 'correct' subgraphs of each rooted universal cover of depth $l-1$ belonging to adjacent processors, and to connect all of these subgraphs at a vertex to form $\mathrm{U}_v^l\vec{x}$. (See Example 2.5.1 below.) This can be done as follows:

At step l, processor v receives $\mathrm{U}_u^{l-1}\vec{x}$ from each adjacent processor u. From each of these rooted universal covers processor v constructs a pair $\{T_1, T_2\}$ of subtrees, as described below. All such subtrees are then merged at the root to form $\mathrm{U}_v^l\vec{x}$.

The two subtrees T_1 and T_2 of a neigboring universal cover $\mathrm{U}_u^{l-1}\vec{x}$ are given as follows. Suppose that $\mathrm{U}_u^{l-1}\vec{x}$ is sent to processor v via a link $\{(u,i),(v,j)\}$. Let $\mathbf{e}_1 = \langle \tilde{u}, (i,j)u_1 \rangle$ and $\mathbf{e}_2 = \langle u_2, (j,i), \tilde{u} \rangle$ be edges adjacent to the root $\tilde{u}$ of $\mathrm{U}_u^{l-1}\vec{x}$. Let T_1' be the subgraph

of $\mathrm{U}_u^{l-1}\vec{x}$ obtained by removing all edges adjacent to the root except for $\mathbf{e}_1$. Then T_1 is the connected component of T_1' containing u_1. T_2 is constructed similarly.

□

EXAMPLE 2.5.1: We will describe part of the construction of U_v^3, omitting the input $\vec{x}$ for simplicity. At step 2, processor v receives U_w^2 and U_u^2. The trees T_1 and T_2 constructed from U_u^2 are pictured. (To save space we have drawn the trees and universal covers schematically, with triangles representing subtrees.) The roots of T_1 and T_2 are identified, and the resulting graph forms part of U_v^3.

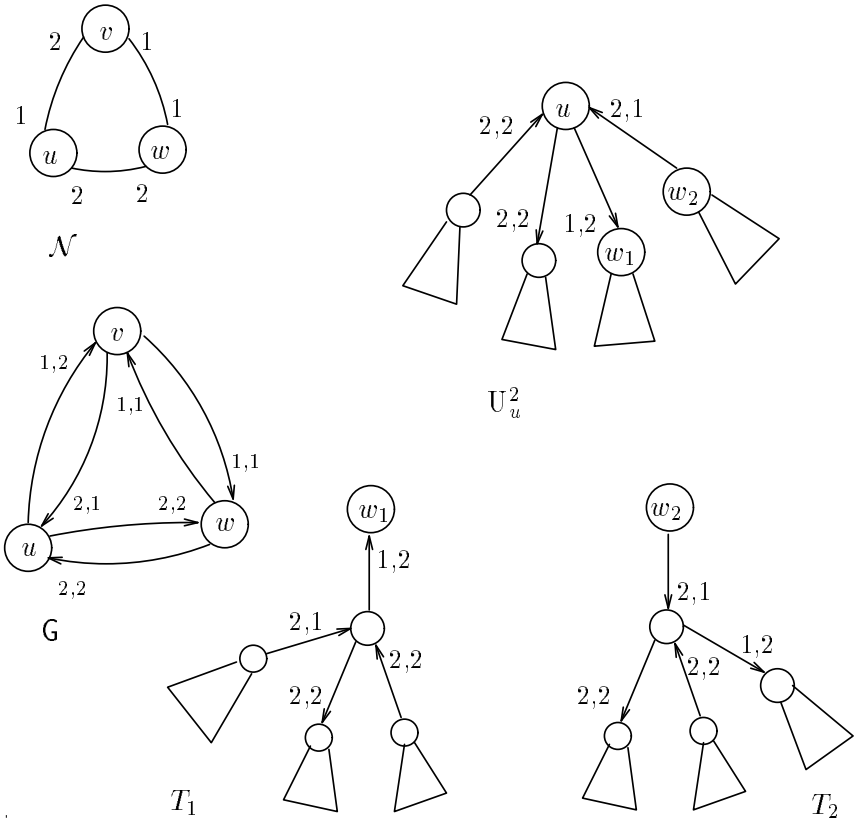


Figure 2.8: Computing U_v^3

The next theorem characterizes the set of functions a given network can compute anonymously. In [YK87a], Yamashita and Kameda obtained similar results for scalar valued functions: See Theorem 4.1 in [YK87a], and also the related results section at the end of this chapter.

Theorem 2.5.1: *Let* G *be a network. Any computable function* $f : \mathbf{I^n} \to \mathbf{O^n}$ *can be computed anonymously by* G *iff* f *satisfies: For all inputs* $\vec{x}$ *and* $\vec{y}$ *and for all processors* i *and* j, *if* $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ *then* $f(\vec{x})_i = f(\vec{y})_j$.

Let us defer the proof of the theorem until the end of the section. We next give some examples illustrating the theorem.

EXAMPLE 2.5.2: **(Computing Processor id's)**

Let f be the constant function given by: $f(\vec{x}) = (1, \ldots, n)$ for all $\vec{x} \in \mathbf{I^n}$. A network which computes f has computed the id of each processor (Definition 2.2.2) in the network. By the theorem, this function is computable by a network G if and only if $\mathrm{U}_i\vec{x} \not\simeq \mathrm{U}_j\vec{y}$ for all processors i and j in G and all inputs $\vec{x}$ and $\vec{y}$. This holds iff all processors in the network have distinct rooted universal covers, ie, iff $\mathrm{U}_i \not\simeq \mathrm{U}_j$ for any processors i and j in the network.

EXAMPLE 2.5.3: Let $\vec{x} = (x, x, y, y)$ and $\vec{y} = (y, y, x, x)$. Then for G in Figure 2.9 we have $\mathrm{U}_1\vec{x} \simeq \mathrm{U}_2\vec{x} \simeq \mathrm{U}_3\vec{y} \simeq \mathrm{U}_4\vec{y}$, so any function f which G computes must at least satisfy:

$f(\vec{x})_1 = f(\vec{x})_2 = f(\vec{y})_3 = f(\vec{y})_4$.

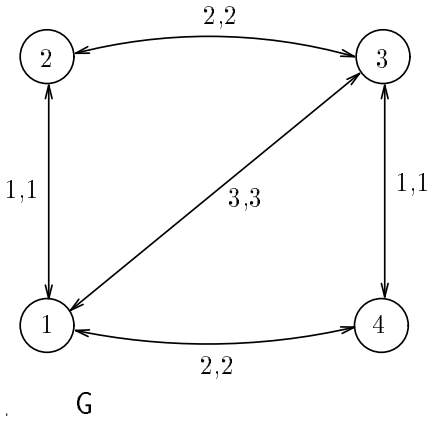


Figure 2.9: Computable functions

We now give a brief outline of the proof of Theorem 2.5.1. The "$\Longrightarrow$" direction, showing that an anonymously computable function satisfies the given conditions, is easy. Briefly; if $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ then processors i and j cannot distinguish themselves in a synchronous computation with inputs $\vec{x}$ and $\vec{y}$, and so must compute the same output. The proof of the "$\Longleftarrow$" part of the theorem requires finding an algorithm which anonymously computes $f(\vec{x})_i$ on each processor i. Before we present this algorithm we will show that each processor can anonymously compute a particular *permutation* $\vec{x}'$ of an input-vector $\vec{x}$, where $\vec{x}'$ is such that the set of rooted universal covers of $(\mathsf{G}, \vec{x})$ equals the set of rooted universal covers of $(\mathsf{G}, \vec{x}')$. An algorithm for computing $\vec{x}'$ is given in Lemma 2.5.3 below. Given this result, it will be easy to show that the following procedure, run on processor i, computes $f(\vec{x})_i$:

Processor i first computes $\mathrm{U}_i^{2n-1}\vec{x}$. Processor i then computes the permutation $\vec{x}'$ mentioned above, and finds a vertex v in G for which $\mathrm{U}_v^{2n-1}\vec{x}' \simeq \mathrm{U}_i^{2n-1}\vec{x}$. Finally, processor i computes $f(\vec{x}')$ and outputs $f(\vec{x}')_v$.

The next task before giving a detailed proof of Theorem 2.5.1 will be to show that the permutation $\vec{x}'$ discussed above is anonymously computable. It might seem at first that computing *any* permutation of $\vec{x}$ anonymously is impossible. Processors can exchange

their components of $\vec{x}$ until each has a complete set, but there is no obvious way to tell with what multiplicity each component should occur. However, consider the following two facts:

(1) If $i \in \mathsf{V}(\mathsf{G})$, the tree $\mathrm{U}_i^{2n-1}\vec{x}$ can sometimes be "mapped onto" G at a vertex $v \in \mathsf{V}(\mathsf{G})$, as follows: For each path $\mathcal{P}$ from the root of $\mathrm{U}_i^{2n-1}\vec{x}$, let $\mathcal{P}'$ be the path (if any) from v in G having the same word as $\mathcal{P}$. If $\mathcal{P}'$ exists, map the terminal vertex $\tilde{w}$ of $\mathcal{P}$ to the terminal vertex w of $\mathcal{P}'$, and label the vertex $w \in \mathsf{G}$ with the vertex-label $x_{\tilde{w}}$ of $\tilde{w}$. If every path from the root of $\mathrm{U}_i^{2n-1}\vec{x}$ corresponds to a path from v in G, *and* if the resulting vertex-labeling of G is consistent (i.e., no vertex gets two distinct labels) we will say that $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v. If $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v then the mapping procedure labels the vertices of G with the components of $\vec{x}$. That is, there is a vector $\vec{y}_v$ whose components equal those of $\vec{x}$, such that the vertex- labeling obtained by the mapping equals $(\mathsf{G}, \vec{y}_v)$. We will see that the set of rooted universal covers of $(\mathsf{G}, \vec{x})$ equals the set of rooted universal covers of $(\mathsf{G}, \vec{y}_v)$.

(2) It does not violate anonymity to allow an algorithm running on each processor of a network to have a copy of the graph of the network, complete with edge-labels and processor ids, but without input $\vec{x}$.

This suggests the following procedure for computing $\vec{x}'$: Each processor i first computes $\mathrm{U}_i^{2n-1}\vec{x}$, and then attempts sucessively to "map" $\mathrm{U}_i^{2n-1}\vec{x}$ onto vertex 1, vertex 2,..., and so on, of its copy of G. If v is the first vertex at which $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G, processor i takes the vector $\vec{y}_v$ described above to be $\vec{x}'$. In this way all processors compute the same vector $\vec{x}'$.

A more formal description of this algorithm will be given in the proof of Lemma 2.5.3.

EXAMPLE 2.5.4: It can be checked that $\mathrm{U}_1 \simeq \mathrm{U}_2 \simeq \mathrm{U}_3$, but $\mathrm{U}_1\vec{x} \not\simeq \mathrm{U}_2\vec{x} \not\simeq \mathrm{U}_3\vec{x}$, in Figure 2.10. For this reason $\mathrm{U}_2^5\vec{x}$ can only be "mapped onto" G at vertex 2. $\mathrm{U}_2^5\vec{x}$ does not map onto G at vertex 1, for instance, because under the attempted mapping, vertex 2 gets two distinct labels — label "x" from the path from the root of $\mathrm{U}_2^5\vec{x}$ having word "b", and also label "z", from the path with word "a".

Definition 2.5.1: Let G be a network and let $\vec{x}$ and $\vec{y}$ be input-vectors for G. We will say that $\vec{x}$ and $\vec{y}$ are *equivalent* if for each $j = 1, \ldots, n$ there is a vertex $k \in \mathsf{V}(\mathsf{G})$ such that $\mathrm{U}_j\vec{x} \simeq \mathrm{U}_k\vec{y}$.

EXAMPLE 2.5.5: For instance, let G be as pictured in Figure 2.11. If $\vec{x} = (x_1, x_2, x_3, x_4)$ and $\vec{y} = (x_3, x_4, x_1, x_2)$ then $\vec{x}$ and $\vec{y}$ are equivalent, since $\mathrm{U}_1\vec{x} \simeq \mathrm{U}_3\vec{y}$; $\mathrm{U}_2\vec{x} \simeq \mathrm{U}_4\vec{y}$; $\mathrm{U}_3\vec{x} \simeq \mathrm{U}_1\vec{y}$, and $\mathrm{U}_4\vec{x} \simeq \mathrm{U}_2\vec{y}$.

Remark 2.5.2: Note that $\vec{x}$ and $\vec{y}$ are equivalent iff there are vertices i and j in G such that $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$. This follows from Lemma 2.4.2, for suppose that $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ and that v is any vertex in G. Since G is connected, there is a path with word a from i to v. By Lemma 2.4.2, $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_{f_a(i)}\vec{x} \simeq \mathrm{U}_{f_a(j)}\vec{y}$.

Lemma 2.5.3: *If a network G is given input $\vec{x}$, then each processor in G can anonymously compute a vector $\vec{x}'$ equivalent to $\vec{x}$, and all processors compute the same $\vec{x}'$.*

Proof We will present an algorithm for computing $\vec{x}$ and show that the algorithm is correct. The algorithm is as follows:

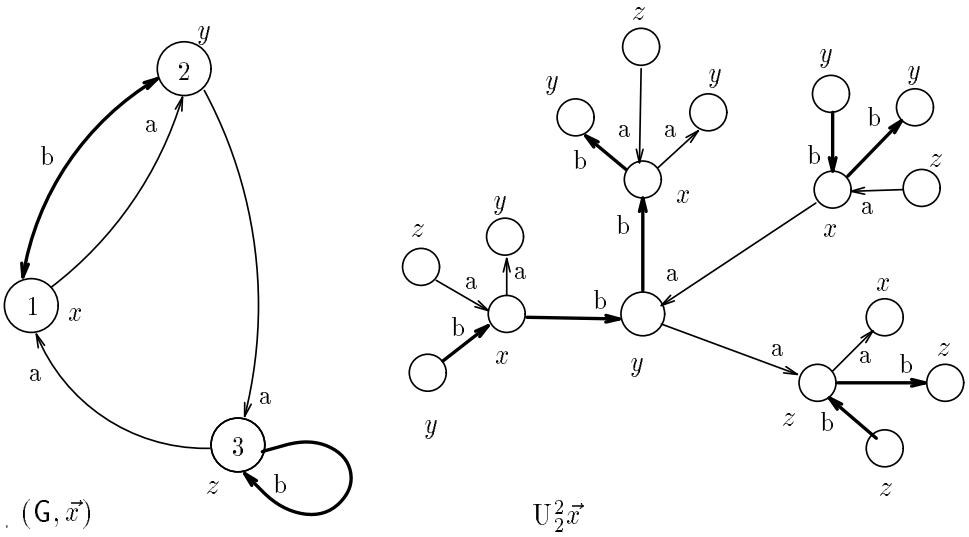


Figure 2.10: Mapping $U_2^5\vec{x}$ onto G

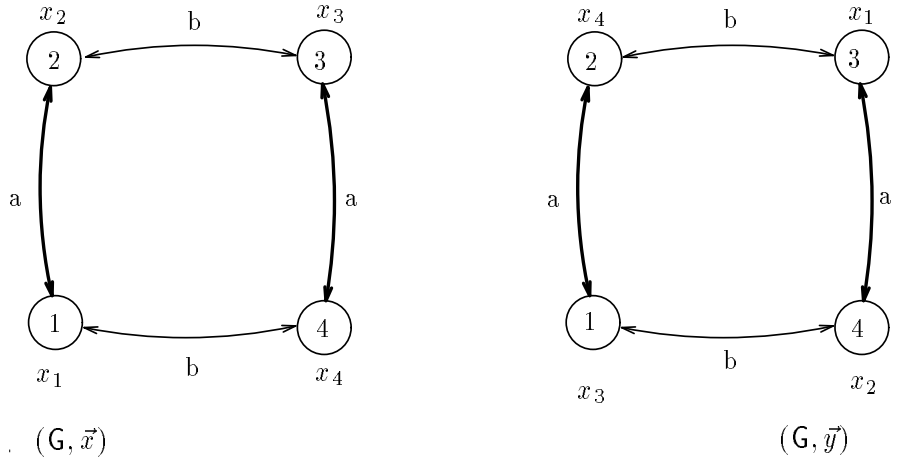


Figure 2.11: Equivalent input-vectors

Algorithm 2.5.2: (Run at each processor i, for computing a vector $\vec{x}'$ equivalent to $\vec{x}$)

Step 1: Compute $U_i^{2n-1}\vec{x}$.

Step 2: For each vertex $v = 1, \ldots, n \in V(G)$, run the procedure LABEL-G given below.

Step 3: For v the first vertex in G such that $U_i^{2n-1}\vec{x}$ maps onto G at v, take $\vec{x}' = (x'_1, \ldots, x'_n)$, where x'_j is the label of vertex j in G obtained from the procedure LABEL-G.

Procedure LABEL-G

- For each simple path $\mathcal{P}$ from the root $\tilde{i}$ of $\mathrm{U}_i^{2n-1}\vec{x}$, check for a path $\mathcal{P}'$ from v in G having the same word as $\mathcal{P}$. Conversely, for each simple path $\mathcal{P}'$ from v, check for a simple path $\mathcal{P}$ from $\tilde{i}$ having the same word. If no such corresponding path exists, output: "$\mathrm{U}_i^{2n-1}\vec{x}$ does not map onto G at v". Otherwise, label the terminal vertex w of $\mathcal{P}'$ with the label $x_{\tilde{w}} \in \{x_1, \ldots, x_n\}$ of the terminal vertex of $\mathcal{P}$.
- If w already has a label $x_j \neq x_{\tilde{w}}$, output: "$\mathrm{U}_i^{2n-1}\vec{x}$ does not map onto G at v".
- If all simple paths from $\tilde{i}$ in $\mathrm{U}_i^{2n-1}\vec{x}$ have corresponding paths $\mathcal{P}'$ from v and conversely, and if no vertex in G gets two distinct labels, output: "$\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v".

□

Proof of correctness: We will show that if $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v and the algorithm computes a vector $\vec{x}'$, then $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_v\vec{x}'$. That $\vec{x}$ and $\vec{x}'$ are equivalent then follows from Remark 2.5.2.

Suppose that $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v. Let $\alpha : \mathrm{U}_i^{2n-1} \to \mathrm{U}_v^{2n-1}$ be the map which takes the terminal vertex of each path $\mathcal{P}_1$ from the root of U_i^{2n-1} to the terminal vertex of the corresponding path $\mathcal{P}_2$ from the root of U_v^{2n-1}, where $\mathcal{P}_1$ and $\mathcal{P}_2$ have the same word. Since $\mathrm{U}_i^{2n-1}\vec{x}$ maps onto G at v we have that $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ is defined on the root of U_i iff $f_a \in \mathcal{E}(\mathsf{G})$ is defined on v. Since U_v covers G, we have f_a is defined on v iff $\tilde{f}_a$ is defined on the root of U_v. Hence α is defined everywhere. The proof of Lemma 2.4.1 shows that α is an isomorphism. By construction, any pair of vertices $\tilde{r} \in \mathrm{U}_i^{2n-1}$ and $\alpha(\tilde{r}) \in \mathrm{U}_v^{2n-1}$ share the same vertex-label, and so $\mathrm{U}_i^{2n-1}\vec{x} \simeq \mathrm{U}_v^{2n-1}\vec{x}'$ (Definition 2.4.3). By Proposition 2.4.1, $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_v\vec{x}'$. □

We can now give a formal proof of Theorem 2.5.1.

Proof (of Theorem 2.5.1)

($\Longrightarrow$) Suppose that f can be computed by G, that is, that there is an algorithm $\mathcal{A}$ for computing f on G. Then any execution, in particular, the synchronous execution of $\mathcal{A}$, computes f on G. Suppose that $\vec{x}$, $\vec{y}$, i and j are such that $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$. Then by Lemma 2.5.1 above, at each step of the synchronous execution of $\mathcal{A}$ on G with input $\vec{x}$, processor i computes the same values that processor j produces during the same step of the synchronous execution of $\mathcal{A}$ with input $\vec{y}$. Hence the value processor i computes for $f(\vec{x})_i$ is the same as the value that processor j computes for $f(\vec{y})_j$, and $f(\vec{x})_i = f(\vec{y})_j$.

($\Longleftarrow$) Suppose first that f is computable and satisfies the conditions of the theorem. We will argue that f is anonymously computable on G.

Algorithm 2.5.3: (for computing f on G)

To compute $f(\vec{x})_i$, processor i first computes $\mathrm{U}_i^{2n-1}\vec{x}$. Processor i then runs Algorithm 2.5.2 to find a vector $\vec{x}'$ equivalent to $\vec{x}$ and a vertex v for which $\mathrm{U}_i^{2n-1}\vec{x} \simeq \mathrm{U}_v^{2n-1}\vec{x}'$. Finally, processor i computes $f(\vec{x}')$ and outputs $f(\vec{x}')_v$.

By Lemma 2.5.2, any processor i can compute $\mathrm{U}_i^{2n-1}\vec{x}$. By Lemma 2.5.3 any processor can compute $\vec{x}'$ and all processors compute the same vector $\vec{x}'$. The vector $f(\vec{x}')$ can be computed since f is a computable function. By hypothesis, $f(\vec{x})_i = f(\vec{x}')_v$ whenever $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_v\vec{x}'$. By Proposition 2.4.1, we have $f(\vec{x})_i = f(\vec{x}')_j$ whenever $\mathrm{U}_i^{2n-1}\vec{x} \simeq \mathrm{U}_v^{2n-1}\vec{x}'$, and so processor i computes the correct value. □

2.6 Related Work

The Edge-Label Monoid: There is an extensive literature on the semigroups associated with finite-state machines, usually presented under the heading of "algebraic automata theory" (e.g., see Holcombe's book, [Hol82]). Any finite-state machine M has associated with it a graph $\mathsf{G} = \langle \mathsf{V}(\mathsf{G}), \mathsf{E}(\mathsf{G}), \mathsf{A}(\mathsf{G}) \rangle$, where G need not satisfy Property 2. The semigroup of M is defined to be the quotient of the free semigroup on $\mathsf{A}(\mathsf{G})$ by the relation $\sim$, where $a_1 \sim a_2$ for a_1 and $a_2 \in \mathsf{A}(\mathsf{G})^*$ if $f_{a_1}(v) = f_{a_2}(v)$ for all $v \in \mathsf{V}(\mathsf{G})$, for f_{a_i} defined as in Subsection 2.3.1. The edge-label monoid as we have defined it is a special case of this, for which the functions f_a are total and are one-to-one on $\mathsf{V}(\mathsf{G})$, and for which an identity and partial inverses are given.

From combinatorial group theory we obtain the related notion of a "groupoid". (See [Coh89].) A *groupoid* is a set G together with a partial multiplication which is associative where defined, such that any element in G has a partial inverse. For instance, the set of vertices of a graph G, together with the set $\{f_a : a \in \mathsf{A}(\mathsf{G})\}$ of partial edge-label functions, forms a groupoid under partial function composition. Cohen associates a graph with a groupoid as follows: the vertices are the partial identity elements. If an element $b \in G$ has left and right identities e and f, respectively, then there is an edge labeled "b" from e to f in the graph. Thus instead of completing the partial functions in our edge-label monoid, we could have defined the "edge-label groupoid" of a graph in this paper.

Covering Maps: The definition of covering map we use is borrowed more-or-less intact from algebraic topology (for instance, see [Mas67]). Algebraic automata theory uses a generalization of this notion of covering. In our notation, it is given as follows: Let G and H be edge-labeled digraphs, not necessarily satisfying Property 2. A *covering map* from G to H is a pair (ρ, γ) of maps, where $\rho : \mathsf{V}(\mathsf{G}) \to \mathsf{V}(\mathsf{H})$ is a partial function and $\gamma : \mathsf{A}(\mathsf{G}) \to \mathsf{A}(\mathsf{H})$ is a function such that $f_a\rho(v) = \rho(f_{\gamma(a)}(v))$ for all $v \in \mathsf{V}(\mathsf{G})$ on which ρ is defined. ([Hol82], page 43)

Covering maps (as we define them) are used in distributed computing to capture the notion of two networks being "locally the same". Angluin showed that if the graphs of two computer networks have a common finite cover, then the behavior of the networks is indistinguishable if the networks have a "uniform initial configuration" and if all processors of the same degree run the same algorithm ([Ang80]). She exhibited a polynomial-time algorithm for determining whether two networks have isomorphic rooted universal covers, and Leighton ([Lei82]) showed that graphs having isomorphic universal covers share a common finite cover. Fischer, Lynch and Merritt ([FLM85]) used graph covers to simplify proofs in fault-tolerant computing. Attiya, Snir and Warmuth ([ASW88]), studying rings of anonymous processors, made extensive use of the notion of the "k-neigborhood" of a

vertex v in a ring. The k-neighborhood of a vertex uniquely specifies the rooted universal cover with root v, truncated at depth k.

In their paper ([YK87b]), Yamashita and Kameda introduced the notion of the "view" of a vertex of an edge-labeled directed graph. The *view* T_v of a vertex v in a graph G is a rooted subgraph of U_v induced by the collection of paths directed away from the root of U_v. In ([YK87b]) Yamashita and Kameda showed that the view of a processor in an anonymous network represents what the processor can learn of the topology of its network by exchanging messages with its neighbors. Kranakis, Krizanc and van den Berg ([KKvdB90]) made use of the view in deriving a number of results on computing boolean functions on anonymous networks.

The view (although not called by that name) also makes an appearance in algebraic automata theory. States s_1 and s_2 in a deterministic finite-state machine are called *k-equivalent* if T_{s_1}, truncated at depth k, is isomorphic to T_{s_2} truncated at depth k. In 1956 Moore ([Moo56]) showed that states s_1 and s_2 are $n-2$ equivalent in an n state machine iff they are k-equivalent for all positive k. An argument similar to Moore's gives us Proposition 2.4.1 in this paper.

Anonymous Computing: The papers [Ang80], [YK87b,YK87a,YK88] and [KKvdB90] all require that an algorithm which runs on an anonymous network work for *any* edge-labeling of the network satisfying Property 2. This means that an algorithm must work on every graph in the *family* $\{(\mathsf{G},\sigma) : \sigma \in \Sigma\}$ of networks, where G is a graph without edge-labels and each $\sigma \in \Sigma$ is an edge-labeling of G satisfying Property 2. As a consequence, the conditions these papers derive for functions to be computable are more restrictive than the conditions we derive here. The viewpoint of this paper is that a network's edge-labeling is an intrinsic part of the network, and we require an algorithm to work only on a particular edge-labeling.

In [ASW88], Attiya, Snir and Warmuth gave the following characterization of the functions computable on a ring of n processors:

Theorem 2.6.1:

1. *Let G be an "oriented" ring of n processors. That is, the links of G are $\{(v_1,1),(v_2,2)\}$, $\{(v_2,1),(v_3,2)\}$, ..., $\{(v_{n-1},1),(v_n,2)\}$, $\{(v_n,1),(v_1,2)\}$.*
   Then any function $f : \mathbf{I^n} \to \mathbf{O}$ is computable by G iff f is invariant under cyclic shifts of the input. (Here G is said to "compute" $f(x_1,\ldots,x_n) = y$ if each processor i computes the same value y when given input x_i.)
2. *There exists an algorithm that computes f on any ring with n processors (that is, with any edge-labeling) iff f is invariant under cyclic shifts and reversals of the input.*

In [YK87a], Yamashita and Kameda generalized the second part of this result to arbitrary anonymous networks. As in the above, a network G is said to compute a function $f(x_1,\ldots,x_n) = y$ if each processor i computes the same value y when given x_i as input. In our notation, Yamashita and Kameda's characterization is as follows:

Theorem 2.6.2: *(Theorem 4.1 in [YK87a])*

Let G be a network with n processors. Let $\sim$ be the equivalence-relation on the set of all input-vectors $\vec{x} \in \mathbf{I^n}$, given as follows:

$\vec{x} \sim \vec{y}$ iff there exist two edge-labelings (σ_1 and σ_2) of the edges of G *such that the sets* $\{\mathrm{U}_i\vec{x} : i \in \mathsf{V}(\mathsf{G}),\ \mathsf{G}$ *labeled with* $\sigma_1\}$ *and* $\{\mathrm{U}_i\vec{y} : i \in \mathsf{V}(\mathsf{G}),\ \mathsf{G}$ *labeled with* $\sigma_2\}$ *are equal. Then* G *computes* $f : \mathbf{I^n} \to \mathbf{O}$ *iff* $f(\vec{x}) = f(\vec{y})$ *whenever* $\vec{x} \sim \vec{y}$.

The characterization we obtain for vector-valued functions is clearly quite similar to this. The partition associated with the equivalence relation we give on the set of inputs (Definition 2.5.1) is a refinement the partition from Yamsahita and Kameda's definition, and our Theorem 2.5.1 implies that a network G computes $f : \mathbf{I^n} \to \mathbf{O}$ iff $f(\vec{x}) = f(\vec{y})$ whenever $\vec{x}$ and $\vec{y}$ are equivalent, by our definition of equivalence.

The anonymous computing literature explores a number of issues which we do not address in this paper. For instance, [ASW88] and [YK88] show that certain computations are impossible for an anonymous network if the network does not 'know' how many processors it has. Most of the papers on anonymous computing consider the running time of algorithms, which we do not, except to differentiate hard from easy problems. Yamashita and Kameda ([YK87b,YK88]) examine a network's capacity under various assumptions about how much the network 'knows' about its topology; whereas in the proof of Theorem 2.5.1 we assume that every processor knows the graph of the network.

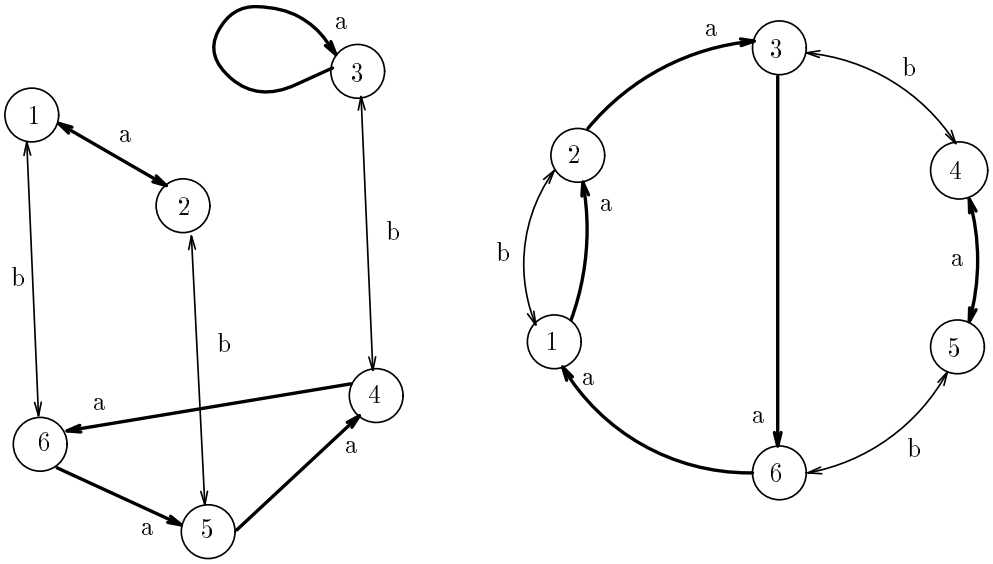
3. The Symmetries of a Network

3.1 Introduction

In the last chapter, we found a characterization of the set of functions that a given network can compute. In this chapter we begin work on a pair of classification problems. To describe these, let us say that two networks are "f-equivalent" if the set of functions each can compute are the same, and "p-equivalent" if the set of functions are the same "up to a permutation", that is, if they would be the same if the vertex-labels of one graph were changed by a permutation. We will look for a collection of topological or algebraic features of the graphs of networks which correctly place them in their equivalence-classes under these two relations.

What graph features might we expect to succeed at the first classification job? A logical first choice might be the set of automorphisms of a graph: Perhaps two networks can compute the same functions iff their automorphism groups are identical. In Chapter 5, in fact, we will see that having identical automorphism groups is a necessary condition for two graphs to compute the same set of functions. It is not a sufficient condition, however, as Example 3.1.1 shows.

EXAMPLE 3.1.1: It can easily be checked that the graphs G_1 and G_2 in Figure 3.1 both have the trivial automorphism group. For instance, any automorphism of G_1 must map the vertex "3" to itself, and by Corollary 2.3.1, an automorphism is determined by its action on a single vertex. In G_2 it can be checked that there is no automorphism mapping



G_1 G_2

Figure 3.1: G_1 and G_2 have identical automorphism groups but compute different sets of functions.

vertex 1 to vertex 2, or mapping vertex 1 to vertex 3, and so on, and hence no nontrivial automorphisms. However, if we choose inputs $\vec{x} = (x, x, x, y, y, y)$ and $\vec{y} = (y, y, y, x, x, x)$, then it can be checked by hand that, for instance, $\mathrm{U}_1^{n-1}\vec{x} \simeq \mathrm{U}_4^{n-1}\vec{y}$ in G_1 but not in G_2. By Theorem 2.5.1, this implies that G_2 can compute a function that G_1 cannot compute.

The graph G_1 in the above example has, in a sense, more 'symmetry' than G_2: There are input vectors for G_1 under which a number of its vertices have isomorphic rooted universal covers, but this is not true of G_2. (This can be checked using techniques developed later in this chapter.) In general, networks which are highly "symmetrical" in this sense can compute fewer functions than asymmetrical networks. One aim of this chapter will be to make precise this notion of network symmetry. The graph properties that we call "network symmetries" will turn out, in fact, to be the features we are seeking for classifying networks.

Chapter Outline and Main Results:

Our first step in defining the symmetries of a network G will be to describe a class of partitions of the vertex-set of a graph which are preserved by the edge-label monoid of the graph. If π is such a partition, there is a natural quotient-graph G/π, obtained by identifying the vertices in each block of π. In Section 3.2 of this chapter we will define these "correct partitions" and explore their relationship with universal covers. We will prove that there is a unique "coarsest" correct partition (Proposition 3.2.2.) In Section 3.3 we will define the quotient graph G/π, where π is a correct partition, and lay out some of the properties of quotient-graph isomorphisms. In Section 3.4, we will define 'network symmetry' and what it means for a network to satisfy a symmetry, and prove the following: (1) A network computes a function iff the function satisfies all of the network's symmetries (Theorem 3.4.1), and (2) Networks compute the same set of functions iff their symmetry-sets are identical. (Theorem 3.4.2). Finally, in the last section, we will consider networks which *would* compute the same set of functions if the processor id's of one of the networks were changed by a permutation. We will say that two such networks "differ by a permutation", and show that two networks differ by a permutation iff their symmetry sets "differ by a permutation" also.

3.2 Universal Covers And Vertex Partitions

The next two sections fill in some necessary background which will be used to track down a notion of 'network symmetry'. In this section we will look at a family of partitions of $\mathsf{V}(\mathsf{G})$ which are preserved by the graph monoid.

We will need the following notation:

Notation: Let G be a graph. If $\mathsf{B} = \{i_1, i_2, \ldots, i_k\} \subseteq \mathsf{V}(\mathsf{G})$ and $f_a \in \mathcal{E}(\mathsf{G})$, we will write $f_a\mathsf{B}$ for the image of B under f_a, i.e., for the set $\{f_a(i_1), \ldots, f_a(i_k)\}$.

Definition 3.2.1: Let G be a graph, and Let $\pi = \{\mathsf{B}_1, \ldots, \mathsf{B}_k\}$ be a partition of $\mathsf{V}(\mathsf{G})$. We call π a *correct partition with respect to* $\mathcal{E}(\mathsf{G})$, or *c-partition* for short, if it satisfies:

1. For each block $\mathsf{B} \in \pi$, any $f_a \in \mathcal{E}(\mathsf{G})$ is defined on all or none of the elements of B.

2. For any block $\mathsf{B} \in \pi$ and for any $f_a \in \mathcal{E}(\mathsf{G})$, if f_a is defined on the elements of B then $f_a\mathsf{B} \in \pi$.

EXAMPLE 3.2.1: In Figure 3.2 below, the c-partitions of G are $\pi_1 = 1/2/3/4$ (i.e., $\pi_1 = \{\mathsf{B}_1, \mathsf{B}_2, \mathsf{B}_3, \mathsf{B}_4\}$ where $\mathsf{B}_1 = \{1\}, \ldots, \mathsf{B}_4 = \{4\}$) and $\pi_2 = 1, 4/2, 3$.

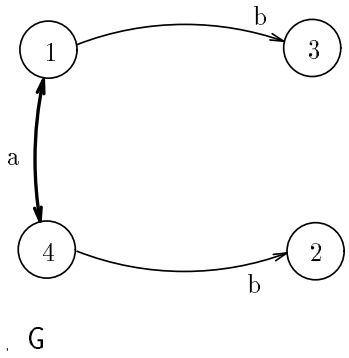


Figure 3.2: A graph and its c-partitions

Notation: If π is a partition of $\mathsf{V}(\mathsf{G})$, we will write $[i]$ for the block of π containing a point $i \in \mathsf{V}(\mathsf{G})$. If π_1 and π_2 are two c-partitions, write $[i]_{\pi_1}$ for the block of π_1 containing i and $[i]_{\pi_2}$ for the block of π_2 containing i.

We will see next that the c-partitions of a graph are the partitions associated with a certain class of equivalence relations, the 'congruences'.

Definition 3.2.2: Let G be a graph. A *congruence relation*[1] $\sim$ on $\mathsf{V}(\mathsf{G})$ is an equivalence relation satisfying:

1. For all $f_a \in \mathcal{E}(\mathsf{G})$ and $v, w \in \mathsf{V}(\mathsf{G})$, if $v \sim w$ then f_a is defined on both or neither of v and w.
2. For all $f_a \in \mathcal{E}(\mathsf{G})$ and $v, w \in \mathsf{V}(\mathsf{G})$, $v \sim w$ implies that $f_a(v) \sim f_a(w)$ if f_a is defined on v and w.

The following is immediate from the definintion of c-partition:

Lemma 3.2.1: *Let π be a partition of $\mathsf{V}(\mathsf{G})$ and let $\sim$ be the equivalence relation defined by π, i.e., $i \sim j$ iff $[i] = [j]$ in π. Then π is a c-partition iff $\sim$ is a congruence relation.*

□

The next proposition shows that all blocks of a given c-partition are the same size.

Proposition 3.2.1: *If π is a c-partition of G then $|[v]| = |[w]|$ for all $[v]$ and $[w]$ in π.*

Proof Let $[v]$ and $[w]$ be two blocks in π and let $i \in [v]$ and $j \in [w]$. Since G is connected, there is some $f_a \in \mathcal{E}(\mathsf{G})$ such that $f_a(i) = j$. Then $f_a([v]) = [w]$, since if $i, k \in [i]$ and $f_a(i) \in [w]$ then $f_a(k) \in [w]$, by definition of congruence relation. By Property 2 in Chapter 2 the elements of $\mathcal{E}(\mathsf{G})$ are one-to-one on $\mathsf{V}(\mathsf{G})$, and so f_a induces a bijection from $[v]$ to $[w]$. Hence $|[v]| = |[w]|$. □

[1] [B$\ddot{8}$9] page 91, also [BS81]

The next lemma gives a relationship between graph covering maps and c-partitions. It states that the partition induced by the inverse image of a covering map is a c-partition.

Lemma 3.2.2: *Let* G *and* H *be graphs, and let* $\beta : \mathsf{G} \to \mathsf{H}$ *be a covering map. Define an equivalence relation* $\sim_\beta$ *on* $\mathsf{V}(\mathsf{G})$ *by:* $v \sim_\beta w$ *iff* $\beta(v) = \beta(w)$*. Then* $\sim_\beta$ *is a congruence and so the related partition* π_β *of* $\mathsf{V}(\mathsf{G})$ *is a c-partition.*

EXAMPLE 3.2.2: Let β be the covering map taking 1 and 2 to u, and 3 and 4 to v in Figure 3.3 below. Then $\pi = 1, 2/3, 4$ is a c-partition.

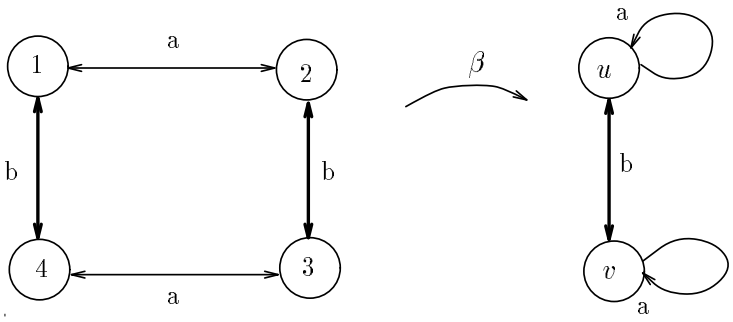


Figure 3.3: Example for Lemma 3.2.2

Proof of the lemma: Suppose first that $v \sim_\beta w$ and that $f_a \in \mathcal{E}(\mathsf{G})$ is defined on v. Then since β is a covering map, $g_a \in \mathcal{E}(\mathsf{H})$ is defined on $\beta(v) \in \mathsf{H}$ and so f_a is defined on w, by Proposition 2.3.1 in Chapter 2. Similarly, if f_a is defined on w then it is defined on v. Next, suppose that $f_a \in \mathcal{E}(\mathsf{G})$ is defined on vertices v and $w \in \mathsf{V}(\mathsf{G})$. Then since β is a covering map, we have $\beta f_a(v) = g_a \beta(v)$ and $\beta f_a(w) = g_a \beta(w)$ (again by Proposition 2.3.1 in Chapter 2). Since $\beta(v) = \beta(w)$, we have $\beta(f_a(v)) = \beta(f_a(w))$, and so $f_a(v) \sim_\beta f_a(w)$, and $\sim_\beta$ is a congruence. □

The next three lemmas give conditions that insure that two rooted universal covers are isomorphic. Lemma 3.2.3 is a generalization of Lemma 2.4.1 in Chapter 2. It states that two rooted universal covers are isomorphic if the paths from the root of one are isomorphic to the paths from the root of the other. Lemma 3.3.3 is a corollary of Lemma 3.2.3.

Lemma 3.2.3: *Let* G *have universal cover* U *and let* U_v *and* U_w *have roots* $\tilde{v}$ *and* $\tilde{w}$*, respectively. Suppose that each* $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ *is defined on* $\tilde{v}$ *iff it is defined on* $\tilde{w}$*. Then there is an isomorphism* $\tilde{\delta} : \mathrm{U}_v \to \mathrm{U}_w$ *given as follows:* $\tilde{\delta}(\tilde{v}) = \tilde{w}$*. For all* $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ *defined on* $\tilde{v}$ *for which* a *is reduced, define* $\tilde{\delta}\tilde{f}_a(\tilde{v})$ *to be the vertex* $\tilde{f}_a\tilde{\delta}(\tilde{v}) = \tilde{f}_a(\tilde{w})$*. For each* $\tilde{r} \in \mathsf{V}(\mathrm{U})$*, define* $\tilde{\delta}(\tilde{r})$ *to be* $\tilde{\delta}\tilde{f}_a(\tilde{v})$*, where* $\tilde{f}_a$ *is the unique element of* $\mathcal{E}(\mathrm{U})$ *mapping* $\tilde{v}$ *to* $\tilde{r}$ *for which* a *is reduced.*

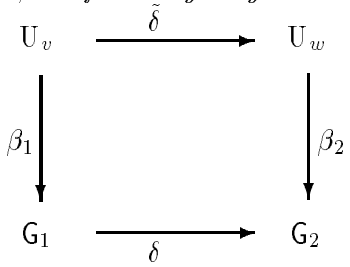
Proof The proof of Lemma 2.4.1, Chapter 2, shows that $\tilde{\delta}$ is an isomorphism. □

Lemma 3.2.4: *Let* π *be a c-partition of* G *and let* i *and* $j \in \mathsf{V}(\mathsf{G})$ *be in the same block of* π*. Then* $\mathrm{U}_i \simeq \mathrm{U}_j$*.*

Proof Let i and $j \in \mathsf{V}(\mathsf{G})$ be such that $[i] = [j] \in \pi$. Let U_i and U_j have roots $\tilde{i}$ and $\tilde{j}$, respectively, and canonical covering maps β_1 and β_2. The conclusion will follow from Lemma 3.2.3 if we show that $f_a \in \mathcal{E}(\mathsf{G})$ is defined on $\tilde{i}$ iff it is defined on $\tilde{j}$. Suppose that $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ is defined on $\tilde{i}$. Then $f_a \in \mathcal{E}(\mathsf{G})$ is defined on $\beta_1(\tilde{i}) = i$ since β_1 is a covering map. Since i and j are in the same block of π, f_a is defined on j, and hence $\tilde{f}_a$ is defined on $\tilde{j}$ since β_2 is a covering map. The same argument shows that if $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ is defined on $\tilde{j}$ then it is defined on $\tilde{i}$. □

The next lemma will be used in the proof of Proposition 3.3.5. It describes conditions under which an isomorphism between two graphs 'lifts' to an isomorphism between rooted universal covers.

Lemma 3.2.5: *Let* G_1 *and* G_2 *have the same universal cover* U *and let* $\delta : \mathsf{G}_1 \to \mathsf{G}_2$ *be an isomorphism such that* $\delta(v) = w$ *for some* $v \in \mathsf{V}(\mathsf{G}_1)$ *and* $w \in \mathsf{V}(\mathsf{G}_2)$. *Then there is an isomorphism* $\tilde{\delta} : \mathrm{U}_v \to \mathrm{U}_w$ *such that* $\beta_2\tilde{\delta} = \delta\beta_1$ *for* $\beta_1 : \mathrm{U}_v \to \mathsf{G}_1$ *and* $\beta_2 : \mathrm{U}_w \to \mathsf{G}_2$ *the canonical covering maps. That is, the following diagram commutes:*



Proof Define $\tilde{\delta}$ as in Lemma 3.2.3. We first use Lemma 3.2.3 to show that $\tilde{\delta}$ is an isomorphism. If $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ is defined on the root $\tilde{v}$ of U_v then $g_a \in \mathcal{E}(\mathsf{G}_1)$ is defined on v since β_1 is a covering map. Also, $h_a \in \mathcal{E}(\mathsf{G}_2)$ is defined on $\delta(v)$ since δ is an isomorphism, and so $\tilde{f}_a$ is defined on the root $\tilde{w}$ of U_w since β_2 is a covering map. Similarly, $\tilde{f}_a$ being defined on $\tilde{w}$ implies that $\tilde{f}_a$ is defined on $\tilde{v}$, and so by Lemma 3.2.3, $\tilde{\delta}$ is an isomorphism.

Let $\tilde{r} \in \mathsf{V}(\mathrm{U})$ and let $\tilde{f}_a$ be the unique element of $\mathcal{E}(\mathrm{U})$ mapping $\tilde{v}$ to $\tilde{r}$ for which a is reduced. If $\beta_1 : \mathrm{U}_v \to \mathsf{G}_1$ and $\beta_2 : \mathrm{U}_w \to \mathsf{G}_2$ are the canonical covering maps then by Proposition 2.3.1 in Chapter 2 we have $\delta\beta_1(\tilde{r}) = \delta\beta_1\tilde{f}_a(\tilde{v}) = \delta f_a\beta_1(\tilde{v})$. Furthermore, $\delta f_a\beta_1(\tilde{v}) = \delta f_a(v) = f_a\delta(v) = f_a(w)$, and $\beta_2\tilde{\delta}(\tilde{r}) = \beta_2\tilde{f}_a(\tilde{w}) = f_a\beta_2(\tilde{w}) = f_a(w)$, and so $\delta\beta_1(\tilde{r}) = \beta_2\tilde{\delta}(\tilde{r})$. Since a covering map is determined by its action on a single vertex, we have $\delta\beta_1 = \beta_2\tilde{\delta}$. □

The last proposition in this section shows that any graph has a unique c-partition with largest block-size. We will use this proposition in Chapter 6.

Definition 3.2.3: We will say that a partition π_1 is a *refinement* of a partition π_2, written $\pi_1 \preceq \pi_2$, if whenever i and j are in the same block of π_1 they are also in the same block of π_2.

Proposition 3.2.2: *Let* G *be a graph. Then there is a unique "coarsest" c-partition* Π *of* $\mathsf{V}(\mathsf{G})$ *such that any other c-partition of* $\mathsf{V}(\mathsf{G})$ *is a refinement of* Π. *(Figure 3.4)*

EXAMPLE 3.2.3: The graph in Figure 3.4 has "coarsest c-partition" $\Pi = 1,2,3/4,5,6$.

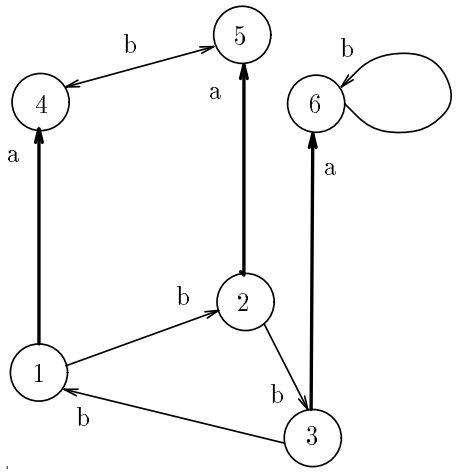


Figure 3.4: Illustrating the "coarsest c-partition" of a graph

Proof of Proposition 3.2.2 Let G have universal cover U. Put a relation $\sim$ on $\mathsf{V}(\mathsf{G})$ by $v \sim w$ if $\mathrm{U}_v \simeq \mathrm{U}_w$. It is easy to see that $\sim$ is an equivalence relation. An argument similar to that used in the proof of Corollaries 3.2.4 and 3.2.5 shows that if $v \sim w$ then $f_a \in \mathcal{E}(\mathsf{G})$ is defined on v iff it is defined on w. By Lemma 2.4.2 in Chapter 2, if f_a is defined on v and $v \sim w$ then $\mathrm{U}_{f_a(v)} \simeq \mathrm{U}_{f_a(w)}$, and so $f_a(v) \sim f_a(w)$. Thus $\sim$ is a congruence relation and the associated partition Π is a c-partition. Let π be any other c-partition of G , with $\sim_\pi$ its associated congruence relation. If $v \sim_\pi w$ for $v, w \in \mathsf{V}(\mathsf{G})$, then by Corollary 3.2.4 above, $\mathrm{U}_v \simeq \mathrm{U}_w$ and so $v \sim w$. That is, π is a refinement of Π. □

3.3 Quotient Graphs And Their Isomorphisms

This section examines a relationship between the set of graphs covered by a graph G and the set of c-partitions of G. In Proposition 3.3.2, we will find that these sets are in one-to-one correspondence, that is, for every c-partition π there is a "quotient graph" G/π which G covers. In Proposition 3.3.5 we will find a relationship between isomorphisms of rooted universal covers with input and isomorphisms of quotient graphs.

Definition 3.3.1: If π is a c-partition of G, the *quotient graph* G/π is defined as follows: The vertices of G/π are the blocks of π. A triple $\langle [v]\, a\, [w] \rangle$ is an edge of G/π for $[v]$ and $[w] \in \pi$ iff there is an edge $\langle v\, a\, w \rangle \in \mathsf{E}(\mathsf{G})$ with $v \in [v]$ and $w \in [w]$.

It is easy to see that G/π is well-defined and is a graph. (See Example 3.3.1.)

EXAMPLE 3.3.1: Graphs G and G/π are pictured in Figure 3.5, for $\pi = 1,2,3/4,5,6$.

To simplify notation in future proofs, we will rephrase Proposition 2.3.1 from Chapter 2 for quotient-graphs:

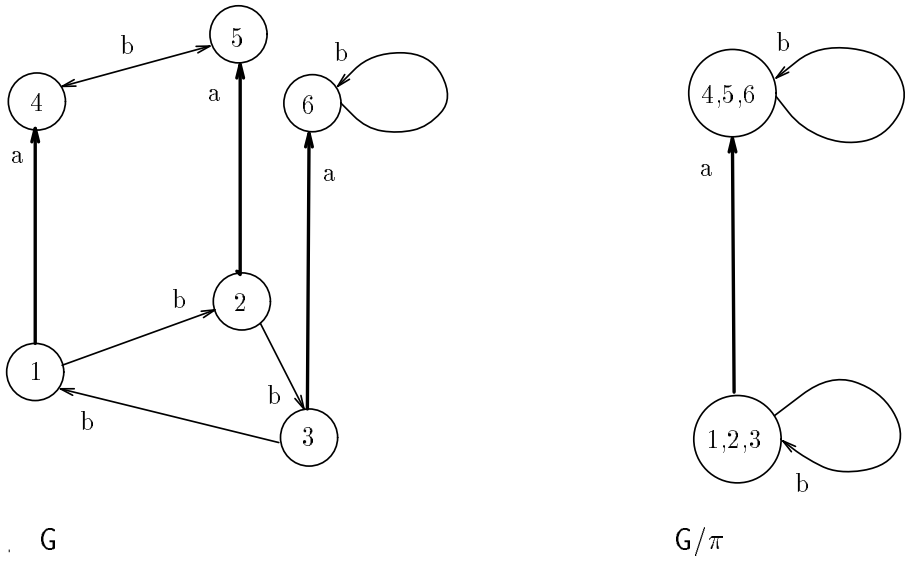


Figure 3.5: G and G/π, where $\pi = 1,2,3/4,5,6$.

Proposition 3.3.1: *Let π_1 and π_2 be c-partitions of a graph G. A surjective map $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is a covering map iff δ satisfies both of the following:*

(1) For all blocks $\mathsf{B} \in \pi_1$, any $f_a \in \mathcal{E}(\mathsf{G})$ is defined on B iff f_a is defined on $\delta(\mathsf{B})$.

(2) For all $f_a \in \mathcal{E}(\mathsf{G})$ and for all blocks $\mathsf{B} \in \pi_1$ on which f_a is defined, $\delta f_a \mathsf{B} = f_a \delta(\mathsf{B})$.

Proof By construction, the maps $g_a \in \mathcal{E}(\mathsf{G}/\pi_1)$ and $h_a \in \mathcal{E}(\mathsf{G}/\pi_2)$ are defined on $\mathsf{B} \in \pi_1$ and on $\delta(\mathsf{B}) \in \pi_2$, respectively, iff $f_a \in \mathcal{E}(\mathsf{G})$ is defined on all of the elements of B and on all of the elements of $\delta(\mathsf{B})$. Also, $\delta g_a \mathsf{B} = h_a \delta(\mathsf{B})$ iff $\delta f_a \mathsf{B} = f_a \delta(\mathsf{B})$ by construction of the quotient graph. The conclusion then follows from Proposition 2.3.1 in Chapter 2. □

The next proposition gives a correspondence between the c-partitions of a graph G and the graphs covered by G.

Proposition 3.3.2: *Let G and H be graphs. If π is a c-partition of $\mathsf{V}(\mathsf{G})$ then G covers G/π. Conversely, if G covers H then $\mathcal{H} \simeq \mathsf{G}/\pi$ for some c-partition π of G.*

Proof Let π be a c-partition of G. We will show first that G covers G/π. Let $\beta : \mathsf{V}(\mathsf{G}) \to \mathsf{V}(\mathsf{G}/\pi)$ map each vertex $v \in \mathsf{V}(\mathsf{G})$ to $[v] \in \mathsf{V}(\mathsf{G}/\pi)$. Then β is a covering map: Clearly β is onto. By definition of c-partition, $f_a \in \mathcal{E}(\mathsf{G})$ is defined on $v \in \mathsf{V}(\mathsf{G})$ iff it is defined on all elements of $[v]$. Suppose that $f_a(v) = w$ for some $v \in \mathsf{V}(\mathsf{G})$ and $f_a \in \mathcal{E}(\mathsf{G})$. Then by the definition of c-partition, $f_a[v] = [w]$. By construction, $\beta f_a(v) = \beta(w) = [w]$, whereas $f_a \beta(v) = f_a[v] = [w]$. Thus $\beta f_a(v) = f_a \beta(v)$, and by Proposition 3.3.1 above, β is a covering map.

Suppose now that G covers a graph H via a covering map β. We show that there is a c-partition π such that $\mathsf{H} \simeq \mathsf{G}/\pi$. Put a relation $\sim_\beta$ on $\mathsf{V}(\mathsf{G})$ by: $v \sim_\beta w$ iff $\beta(v) = \beta(w)$. By Lemma 3.2.2, this relation is a congruence and so the related partition π of $\mathsf{V}(\mathsf{G})$ is a c-partition. Define a map $\delta : \mathsf{H} \to \mathsf{G}/\pi$ by $\delta(w) = \beta_1 \beta^{-1}(w) \in \pi$, where $\beta_1 : \mathsf{G} \to \mathsf{G}/\pi$ maps each i to $[i]_\pi$. It is straightforward to show that δ is an isomorphism. □

Definition 3.3.2: Let π_1 and π_2 be c-partitions such that $\pi_1 \preceq \pi_2$. The *inclusion covering map* $\beta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is the map which takes each block $[i] \in \pi_1$ to $[i] \in \pi_2$.

The argument used in the first part of the proof of Proposition 3.3.2 also gives us the following:

Lemma 3.3.1: *The inclusion covering map is a covering map. That is, if $\pi_1 \preceq \pi_2$ for c-partitions π_1 and π_2, then G/π_1 covers G/π_2.*

□

The next two results are ones we would expect to hold if "isomorphism" and "covering map" are defined correctly. The first result will be used in Chapter 5.

Proposition 3.3.3: *Let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism, and suppose that G/π_2 covers G/π_4. Then there is a c-partition π_3 of G such that G/π_1 covers G/π_3, for which there is an isomorphism $\delta' : \mathsf{G}/\pi_3 \to \mathsf{G}/\pi_4$; where $\delta' = \beta_2 \delta {\beta_1}^{-1}$ for $\beta_1 : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_3$ and $\beta_2 : \mathsf{G}/\pi_2 \to \mathsf{G}/\pi_4$ the inclusion covering maps.*

Proof Define a partition π_3 of $\mathsf{V}(\mathsf{G})$ such that v and w are in the same block of π_3 iff $\beta_2\delta([v]) = \beta_2\delta([w])$. By Lemma 3.2.2, π_3 is a c-partition, and $\pi_1 \preceq \pi_3$ by construction. By Lemma 3.3.1, G/π_1 covers G/π_3. As in Proposition 3.3.2, it is straightforward to show that $\delta' = \beta_2\delta{\beta_1}^{-1}$ is an isomorphism: $\mathsf{G}/\pi_3 \to \mathsf{G}/\pi_4$ for β_1 and β_2 the inclusion covering maps. □

Proposition 3.3.4: *Let π_1, π_2 and π_3 be c-partitions, and let $\delta_1 : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_3$ and $\delta_2 : \mathsf{G}/\pi_2 \to \mathsf{G}/\pi_3$ be isomorphisms such that $\delta_1([1]_{\pi_1}) = \delta_2([1]_{\pi_2})$. Then $\delta_1 = \delta_2$ and $\pi_1 = \pi_2$.*

Proof Let $\beta_1 : \mathsf{G} \to \mathsf{G}/\pi_1$ and $\beta_2 : \mathsf{G} \to \mathsf{G}/\pi_2$ be the inclusion covering maps (That is, $\beta_1(i) = [i]_{\pi_1}, \beta_2(i) = [i]_{\pi_2}$). Then since graph covering map are determined by their action on a single point, we have $\delta_1\beta_1(1) = \delta_2\beta_2(1)$ implies that $\delta_1\beta_1 = \delta_2\beta_2$. Since δ_1 and δ_2 are isomorphisms, the blocks of π_1, π_2 and π_3 are the same size. Thus if $v \in [1]_{\pi_1}$ but $v \notin [1]_{\pi_2}$ then $\delta_1\beta_1(v) \neq \delta_2\beta_2(v)$. Hence $[1]_{\pi_1} = [1]_{\pi_2}$. Since $\delta_1\beta_1 = \delta_2\beta_2$, we can use the same argument to show that $[v]_{\pi_1} = [v]_{\pi_2}$ for any $v \in \mathsf{V}(\mathsf{G})$. Thus $\pi_1 = \pi_2$ and so $\delta_1 = \delta_2$. □

We will now introduce a c-partition which is associated with an input vector.

Notation: Let $\vec{x}$ be an input for G. Define an equivalence relation $\sim_{\vec{x}}$ on $\mathsf{V}(\mathsf{G})$ by: $v \sim_{\vec{x}} w$ iff $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{x}$. This relation induces a partition, which we denote by $\pi_{\vec{x}}$, on $\mathsf{V}(\mathsf{G})$.

We have:

Lemma 3.3.2: *$\pi_{\vec{x}}$ is a c-partition of $\mathsf{V}(\mathsf{G})$ for any input $\vec{x}$ to G.*

Proof Let $[v] \in \pi_{\vec{x}}$ and $f_a \in \mathcal{E}(\mathsf{G})$ be defined on $v \in [v]$. Then for all $w \in [v]$ we have $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{x}$, and so f_a is defined on w also, by Proposition 2.3.1 in Chapter 2. By Lemma 2.4.2 in Chapter 2, if $\mathrm{U}_v\vec{x} \simeq \mathrm{U}_w\vec{x}$ and f_a is defined on v and w then $\mathrm{U}_{f_a(v)}\vec{x} \simeq \mathrm{U}_{f_a(w)}\vec{x}$. Hence if f_a is defined on the elements of $[v]$ then $f_a([v]) \in \pi$. □

Remark: The "coarsest" c-partition Π of a graph G equals $\pi_{\vec{x}}$, where $\vec{x}$ is any input vector, all of whose components are equal. Similarly, the finest c-partition $\pi = 1/2/\ldots/n$ equals $\pi_{\vec{x}}$, for any $\vec{x}$ with all distinct components. In general, for any c-partition π there exist inputs $\vec{x}$ such that $\pi = \pi_{\vec{x}}$.

Note that since $\pi_{\vec{x}}$ is a c-partition, $\mathsf{G}/\pi_{\vec{x}}$ is defined (as before) and G covers $\mathsf{G}/\pi_{\vec{x}}$.

The last proposition in this section gives a relationship between isomorphisms of rooted universal covers $\mathrm{U}_i\vec{x}$ and isomorphisms of quotient-graphs $\mathsf{G}/\pi_{\vec{x}}$. This proposition will be used to prove Theorem 3.4.1 in the next section.

Before we present the proposition, we will need a consistent notion of what it means for an isomorphism on a quotient-graph G/π to act on an input vector $\vec{x}$, if π is a refinement of $\pi_{\vec{x}}$. For instance, let $\pi_1 = 1,6/2,5/3,4/7,8$ and $\pi_2 = 1,2/3,8/4,7/5,6$, and let $\delta : \{1,6\} \to \{4,7\}$; $\{2,5\} \to \{3,8\}$; $\{3,4\} \to \{1,2\}$; and $\{7,8\} \to \{5,6\}$. (See Example 3.3.2.) If $\vec{x} = (x_1, x_2, \ldots, x_n) = (x,x,y,z,r,r,z,y)$ then $\pi_2 = \pi_{\vec{x}}$ and it is reasonable to define $\delta(\vec{x})$ to be that vector whose ith component is the $\delta([i])$th component of $\vec{x}$, i.e., $\delta(\vec{x}) = (x_4, x_3, x_1, x_1, x_3, x_4, x_5, x_5) = (z,y,x,x,y,z,r,r)$. This motivates the following notation:

Notation:

1. Let π be a c-partition of a graph and let $\vec{x}$ be such that $\pi \preceq \pi_{\vec{x}}$. We will write $x_{[i]}$ for x_i, where $i \in [i] \in \pi$.

   This is well-defined since $x_i = x_j$ whenever $[i] = [j]$ in π.
2. Let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism and let $\vec{x}$ be such that $\pi_2 \preceq \pi_{\vec{x}}$. We will write $\delta(\vec{x})$ for the vector $(x_{\delta([1])}, \ldots, x_{\delta([n])})$. That is, $\delta(\vec{x}) = (x_{i_1}, \ldots, x_{i_n})$ where $i_j \in \delta([j])$.

We will need the following lemma:

Lemma 3.3.3: *Suppose that a network G has a c-partition π and an input $\vec{x}$ satisfying: $x_i = x_j$ whenever $[i] = [j]$ in π. Then if $[i] = [j]$ in π then $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{x}$, and so $\pi \preceq \pi_{\vec{x}}$.*

Proof Let B be a block in π and let i and $j \in \mathsf{B}$. By Lemma 3.2.4, $\mathrm{U}_i \simeq \mathrm{U}_j$. Define an isomorphism $\delta : \mathrm{U}_i \to \mathrm{U}_j$ as Lemma 3.2.3. Let $\tilde{v}$ be a vertex in U_i and $\tilde{w} = \delta(\tilde{v})$ be a vertex in U_j. We will show that $x_v = x_w$, and conclude that $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{x}$. Let $\tilde{f}_a \in \mathcal{E}(\mathrm{U})$ map $\tilde{i}$ to $\tilde{v}$. Then $\tilde{w} = \delta(\tilde{v}) = \delta\tilde{f}_a(\tilde{i}) = \tilde{f}_a(\tilde{j})$. Since $f_a(\mathsf{B})$ is a block of π, it follows that v and w are in the same block of π and so by hypothesis, $x_v = x_w$. Hence $\tilde{v}$ and $\tilde{w}$ both have the same label x_v, and so $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$. □

We have:

Proposition 3.3.5: *Let G be a network. Then:*

1. *If there are inputs $\vec{x}$ and $\vec{y}$ and vertices i and j of G such that $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$, then there is an isomorphism $\delta : \mathsf{G}/\pi_{\vec{x}} \to \mathsf{G}/\pi_{\vec{y}}$ such that $\delta([i]) = [j]$ and $\vec{x} = \delta(\vec{y})$.*
2. *If $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is an isomorphism for c-partitions π_1 and π_2 of G, then for all $\vec{y}$ such that $\pi_2 \preceq \pi_{\vec{y}}$, we have $\pi_1 \preceq \pi_{\delta(\vec{y})}$ and $\mathrm{U}_i\delta(\vec{y}) \simeq \mathrm{U}_j\vec{y}$ for any i and j such that $\delta([i]) = [j]$.*

EXAMPLE 3.3.2: Examples of this proposition are given in Figures 3.6 and 3.7 below.

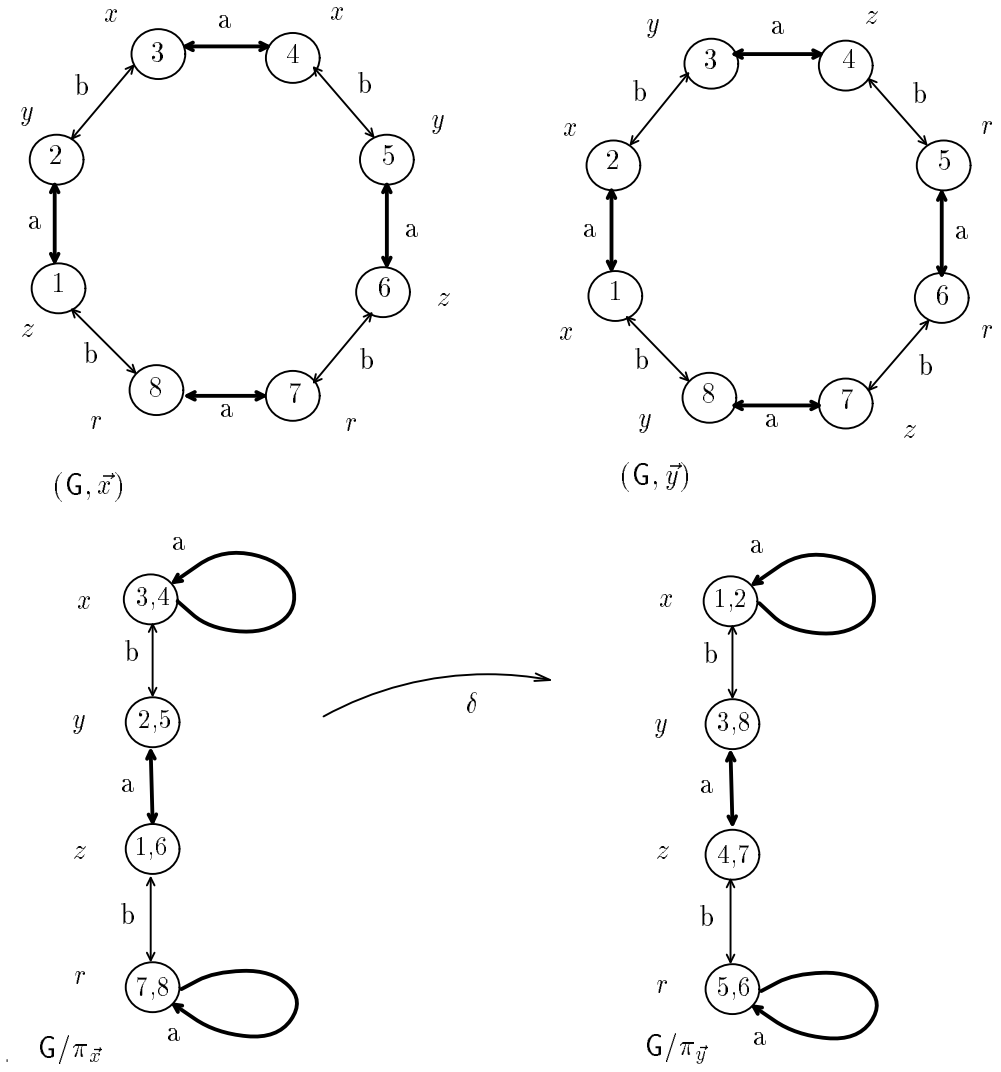


Figure 3.6: Example for Propostition 3.3.5 part 1.

In Figure 3.6, let $\vec{x} = (z, y, x, x, y, z, r, r)$ and $\vec{y} = (x, x, y, z, r, r, z, y)$. Then $U_1\vec{x} \simeq U_4\vec{y}$, for instance, and the map δ: $1,6 \to 4,7$, $2,5 \to 3,8$, $3,4 \to 1,2$, $7,8 \to 5,6$ is an isomorphism from $G/\pi_{\vec{x}}$ to $G/\pi_{\vec{y}}$ for which $\delta(\vec{y}) = \vec{x}$.

In Figure 3.7 take G_1, G_2, and δ as above, and let $\pi_1 = \pi_{\vec{x}} = 1,6/2,5/3,4/7,8$ and $\pi_2 = \pi_{\vec{y}} = 1,2/3,8/4,7/5,6$. Let $\vec{y} = (x, x, x, x, r, r, x, x)$. Then $\delta(\vec{y}) = (x, x, x, x, x, x, r, r)$ and, for instance, $U_1\delta(\vec{y}) \simeq U_4\vec{y}$.

Proof of the proposition:

(1) Suppose that there are inputs $\vec{x}$ and $\vec{y}$ and vertices i and j such that $U_i\vec{x} \simeq U_j\vec{y}$. Let $\beta_1 : G \to G/\pi_{\vec{x}}$ and $\beta_2 : G \to G/\pi_{\vec{y}}$ be the inclusion covering maps. We will find a map

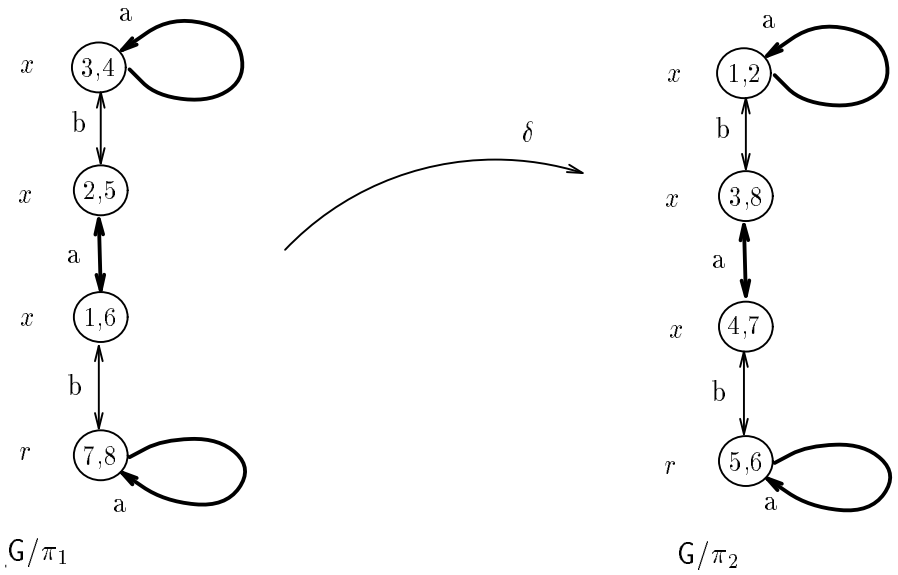


Figure 3.7: Example for Propostition 3.3.5 part 2.

$\delta : \mathsf{G}/\pi_{\vec{x}} \to \mathsf{G}/\pi_{\vec{y}}$ such that $\beta_2 = \delta\beta_1$ and show that δ is an isomorphism which maps $[i]$ to $[j]$.

Define a map $\delta : \mathsf{G}/\pi_{\vec{x}} \to \mathsf{G}/\pi_{\vec{y}}$ as follows: $\delta([i]) = [j]$ for $[i] \in \pi_{\vec{x}}$ and $[j] \in \pi_{\vec{y}}$. For any $f_b \in \mathcal{E}(\mathsf{G})$ for which b is reduced, let $\delta f_b([i]) = f_b\delta([i]) = f_b([j])$. Since G is connected, δ is defined on all blocks of $\pi_{\vec{x}}$. It is also well-defined: For suppose that $\delta([k]) = [l_1]$ and $\delta([k]) = [l_2]$ for $[k] \in \pi_{\vec{x}}$ and $[l_1]$ and $[l_2] \in \pi_{\vec{y}}$. Then there are elements f_a and $f_b \in \mathcal{E}(\mathsf{G})$ such that $f_a([i]) = f_b([i]) = [k] \in \pi_{\vec{x}}$ and $f_a([j]) = [l_1]$, and $f_b([j]) = [l_2] \in \pi_{\vec{y}}$. By Lemma 2.4.2 in Chapter 2, if $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ then $\mathrm{U}_{f_a(i)}\vec{x} \simeq \mathrm{U}_{f_a(j)}\vec{y}$ and $\mathrm{U}_{f_b(i)}\vec{x} \simeq \mathrm{U}_{f_b(j)}\vec{y}$ for f_a and f_b defined on i. Since $f_a([i]) = f_b([i])$, we have $\mathrm{U}_{f_a(i)}\vec{x} \simeq \mathrm{U}_{f_b(i)}\vec{x}$. By transitivity, then, $\mathrm{U}_{f_a(j)}\vec{y} \simeq \mathrm{U}_{f_b(j)}\vec{y}$ and so $f_a([j]) = f_b([j]) \in \pi_{\vec{y}}$. Thus $[l_1] = [l_2]$ and δ is well-defined. A similar argument shows that δ is one-to-one. If B is any block in $\pi_{\vec{y}}$ then since G is connected there is an element $f_a \in \mathcal{E}(\mathsf{G})$ such that $f_a([j]) = \mathsf{B}$. Then $\delta f_a([i]) = \mathsf{B}$, so δ is onto. That δ is a covering map follows from Proposition 3.3.1 above: For suppose that $f_b \in \mathcal{E}(\mathsf{G})$ is defined on a block $f_a([i]) \in \pi_{\vec{x}}$. Then f_bf_a is defined on $[i]$ and hence on $[j]$, since $\mathrm{U}_{f_bf_a(i)}\vec{x} \simeq \mathrm{U}_{f_bf_a(j)}\vec{y}$ by Lemma 2.4.2 in Chapter 2. Thus f_b is defined on $f_a([j]) \in \pi_{\vec{y}}$. The same argument shows that if f_b is defined on $f_a([j]) \in \pi_{\vec{y}}$ then it is defined on $f_a([i]) \in \pi_{\vec{x}}$. The map δ commutes with the elements of $\mathcal{E}(\mathsf{G})$ by construction, and so by the abovementioned proposition, δ is a covering map. Since $y_{\delta([l])} = x_{[l]}$ for $l = 1, \ldots, n$, we have $\delta(\vec{y}) = \vec{x}$. This proves (1).

(2) Let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism. Let $\vec{y}$ be such that $\pi_2 \preceq \pi_{\vec{y}}$, and write $\delta(\vec{y}) = \vec{x}$. Then:

1. $\pi_1 \preceq \pi_{\vec{x}}$: This will follow from Lemma 3.3.3 if $x_i = x_j$ whenever $[i] = [j] \in \pi_1$. Suppose that $[i] = [j]$ in π_1 Then the following three facts hold:

(a) Since δ is an isomorphism from G/π_1 to G/π_2, we have $[i] = [j] \in \pi_1$ iff $\delta([i]) = \delta([j])$.

(b) $\delta([i]) = \delta([j])$ implies that $y_{\delta([i])} = y_{\delta([j])}$.

(c) Because $\vec{x} = \delta(\vec{y})$, we have $y_{\delta([i])} = y_{\delta([j])}$ iff $x_i = x_j$.

Hence $x_i = x_j$ and $\pi_1 \preceq \pi_{\vec{x}}$.

2. $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ *whenever* $\delta([i]) = [j]$: Suppose that $\delta([i]) = [j]$. By Lemma 3.2.5, $\mathrm{U}_i \simeq \mathrm{U}_j$ via an isomorphism $\tilde{\delta}$ such that $\beta_2\tilde{\delta} = \delta\beta_1$ for β_1 and β_2 the canonical covering maps. Let $\tilde{v} \in \mathsf{V}(\mathrm{U}_i)$ and let $\tilde{\delta}(\tilde{v}) = \tilde{r}$. Then $\delta([v]) = [r]$ and so $x_v = y_{\delta([v])} = y_{[r]} = y_r$. Hence $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$.

□

3.4 Symmetry And Its Consequences

We now have the machinery for finding a set of network features which classify networks according to the functions they can compute. Let us consider again what these features might be. We have already seen that the graph automorphisms do not reliably distinguish between networks. Another plausible choice might be the elements of the edge-label monoid of a graph. However, this also fails to consistently classify networks. In Chapter 5 we will give an example of two networks having distinct edge-label monoids which nonethless can compute the same functions. It is true, however, that if two networks have identical (not just isomorphic) edge-label monoids, that the set of functions each can compute is the same. (See Remark 5.3.1 in Chapter 5.)

Perhaps two networks compute the same functions iff they have the same c-partitions? Unfortunately, this is also false, as Example 3.4.1 shows.

EXAMPLE 3.4.1: The graphs G_1 and G_2 in Figure 3.8 share the same c-partitions, namely, $\pi_1 = 1/2/3/4$, $\pi_2 = 1,2/3,4$, and $\pi_3 = /1,2,3,4/$. However, they do not have the same quotient-graph isomorphisms. G_1, for instance, has an automorphism $\delta = (1,2,3,4)$ which G_2 does not have. By Proposition 3.3.5 in the previous section, this implies (for instance) that in G_1, $\mathrm{U}_4\delta(\vec{x}) \simeq \mathrm{U}_1\vec{x}$ for any $\vec{x}$. This does not hold for G_2, so G_2 can compute a function that G_1 cannot compute.

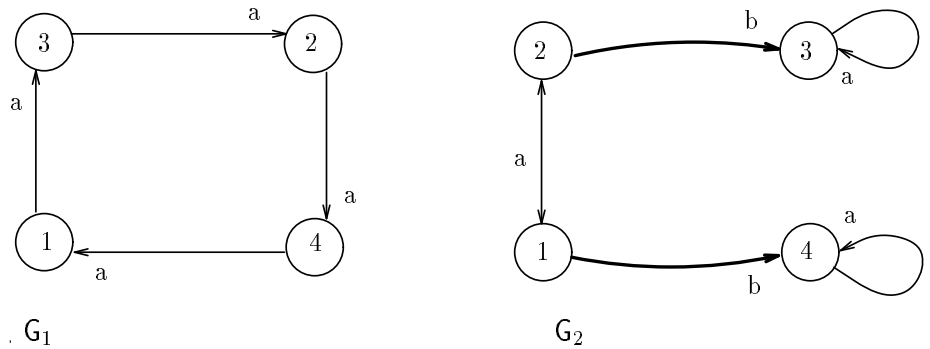


Figure 3.8: G_1 and G_2 have the same c-partitions but compute different functions.

In this section we will see that the set of distinguishing features or 'symmetries' – as we call them – of a graph consists of the c-partitions of the graph *and* its quotient-graph isomorphisms. This section's first theorem gives an alternate characterization of the functions computable by a network in terms of its "symmetries". The second theorem shows that the set of symmetries correctly classifies networks: Two networks compute the same functions iff they have the same symmetry set.

We define a 'network symmetry' to be a triple consisting of two c-partitions and the associated quotient-graph isomorphism. More formally, we have:

Definition 3.4.1: Let G be a graph, and let $\delta : \mathsf{G}/\pi_1 \rightarrow \mathsf{G}/\pi_2$ be an isomorphism of quotient-graphs G/π_1 and G/π_2. Then δ induces a bijection, which we also call δ, between the blocks of π_1 and π_2. We will call a triple $\langle \pi_1, \pi_2, \delta \rangle$ a *symmetry of* G if $\delta : \mathsf{G}/\pi_1 \rightarrow \mathsf{G}/\pi_2$ is an isomorphism. The *symmetry set* of a graph G is the set of symmetries of G.

We will show next that a network can compute a function f iff f "satisfies the symmetries" of the network. What should it mean for a function to 'satisfy a symmetry'? Consider the network G in Example 3.3.2. The triple $\langle \pi_1, \pi_2, \delta \rangle$ is a symmetry for G, where π_1, π_2 and δ are as given in the example. Let $\vec{y} = (x, x, y, z, r, r, z, y)$ and $\delta(\vec{y}) = (z, y, x, x, y, z, r, r)$ as in the example. By Theorem 2.5.1, f must satisfy $f(\delta(\vec{y}))_i = f(\vec{y})_{\delta([i])}$ for all $i = 1, \ldots, n$, since $\mathrm{U}_i \delta(\vec{y}) \simeq \mathrm{U}_{\delta([i])} \vec{y}$ for $i = 1, \ldots, n$. That is, $f(\delta(\vec{y})) = \delta f(\vec{y})$ for this $\vec{y}$ for which $\pi_2 = \pi_{\vec{y}}$. This motivates the following definition:

Definition 3.4.2: We will say that a function $f : \mathbf{I^n} \rightarrow \mathbf{O^n}$ *satisfies a symmetry* $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ if for all inputs $\vec{x}$ such that $\pi_2 \preceq \pi_{\vec{x}}$, we also have $\pi_2 \preceq \pi_{f(\vec{x})}$ and $f(\delta(\vec{x})) = \delta(f(\vec{x}))$.

The clause "$\pi_2 \preceq \pi_{f(\vec{x})}$" insures that $\delta(f(\vec{x}))$ is defined.

We have:

Theorem 3.4.1: *A network with graph* G *computes a function* f *iff* f *satisfies all symmetries in the symmetry-set of* G.

Proof Suppose first that f satisfies all symmetries of G. Let $i, j \in \mathsf{V}(\mathsf{G})$ and $\vec{x}, \vec{y} \in \mathbf{I}$ be such that $\mathrm{U}_i \vec{x} \simeq \mathrm{U}_j \vec{y}$. By part 1 of Proposition 3.3.5, there is a symmetry $\mathbf{s} = \langle \pi_{\vec{x}}, \pi_{\vec{y}}, \delta \rangle$ with $\vec{x} = \delta(\vec{y})$ and $[j] = \delta([i])$. Since f satisfies $\mathbf{s}$, we have $f(\delta(\vec{y})) = \delta f(\vec{y})$. That is, $f(\vec{x})_i = f(\vec{y})_j$, and so by Theorem 2.5.1 in Chapter 2, G computes f.

Conversely, suppose that G computes f. Let $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ be a symmetry of G and let $\vec{x}$ be an input-vector such that $\pi_2 \preceq \pi_{\vec{x}}$. If $[i] = [j]$ in π_2 then $\mathrm{U}_i \vec{x} \simeq \mathrm{U}_j \vec{x}$ (by definition of $\pi_{\vec{x}}$), and we have $f(\vec{x})_i = f(\vec{x})_j$, since G computes f (Theorem 2.5.1, Chapter 2.) By Lemma 3.3.3, this implies that $\pi_2 \preceq \pi_{f(\vec{x})}$. To complete the proof that f satisfies $\mathbf{s}$ we need to show that $f(\delta(\vec{x})) = \delta f(\vec{x})$, i.e., that $f(\delta(\vec{x}))_i = f(\vec{x})_{\delta([i])}$ for $i = 1, \ldots, n$. Since $\pi_2 \preceq \pi_{\vec{x}}$ we have by Proposition 3.3.5 part 2 that $\mathrm{U}_i \delta(\vec{x}) \simeq \mathrm{U}_{\delta([i])} \vec{x}$. Then by Theorem 2.5.1, f must satisfy $f(\delta(\vec{x}))_i = f(\vec{x})_{\delta([i])}$, since G computes f. Thus f satisfies $\mathbf{s}$. □

The next lemma will be used to prove Theorem 3.4.2.

Notation: If G_1 is a network, we will write π^1 for a c-partition of G_1, and i_1 or j_1 will denote vertices in G_1. We will denote a c-partition $\pi_{\vec{x}}$ of G_1 by $\pi^1_{\vec{x}}$.

Lemma 3.4.1: *Let* G_1 *and* G_2 *be networks with* $|\mathsf{V}(\mathsf{G}_1)| = |\mathsf{V}(\mathsf{G}_2)|$. *Let* π^1 *and* π^2 *be c-partitions of* G_1 *and* G_2, *respectively, such that* $\pi^1 = \pi^2$. *If* $\pi^1 \preceq \pi^1_{\vec{x}}$ *for some* $\vec{x}$ *then* $\pi^2 \preceq \pi^2_{\vec{x}}$.

Proof If i_2 and j_2 are in the same block of π^2 then i_1 and j_1 are in the same block of π^1 since $\pi^1 = \pi^2$. If i_1 and j_1 are in the same block of π^1 then $\mathrm{U}_{i_1}\vec{x} \simeq \mathrm{U}_{j_1}\vec{x}$ (since $\pi^1 \preceq \pi^1_{\vec{x}}$), and so $x_i = x_j$. Hence if $[i_2] = [j_2] \in \pi^2$ then $x_i = x_j$. Thus by Lemma 3.3.3, $\pi^2 \preceq \pi^2_{\vec{x}}$. □

Theorem 3.4.2: *Let* G_1 *and* G_2 *be networks and suppose that* $|\mathsf{V}(\mathsf{G}_1)| = |\mathsf{V}(\mathsf{G}_2)|$. *Then the sets of functions which* G_1 *and* G_2 *can compute are equal iff their symmetry-sets are identical.*

Proof Let G_1 and G_2 have symmetry-set S_1 and S_2, respectively.

($\Longleftarrow$) Suppose first that $\mathsf{S}_1 = \mathsf{S}_2$ and that G_1 computes a function f. We will show that G_2 also computes f, by showing that f satisfies all of the symmetries in S_2. Let $\mathbf{s}_1 = \langle \pi^1_a, \pi^1_b, \delta \rangle \in \mathsf{S}_1$. We show that f satisfies $\mathbf{s}_2 = \langle \pi^2_a, \pi^2_b, \delta \rangle \in \mathsf{S}_2$, where $\mathbf{s}_1 = \mathbf{s}_2$. Let $\vec{x}$ be an input-vector such that $\pi^2_b \preceq \pi^2_{\vec{x}}$. We need to show that $\pi^2_b \preceq \pi^2_{f(\vec{x})}$ and that $f(\delta(\vec{x})) = \delta f(\vec{x})$. Since $\pi^1_b = \pi^2_b$, we have $\pi^1_b \preceq \pi^1_{\vec{x}}$, by Lemma 3.4.1. By Proposition 3.3.3 there is a symmetry $\mathbf{s} = \langle \pi^1_c, \pi^1_{\vec{x}}, \delta' \rangle \in \mathsf{S}_1$, where $\delta' = \beta_2 \delta {\beta_1}^{-1}$ for $\beta_1 : \mathsf{G}_1/\pi^1_a \to \mathsf{G}_1/\pi^1_c$ and $\beta_2 : \mathsf{G}_1/\pi^1_b \to \mathsf{G}_1/\pi^1_{\vec{x}}$ the inclusion covering maps. Since f satisfies all of the symmetries in S_1, it satisfies $\mathbf{s}$, so $\pi^1_{\vec{x}} \preceq \pi^1_{f(\vec{x})}$ and $\delta' f(\vec{x}) = f(\delta'(\vec{x}))$. Since $\pi^1_b \preceq \pi^1_{\vec{x}}$, we have $\pi^1_b \preceq \pi^1_{f(\vec{x})}$. Since $\pi^1_b = \pi^2_b$, we have, again by Lemma 3.4.1, that $\pi^2_b \preceq \pi^2_{f(\vec{x})}$. That $f(\delta(\vec{x})) = \delta f(\vec{x})$ follows from the easily-verified fact that $\delta(\vec{y}) = \delta'(\vec{y})$ for all $\vec{y}$ on which δ' is defined. In particular, $\delta(\vec{x}) = \delta'(\vec{x})$ and $\delta f(\vec{x}) = \delta' f(\vec{x})$, and $f(\delta'(\vec{x})) = \delta' f(\vec{x})$ since f satisfies $\mathbf{s}$. Hence f satisfies $\mathbf{s}_2$. Since $\mathsf{S}_1 = \mathsf{S}_2$, this shows that f satisfes all symmetries in S_2 and thus, by Theorem 3.4.1, that G_2 computes f. The same argument shows that if G_2 can compute a function f, then G_1 can compute it.

($\Longrightarrow$) Conversely, suppose that G_1 and G_2 compute the same functions. We will show that $\mathsf{S}_1 = \mathsf{S}_2$, where S_1 and S_2 are the symmetry-sets of G_1 and G_2, respectively.

Suppose on the contrary that G_2 has a symmetry $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ that G_1 does not have. We will construct a function f which G_1 can compute but G_2 cannot. Now $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle \notin \mathsf{S}_1$ means that either (1) π_1 or π_2 are not c-partitions of G_1, or (2) δ is not an isomorphism from G_1/π_1 to G_1/π_2. Let us consider these two cases separately.

Case 1: Suppose that π_1 is not a c-partition of G_1. Then there is a block $\mathsf{B} \in \pi_1$, $g_a \in \mathcal{E}(\mathsf{G}_1)$, and i and $j \in \mathsf{B}$ such that either $g_a(i)$ and $g_a(j)$ are in different blocks of π_1, or g_a is defined on one but not both of $\{i, j\}$. Define a function f by $f(\vec{x})_l = x_{g_a(l)}$ for $l \in \{1, \ldots, n\}$ on which g_a is defined, and $f(\vec{x})_l = y \notin \{x_1, \ldots, x_n\}$ for l on which g_a is undefined. Then if $\vec{x}' \in \mathbf{I^n}$ is such that $x'_r = x'_s$ iff r and s are in the same block of π_1, then $f(\vec{x}')_i \neq f(\vec{x}')_j$ for i, j as above. Note that f is computable by G_1 by construction, since each processor l can request $x_{g_a(l)}$ from processor $g_a(l)$ if such exists, or output 'y' if g_a is not defined on l. However, f is not computable by G_2: $\mathrm{U}_i\vec{x}' \simeq \mathrm{U}_j\vec{x}'$ for i, j, and $\vec{x}'$ as above, and by Theorem 2.5.1 G_2 computes f only if $f(\vec{x}')_i = f(\vec{x}')_j$.

Case 2: Suppose now that π_1 and π_2 are c-partitions of G_1 but that δ is not an isomorphism from G_1/π_1 to G_1/π_2. Then there is a block $\mathsf{B} \in \pi_1$ and an element $g_a \in \mathcal{E}(\mathsf{G}_1)$ such that either g_a is defined on B but not on $\delta(\mathsf{B})$, or vice-versa, or $\delta g_a(\mathsf{B}) \neq g_a \delta(\mathsf{B})$. Define f as above in Case 1. Let $\vec{x}' \in \mathbf{I^n}$ be such that $x_r = x_s$ iff r and s are in the same block of π_2. Then $\delta f(\vec{x}') \neq f\delta(\vec{x}')$, since for each $i \in \{1, \ldots, n\}$ we have:

$$\delta f(\vec{x}')_i = \begin{cases} x'_{\delta([g_a(i)])} & \text{for } i \text{ on which } g_a \text{ is defined} \\ y & \text{otherwise} \end{cases}$$

$$f(\delta(\vec{x}'))_i = \begin{cases} x'_{g_a\delta([i])} & \text{for } \delta([i]) \text{ on which } g_a \text{ is defined} \\ y & \text{otherwise} \end{cases}$$

Thus there exists $i \in \mathsf{B}$ such that $x'_{\delta([g_a(i)])} \neq x'_{g_a\delta([i])}$, by construction.

By construction, G_1 can compute f; but by Theorem 2.5.1 in Chapter 2, G_2 cannot. □

3.5 Networks Differing By A Permutation

Consider the graphs G_1 and G_2 in Figure 3.9 below.

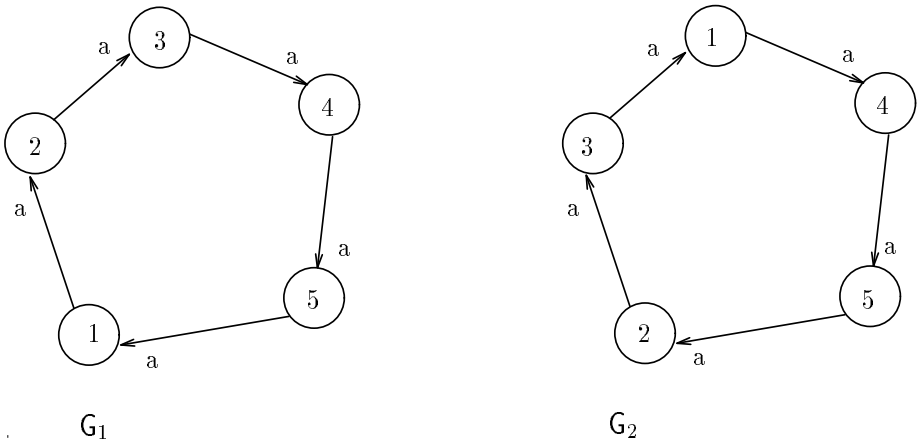


Figure 3.9: Networks differing by a permutation

G_2 is identical to G_1, except that its first three vertex-labels have been cyclically permuted. It would be reasonable to expect that G_1 and G_2 would compute the same set of functions. However, due to the way we have defined function-computation on a network, they only 'almost' compute the same functions. For instance, to be computable by G_1, a function f must satisfy $f(\delta_1(\vec{x})) = \delta_1 f(\vec{x})$ for all $\vec{x} \in \mathbf{I^n}$ and for $\delta_1 = (1,2,3,4,5)$. A function computable by G_2, on the other hand, instead satisfies $f(\delta_2(\vec{x})) = \delta_2 f(\vec{x})$ for $\delta_2 = (1,4,5,2,3)$.

In this section we will make precise this artifact of our model. We will show that two networks have symmetry-sets "differing by a permutation" iff the functions they compute differ by the same permutation.

Notation: Let ρ be a permutation of $\{1,\ldots,n\}$. Then ρ induces a map, which we also call ρ, on the set $\mathbf{I^n}$ of input-vectors for G, by $\rho(x_1,\ldots,x_n) = (x_{\rho(1)},\ldots,x_{\rho(n)})$. That is, $\rho(\vec{x})_i = \vec{x}_{\rho(i)}$. Let f_1 and f_2 be functions defined on $\mathbf{I^n}$. We will say that f_2 *differs from* f_1 *by a permutation* ρ if $\rho f_1((\vec{x})) = f_2(\vec{x})$ for all $\vec{x} \in \mathbf{I^n}$.

If G is a graph and ρ is a permutation of $\{1,\ldots,n\}$, write $\rho\mathsf{G}$ for G with its vertices relabeled by ρ. That is, $\rho(v) \in \mathsf{V}(\rho\mathsf{G})$ iff $v \in \mathsf{V}(\mathsf{G})$, and $\langle\rho(v)a\rho(w)\rangle \in \mathsf{E}(\rho\mathsf{G})$ iff $\langle v\,a\,w\rangle \in \mathsf{E}(\mathsf{G})$.

Let us look again at the set of functions graphs G and $\rho\mathsf{G}$ can compute. If G is given input $\vec{x}$ and $\rho\mathsf{G}$ is given input $\rho^{-1}(\vec{x})$, then the vertex called "i" in $\rho\mathsf{G}$ is called "$\rho^{-1}(i)$" in G, and both vertices receive the same input-value. In $\rho\mathsf{G}$, processor i gets input $x_{\rho^{-1}(i)}$ and computes $f(\vec{x})_{\rho^{-1}(i)}$. In G, processor $\rho^{-1}(i)$ also gets input $x_{\rho^{-1}(i)}$ and computes $f(\vec{x})_{\rho^{-1}(i)}$. Thus:

Remark 3.5.1: *Let* G *be a network,* $\vec{x}$ *an input-vector,* ρ *a permutation of* $\{1,\ldots,n\}$ *and* f *a function from* $\mathbf{I^n}$ *to* $\mathbf{O^n}$. *Then* G *computes* $f(\vec{x})$ *when given input* $\vec{x}$ *iff* $\rho\mathsf{G}$ *computes* $\rho^{-1}f(\vec{x})$, *given input* $\rho^{-1}(\vec{x})$. *That is,* G *computes a function* f *iff* $\rho\mathsf{G}$ *computes* $\rho^{-1}f\rho$.

□

This suggests the following definition:

Definition 3.5.1: Let F_1 and F_2 be two sets of functions: $\mathbf{I^n} \to \mathbf{O^n}$. Let ρ be a permutation of $\{1,\ldots,n\}$ and write $\rho(x_1,\ldots,x_n) = (x_{\rho(1)},\ldots,x_{\rho(n)})$ as before. We will say that F_1 *differs by* ρ *from* F_2 if ρ induces a bijection: $f \to \rho^{-1}f\rho$ from F_1 to F_2.

Remark 3.5.1 then gives us the following:

Remark 3.5.2: *The set of functions computable by a network* G *differs by* ρ *from the set of functions computable by* $\rho\mathsf{G}$, *for any permutation* ρ *of* $\{1,\ldots,n\}$.

□

Notation: Let π be a partition of $\{1,\ldots,n\}$ and let $\rho : \{1,\ldots,n\} \to \{1,\ldots,n\}$ be a permutation. Write $\rho(\pi)$ for the partition of $\{1,\ldots,n\}$ satisfying: If $[i] = [j] \in \pi$ then $[\rho(i)] = [\rho(j)]$ in $\rho\pi$. If $\delta : \pi_1 \to \pi_2$ is a bijection from the blocks of π_1 to the blocks of π_2, we will write $\rho\delta$ for the bijection from $\rho(\pi_1)$ to $\rho(\pi_2)$ which takes each block $\rho(\mathsf{B})$ of $\rho(\pi_1)$ to a block $\rho(\delta(\mathsf{B}))$ of $\rho(\pi_2)$.

EXAMPLE 3.5.1: If $\pi_1 = 1,2,3/4,5,6/7,8,9$ and $\rho = (1,4,5,8)(3,7,9)$
then $\rho(\pi_1) = 2,4,7/5,6,8/1,3,9$ because $\rho(1) = 4$, $\rho(2) = 2$, $\rho(3) = 7$, and so on.
If $\pi_2 = 1,4,7/2,5,9/3,6,8$ then $\rho(\pi_2) = 4,5,9/2,3,8/1,6,7$
If $\delta : \{1,2,3\} \to \{1,4,7\}$; $\{4,5,6\} \to \{2,5,9\}$, and $\{7,8,9\} \to \{3,6,8\}$, then
$\rho\delta : \{2,4,7\} \to \{4,5,9\}$; $\{5,6,8\} \to \{2,3,8\}$, and $\{1,3,9\} \to \{1,6,7\}$.

Notation: Suppose that networks G_1 and G_2 have symmetry-sets S_1 and S_2, respectively. We will write $S_2 = \rho S_1$ if ρ induces a bijection from S_1 to S_2 such that any triple $\langle\pi_1,\pi_2,\delta\rangle$ is a symmetry in S_1 iff $\langle\rho(\pi_1),\rho(\pi_2),\rho\circ\delta\rangle$ is a symmetry in S_2.

By constuction, we have:

Remark 3.5.3: *A triple* $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ *is a symmetry of* G *iff* $\rho S = \langle \rho\pi_1, \rho\pi_2, \rho\delta \rangle$ *is a symmetry of* $\rho\mathsf{G}$.

□

Finally we can show:

Theorem 3.5.1: *Let* G_1 *and* G_2 *be networks with symmetry-sets* S_1 *and* S_2*, respectively. Then* $S_2 = \rho S_1$ *for some permutation* ρ *iff the set of functions computable by* G_1 *differs by* ρ *from the set of functions computable by* G_2.

Proof By Remark 3.5.3, $\rho\mathsf{G}_1$ has symmetry-set ρS_1. Suppose first that $S_2 = \rho S_1$. Then by Theorem 3.4.1, $\rho\mathsf{G}_1$ and G_2 compute the same set of functions. By Remark 3.5.2, the set of functions computed by G_1 differs by ρ from the set of functions computed by $\rho\mathsf{G}_1$, and the conclusion follows. Conversely, suppose that the functions computable by G_1 differ by ρ from those computable by G_2. Then G_2 and $\rho\mathsf{G}_1$ compute the same set of functions, so $S_2 = \rho S_1$. □

3.6 Related Work

A generalization of our "correct partition" appears in algebraic automata theory under the name "admissible partition". (See [Hol82]). In our notation, an *admissible relation* of the states of a finite-state machine is an equivalence-relation $\sim$ on the vertices of an edge-labeled, directed graph G, satisfying: For all words a over $\mathsf{A}(\mathsf{G})$, if $v \sim w$ and if f_a is defined on v and w then $f_a(v) \sim f_a(w)$. A partition of $\mathsf{V}(\mathsf{G})$ is *admissible* if the associated equivalence-relation is admissible. Note that all congruence-relations (as we define them) are admissible, but not all admissible relations are congruences. There is a well-defined quotient of G associated with each admissible partition.

4. Group Graphs

4.1 Introduction

In the last chapter we described a classification for networks, such that two networks are in the same class iff the set of functions each can compute is the same. We found a set of network features — the symmetries — having the property that each equivalence-class of networks is uniquely specified by a set of symmetries. Our task in the remaining three chapters is to show that classifying networks is *easy*. In Chapter 5 we will show that classifying "group graphs" is easy, where a group graph is a graph whose edge-label monoid is a group. In Chapter 6, we will show that classifying graphs with arbitrary edge-label monoids is also easy. To show that classification is easy, we will find a small, easily-computed subset of the set of symmetries of a graph which generates the whole symmetry-set under certain operations. This subset can be used in place of the symmetry-set to characterize an equivalence-class of graphs.

Chapter Summary and Main Results: This chapter develops the necessary group-and-graph background for the classification effort. Two propositions from this chapter will be used in Chapter 5: Proposition 4.6.5, which gives a correspondence between subgroup conjugacy and quotient-graph isomorphism, and Proposition 4.6.6, which gives a one-to-one correspondence between the symmetries of a graph G and certain cosets of subgroups of $\mathcal{E}(\mathsf{G})$. Most of the rest of the chapter is preparation for these results.

In Section 4.2, we will review some facts about the block-systems of permutation groups. In Section 4.3, we review the idea of a group acting on a set, and in Section 4.4, use these group actions to construct graphs – "operator graphs", of which group-graphs are an instance. In Section 4.5 we will show that the set of graphs covered by a group-graph G forms a lattice isomorphic with a sublattice of the lattice of subgroups of $\mathcal{E}(\mathsf{G})$. Finally, in Section 4.6, we will find the desired relationships between graph ismorphism and subgroup conjugacy and between cosets and symmetries.

4.2 Block Systems

In this section we will review some basic facts about "block-systems" of permutation groups: If $\mathcal{G}$ is a transitive permutation group on a set S, a block-system is a partition of S which is preserved by $\mathcal{G}$. We will find a close correspondence between block-systems and subgroups, and between the cosets of a subgroup and the blocks of a block-system.

The results in this section are well-known; see [Jac74], [Rob82], [Sco87], and [Wei64].

Notation:

1. Write $(\mathcal{G}, S)$ for a permutation group $\mathcal{G}$ on a finite set S.
2. Write $Sym(S)$ for the symmetric group on S.
3. Write $\mathcal{G}_v$ for the stabilizer subgroup of $\mathcal{G}$ of the point $v \in \mathsf{S}$. That is, $\mathcal{G}_v = \{g \in \mathcal{G} : g(v) = v\}$.

Lemma 4.2.1: *([Jac74], page 74) If $(\mathcal{G}, S)$ is a transitive permutation group then $\mathcal{G}_v$ and $\mathcal{G}_w$ are conjugate subgroups for any $v, w \in \mathsf{S}$. In particular, if $g(v) = w$ for $g \in \mathcal{G}$ and v, $w \in \mathsf{S}$ then $\mathcal{G}_w = g\mathcal{G}_v g^{-1}$.*

Definition 4.2.1: Let $(\mathcal{G}, \mathsf{S})$ be a transitive permutation group. A subset B of S is called a *block*, if for all $g \in \mathcal{G}$, either $g\mathsf{B} = \mathsf{B}$ or $\mathsf{B} \cap g\mathsf{B} = \emptyset$.

In the sequel we will assume that all permutation groups are transitive, unless stated otherwise.

Weilandt proves the following about blocks:

Proposition 4.2.1: *(Proposition 6.2 in [Wei64])*

If B is a block of a transitive permutation group $\mathcal{G}$, then $g\mathsf{B}$ is a block of $\mathcal{G}$, for all $g \in \mathcal{G}$.

We also have:

Proposition 4.2.2: *(Proposition 6.3 in [Wei64])*

If B is a block of $\mathcal{G}$ then $|\mathsf{B}|$ divides $|\mathsf{S}|$.

Definition 4.2.2: A *block-system* or *system of imprimitivity* for a transitive permutation group $\mathcal{G}$ is a set π of blocks such that S is the disjoint union of the blocks in π, and if $\mathsf{B} \in \pi$ and $g \in \mathcal{G}$, then $g\mathsf{B} \in \pi$.

EXAMPLE 4.2.1: If G is a connected graph for which $\mathcal{E}(\mathsf{G})$ is a group, then the block-systems of $\mathcal{E}(\mathsf{G})$ are the c-partitions of G.

Proposition 4.2.3: *(Proposition 10.5.4 in [Sco87]) Let B be a block and π a collection of blocks containing B. Then π is a block-system iff π is the set of all distinct blocks $\{g\mathsf{B} : g \in \mathcal{G}\}$.*

Note that:

Remark 4.2.1: *A block-system is uniquely determined by a single block.*

Notation: If $\mathcal{H} \leq \mathcal{G}$ and $v \in \mathsf{S}$, we will write $\mathcal{H}(v)$ for the set $\{h(v) : h \in \mathcal{H}\}$.

If we fix an element $v \in S$, there is a one-to-one correspondence between the blocks of $\mathcal{G}$ containing v and the subgroups of $\mathcal{G}$ containing the stabilizer subgroup $\mathcal{G}_v$:

Proposition 4.2.4: *(Proposition 10.5.6 in [Sco87]) There is a bijection T from the set of all subgroups of $\mathcal{G}$ containing $\mathcal{G}_v$ onto the set of all blocks of $\mathcal{G}$ containing v. T is given by: $T(\mathcal{H}) = \mathcal{H}(v)$.*

Corollary 4.2.1: *Let B be a block, $v \in \mathsf{B}$ and let $\mathcal{H}$ be a subgroup such that $\mathcal{G}_v \leq \mathcal{H}$ and $\mathcal{H}(v) = \mathsf{B}$. Then $\mathcal{H} = \{g \in \mathcal{G} : g\mathsf{B} = \mathsf{B}\}$.*

Proof Let $\mathcal{J} = \{g \in \mathcal{G} : g(v) \in \mathsf{B}\}$. It is easy to verify that $\mathcal{J}$ is a subgroup of $\mathcal{G}$. Note that $\mathcal{G}_v \leq \mathcal{J}$. We claim that $\mathcal{J} = \mathcal{H}$. First, $\mathcal{H} \leq \mathcal{J}$, and so $\mathcal{H}(v) \subseteq \mathcal{J}(v)$. Since $\mathcal{J}(v) \subseteq \mathcal{H}(v)$, the blocks $\mathcal{H}(v)$ and $\mathcal{J}(v)$ are equal and so by Proposition 4.2.4, $\mathcal{H} = \mathcal{J}$. The conclusion will follow if the sets $\{g : g\mathsf{B} = \mathsf{B}\}$ and $\{g : g(v) \in \mathsf{B}\} = \mathcal{H}$ are equal.

Let $g \in \mathcal{G}$. If $g\mathsf{B} = \mathsf{B}$ then $g(v) \in \mathsf{B}$, so $\{g : g\mathsf{B} = \mathsf{B}\} \subseteq \mathcal{H}$. Conversely, if $h \in \mathcal{H}$, then $h(v) \in \mathsf{B}$ and so $h\mathsf{B} = \mathsf{B}$. Thus $\mathcal{H} \subseteq \{g \in \mathcal{G} : g\mathsf{B} = \mathsf{B}\}$. □

Definition 4.2.3: A permutation group $\mathcal{G}$ on a set S is called *regular* if it is transitive and fixed-point free; that is, if it satisfies:

- For all $v, w \in S$ there is an element $g \in \mathcal{G}$ such that $g(v) = w$
- If $g(v) = v$ for some $v \in S$ then g is the identity element in $\mathcal{G}$.

If $\mathcal{G}$ is regular and $|S| = n$ then $\mathcal{G}$ has order n also: For fix $v \in S$. Then for every $w \in S$ there is a unique element of $\mathcal{G}$ which maps v to w.

Note also that if $(\mathcal{G}, s)$ is regular, then $\mathcal{G}_v$ is always trivial for any $v \in S$, since $\mathcal{G}$ is fixed-point free. Hence if $(\mathcal{G}, S)$ is regular then the blocks of S containing v are in one-to-one correspondence with the subgroups of $\mathcal{G}$, by Proposition 4.2.4.

Since each block of a permutation group $(\mathcal{G}, S)$ uniquely determines a block-system, Proposition 4.2.4 gives a one-to-one correspondence between the subgroups of $\mathcal{G}$ containing $\mathcal{G}_v$ and the set of block-systems of $\mathcal{G}$, with a subgroup $\mathcal{H}$ corresponding to the block-system containing $\mathcal{H}(v)$. More formally, we have:

Definition 4.2.4: Fix $v \in S$. We will call a block-system π the *block-system corresponding to subgroup $\mathcal{H}$ with respect to* v, and subgroup $\mathcal{H}$ is called the *subgroup corresponding to π with respect to v* if $\mathcal{G}_v \leq \mathcal{H}$ and $\mathcal{H}(v)$ is a block in π.

EXAMPLE 4.2.2: Let $\mathcal{G}$ be the dihedral group D_8 on the set $\{1, \ldots, 8\}$ and generated by:

$f_2 = (1,2)(3,4)(5,6)(7,8)$ and

$f_8 = (1,8)(2,3)(4,5)(6,7)$.

(Refer to Example 4.5.1 for the whole group.)

Note that $\pi = 1, 2/3, 8/4, 7/5, 6$ is a block-system of $\mathcal{G}$. Then π corresponds to the subgroup $\mathcal{H}$ generated by f_2 with respect to any of the points $1, 2, 5$, or 6, since, for instance, $\langle f_2 \rangle(1) = \{1, 2\} \in \pi$; $\langle f_2 \rangle(5) = \{5, 6\} \in \pi$, and so on.

If we fix a different point v in S, we have the following:

Proposition 4.2.5: *Let $(\mathcal{G}, S)$ be a transitive permutation group. Let $v, w \in S$, let π be a block-system for $\mathcal{G}$ and let $\mathcal{H}$ be the subgroup of $\mathcal{G}$ corresponding to π with respect to v. Then the unique subgroup $\mathcal{J}$ corresponding to π with respect to w is $\mathcal{J} = g\mathcal{H}g^{-1}$, where $g(v) = w$.*

Proof Suppose that $\mathcal{J}$ corresponds to π with respect to w. Choose $g \in \mathcal{G}$ such that $g(v) = w$. Then $g\mathcal{H}g^{-1}(w) = g\mathcal{H}(v) = \mathcal{J}(w) \in \pi$, so by Proposition 4.2.4, $\mathcal{J} = g\mathcal{H}g^{-1}$. □

We also have the following correspondence:

Proposition 4.2.6: *Let $\mathcal{G}_v \leq \mathcal{H} \leq \mathcal{G}$. There is a one-to-one correspondence between the left cosets of $\mathcal{H}$ and the blocks in the corresponding block-system, given by: $g\mathcal{H} \to g\mathcal{H}(v)$.*

Proof Let $\rho : g\mathcal{H} \to g\mathcal{H}(v)$. Note first that $\{g\mathcal{H}(v) : g \in \mathcal{G}\}$ is a block-system: By Proposition 4.2.4, $\mathcal{H}(v)$ is a block; say, $\mathcal{H}(v) = \mathsf{B}$. By Proposition 4.2.3, $\{g\mathsf{B} : g \in \mathcal{G}\}$ is a block-system, and so ρ is onto. We show that ρ is well-defined and one-to-one. We have $g_1\mathsf{B} = g_2\mathsf{B}$ iff $g_2^{-1}g_1\mathsf{B} = \mathsf{B}$. By Corollary 4.2.1, $\mathcal{H} = \{g : g\mathsf{B} = \mathsf{B}\}$, so $g_2^{-1}g_1\mathsf{B} = \mathsf{B}$ iff $g_2g_1^{-1} \in \mathcal{H}$ or $g_1\mathcal{H} = g_2\mathcal{H}$. Hence ρ is well-defined and one-to-one. Thus ρ is a bijection, as claimed. □

Corollary 4.2.2: *Let $\mathcal{G}_v \leq \mathcal{H} \leq \mathcal{G}$. Then for any left coset $g'\mathcal{H}$ of $\mathcal{H}$, we have $g'\mathcal{H} = \{g \in \mathcal{G} : g\mathcal{H}(v) = g'\mathcal{H}(v)\}$.*

Proof Let $S = \{g : g\mathcal{H}(v) = g'\mathcal{H}(v)\}$. If $h \in g'\mathcal{H}$ then $h\mathcal{H} = g'\mathcal{H}$ and so $h\mathcal{H}(v) = g'\mathcal{H}(v)$, by Proposition 4.2.6, and so $h \in S$. Thus $g'\mathcal{H} \subseteq S$. Conversely, if $g \in S$ then by Proposition 4.2.6 we have $g\mathcal{H} = g'\mathcal{H}$. Thus $S \subseteq g'\mathcal{H}$. □

In the next section we will see that $\mathcal{G}$ permutes the blocks of any block-system in the same way that it permutes the left cosets of the corresponding subgroup.

4.3 Group Actions

Perhaps the most natural way to think of the edge-label group of a graph G is as a group acting on the vertices of G, where any $f_a \in \mathcal{E}(\mathsf{G})$ acts on the vertices by permuting them. In this section we will review the notion of a group action and define what it means for two actions to be "equivalent". The aim is to formalize the idea of a group action for use in the next section, where we will construct graphs from groups and actions. Again, the material in this section is review. See [Jac74] for a description of action and equivalence and [Mas67] for a definition of equivariant maps.

Definition 4.3.1: An *action* of a group $\mathcal{G}$ on a set S is a mapping $T : \mathcal{G} \times S \to S$ satisfying:

(1) $T(1, s) = s$ for all $s \in S$ (where 1 is the identity of $\mathcal{G}$)

(2) $T(g_1 g_2, s) = T(g_1, T(g_2, s))$ for all g_1 and $g_2 \in \mathcal{G}$ and $s \in S$.

An action T on $\mathcal{G} \times S$ is said to be *transitive*, and $\mathcal{G}$ is said to *act transitively on* S if for any two points s_1 and $s_2 \in S$ there is an element $g \in \mathcal{G}$ such that $T(g, s_1) = s_2$.

Unless otherwise stated we will assume that $\mathcal{G}$ and S are finite and that $\mathcal{G}$ acts transitively on S.

EXAMPLE 4.3.1: (Action By Left Or Right Multiplication)

Let $S = \mathcal{G}$ and let $T_l : \mathcal{G} \times S \to S$ be defined by: $T_l(g, h) = gh$. It is easy to see that T_l is an action. It is called the *action of $\mathcal{G}$ on itself by left multiplication.* If we let $S = \mathcal{G}$ and define $T_r : \mathcal{G} \times S \to S$ by: $T_r(g, h) = hg^{-1}$, then T_r is an action, called the *action of $\mathcal{G}$ on itself by right multiplication.*[1]

If we fix a group $\mathcal{G}$ and a set S, there is a one-to-one correspondence between the set of group actions $T : \mathcal{G} \times S \to S$ and the set of group homomorphisms $\rho_T : \mathcal{G} \to Sym(S)$ (See Proposition 1.6.5 in [Rob82]). This is given as follows: If T is an action: $\mathcal{G} \times S \to S$, then the corresponding homomorphism ρ_T maps each group element $g \in \mathcal{G}$ to the permutation $s \to T(g, s)$. That is, $\rho_T(g)(s) = T(g, s)$. This gives us the following definition:

Definition 4.3.2: We call the group $\rho_T(\mathcal{G}) \leq Sym(S)$ the *permutation group corresponding to the action T.*

EXAMPLE 4.3.2: Let $\mathcal{G} = \mathbf{Z}_4 = \{1, \alpha, \alpha^2, \alpha^3\}$ act on $S = \{a, b\}$ by $T(1, a) = T(\alpha^2, a) = a$; $T(1, b) = T(\alpha^2, b) = b$, and $T(\alpha, a) = T(\alpha^3, a) = b$, and $T(\alpha, b) = T(\alpha^3, b) = a$. Then the permutation group $\rho_T(\mathcal{G})$ is $\mathbf{Z}_2$.

[1] If $T_r(g, h) = hg$ instead of hg^{-1} then T_r is not an action, since it fails condition (2) of the definition.

Suppose that $\mathcal{G} = \mathbf{Z}_3 = \{1, \alpha, \alpha^2\}$ and that $\mathcal{G}$ acts on a set $S_1 = \{a, b, c\}$ by permuting it: $T_1(\alpha, a) = b; T_1(\alpha, b) = c$, and $T_1(\alpha, c) = a$. Suppose that $\mathcal{G}$ acts on a set $S_2 = \{1, 2, 3\}$ "in the same way" as it acts on S_1; i.e., $T_2(\alpha, 1) = 2; T_2(\alpha, 2) = 3$; and $T_2(\alpha, 3) = 1$. Then the actions T_1 and T_2 are identical up to a bijection from S_1 to S_2. It will be convenient to have a formalization of the notion of a group "acting in the same way" on two sets. Following is such a formalization:

Definition 4.3.3: Let $T_1 : \mathcal{G} \times S_1 \to S_1$ and $T_2 : \mathcal{G} \times S_2 \to S_2$ be actions. A surjective map $\rho : S_1 \to S_2$ is called a $\mathcal{G}-$*equivariant*[2] *map from* T_1 *to* T_2 if ρ commutes with T_1 and T_2, that is, if $\rho T_1(g, s) = T_2(g, \rho(s))$ for all $g \in \mathcal{G}$ and $s \in S_1$. The actions T_1 and T_2 are called *equivalent actions* and ρ is called an *equivalence* if ρ is a bijective $\mathcal{G}-$equivariant map from T_1 to T_2.

We have:

Proposition 4.3.1: *Let $T_1 : \mathcal{G} \times S_1 \to S_1$ and $T_2 : \mathcal{G} \times S_2 \to S_2$ be actions. If there is a $\mathcal{G}$-equivariant map ρ from T_1 to T_2 then there is a group epimorphism from $\mathcal{K}$ to $\mathcal{H}$, where $\mathcal{K}$ and $\mathcal{H}$ are the permutation groups corresponding to T_1 and T_2, respectively.*

Proof We will use the following fact, which derives immediately from the second isomorphism theorem for groups:

(*) *Let $\mathcal{G}, \mathcal{H}$ and $\mathcal{K}$ be groups, and $\rho_1 : \mathcal{G} \to \mathcal{K}$ and $\rho_2 : \mathcal{G} \to \mathcal{H}$ epimorphisms such that $ker(\rho_1) \leq ker(\rho_2)$. Then there is an epimorphism from $\mathcal{K}$ to $\mathcal{H}$.*

Proof of (*): We want an epimorphism: $\mathcal{G}/ker\rho_1 \to \mathcal{G}/ker\rho_2$, because $\mathcal{K} \simeq \mathcal{G}/ker\rho_1$ and $\mathcal{H} \simeq \mathcal{G}/ker\rho_2$. By the second isomorphism theorem, $\mathcal{G}/ker\rho_2 \simeq (\mathcal{G}/ker\rho_1)/(ker\rho_2/ker\rho_1)$.

By (*) it suffices to show that $ker(\rho_{T_1}) \leq ker(\rho_{T_2})$, where $\rho_{T_1} : \mathcal{G} \to \mathcal{K}$ and $\rho_{T_2} : \mathcal{G} \to \mathcal{H}$ are the group epimorphisms corresponding to T_1 and T_2, respectively. Now $g \in ker(\rho_{T_1})$ iff $T_1(g, s) = s$ for all $s \in S_1$. Suppose that $g \in ker(\rho_{T_1})$. Then $\rho T_1(g, s) = \rho(s) = T_2(g, \rho(s))$, so $g \in ker\rho_{T_2}$, since ρ is onto. □

The next proposition shows that a group permutes the blocks of a block-system in the same way that it permutes the left cosets of the corresponding subgroup.

Proposition 4.3.2: *Let $(\mathcal{G}, S)$ be a transitive permutation group; let $v \in S$ and let π be a block-system corresponding to a subgroup $\mathcal{H}$ of $\mathcal{G}$ with respect to v. Then the action of $\mathcal{G}$ on the left cosets of $\mathcal{H}$ by left multiplication is equivalent to the action of $\mathcal{G}$ on the blocks of π by left multiplication. In particular, the map $\rho : \{g\mathcal{H} : g \in \mathcal{G}\} \to \{\mathsf{B} \in \pi\}$ given by $\rho(g\mathcal{H}) = g\mathcal{H}(v)$ is an equivalence between the two actions.*

Proof By Proposition 4.2.6, ρ is a bijection. Also, ρ commutes with the group actions since $\rho(g_1 g\mathcal{H}) = g_1 g\mathcal{H}(v) = g_1\rho(g\mathcal{H})$ for any g_1 and $g \in \mathcal{G}$. □

Remark: The proposition holds in particular for B the trivial block $\mathsf{B} = \{v\}$. In this case $\mathcal{H} = \mathcal{G}_v$ and Proposition 4.3.2 shows that $\mathcal{G}$ acts on the left cosets of $\mathcal{G}_v$ in the same way that it acts on S.

---

[2] Massey pg 255. The notion of equivalent actions is from [Jac74], page 72.

4.4 Operator Graphs

Recall that a *group-graph* is a graph whose edge-label monoid is a group. The next three sections examine the structure of group-graphs. A group graph G can be thought of as a picture of the action of a group ($\mathcal{E}(\mathsf{G})$) on a set ($\mathsf{V}(\mathsf{G})$), with the labeled edges representing group elements acting on the vertices. In this section we will describe a class of graphs, the "operator graphs", which are constructed from group actions in this way. A well-known example of an operator graph is a Cayley graph, which is constructed from the action of a group on itself by left (or right) multiplication. More generally, any action specifies a graph once a generator-set for the group is chosen.

Definition 4.4.1: Let $\mathcal{G}$ be a group with generator-set X, let S be a set, and $T : \mathcal{G} \times S \to S$ be a transitive group action. The *operator graph*[3] of $\mathcal{G}$ with respect to X and T is a graph $\mathsf{G} = \langle \mathsf{V}(\mathsf{G}), \mathsf{E}(\mathsf{G}), \mathsf{A}(\mathsf{G}) \rangle$, given as follows:

- $\mathsf{V}(\mathsf{G}) = S$;
- $\mathsf{A}(\mathsf{G}) = X$; and
- $\mathsf{E}(\mathsf{G})$ is the set of triples $\langle v\, g\, w \rangle$, where v and $w \in S$ and $g \in X$, and $T(g, v) = w$.

It is immediate that these graphs satisfy the "edge-label property" (Property 2 in Chapter 2, since elements of $\mathcal{G}$ are one-to-one on S.

Remark 4.4.1: Suppose that $\mathcal{G}$ is a group, T is an action and G is a corresponding operator graph. Then $\mathcal{E}(\mathsf{G})$ is the permutation group $\rho_T(\mathcal{G}) \leq Sym(S)$ associated with T, because if $g \in \mathcal{G}$ then $\rho_T(g)(v) = T(g, v)$ for all $v \in S$.

EXAMPLE 4.4.1: Let G be a graph such that $\mathcal{E}(\mathsf{G})$ is a group. Define the *natural action* T of $\mathcal{E}(\mathsf{G})$ on $\mathsf{V}(\mathsf{G})$ by: $T(f_a, v) = f_a(v)$. Let G' be the operator graph of the natural action of $\mathcal{E}(\mathsf{G})$ on $\mathsf{V}(\mathsf{G})$ with respect to the generator set $X = \{f_a : a \in \mathsf{A}(\mathsf{G})\}$. Then G' is the graph obtained from G by replacing every edge-label a in an edge in G with the label $f_a \in X$. That is, $\langle v\, a\, w \rangle \in \mathsf{E}(\mathsf{G})$ iff $\langle v\, f_a\, w \rangle \in \mathsf{E}(\mathsf{G}')$. We will usually equate G' with G and call G the *operator graph of* $\mathcal{E}(\mathsf{G})$ *with respect to the natural action.*

EXAMPLE 4.4.2: **(Cayley graphs)** Let $\mathcal{G}$ be a group, X a set of generators for $\mathcal{G}$, and $T : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ the action of $\mathcal{G}$ on itself by left multiplication. The operator graph G associated with $\mathcal{G}, X$ and T is called a (left) *Cayley graph* of $\mathcal{G}$. That is, $\mathsf{V}(\mathsf{G}) = \mathcal{G}$, $\mathsf{A}(\mathsf{G}) = X$, and $\mathsf{E}(\mathsf{G})$ is the set of triples $\langle g_1, g_2, g_3 \rangle$ with $g_1 \in X$ and g_2 and $g_3 \in \mathcal{G}$ being such that $g_2 g_1 = g_3$.

The next proposition shows that the covering maps between two quotient-graphs $\mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ are precisely the $\mathcal{E}(\mathsf{G})$-equivariant maps between the natural actions on G/π_1 and G/π_2.

Proposition 4.4.1:
(1) Let $\mathcal{G}$ be a group with generator-set X, and let $T_1 : \mathcal{G} \times \mathsf{V}_1 \to \mathsf{V}_1$ and $T_2 : \mathcal{G} \times \mathsf{V}_2 \to \mathsf{V}_2$ be actions such that there is a $\mathcal{G}$-equivariant map $\rho : T_1 \to T_2$. Then ρ is a covering map from G_1 to G_2, where G_1 and G_2 are the operator graphs with respect to X, for T_1 and T_2, respectively.

---

[3] The term "operator graph" may be due to F.R.K. Chung of Bellcore. See also [ABR87]. Every operator graph is isomorphic with a *Schrier coset graph* [Bol79], which is an operator graph in which $\mathsf{V}(\mathsf{G})$ is the set of left cosets of a subgroup of $\mathcal{G}$ and $\mathcal{G}$ acts on the left cosets by left multiplication. (Corollary 4.4.1).

(2) Conversely, if G_1 *and* G_2 *are graphs and if there is a covering map* $\rho : \mathsf{G}_1 \to \mathsf{G}_2$ *then there are actions* $T_1 : \mathcal{E}(\mathsf{G}_1) \times \mathsf{V}(\mathsf{G}_1) \to \mathsf{V}(\mathsf{G}_1)$ *and* $T_2 : \mathcal{E}(\mathsf{G}_1) \times \mathsf{V}(\mathsf{G}_2) \to \mathsf{V}(\mathsf{G}_2)$ *such that* ρ *is an* $\mathcal{E}(\mathsf{G}_1)$*-equivariant map from* T_1 *to* T_2*. The graphs* G_1 *and* G_2 *are operator graphs of* T_1 *and* T_2*, respectively, with respect to the generator-set* $X = \{f_a : a \in \mathsf{A}(\mathsf{G}_1)\}$*.*

Proof

(1) Let $\mathcal{G}$, T_1, T_2 and ρ be as given. Note first that $\mathsf{A}(\mathsf{G}_1) = \mathsf{A}(\mathsf{G}_2) = X$. We will show that $\rho : \mathsf{V}_1 \to \mathsf{V}_2$ is a covering map from G_1 to G_2. This follows from Proposition 2.3.1 in Chapter 2, for let $a \in \mathcal{G}$ and write $f_a{}'$ for $\rho_{T_1}(a) \in \mathcal{E}(\mathsf{G}_1)$ and $f_a{}''$ for $\rho_{T_2}(a) \in \mathcal{E}(\mathsf{G}_2)$. Then:

(a) All elements of $\mathcal{E}(\mathsf{G}_1)$ and $\mathcal{E}(\mathsf{G}_2)$ are defined on all elements of V_1 and V_2 respectively, since T_1 and T_2 are defined on all of $\mathcal{G} \times \mathsf{V}_1$ and $\mathcal{G} \times \mathsf{V}_2$, respectively.

(b) For all $v \in \mathsf{V}_1$ and $a \in \mathcal{G}$, $\rho T_1(a, v) = T_2(a, \rho(v))$. That is, $\rho(f_a{}'(v)) = f_a{}''\rho(v)$. Thus by Proposition 2.3.1 in Chapter 2, ρ is a covering map .

(2) Let G_1 and G_2 be graphs and let $\rho : \mathsf{G}_1 \to \mathsf{G}_2$ be a covering map. Let $\mathcal{G} = \mathcal{E}(\mathsf{G}_1)$, let $T_1(f_a, v) = f_a(v)$, and let $T_2(f_a, \rho(v)) = \rho(T_1(f_a, v))$. Then T_1 and T_2 are easily seen to be actions, and the map $\rho : T_1 \to T_2$ is $\mathcal{G}$-equivariant by construction. By Example 4.4.1, G_1 is the operator graph of T_1 with respect to X. Edges in the operator graph of T_2 are of the form $\langle \rho(v)\, f_a\, \rho(w) \rangle$, where $\rho(w) = T_2(f_a, \rho(v)) = \rho f_a(v)$. Since ρ is a covering map, edges in G_2 are also of the form $\langle \rho(v)\, a\, \rho(w) \rangle$, where $\rho(w) = \rho(f_a(v))$. Thus G_2 is the operator graph of T_2 with respect to X. □

We will show next that any operator graph is covered by a Cayley graph of the same group.

Notation: If G is a graph and $\mathcal{E}(\mathsf{G})$ is a group, we will write "R" for the left Cayley graph of $\mathcal{E}(\mathsf{G})$ with respect to the generator set $\{f_a : a \in \mathsf{A}(\mathsf{G})\}$.

Note that $\mathcal{E}(\mathsf{R})$ is a regular permutation group isomorphic to $\mathcal{E}(\mathsf{G})$.

Corollary 4.4.1: *Let* $\mathcal{E}(\mathsf{G})_v \leq \mathcal{H} \leq \mathcal{E}(\mathsf{G})$*. Denote the left cosets of* $\mathcal{H}$ *in* $\mathcal{E}(\mathsf{G})$ *by* $\pi_{\mathcal{H}}$ *and the partition of* $\mathsf{V}(\mathsf{G})$ *associated with* $\mathcal{H}$ *by* π*. Then* $\mathsf{R}/\pi_{\mathcal{H}} \simeq \mathsf{G}/\pi$ *(and so* R *covers* G/π*). In particular, if* $\mathcal{H} = \mathcal{E}(\mathsf{G})_v$ *then* $\mathsf{R}/\pi_{\mathcal{H}} \simeq \mathsf{G}$*.*

Proof Let T_1 be the action of $\mathcal{E}(\mathsf{G})$ on $\pi_{\mathcal{H}}$ by left multiplication; i.e., $T_1(f_a, f_b\mathcal{H}) = f_a f_b \mathcal{H}$ for all blocks $f_b\mathcal{H} \in \pi_{\mathcal{H}}$. Define an action T_2 of $\mathcal{E}(\mathsf{G})$ on π similarly, i.e., $T_2(f_a, \mathsf{B}) = f_a \mathsf{B}$ for any $\mathsf{B} \in \pi$. Let $\rho : \pi_{\mathcal{H}} \to \pi$ be given by $\rho(f_a\mathcal{H}) \to f_a\mathcal{H}(v)$. By Proposition 4.3.2 this is an equivalence (that is, a bijective $\mathcal{E}(\mathsf{G})$-equivariant map) and so by Proposition 4.4.1 part 1, ρ is a covering map from $\mathsf{R}/\pi_{\mathcal{H}}$ to G/π. Since ρ is a bijection, it is an isomorphism. □

Corollary 4.4.2: *Let* G_1 *and* G_2 *be graphs. If there is a covering map:* $\mathsf{G}_1 \to \mathsf{G}_2$ *then there is a group epimorphism:* $\mathcal{E}(\mathsf{G}_1) \to \mathcal{E}(\mathsf{G}_2)$*. In particular, if* G_1 *and* G_2 *are isomorphic then* $\mathcal{E}(\mathsf{G}_1)$ *and* $\mathcal{E}(\mathsf{G}_2)$ *are isomorphic also.*

Proof Suppose that there is a covering map from G_1 to G_2. By Proposition 4.4.1 part 2, there is an $\mathcal{E}(\mathsf{G}_1)$-equivariant map $\rho : T_1 \rightarrow T_2$ such that G_1 and G_2 are the operator graphs for T_1 and T_2 with respect to the generator set $X = \{f_a \in \mathcal{E}(\mathsf{G}_1) : a \in \mathsf{A}(\mathsf{G}_1)\}$. By Proposition 4.3.1, there is a group epimorphism from the permutation group associated with T_1 to the permutation group associated with T_2. By Remark 4.4.1, these permutation groups are equal to $\mathcal{E}(\mathsf{G}_1)$ and $\mathcal{E}(\mathsf{G}_2)$, respectively. □

Remark 4.4.2: This section gives us a way of constructing nonisomorphic graphs which share the same symmetries. Let $T : \mathcal{G} \mathsf{X} S \rightarrow S$ be an action of a group $\mathcal{G}$ on a set S. If G_1 and G_2 are operator-graphs of T with respect to two unequal generator-sets X and Y of $\mathcal{G}$, then G_1 and G_2 are nonisomorphic. G_1 and G_2 have the same symmetries, however, since a symmetry is determined only by the action of $\mathcal{G}$ on S, not by the set of generators chosen for $\mathcal{G}$.

4.5 The Lattice Of Block-Systems And The Lattice Of Quotient-Graphs

If $\mathcal{E}(\mathsf{G})$ is a group, the set of subgroups of $\mathcal{E}(\mathsf{G})$ containing a stabilizer-subgroup $\mathcal{E}(\mathsf{G})_v$ forms a lattice under the operations subgroup-join and intersection. The set of block-systems of G inherits this lattice structure, since by Proposition 4.2.4, the block-systems are in one-to-one correspondence with the subgroups containing $\mathcal{E}(\mathsf{G})_v$. Since the block-systems are also in one-to-one correspondence with the graphs covered by G (Proposition 3.3.2 in Chapter 3), these graphs also inherit the lattice structure of the lattice of subgroups. This section briefly describes these lattices.

We begin by defining the lattice of block-systems (that is, of c-partitions) of a graph G.

Definition 4.5.1: Let G be a graph for which $\mathcal{E}(\mathsf{G})$ is a group and let $v \in \mathsf{V}(\mathsf{G})$. Define a lattice $\mathsf{L}_\mathsf{B} v$ of block-systems of G as follows:

- The partial order on $\mathsf{L}_\mathsf{B} v$ is given by $\pi_1 \leq \pi_2$ iff π_1 is a refinement of π_2.

If π_1 and π_2 are block-systems corresponding to subgroups $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively, with respect to v, then

- $\pi_1 \wedge \pi_2$ is the block-system corresponding to $\mathcal{H}_1 \cap \mathcal{H}_2$, i.e., the block-system containing the block $[v] = (\mathcal{H}_1 \cap \mathcal{H}_2)(v)$.
- $\pi_1 \vee \pi_2$ is the block-system corresponding to the subgroup join $\langle \mathcal{H}_1, \mathcal{H}_2 \rangle$ of $\mathcal{H}_1$ and $\mathcal{H}_2$, i.e., the block-system containing the block $[v] = \langle \mathcal{H}_1, \mathcal{H}_2 \rangle(v)$.

The next lemma shows that $\mathsf{L}_\mathsf{B} v$ is independent of the choice of v.

Lemma 4.5.1: $\mathsf{L}_\mathsf{B} v = \mathsf{L}_\mathsf{B} w$ *for all* $v, w \in \mathsf{V}(\mathsf{G})$.

Proof Let L_v be the lattice of subgroups of $\mathcal{E}(\mathsf{G})$ containing $\mathcal{E}(\mathsf{G})_v$ and L_w, the lattice of subgroups containing $\mathcal{E}(\mathsf{G})_w$. By Lemma 4.2.1, $\mathcal{E}(\mathsf{G})_v$ and $\mathcal{E}(\mathsf{G})_w$ are conjugate via some $f_a \in \mathcal{E}(\mathsf{G})$, where $f_a(v) = w$. Then f_a induces a lattice isomorphism (preserving lattice join and meet) from L_v to L_w by: $\mathcal{H} \rightarrow f_a \mathcal{H} {f_a}^{-1}$ for all $\mathcal{H} \in \mathsf{L}_v$. This holds because the inner automorphism is a group isomorphism, and so preserves subgroups, intersections,

and joins of subgroups. By Proposition 4.2.5, if a block-system π corresponds to $\mathcal{H}$ with respect to v then it corresponds to $f_a \mathcal{H} {f_a}^{-1}$ with respect to w. Thus $\mathsf{L_B} v = \mathsf{L_B} w$. □

Since $\mathsf{L_B} v$ is independent of v, we will write $\mathsf{L_B}$ instead of $\mathsf{L_B} v$ in the sequel.

The next lemma will be used in Chapter 5 to show that the symmetries of a graph form a lattice.

Lemma 4.5.2: *Let $\mathcal{G}_v$ be contained in subgroups $\mathcal{H}$ and $\mathcal{J}$ of $\mathcal{G}$. Then $g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v) = (g_1\mathcal{H} \cap g_2\mathcal{J})(v)$ for any $g_1, g_2 \in \mathcal{G}$.*

Proof Let $w \in (g_1\mathcal{H} \cap g_2\mathcal{J})(v)$. Then there is an element $g' \in g_1\mathcal{H} \cap g_2\mathcal{J}$ such that $g'(v) = w$. Thus $g'(v) \in g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v)$, and $(g_1\mathcal{H} \cap g_2\mathcal{J})(v) \subseteq g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v)$.

Now let $w \in g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v)$. Then there exists $g_1 h \in g_1\mathcal{H}$ and $g_2 j \in g_2\mathcal{J}$ such that $g_1 h(v) = g_2 j(v) = w$. Since $g_2\mathcal{J}(v)$ is a block and since $g_1 h$ maps $v \in \mathcal{J}(v)$ to a point in $g_2\mathcal{J}(v)$, it maps $\mathcal{J}(v)$ bijectively onto $g_2\mathcal{J}(v)$. By Corollary 4.2.2, $g_2\mathcal{J} = \{g : g\mathcal{J}(v) = g_2\mathcal{J}(v)\}$, and so $g_1 h \in g_2\mathcal{J}$. Then $w = g_1 h(v) \in (g_1\mathcal{H} \cap g_2\mathcal{J})(v)$, and so $g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v) \subseteq (g_1\mathcal{H} \cap g_2\mathcal{J})(v)$, and the two sets are equal. □

Corollary 4.5.1: *The intersection of two blocks is a block.*

□

Corollary 4.5.2: *Let $\mathcal{G}_v \leq \mathcal{H}, \mathcal{J} \leq \mathcal{G}$. If $g_1\mathcal{H}(v) \subseteq g_2\mathcal{J}(v)$ then $g_1\mathcal{H} \subseteq g_2\mathcal{J}$.*

Proof By Propositions 4.2.4 and 4.2.6 there is a one-to-one correspondence between the set of left cosets of subgroups of $\mathcal{G}$ containing $\mathcal{G}_v$ and the set of blocks of $\mathcal{G}$, given by: $g\mathcal{H} \rightarrow g\mathcal{H}(v)$. Suppose that $g_1\mathcal{H}(v) \subseteq g_2\mathcal{J}(v)$. By Lemma 4.5.2 we have $g_1\mathcal{H}(v) = g_1\mathcal{H}(v) \cap g_2\mathcal{J}(v) = (g_1\mathcal{H} \cap g_2\mathcal{J})(v)$. Since $g_1\mathcal{H} \cap g_2\mathcal{J}$ is a left coset of $\mathcal{H} \cap \mathcal{J}$ we have $g_1\mathcal{H} = g_1\mathcal{H} \cap g_2\mathcal{J}$, or $g_1\mathcal{H} \subseteq g_2\mathcal{J}$. □

The set of quotient graphs of G also forms a lattice $\mathsf{L_G}$ under the partial order "$\preceq$", where $\mathsf{G}/\pi_1 \preceq \mathsf{G}/\pi_2$ if π_1 is a refinement of π_2 (i.e., if G/π_2 is a quotient of G/π_1). Its join and meet are described as follows:

- $\mathsf{G}/\pi_1 \wedge \mathsf{G}/\pi_2 = \mathsf{G}/\pi_1 \wedge \pi_2$
- $\mathsf{G}/\pi_1 \vee \mathsf{G}/\pi_2 = \mathsf{G}/\pi_1 \vee \pi_2$.

Then $\mathsf{L_G}$ inherits the lattice structure from $\mathsf{L_B}$.

EXAMPLE 4.5.1: Figure 4.1 pictures a Cayley graph of the dihedral group D_8, its lattice of quotient-graphs and the corresponding lattice of subgroups. Figure 4.2 shows the quotient-graphs of G.

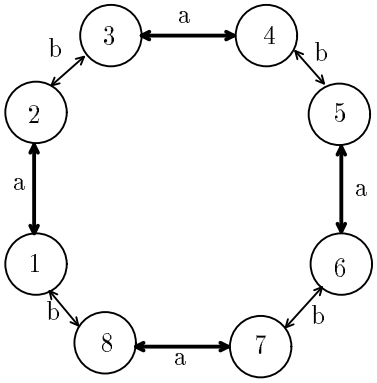
The elements of $\mathcal{E}(\mathsf{G})$ are as follows:

$f_1 = id$ $\quad f_5 = (1,5)(2,6)(3,7)(4,8)$
$f_2 = (1,2)(3,4)(5,6)(7,8)$ $\quad f_6 = (1,6)(2,5)(3,8)(4,7)$
$f_3 = (1,3,5,7)(2,8,6,4)$ $\quad f_7 = (1,7,5,3)(2,4,6,8)$
$f_4 = (1,4)(2,7)(3,6)(5,8)$ $\quad f_8 = (1,8)(2,3)(4,5)(6,7)$

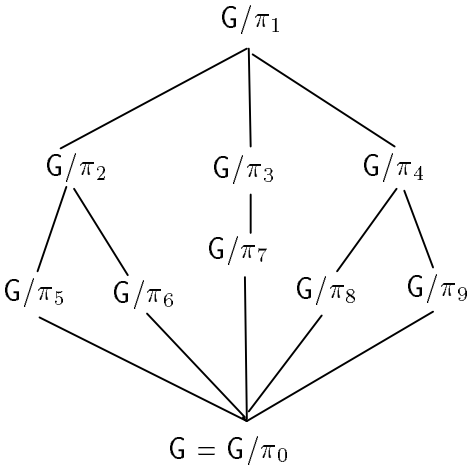
(Here "f_i" denotes the group-element mapping 1 to i. The subscript "i" is not a word in $\mathsf{A}(\mathsf{G})^*$.)

The block-systems for G and their corresponding subgroups with respect to vertex 1 are given as follows:

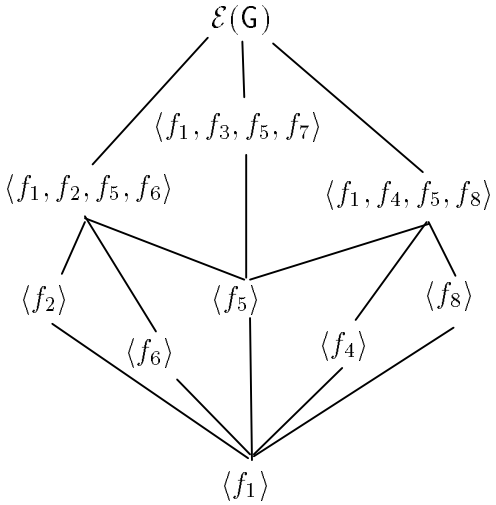
$\pi_1 = /1,2,\ldots,8/ \sim \mathcal{E}(\mathsf{G})$
$\pi_2 = 1,2,5,6/3,4,7,8 \sim \langle f_1, f_2, f_5, f_6\rangle$
$\pi_3 = 1,3,5,7/2,4,6,8 \sim \langle f_1, f_3, f_5, f_7\rangle$
$\pi_4 = 1,4,5,8/2,3,6,7 \sim \langle f_1, f_4, f_5, f_8\rangle$
$\pi_5 = 1,2/3,8/4,7/5,6 \sim \langle f_1, f_2\rangle$
$\pi_6 = 1,6/2,5/3,4/7,8 \sim \langle f_1, f_6\rangle$
$\pi_7 = 1,5/2,6/3,7/4,8 \sim \langle f_1, f_5\rangle$
$\pi_8 = 1,4/2,3/5,8/6,7 \sim \langle f_1, f_4\rangle$
$\pi_9 = 1,8/2,7/3,6/4,5 \sim \langle f_1, f_8\rangle$
$\pi_{10} = 1/2/3/4/5/6/7/8 \sim \langle f_1\rangle$



G is a Cayley graph of D_8, generated by f_2 and f_8.



The lattice of quotients



The lattice of subgroups

Figure 4.1: A graph G and its lattice of subgroups and lattice of quotient-graphs

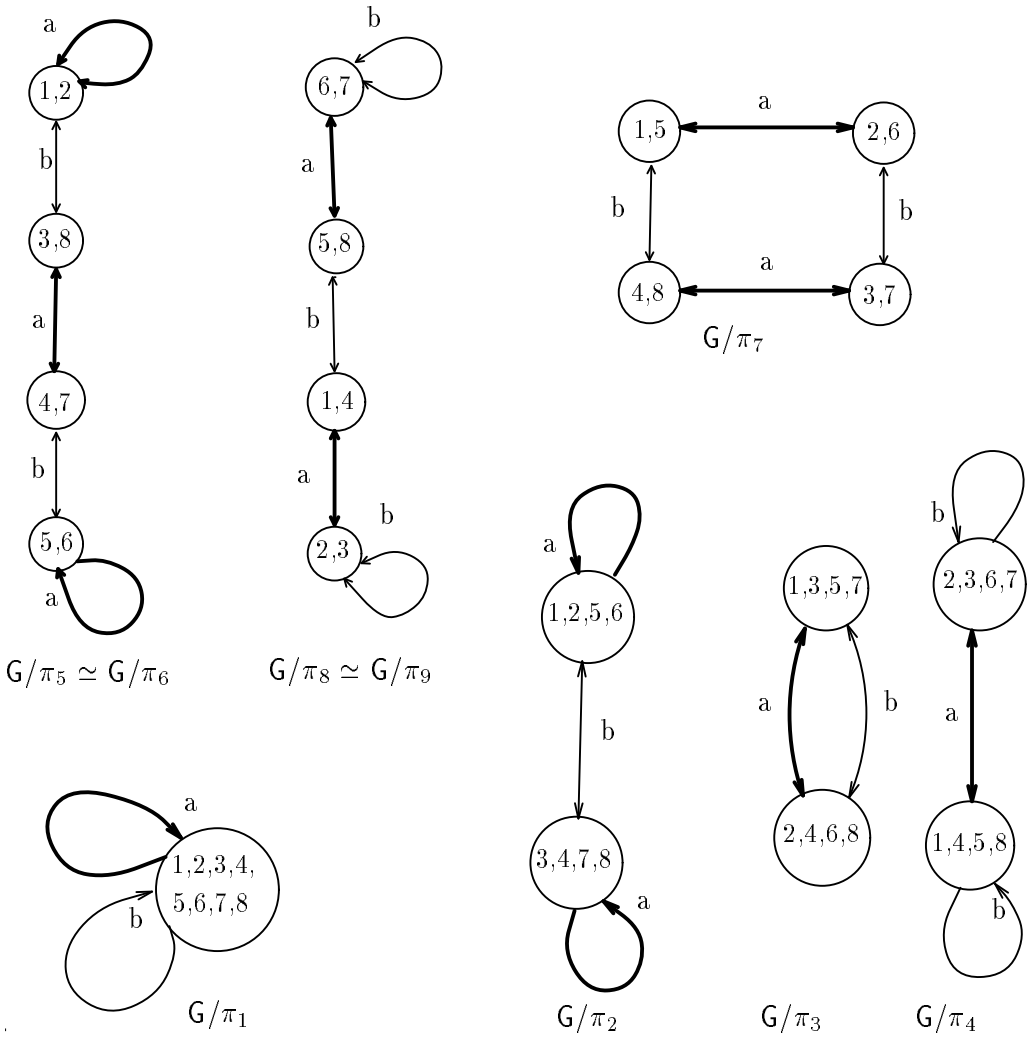


Figure 4.2: The quotient-graphs of G

4.6 Graph Isomorphisms And Conjugate Subgroups

One of our aims in the next chapter will be to find an algorithm which generates all of the symmetries of a given network. In this section we will show that there is a one-to-one map from the set of symmetries of a graph G (or more particularly, the set of all isomorphisms of quotients of G) into the set of all left cosets of subgroups of $\mathcal{E}(\mathsf{G})$. This will allow us to characterize each symmetry uniquely as a left coset of a subgroup of $\mathcal{E}(\mathsf{G})$. We will also find a correspondence between subgroup conjugacy and quotient-graph isomorphism.

First we need a definition:

Definition 4.6.1: If $\mathcal{G}$ is a permutation group on a set S, the *centralizer* $\mathsf{C}_{Sym(S)}\mathcal{G}$ of $\mathcal{G}$ in the symmetric group $Sym(S)$ is the set of all elements in $Sym(S)$ which commute with all elements of $\mathcal{G}$:

$\mathsf{C}_{Sym(S)}\mathcal{G} = \{h \in Sym(S) : hg = gh \text{ for all } g \in \mathcal{G}\}$.

Note that:

Proposition 4.6.1: *(Proposition 10.3.6 in [Sco87]) If $(\mathcal{G}, S)$ is a regular permutation group then $\mathsf{C}_{Sym(S)}\mathcal{G}$ is regular and isomorphic to $\mathcal{G}$.*

The next proposition is an immediate corollary of Proposition 2.3.1 in Chapter 2.

Proposition 4.6.2: *If G is a graph such that $\mathcal{E}(\mathsf{G})$ is a group, then the automorphism group of G is the centralizer in $Sym(\mathsf{V}(\mathsf{G}))$ of $\mathcal{E}(\mathsf{G})$.*

Proof Let α be an automorphism of G. By Proposition 2.3.1 in Chapter 2, α commutes with all elements of $\mathcal{E}(\mathsf{G})$ and so is an element of the centralizer. Conversely, let h be an element of the centralizer in $Sym(\mathsf{V}(\mathsf{G}))$ of $\mathcal{E}(\mathsf{G})$. Since $\mathcal{E}(\mathsf{G})$ is a group, any $f_a \in \mathcal{E}(\mathsf{G})$ is defined on all $v \in \mathsf{V}(\mathsf{G})$. Since $h \in \mathsf{C}_{Sym(\mathsf{V}(\mathsf{G}))}\mathcal{E}(\mathsf{G})$, we have $hf_a(v) = f_a h(v)$ for all $f_a \in \mathcal{E}(\mathsf{G})$ and $v \in \mathsf{V}(\mathsf{G})$. By Proposition 2.3.1 in Chapter 2, h is an automorphism of G. □

EXAMPLE 4.6.1: In Figure 4.1, G is the left Cayley graph of D_8, generated by $f_2 = (1,2)(3,4)(5,6)(7,8)$ and $f_8 = (1,8)(2,3)(4,5)(6,7)$. (See Example 4.5.1.) The automorphism group is the permutation group associated with D_8 acting on itself, generated by $\alpha_2 = (1,2)(3,8)(4,7)(5,6)$ and $\alpha_8 = (1,8)(2,7)(3,6)(4,5)$. See Proposition 4.6.4.

We will need the following proposition from group theory:

Proposition 4.6.3: *([Koc70], page 118)*

- *Let $\mathcal{G}$ be a group and $T_l : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ and $T_r : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$ the actions by left and right multiplication, respectively. Write $\mathcal{G}_l$ for the permutation group $\rho_{T_l}(\mathcal{G})$ associated with T_l and $\mathcal{G}_r$ for the permutation group associated with T_r. Then $\mathcal{G}_l = \mathsf{C}_{Sym(\mathcal{G})}\mathcal{G}_r$ and $\mathcal{G}_r = \mathsf{C}_{Sym(\mathcal{G})}\mathcal{G}_l$.*
- *$\mathcal{G}_l$ and $\mathcal{G}_r$ are regular permutation groups.*

□

As a corollary of Propositions 4.6.2 and 4.6.3 we have:

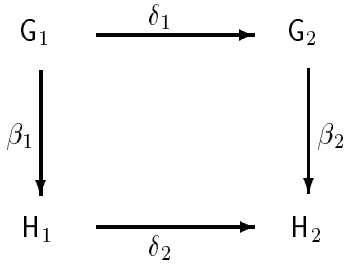
Proposition 4.6.4: *Let G be a graph and let R be the left Cayley graph of $\mathcal{E}(\mathsf{G})$ with respect to the generator-set $\{f_a : a \in \mathsf{A}(\mathsf{G})\}$. Define $\mathcal{E}(\mathsf{G})_l$ and $\mathcal{E}(\mathsf{G})_r$ as in Proposition 4.6.3 above. Then $\mathcal{E}(\mathsf{R}) = \mathcal{E}(\mathsf{G})_l$ and the automorphism group of R is $\mathcal{E}(\mathsf{G})_r$.*

Proof $\mathcal{E}(\mathsf{R}) = \mathcal{E}(\mathsf{G})_l$ by the construction of left Cayley graphs. By Proposition 4.6.2, the automorphism group of R is $C_{Sym(\mathsf{V}(\mathsf{R}))}\mathcal{E}(\mathsf{R}) = C_{Sym(\mathsf{V}(\mathsf{R}))}\mathcal{E}(\mathsf{G})_l = C_{Sym(\mathcal{E}(\mathsf{G}))}\mathcal{E}(\mathsf{G})_l$. By Proposition 4.6.3, this latter equals $\mathcal{E}(\mathsf{G})_r$. □

Informally this says that $\mathcal{E}(\mathsf{R})$ is the group $\mathcal{E}(\mathsf{G})$ acting on itself by left multiplication, and the automorphism group of R is $\mathcal{E}(\mathsf{G})$ acting on itself by right multiplication. (See Example 4.3.1 for a definition of these actions.)

The next lemma shows that any isomorphism of quotient-graphs "lifts" to an automorphism of R, where "lifting" is as defined below.

Definition 4.6.2: Let graphs G_1 cover H_1 via a covering map β_1, let G_2 cover H_2 via β_2, and let δ_1 be an isomorphism from G_1 to G_2 and δ_2 be an isomorphism from H_1 to H_2. We will say that δ_2 *lifts to* δ_1 *via* β_1 *and* β_2 if the following commutes:



EXAMPLE 4.6.2: In Figure 4.3 the automorphism $\delta_2 = (r, s)$ lifts to the automorphism $\delta_1 = (1,2)(3,4)$ via covering maps $\beta_1 = \beta_2 : \{1,4\} \to r$ and $\{2,3\} \to s$.

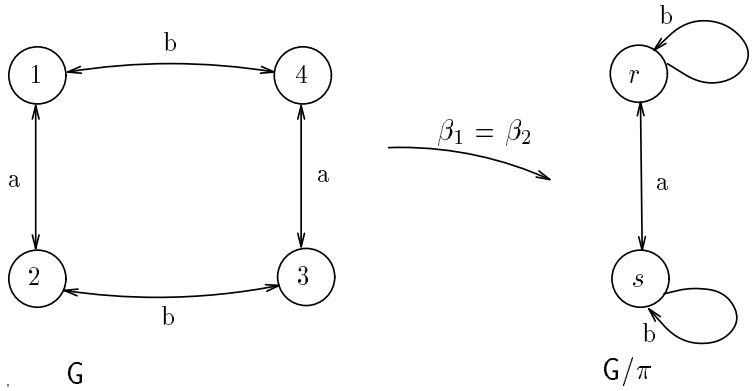


Figure 4.3: Lifting

We have:

Lemma 4.6.1: *Let* $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ *be a graph isomorphism and let* $\mathcal{H} \leq \mathcal{E}(\mathsf{G})$ *correspond to* π_1 *and* $\mathcal{J} \leq \mathcal{E}(\mathsf{G})$ *correspond to* π_2*, with respect to* $v \in \mathsf{V}(\mathsf{G})$*. Then for any covering maps* $\beta_1 : \mathsf{R} \to \mathsf{G}/\pi_1$ *and* $\beta_2 : \mathsf{R} \to \mathsf{G}/\pi_2$*, there is an automorphism* α *of* R *such that* δ *lifts to* α *via* β_1 *and* β_2*. If* β_1 *and* β_2 *are the inclusion covering maps then* α *induces a bijection between the left cosets of* $\mathcal{H}$ *and the left cosets of* $\mathcal{J}$*.*

Proof Let $\beta_1 : \mathsf{R} \to \mathsf{G}/\pi_1$ and $\beta_2 : \mathsf{R} \to \mathsf{G}/\pi_2$ be any covering maps, and let v and $w \in \mathsf{V}(\mathsf{R})$ be such that $\delta\beta_1(v) = \beta_2(w)$. Let $\alpha : \mathsf{R} \to \mathsf{R}$ be an automorphism such that $\alpha(v) = w$ This automorphism exists because $\mathcal{E}(\mathsf{R})$ is regular (Proposition 4.6.4).

Then $\beta_2\alpha(v) = \delta\beta_1(v)$, so $\beta_2\alpha = \delta\beta_1$ since $\beta_2\alpha$ and $\delta\beta_1$ are graph epimorphisms and hence determined by their action on a single vertex. Thus δ lifts to α via β_1 and β_2.

Suppose that $\beta_1 : \mathsf{R} \to \mathsf{G}/\pi_1$ and $\beta_2 : \mathsf{R} \to \mathsf{G}/\pi_2$ are the inclusion covering maps; that is, that $\beta_1(f_b h) = f_b\mathcal{H}(v)$ for each $f_b h \in f_b\mathcal{H}$, and β_2 is defined similarly. Since $\delta\beta_1(v) = \beta_2\alpha(v)$ for all $v \in \mathsf{V}(\mathsf{R})$, the automorphism α must map cosets onto cosets. □

We will need the next two propositions in Chapter 5. Proposition 4.6.5 (next) shows that there is a close correspondence between quotient-graph isomorphisms and subgroup conjugacy.

Proposition 4.6.5: *Let $\mathcal{H}$ correspond to π_1 with respect to a vertex v and let $\mathcal{J}$ correspond to π_2 with respect to v. Then:*

1. *If $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is an isomorphism and $\delta(\mathcal{H}(v)) = f_a\mathcal{J}(v)$ for some $f_a \in \mathcal{E}(\mathsf{G})$ then $\mathcal{H}f_a = f_a\mathcal{J}$.*
2. *Conversely, if $\mathcal{H}f_a = f_a\mathcal{J}$ for some $f_a \in \mathcal{E}(\mathsf{G})$ then there is an isomorphism $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ such that $\delta(\mathcal{H}(v)) = f_a\mathcal{J}(v)$.*

In particular, $\mathcal{H}$ and $\mathcal{J}$ are conjugate in $\mathcal{E}(\mathsf{G})$ iff G/π_1 and G/π_2 are isomorphic.

EXAMPLE 4.6.3: In Example 4.5.1, the block-system $\pi_1 = 1,2/3,8/4,7/5,6$ corresponds to the subgroup $\mathcal{H}$ generated by f_2, and $\pi_2 = 1,6/2,5/3,4/7,8$ corresponds to the subgroup $\mathcal{J}$ generated by f_6. Here $\mathcal{H}$ and $\mathcal{J}$ are conjugate: $\mathcal{H}f_3 \simeq f_3\mathcal{J}$. The quotient-graphs G/π_1 and G/π_2 are isomorphic via the isomorphism $\delta : \{1,2\} \to \{3,4\}$; $\{3,8\} \to \{2,5\}$; $\{4,7\} \to \{1,6\}$, and $\{5,6\} \to \{7,8\}$.

Proof (of Proposition 4.6.5) Suppose first that $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is an isomorphism and that $\delta(\mathcal{H}(v)) = f_a\mathcal{J}(v)$ for some $f_a \in \mathcal{E}(\mathsf{G})$. By Lemma 4.6.1 above, δ lifts to an automorphism α of R via the inclusion covering maps β_1 and β_2. By construction, α maps $\mathcal{H}$ to $f_a\mathcal{J}$. By Proposition 4.6.4, α is an element ${f_b}^{-1}$ of $\mathcal{E}(\mathsf{G})$ acting on $\mathcal{E}(\mathsf{G})$ by right multiplication: If α maps $\mathcal{H}$ to $f_a\mathcal{J}$ then $\mathcal{H}f_b = f_a\mathcal{J}$. Then $f_b = f_aj$ for some $j \in \mathcal{J}$, and so $\mathcal{H}f_b = \mathcal{H}f_aj = f_a\mathcal{J}$, or $\mathcal{H}f_a = f_a\mathcal{J}$.

Conversely, suppose that $\mathcal{H}f_a = f_a\mathcal{J}$ for some $f_a \in \mathcal{E}(\mathsf{G})$. By Proposition 4.6.4, ${f_a}^{-1}$, viewed as an element of $\mathcal{E}(\mathsf{G})$ acting on $\mathcal{E}(\mathsf{G})$ by right multiplication, is an automorphism of R. In fact, ${f_a}^{-1}$ induces a bijection α between the left cosets of $\mathcal{H}$ and the left cosets of $\mathcal{J}$ by $\alpha(f_b\mathcal{H}) = f_b\mathcal{H}f_a = f_bf_a\mathcal{J}$. Let $\beta_1 : \mathsf{R} \to \mathsf{G}/\pi_1$ and $\beta_2 : \mathsf{R} \to \mathsf{G}/\pi_2$ be the inclusion covering maps and define $\delta = \beta_2\alpha{\beta_1}^{-1} : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$. Then δ is well-defined and is easily seen to be an isomorphism. By construction, $\delta(\mathcal{H}(v)) = f_a\mathcal{J}(v)$. □

Finally, we have:

Proposition 4.6.6: G, R *as before. Fix $v \in \mathsf{V}(\mathsf{G})$. There is an injective map ρ from the set of all isomorphisms of quotients of G into the set of left cosets of the subgroups of $\mathcal{E}(\mathsf{G})$ which contain $\mathcal{E}(\mathsf{G})_v$. This is given as follows:*

Let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism and let π_1 correspond to $\mathcal{H}$ and π_2 correspond to $\mathcal{J}$ with respect to v. Then ρ maps δ to $f_a\mathcal{J}$, where $\mathcal{H}f_a = f_a\mathcal{J}$ and $f_a\mathcal{J}(v) = \delta(\mathcal{H}(v))$. That is, if $\delta([v]_{\pi_1}) = [w]_{\pi_2}$, then $\rho : \delta \to f_a\mathcal{J}$ where $f_a\mathcal{J}(v) = [w]_{\pi_2}$.

EXAMPLE 4.6.4: In Example 4.6.3, δ corresponds to $f_3\mathcal{J} = \{f_3, f_4\}$. We have $f_3\mathcal{J}(1) = \{3,4\}$. Indeed, $\delta(\{1,2\}) = \{3,4\}$.

Proof of Proposition 4.6.6 Let ρ, δ, $\mathcal{H}$ and $\mathcal{J}$ be as given. We will show that ρ is well-defined on the set of all isomorphisms of quotients of G, and that it is one-to-one. Let $\beta_1 : \mathsf{R} \to \mathsf{G}/\pi_1$ and $\beta_2 : \mathsf{R} \to \mathsf{G}/\pi_2$ be the inclusion covering maps. By Lemma 4.6.1, δ lifts to an automorphism α of R via β_1 and β_2, and α maps the left cosets of $\mathcal{H}$ bijectively onto the left cosets of $\mathcal{J}$. By definition of lifting, $\delta(\mathcal{H}(v)) = \delta\beta_1(h) = \beta_2\alpha(h) = f_a\mathcal{J}(v)$ for some $f_a \in \mathcal{E}(\mathsf{G})$ and $h \in \mathcal{H}$, and so ρ is defined on δ. If $f_a\mathcal{J}(v) = f_b\mathcal{J}(v)$ for some f_a and $f_b \in \mathcal{E}(\mathsf{G})$ then $f_a\mathcal{J} = f_b\mathcal{J}$ by Proposition 4.2.6, and so ρ is well-defined. By Proposition 4.6.5, if ρ maps δ to $f_a\mathcal{J}$ then $\mathcal{H}f_a = f_a\mathcal{J}$. Finally, ρ is one-to-one, for suppose that ρ maps two isomorphisms to the same left coset:

$\delta_1(\mathcal{H}_1(v)) = f_a\mathcal{J}(v)$ and

$\delta_2(\mathcal{H}_2(v)) = f_a\mathcal{J}(v)$ also. By Proposition 4.6.5, $\mathcal{H}_1 f_a = \mathcal{H}_2 f_a = f_a\mathcal{J}$, so $\mathcal{H}_1 = \mathcal{H}_2 = f_a\mathcal{J}{f_a}^{-1}$, and $\delta_1 = \delta_2$. □

4.7 Related Results

Operator Graphs: The term "operator graph" may have been coined by F.R.K. Chung of Bellcore. An older name for operator graph is "group action graph"; see [ABR87] and [BR91]. As mentioned in the footnote in Section 4.4, any operator graph of $\mathcal{G}$ is isomorphic with a *Schrier coset graph* of a group $\mathcal{G}$, which is an operator graph of the action of $\mathcal{G}$ on the left cosets of a subgroup of $\mathcal{G}$. See [Bol79].

An operator graph is a special case of a "$\mathcal{G}$-graph", defined in [Coh89]. If $\mathcal{G}$ is a group, a "$\mathcal{G}$-graph" is a graph G such that $\mathcal{G}$ acts on $\mathsf{V}(\mathsf{G})$ and $\mathsf{E}(\mathsf{G})$ in such a way that the orientations and inverses of the edges are preserved. If we let G be an operator graph of a group $\mathcal{G}$ with respect to an action T_1 on $\mathsf{V}(\mathsf{G})$, and let $T_2 : \mathcal{G} \times \mathsf{E}(\mathsf{G}) \to \mathsf{E}(\mathsf{G})$ be the identity action which maps each edge to itself, then G is a $\mathcal{G}$-graph.

An operator graph is also a special case of a "voltage graph". The term *voltage graph* was coined by J Gross ([GT87]) and refers to a digraph G whose edges are given "plus" and "minus" directions, together with a map from the plus-directed edges into a permutation group. Voltage graphs are used in [GT87] to compute covering graphs for given graphs; that is, for reconstructing a graph from one of its quotients.

The lattices of quotient graphs and of subgroups: In [B$\ddot{8}$9], Büchi derives related results for "k-algebras". A *k-algebra* is a tuple $\mathsf{A} = \langle A, E, f_1, \ldots, f_k\rangle$, where A is a set of "states", $E \in A$ is the "start state", and each f_i for $i = 1, \ldots, k$ is a function from A to A. A k-algebra A is associated in a natural way with a graph G: Let $\mathsf{V}(\mathsf{G}) = A$ and let $\langle v\, f_i\, w\rangle \in \mathsf{E}(\mathsf{G})$ whenever $f_i(v) = w$. A k-algebra is called *reduced* if its associated graph is strongly connected.

A *congruence relation* $\sim$ on a k-algebra A is an equivalence relation on A such that $v \sim w$ iff $f_i(v) \sim f_i(w)$ for $i = 1, \ldots, k$. Congruence relations induce quotient algebras in the usual way. Büchi proves the following:

Theorem 4.7.1: *(Theorem 4 in [B$\ddot{8}$9]) The lattice of quotients of a reduced k-algebra* A *is anti-isomorphic to the lattice of all congruences on* A*, where the anti-isomorphism maps each quotient* $\mathsf{A}/\sim$ *to the congruence* $\sim$*.*

The order relation on the lattice of quotients is "homomorphism", where a *homomorphism* from a k-algebra $\mathsf{A} = \langle A, E, f_1, \ldots, f_k\rangle$ to a k-algebra $\mathsf{B} = \langle B, F, g_1, \ldots, g_k\rangle$ is a map α from A onto B such that $g_i\alpha(v) = \alpha f_i(v)$ for $i = 1, \ldots, k$ and for all $v \in A$. (Notice the similarity between Büchi's definition of homomorphism and our definition of covering map.) As might be expected, there is a one-to-one correspondence between the homomorphims from an algebra A and the congruence relations on A.

Büchi also shows the following for a family α_i of congruences on a k-algebra A:

Theorem 4.7.2: *(Theorem 4 part 2 in [B$\ddot{8}$9])*

$A/(\bigcap_i \alpha_i)$ *is isomorphic to* $\bigotimes_i (A/\alpha_i)$*, where '*$\bigcap$*' is the meet-operator on the lattice of congruences and '*$\bigotimes$*' is the "reduced direct sum";*

$A/(\bigcup_i \alpha_i)$ is isomorphic to $\bigwedge_i (A/\alpha_i)$, where '$\bigcup$' is the join-operator on the lattice of congruences, and '$\bigwedge$' is the meet-operator on the lattice of quotients.

5. Classifying Group Graphs

5.1 Introduction

How hard is it to classify networks? This chapter addresses the question for the two classifications described previously, in which networks were called "f-equivalent" if the set of functions each can compute is the same, and "p-equivalent" if the set of functions each can compute is the same up to a permutation. In light of Theorems 3.4.2 and 3.5.1 in Chapter 3, the question of how hard it is to classify networks can be rephrased as follows: Is there a small, easily-computed set of graph-features which two graphs share in common iff they have the same set of quotient-graph isomorphisms (or, respectively, iff their sets of isomorphisms differ by a permutation)? The answer to the first question is "yes" — the first classification task is relatively easy. The second classification problem will turn out to be at least as hard as determining whether two finite permutation groups are isomorphic.

Chapter Summary and Main Results: The first four sections of this chapter address the first classification question. In Section 5.2 we will show that a graph can have a subexponential number of symmetries, and conclude that checking two graphs for equivalence by comparing their symmetry-sets is potentially slow. In Section 5.3 we will see that the symmetry-set of a graph forms a lattice, and in Section 5.4, will find a small generator-set, the "constraint-set", for the lattice of symmetries. We will show that two networks compute the same set of functions iff they have the same constraint-set. Section 5.5 gives an algorithm polynomial in the number of edges of a graph for finding the graph's constraints, and concludes the examination of the first classification question. Section 5.6 addresses the second classification problem, and gives a polynomial-time transformation of this problem to the group-isomorphism problem. Since we are only concerned with the tractability of problems, we will aim for algorithms which are easy to understand, rather than optimally fast.

This chapter concerns itself only with group-graphs. In the final chapter we will see that the results obtained for group-graphs also hold for arbitrary monoid-graphs.

5.2 The Number Of Symmetries Of A Network

In this section we will count the number of block-systems and symmetries of a network. We will see that a network can have a subexponential number of both block-systems and symmetries.

The following proposition is folklore in computational group theory. It may be due to Tarjan (See [Mil78]).

Proposition 5.2.1: *A group $\mathcal{G}$ of order n has $O(n^{\lg n})$ subgroups. $\mathcal{G}$ is generated by a set of size no bigger than* $\lg n$.

Proof sketch Adding one element to a set of generators of a subgroup of $\mathcal{G}$ at least doubles the size of the subgroup generated. Therefore $\mathcal{G}$ has a generator-set of size less than or equal to $\lg n$. There are $\sum_{k=1}^{\lg n} \binom{n}{k}$ subsets of $\mathcal{G}$ of size less than or equal to $\lg n$

and hence no more than $\sum_{k=1}^{\lg n} \binom{n}{k}$ subgroups. The conclusion follows from the well-known fact that $\sum_{k=0}^{m} \binom{n}{k} \leq n^m + 1$. □

A similar argument gives us the following:

Proposition 5.2.2: *Let $(\mathcal{G}, S)$ be a transitive permutation group, with $|S| = n$. Then $\mathcal{G}$ has $O(n^{\lg n})$ block-systems.*

Proof of the proposition: Let $\mathcal{G}_v$ be the stabilizer subgroup of a point $v \in S$. We show that $\mathcal{G}$ has $O(n^{\lg n})$ subgroups containing $\mathcal{G}_v$. The conclusion then follows from Proposition 4.2.4 in Chapter 4, which says that there is a one-to-one correspondence between block-systems and subgroups containing $\mathcal{G}_v$.

Let $\mathcal{J}$ be a subgroup of $\mathcal{G}$ containing $\mathcal{G}_v$, but not containing a given left coset $g\mathcal{G}_v$ of $\mathcal{G}_v$. Then the subgroup $\langle g\mathcal{G}_v, \mathcal{J} \rangle$ generated by $g\mathcal{G}_v$ and $\mathcal{J}$ contains at least twice as many left cosets of $\mathcal{G}_v$ as $\mathcal{J}$ contains. Recall that $|\mathcal{G}|/|\mathcal{G}_v| = n$ for any transitive group $\mathcal{G}$. (See Proposition 1.6.1 in [Rob82].) Therefore, there is a collection of no more than $\lg n$ left cosets of $\mathcal{G}_v$ which generates $\mathcal{G}$ (that is, $\mathcal{G}$ is generated by the set of group-elements comprising these cosets). The same argument shows that any subgroup $\mathcal{J}$ of $\mathcal{G}$ containing $\mathcal{G}_v$ is generated by a set of no more than $\lg n$ left cosets of $\mathcal{G}_v$. Since there are at most $\sum_{k=1}^{\lg n} \binom{n}{k} = O(n^{\lg n})$ sets of left cosets of $\mathcal{G}_v$ of size less than or equal to $\lg n$, there are $O(n^{\lg n})$ subgroups of $\mathcal{G}$ containing $\mathcal{G}_v$. □

Corollary 5.2.1: *If $|\mathsf{V}(\mathsf{G})| = n$ and $\mathcal{E}(\mathsf{G})$ is a group then G has $O(n^{\lg n})$ c-partitions.*

□

Estimate of the number of symmetries of a network:

Let G be a network with n vertices. By Proposition 4.6.6 in Chapter 4, each block of $\mathcal{E}(\mathsf{G})$ represents at most one symmetry, so we will count the number of blocks of $\mathcal{E}(\mathsf{G})$. By Proposition 5.2.2, $\mathcal{E}(\mathsf{G})$ has $O(n^{\lg n})$ block-systems. Each block-system has at most n blocks, so G has at most $O(n^{\lg n+1})$ blocks, and hence $O(n^{\lg n+1})$ symmetries.

Remark 5.2.1: There exist graphs having nearly $n^{\lg n}$ symmetries. For instance, a Cayley graph of the group $\mathbf{Z}_2^n = \mathbf{Z}_2 \times \cdots \times \mathbf{Z}_2$ has roughly $n^{\lg n}$ symmetries, since $\mathbf{Z}_2^n$ has roughly $n^{\lg n}$ subgroups. Each subgroup is associated with a quotient graph and each quotient graph has at least the identity automorphism, so there is at least one symmetry for each subgroup.

5.3 The Lattice Of Symmetries

In this section we will show that the set of symmetries of a network G forms a lattice, the *lattice of symmetries of* G. We saw that the lattice of symmetries of a network with n vertices can be large, but we will see that it has a subset, the "constraint-set", which has at most n elements and generates the lattice of symmetries under the lattice-join operation. Since the constraint set generates the lattice of symmetries, it completely determines the behavior of the network vis-a-vis function-computation. That is, two networks compute the same set of functions iff they have identical constraint-sets.

By Propostition 4.6.6 in Chapter 4 there is an injective map from the set of isomorphisms of quotients of G into the set of left cosets of subgroups of $\mathcal{E}(\mathsf{G})$. This gives us the following definition:

Definition 5.3.1: Fix $v \in \mathsf{V}(\mathsf{G})$ and let $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ be a symmetry of G. By Proposition 4.6.6 of Chapter 4 there are subgroups $\mathcal{H}$ and $\mathcal{J}$ of $\mathcal{E}(\mathsf{G})$ and $f_a \in \mathcal{E}(\mathsf{G})$ such that $\mathcal{H}f_a = f_a\mathcal{J}$ and $f_a\mathcal{J}(v) = \delta([v]_{\pi_1})$. We will call the left-coset $f_a\mathcal{J}$ the *coset representative of the symmetry* $\mathbf{s}$ *with respect to* v.

We will show next that there are meet- and join- operations which make the set of coset-representatives of a graph into a lattice. This induces a lattice structure on the set of symmetries of G in a natural way. We will make use of the following proposition from lattice theory:

Proposition 5.3.1: *(Corollary 2.17 in [DP90]) Let* X *be a set and* L *a family of subsets of* X*, ordered by inclusion, such that*

1. $\mathsf{X} \in \mathsf{L}$
2. $\bigcap_{i \in \mathbf{I}} \mathsf{A}_i \in \mathsf{L}$ *for every non-empty family* $\{\mathsf{A}_i\}_{i \in \mathbf{I}} \subseteq \mathsf{L}$.

Then L *is a lattice with meet and join operations given as follows:* $\mathsf{A}_i \wedge \mathsf{A}_j \equiv \mathsf{A}_i \cap \mathsf{A}_j$, *for* A_i *and* $\mathsf{A}_j \in \mathsf{L}$. $\mathsf{A}_i \vee \mathsf{A}_j \equiv \bigcap\{\mathsf{A} \in \mathsf{L} | \mathsf{A}_i \cup \mathsf{A}_j \subseteq \mathsf{A}\}$.

□

Proposition 5.3.2: *Let* G *be a network and let* C_v *be the set of coset-representatives for* G *with respect to a vertex* v*. Then* C_v *adjoin the empty set forms a lattice under the inclusion order, with the meet and join operations as in Proposition 5.3.1 above.*

EXAMPLE 5.3.1: For G as in Figure 5.1, $\mathcal{E}(\mathsf{G})$ is generated by $f_a = (1,2)(3,4)$ and $f_b = (2,3)$ and has group elements f_a, f_b, $f_c = (1,3,4,2)$, $f_d = (1,2,4,3)$, $f_e = (1,3)(2,4)$, $f_f = (1,4)(2,3)$, and $f_g = (1,4)$.

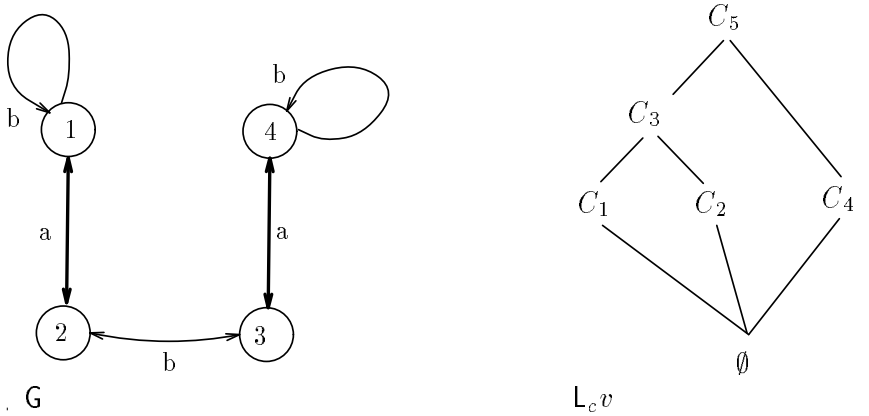


Figure 5.1: Lattice of Coset-Representatives

Choose $v = 1$. Then the symmetries and their corresponding coset-representatives are as follows:

- $\mathbf{s}_1 = \langle 1/2/3/4,\ 1/2/3/4,\ \delta_1 = id\rangle$ with coset representative $C_1 = \mathcal{E}(\mathsf{G})_v = \{id, f_b\}$,
- $\mathbf{s}_2 = \langle 1/2/3/4,\ 1/2/3/4/,\ \delta_2 = (1,4)(2,3)\rangle$ with coset representative $C_2 = f_f\mathcal{E}(\mathsf{G})_v = \{f_f, f_f f_b\} = \{f_f, f_g\}$,
- $\mathbf{s}_3 = \langle 1,4/2,3,\ 1,4/2,3,\ \delta_3 = id\rangle$ with coset representative $C_3 = \mathcal{J} = \langle id, f_b, f_f, f_g\rangle$,
- $\mathbf{s}_4 = \langle 1,4/2,3,\ 1,4/2,3,\ \delta_4 = (\{1,4\},\{2,3\})\rangle$ with coset representative $C_4 = f_a\mathcal{J} = \{f_a, f_c, f_d, f_e\}$,
- $\mathbf{s}_5 = \langle /1,2,3,4/,\ /1,2,3,4/,\ \delta_5 = id\rangle$ with coset representative $C_5 = \mathcal{E}(\mathsf{G})$.

The lattice $\mathsf{L}_c v$ of coset-representatives is pictured.

Proof of the proposition: This will follow from Proposition 5.3.1 above if $\mathsf{C}_v \cup \emptyset$ is closed under intersection. Let $\mathbf{s}_1 = \langle \pi_1, \pi_3, \delta_1\rangle$ and $\mathbf{s}_2 = \langle \pi_2, \pi_4, \delta_2\rangle$ be two symmetries, where π_1, π_2, π_3 and π_4 correspond, respectively, to subgroups $\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3$ and $\mathcal{H}_4$ of $\mathcal{E}(\mathsf{G})$; and $f_a\mathcal{H}_3$ and $f_b\mathcal{H}_4$ are the coset-representatives of $\mathbf{s}_1$ and $\mathbf{s}_2$ with respect to a vertex $v \in \mathsf{V}(\mathsf{G})$. We will show that if $f_a\mathcal{H}_3 \cap f_b\mathcal{H}_4 \neq \emptyset$ then $f_a\mathcal{H}_3 \cap f_b\mathcal{H}_4$ is a coset-representative with respect to v, and conclude that $\mathsf{C}_v \cup \emptyset$ is closed under intersection.

Suppose that $f_a\mathcal{H}_3 \cap f_b\mathcal{H}_4 \neq \emptyset$. Then $f_a\mathcal{H}_3 \cap f_b\mathcal{H}_4$ is a left coset, say $f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$, of $\mathcal{H}_3 \cap \mathcal{H}_4$. By Proposition 4.6.5 of Chapter 4, $\mathcal{H}_1 f_a = f_a\mathcal{H}_3$ and $\mathcal{H}_2 f_b = f_b\mathcal{H}_4$, and so $\mathcal{H}_1 f_a \cap \mathcal{H}_2 f_b = f_a\mathcal{H}_3 \cap f_b\mathcal{H}_4 \neq \emptyset$. The set $\mathcal{H}_1 f_a \cap \mathcal{H}_2 f_b$ is a right coset of $\mathcal{H}_1 \cap \mathcal{H}_2$; say, $\mathcal{H}_1 f_a \cap \mathcal{H}_2 f_b = (\mathcal{H}_1 \cap \mathcal{H}_2) f_d$ for some $f_d \in \mathcal{E}(\mathsf{G})$. So we have:

$(\mathcal{H}_1 \cap \mathcal{H}_2) f_d = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$. This implies that $\mathcal{H}_1 \cap \mathcal{H}_2$ and $\mathcal{H}_3 \cap \mathcal{H}_4$ are conjugate; in fact, $(\mathcal{H}_1 \cap \mathcal{H}_2) f_c = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$. (If $(\mathcal{H}_1 \cap \mathcal{H}_2) f_d = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$ then $f_d = f_c h$ for some $h \in \mathcal{H}_3 \cap \mathcal{H}_4$. Hence $(\mathcal{H}_1 \cap \mathcal{H}_2) f_c h = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$, or $(\mathcal{H}_1 \cap \mathcal{H}_2) f_c = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)h = f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$.) Since $\mathcal{E}(\mathsf{G})_v \leq \mathcal{H}_1 \cap \mathcal{H}_2$ and $\mathcal{H}_3 \cap \mathcal{H}_4$, we have by Proposition 4.6.5 in Chapter 4 that $f_c(\mathcal{H}_3 \cap \mathcal{H}_4)$ is a coset-representative for G with respect to v. □

Notation: We will write $\mathsf{L}_c v$ for the lattice of coset-representatives of a graph with respect to a vertex v.

The Lattice of Symmetries: The lattice structure of $\mathsf{L}_c v$ induces a lattice structure on the symmetries of G, as follows: Let symmetries $\mathbf{s}_1$ and $\mathbf{s}_2$ have coset-representatives C_1 and C_2, respectively. Then $\mathbf{s}_1 \preceq \mathbf{s}_2$ if $C_1 \subseteq C_2$; $\mathbf{s}_1 \vee \mathbf{s}_2$ is the symmetry with coset representative $C_1 \vee C_2$, and $\mathbf{s}_1 \wedge \mathbf{s}_2$ is the symmetry with coset-representative $C_1 \wedge C_2$.

Notation: We will write $\mathsf{L}_S v$ for the lattice of symmetries of a network with respect to a vertex v.

In Chapter 4 we saw that the lattice of block-systems is independent of the choice of a vertex $v \in \mathsf{V}(\mathsf{G})$. This also holds for the lattice of symmetries, as Proposition 5.3.3 below shows. First we will need two lemmas:

Lemma 5.3.1: *Let $f_b(v) = w$ for v and $w \in \mathsf{V}(\mathsf{G})$ and $f_b \in \mathcal{E}(\mathsf{G})$. Then if $f_a\mathcal{H}$ is the coset representative for a symmetry $\mathbf{s}$ with respect to v, then $f_b f_a \mathcal{H} f_b^{-1}$ is the coset-representative for $\mathbf{s}$ with respect to w.*

Proof Let $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$. We will show that $\delta([w]) = f_b f_a \mathcal{H} {f_b}^{-1}(w)$. Since δ is an isomorphism, we have $\delta([w]) = \delta f_b([v]) = f_b \delta([v])$ (Proposition 3.3.1). Since $f_a\mathcal{H}$ is the coset-representative of $\mathbf{s}$ with respect to v, we have $\delta([v]) = f_a\mathcal{H}(v)$. Hence $\delta([w]) = f_b\delta([v]) = f_b f_a \mathcal{H} {f_b}^{-1} f_b(v) = f_b f_a \mathcal{H} {f_b}^{-1}(w)$. Note that $f_b f_a \mathcal{H} {f_b}^{-1}$ is a left coset of $f_b \mathcal{H} {f_b}^{-1}$; namely, $f_b f_a \mathcal{H} {f_b}^{-1} = f_b f_a {f_b}^{-1} f_b \mathcal{H} {f_b}^{-1}$. By Proposition 4.2.5 in Chapter 4, $f_b \mathcal{H} {f_b}^{-1}$ is the subgroup corresponding to π_2 with respect to w. □

Lemma 5.3.2: $\mathsf{L}_c v \simeq \mathsf{L}_c w$ *for any* v *and* $w \in \mathsf{V}(\mathsf{G})$.

Proof Suppose that $w = f_b(v)$ for some $f_b \in \mathcal{E}(\mathsf{G})$. Define $\alpha : \mathsf{L}_c v \to \mathsf{L}_c w$ by: $\alpha(f_a\mathcal{H}) \to f_b f_a \mathcal{H} {f_b}^{-1}$ for each $f_a\mathcal{H} \in \mathsf{L}_c v$. By Lemma 5.3.1, if $f_a\mathcal{H}$ is the coset-representative of a symmetry $\mathbf{s}$ with respect to v then $\alpha(f_a\mathcal{H})$ is the coset-representative of $\mathbf{s}$ with respect to w. Hence α is a bijection. It remains to show that α preserves the lattice meet- and join operations. We recall a proposition from lattice theory:

1: *(Theorem 2.3 in [BS81]). A bijective map* α *between lattices is a lattice isomorphism iff both* α *and* α^{-1} *are order-preserving.*

But this is immediate, for if $f_a\mathcal{H} \subseteq f_c\mathcal{J}$ then $f_b f_a \mathcal{H} {f_b}^{-1} \subseteq f_b f_c \mathcal{J} {f_b}^{-1}$, and conversely. □

Proposition 5.3.3: $\mathsf{L}_S v = \mathsf{L}_S w$ *for any* v *and* $w \in \mathsf{V}(\mathsf{G})$*, where* $\mathsf{L}_S v$ *and* $\mathsf{L}_S w$ *are the lattices of symmetries with respect to* v *and* w*, respectively.*

Proof By Lemma 5.3.2, $\mathsf{L}_c v \simeq \mathsf{L}_c w$ via an isomorphism α. By construction, $\mathsf{L}_S v \simeq \mathsf{L}_c v$ and $\mathsf{L}_S w \simeq \mathsf{L}_c w$, so $\mathsf{L}_S v \simeq \mathsf{L}_S w$. By Lemma 5.3.1, if $f_a\mathcal{H}$ is the coset-representative of a symmetry $\mathbf{s}$ with respect to v then $\alpha(f_a\mathcal{H})$ is the coset-representative of $\mathbf{s}$ with respect to w. Hence $\mathsf{L}_S v = \mathsf{L}_S w$. □

Notation: In light of Proposition 5.3.3, we will write L_S for the lattice of symmetries of G.

The following proposition shows that the lattice of symmetries is reasonably well-behaved.[1]

Proposition 5.3.4: *Let* $\mathbf{s}_1 = \langle \pi_1, \pi_3, \delta_1 \rangle$ *and* $\mathbf{s}_2 = \langle \pi_2, \pi_4, \delta_2 \rangle$ *be symmetries in* L_S*. Then* $\mathbf{s}_1 \preceq \mathbf{s}_2$ *iff* $\pi_1 \preceq \pi_2$ *and* $\pi_3 \preceq \pi_4$ *and* δ_2 *lifts to* δ_1 *via the inclusion covering maps* $\beta_1 : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ *and* $\beta_2 : \mathsf{G}/\pi_3 \to \mathsf{G}/\pi_4$*. In particular, this means that* $\mathbf{s}_1 \preceq \mathbf{s}_2$ *iff* $\delta_1([v]_{\pi_1}) \subseteq \delta_2([v]_{\pi_2})$ *for any* $v \in \mathsf{V}(\mathsf{G})$.

Proof Let π_i correspond to $\mathcal{H}_i$ with respect to v for $i = 1, \ldots, 4$; and let $\mathbf{s}_1$ have coset-representative $f_a\mathcal{H}_3$ and $\mathbf{s}_2$ have coset-representative $f_b\mathcal{H}_4$. Suppose first that $\mathbf{s}_1 \preceq \mathbf{s}_2$. Then by definition, $f_a\mathcal{H}_3 \subseteq f_b\mathcal{H}_4$ and so $\mathcal{H}_3 \leq \mathcal{H}_4$. Since $\mathcal{H}_3$ corresponds to π_3 and $\mathcal{H}_4$ corresponds to π_4 this gives us $\pi_3 \preceq \pi_4$. Note that $\delta_2^{-1}\beta_2\delta_1([w]_{\pi_1}) \in \pi_2$ for all $w \in \mathsf{V}(\mathsf{G})$, and is a covering map, for β_2 the canonical covering map: $\mathsf{G}/\pi_3 \to \mathsf{G}/\pi_4$. Hence $\pi_1 \preceq \pi_2$. If β_1 is the canonical covering map: $\mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ then by definition we have $\beta_1(\mathcal{H}_1(v)) = \mathcal{H}_2(v)$ and $\beta_2(f_a\mathcal{H}_3(v)) = f_b\mathcal{H}_4(v)$, and so $\delta_2\beta_1(\mathcal{H}_1(v)) = f_b\mathcal{H}_4(v) =$

[1] It is not entirely well-behaved. In particular, there is not always an isomorphism δ' such that $\mathbf{s}_1 \vee \mathbf{s}_2 = \langle \pi_1 \vee \pi_2, \pi_3 \vee \pi_4, \delta' \rangle$. (For instance, consider $\mathbf{s}_1 \vee \mathbf{s}_2$ in Figure 5.1.)

$\beta_2\delta_1(\mathcal{H}_1(v))$. Since covering maps are determined by their action on a single point, this implies that $\beta_2\delta_1 = \delta_2\beta_1$.

Conversely, suppose that δ_2 lifts to δ_1 via β_1 and β_2. Then $\delta_2\beta_1([v]) = f_b\mathcal{H}_4(v) = \beta_2\delta_1([v]) = \beta_2 f_a\mathcal{H}_3(v)$, and so $f_a\mathcal{H}_3(v) \subseteq f_b\mathcal{H}_4(v)$. By Corollary 4.5.2 in Chapter 4, we have $f_a\mathcal{H}_3 \subseteq f_b\mathcal{H}_4$, and so $\mathbf{s}_1 \preceq \mathbf{s}_2$. □

Remark 5.3.1: The discussion so far yields the following two observations:
(1) If $\mathcal{E}(\mathsf{G}_1)$ and $\mathcal{E}(\mathsf{G}_2)$ are *identical* then G_1 and G_2 compute the same set of functions, since they then have the same set of symmetries.
(2) The converse is false. To find an example of graphs which compute the same functions but have different edge-label semigroups, we can look for permutation groups $(\mathcal{G}, S)$ and $(\mathcal{H}, S)$ which have no subgroups between $\mathcal{G}$ and $\mathcal{G}_v$ and between $\mathcal{H}$ and $\mathcal{H}_v$, respectively, for some $v \in S$, so that the two corresponding operator graphs have only the trivial symmetry. For instance, let G_1 and G_2 be as follows: $\mathsf{V}(\mathsf{G}_1) = \mathsf{V}(\mathsf{G}_2) = \{u, v, w, r\}$; $\mathcal{E}(\mathsf{G}_1) = Sym(\mathsf{V}(\mathsf{G}))$, and $\mathcal{E}(\mathsf{G}_2) = A_4$. Then $\mathcal{E}(\mathsf{G}_1)_v$ is the symmetric group on three letters and $\mathcal{E}(\mathsf{G}_2)_v = A_3 = Z_3$, and G_1 and G_2 have the same symmetries $\mathbf{s}_1 = \langle 1/2/3/4,\ 1/2/3/4,\ id\rangle$ and $\mathbf{s}_2 = \langle /1,2,3,4/,\ /1,2,3,4/,\ id\rangle$.

This example also shows that networks can have different lattices of coset representatives but share the same lattice of symmetries. That is, $\mathsf{L}_c v$ and $\mathcal{E}(\mathsf{G})$ over-determine the network.

The definition of lattice-join as it is given above has a disadvantage: $\mathbf{s}_1 \vee \mathbf{s}_2$ can be computed only if all of L_S is known. In the next section we will need a definition of join which depends only on $\mathbf{s}_1$ and $\mathbf{s}_2$. How might such a definition be given? Consider the following example. Let $\mathbf{s}_1 = \langle \pi_1,\ \pi_3,\ \delta_1\rangle$ and $\mathbf{s}_2 = \langle \pi_2,\ \pi_4,\ \delta_2\rangle$, and $\mathbf{s}_1 \vee \mathbf{s}_2 = \langle \pi_5,\ \pi_6,\ \delta_3\rangle$, where

$\pi_1 = \pi_3 = \pi_4 = 1,2/3,8/4,7/5,6$; $\pi_2 = 3,4/2,5/1,6/7,8$, and $\delta_1 : \{1,2\} \to \{5,6\}$; $\{3,8\} \to \{4,7\}$; $\{4,7\} \to \{3,8\}$; $\{5,6\} \to \{1,2\}$; and

$\delta_2 : \{3,4\} \to \{1,2\}$; $\{2,5\} \to \{3,8\}$; $\{1,6\} \to \{4,7\}$, and $\{7,8\} \to \{5,6\}$.

By Proposition 5.3.4, δ_3 lifts to δ_1 and δ_2 via the inclusion covering maps. This means, for instance, that since $\delta_1(\{1,2\}) = \{5,6\}$ and $\delta_2(\{1,6\}) = \{4,7\}$, that δ_3 must map $\{1,2\}$ to $\{5,6\}$ *and* map $\{1,6\}$ to $\{4,7\}$. That is, δ_3 must map a block containing $\{1,2,6\}$ to a block containing $\{5,6,4,7\}$. Similarly, since $\delta_1(\{3,8\}) \cap \delta_2(\{1,6\}) \neq \emptyset$, δ_3 must map a block containing $\{1,6,3,8\}$ to a block containing $\{4,7\} = \delta_1(\{3,8\}) \cup \delta_2(\{1,6\})$. Continuing with the same line of reasoning, we find that $\pi_5 = \pi_6 = /1,2,\cdots,8/$ and that δ_3 is the identity map.

This suggests a definition:

Definition 5.3.2: Let $\mathbf{s}_1 = \langle \pi_1,\ \pi_3,\ \delta_1\rangle$ and let $\mathbf{s}_2 = \langle \pi_2,\ \pi_4,\ \delta_2\rangle$. Define $\mathbf{s}_1 \vee' \mathbf{s}_2 = \langle \pi_5,\ \pi_6,\ \delta_3\rangle$ as follows:

- i and j are in the same block of π_5 if any of the following three conditions hold: (1) i and j are in the same block of π_1 or in the same block of π_2, (2) $[i]_{\pi_1} \cap [j]_{\pi_2} \neq \emptyset$, or (3) $\delta_1([i]_{\pi_1}) \cap \delta_2([j]_{\pi_2}) \neq \emptyset$.
- i and j are in the same block of π_6 if any of the following three conditions hold: (4) i and j are in the same block of π_3 or in the same block of π_4, or (5) $[i]_{\pi_3} \cap [j]_{\pi_4} \neq \emptyset$, or (6) $\delta_1^{-1}([i]_{\pi_3}) \cap \delta_2^{-1}([j]_{\pi_4}) \neq \emptyset$.

- Define $\delta_3 = \beta_3\delta_1\beta_1^{-1}$, where $\beta_1 : \pi_1 \to \pi_5$ and $\beta_3 : \pi_3 \to \pi_6$ are the inclusion maps.

We can now show the following:

Proposition 5.3.5: $\mathbf{s}_1 \vee \mathbf{s}_2 = \mathbf{s}_1 \vee' \mathbf{s}_2$ *for any* $\mathbf{s}_1$ *and* $\mathbf{s}_2 \in \mathsf{L}_S$.

Proof Let $\mathbf{s}_1 \vee \mathbf{s}_2 = \langle \pi_a, \pi_b, \delta \rangle$. We will show first that $\mathbf{s}_1 \vee' \mathbf{s}_2$ is a symmetry, and then that $\mathbf{s}_1 \vee' \mathbf{s}_2 = \mathbf{s}_1 \vee \mathbf{s}_2$.

First, π_5 is a c-partition: Let $f_a \in \mathcal{E}(\mathsf{G})$ and choose points i and j in a block of π_5. Assume first that one of the conditions 1, 2, 3 from the definition holds for the pair (i, j). If condition 1 holds then $f_a(i)$ and $f_a(j)$ are in the same block of π_1 (or π_2) and so are in the same block of π_5. If condition (2) holds then $[f_a(i)]_{\pi_1} \cap [f_a(j)]_{\pi_2} \neq \emptyset$, and so $f_a(i)$ and $f_a(j)$ are in the same block of π_5. If condition (3) holds then $f_a(\delta_1([i]_{\pi_1}) \cap \delta_2([j]_{\pi_2})) \neq \emptyset$. Since δ_1 and δ_2 are isomorphisms, they commute with f_a, and so $(\delta_1(f_a[i])_{\pi_1}) \cap \delta_2(f_a[j])_{\pi_2})) \neq \emptyset$, and $f_a(i)$ and $f_a(j)$ are in the same block of π_5. If i and j do not satisfy any of conditions 1, 2, or 3 then there is a sequence $i = i_1, i_2, \ldots, i_k = j$ of points in C such that each pair (i_l, i_{l+1}) for $l = 1, \ldots, k-1$ satisfies one of conditions 1, 2 or 3. Then by the above, for each pair (i_l, i_{l+1}) in a block of π_5, the pair $(f_a(i), f_a(j))$ is in the same block of π_5, and π_5 is a c-partition.

The same argument shows that π_6 is a c-partition.

δ_3 is an isomorphism: First, δ_3 commutes with the elements of $\mathcal{E}(\mathsf{G})$ since β_1, β_2, and δ_1 do. That is, δ_3 is a covering map. By construction, δ_3 lifts to δ_1. Note also that $\beta_3\delta_1\beta_1^{-1} = \beta_4\delta_2\beta_2^{-1}$, where $beta_2 : \mathsf{G}/\pi_2 \to \mathsf{G}/\pi_5$ and $\beta_4 : \mathsf{G}/\pi_4 \to \mathsf{G}/\pi_6$ are the inclusion covering maps. To show this, choose $[i] \in \pi_1$ and $[j] \in \pi_2$ such that $[i]_{\pi_1} \cap [j]_{\pi_2} \neq \emptyset$. Then i and j are in the same block C of π_5 by condition (3) of the definition of π_5, and $\beta_3\delta_1\beta_1^{-1}(\mathsf{C}) = \beta_4\delta_2\beta_2^{-1}(\mathsf{C})$ by condition (6). That is, δ_3 lifts to δ_2 also.

It remains to show that δ_3 is well-defined and is a bijection. To show that δ_3 is one-to-one, suppose that $\delta_3(\mathsf{B}_1) = \delta_3(\mathsf{B}_2) = \mathsf{C}$ for blocks B_1 and $\mathsf{B}_2 \in \pi_5$ and $\mathsf{C} \in \pi_6$. If a pair of vertices i and j in C satisfies one of conditions 4, 5, or 6 in the definition then $\delta_3^{-1}([i])i$ and $\delta_3^{-1}([j])$ are the same block of π_5 by construction. If $\delta_3([i])$ and $\delta_3([j])$ do not satisfy any of conditions 4, 5, or 6 then there is a sequence $i = i_1, i_2, \ldots, i_k = j$ of points in C such that each pair $(\delta_3^{-1}([i_l]), \delta_3^{-1}([i_{l+1}])$ for $l = 1, \ldots, k-1$ satisfies one of conditions 4, 5 or 6. Then each pair (i_l, i_{l+1}) is in the same block of π_5, so i and j are in the same block of π_5 also, by transitivity. That is, any two vertices in C pull back via δ_3 to the same block, so $\mathsf{B}_1 = \mathsf{B}_2$.

A similar argument shows that δ_3 is well-defined. By construction, δ_3 is onto. Since δ_3 lifts to both δ_1 and δ_2 we can use Proposition 5.3.4 to conclude that $\mathbf{s}_1 \preceq \mathbf{s}_1 \vee' \mathbf{s}_2$ and $\mathbf{s}_2 \preceq \mathbf{s}_1 \vee' \mathbf{s}_2$.

It remains to argue that $\mathbf{s}_1 \vee' \mathbf{s}_2 = \mathbf{s}_1 \vee \mathbf{s}_2$. This holds by construction, for since $\delta \in \mathbf{s}_1 \vee \mathbf{s}_2$ lifts to δ_1 and δ_2, we must have $\mathbf{s}_1 \vee \mathbf{s}_2$ satisfying the conditions $1-6$ of the definition of $\mathbf{s}_1 \vee' \mathbf{s}_2$. Thus, if $[i]_{\pi_5} = [j]_{\pi_5}$ then $[i]_{\pi_a} = [j]_{\pi_a}$, and if $[i]_{\pi_6} = [j]_{\pi_6}$ then $[i]_{\pi_b} = [j]_{\pi_b}$; i.e., $\pi_5 \preceq \pi_a$ and $\pi_6 \preceq \pi_b$, and $\delta_3([i]_{\pi_5}) \subseteq \delta([i]_{\pi_a})$ for all $i \in \mathsf{V}(\mathsf{G})$. Hence δ lifts to δ_3. By Proposition 5.3.4 this implies that $\mathbf{s}_1 \vee' \mathbf{s}_2 \preceq \mathbf{s}_1 \vee \mathbf{s}_2$, and so by the definition of the lattice join, the two symmetries are equal. □

The definition of $\mathbf{s}_1 \vee \mathbf{s}_2$ given above does not depend on $\mathcal{E}(\mathsf{G})$ or, for that matter, on G, but only on $\mathbf{s}_1$ and $\mathbf{s}_2$. Therefore we have the following corollary to Proposition 5.3.5:

Corollary 5.3.1: *Let* $\mathbf{s}_a^1 = \mathbf{s}_a^2$ *and* $\mathbf{s}_b^1 = \mathbf{s}_b^2$ *for symmetries* $\mathbf{s}_a^1$ *and* $\mathbf{s}_b^1$ *of a graph* G_1 *and* $\mathbf{s}_a^2$ *and* $\mathbf{s}_b^2$ *of a graph* G_2. *Then* $\mathbf{s}_a^1 \vee \mathbf{s}_b^1 = \mathbf{s}_a^2 \vee \mathbf{s}_b^2$.

□

5.4 Generators For The Lattice Of Symmetries

Although the lattice of symmetries of a network can be large, it contains a small subset, the "constraint set", which generates L_S under the lattice join-operation. In this section we will describe the constraint-set and show the following: (1) The constraint-set has at most $|\mathsf{V}(\mathsf{G})|$ members, and (2) Two networks have identical symmetry sets iff they have identical constraint-sets. Thus the constraint-set can be used in place of the symmetry-set in classifying networks.

Definition 5.4.1: For $i = \{1, \ldots, n\}$, the *minimal coset representative of* i in the lattice $\mathsf{L}_c v$ of coset-representatives is the smallest coset $f_a\mathcal{H} \in \mathsf{L}_c v$ such that $i \in f_a\mathcal{H}(v)$. The *minimal symmetry* or *constraint* of i is the corresponding symmetry. The *constraint-set* of G with respect to v is the collection of all of the minimal symmetries (constraints) in L_S.

Example 5.4.1: For instance, in Example 5.3.1, the minimal symmetry of the vertex 4 with respect to vertex 1 is $\mathbf{s}_2 = \langle 1/2/3/4,\ 1/2/3/4,\ \delta = (1,4)(2,3)\rangle$, since $f_f\mathcal{E}(\mathsf{G})_v$ is the smallest coset representative containing the vertex 4. The constraint-set of G with respect to vertex 1 in Example 5.3.1 is $\{\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_4\}$.

Proposition 5.4.1: *If* G *has* n *vertices, its constraint-set has at most* n *elements. The constraint-set of* G *with respect to a vertex* v *generates the lattice of symmetries under the lattice join.*

Proof First of all, for each $i \in \mathsf{V}(\mathsf{G})$, the minimal coset-representative C_i, and hence the minimal symmetry of i, is unique: For if $i \in f_a\mathcal{H}(v)$ and $i \in f_b\mathcal{J}(v)$ then $i \in f_a\mathcal{H}(v) \cap f_b\mathcal{J}(v)$, which equals $(f_a\mathcal{H} \cap f_b\mathcal{J})(v)$ by Corollary 4.5.2 in Chapter 4. By Proposition 5.3.2, $f_a\mathcal{H} \cap f_b\mathcal{J} \in \mathsf{L}_c v$. Thus the constraint-set has at most n elements.

We next show that $\mathsf{L}_c v$ is generated by the set of minimal coset-representatives under lattice join. From this it follows that L_S is generated by the set of minimal symmetries under lattice join.

Let $f_a\mathcal{H} \in \mathsf{L}_c v$ and suppose that $f_a\mathcal{H}(v) = \{i_1, i_2, \ldots, i_k\}$. We will show that $f_a\mathcal{H} = f_{a_1}\mathcal{H}_1 \vee \ldots \vee f_{a_k}\mathcal{H}_k$, where $f_{a_j}\mathcal{H}_j$ is the minimal coset-representative of i_j for $j = 1, \ldots, k$. First, $f_{a_j}\mathcal{H}_j \subseteq f_a\mathcal{H}$ since $f_{a_j}\mathcal{H}_j$ is the smallest coset such that $i_j \in f_{a_j}\mathcal{H}_j(v)$, and so $f_{a_j}\mathcal{H}_j \cap f_a\mathcal{H} = f_{a_j}\mathcal{H}_j$. This implies that $f_{a_1}\mathcal{H}_1 \vee \ldots \vee f_{a_k}\mathcal{H}_k \subseteq f_a\mathcal{H}$. Now let $g \in f_a\mathcal{H}$. Then $g(v) = i$ for some $i \in \{i_1, \ldots, i_k\}$, that is, $g(v) \in f_{a_i}\mathcal{H}_i(v)$ for some $f_{a_i}\mathcal{H}_i \in \{f_{a_1}\mathcal{H}_1, \ldots, f_{a_k}\mathcal{H}_k\}$. Since $\mathcal{H}_i(v)$ and $f_{a_i}\mathcal{H}_i(v)$ are blocks in the same block-system, we must have $g\mathcal{H}_i(v) = f_{a_i}\mathcal{H}_i(v)$. Hence $g \in f_{a_i}\mathcal{H}_i$ by Proposition 4.2.6 in Chapter 4, and so $f_a\mathcal{H} \subseteq f_{a_1}\mathcal{H}_1 \vee \ldots \vee f_{a_k}\mathcal{H}_k$. Hence $f_a\mathcal{H} = f_{a_1}\mathcal{H}_1 \vee \ldots \vee f_{a_k}\mathcal{H}_k$, and the set $\{f_{a_1}\mathcal{H}_1, \ldots, f_{a_k}\mathcal{H}_k\}$ generates $\mathsf{L}_c v$ under lattice-join. □

Example 5.4.2: Referring back to example 4.5.1 in Chapter 4:

Let G be the Cayley graph of D_8 acting on itself by left multiplication, with generators f_2 and f_8, for $f_2 = (1,2)(3,4)(5,6)(7,8)$ and $f_8 = (1,8)(2,3)(4,5)(6,7)$.

Then for instance if $\mathbf{s} = \langle 1,2/3,8/4,7/5,6 \quad 1,6/2,5/3,4/7,8,\ \delta\rangle$ with $\delta : \{1,2\} \rightarrow \{3,4\}; \{3,8\} \rightarrow \{2,5\}; \{4,7\} \rightarrow \{1,6\}$ and $\{5,6\} \rightarrow \{7,8\}$, then $\mathbf{s}$ has coset representative $f_3\langle id, f_6\rangle$ with respect to vertex 1,

and $\mathbf{s} = \mathbf{s}_1 \vee \mathbf{s}_2$, where

$\mathbf{s}_1 = \langle \pi, \pi, \delta_1 = (1,3,5,7)(2,4,6,8)\rangle$ with coset representative $f_3\{id\}$

and

$\mathbf{s}_2 = \langle \pi,\ \pi,\ \delta_2 = (1,4)(2,3)(5,8)(6,7)\rangle$ with coset representative $f_4\{id\}$,

where $\pi = 1/2/3/4/5/6/7/8$.

The next lemma shows that the minimal symmetries of G can be characterized without reference to the coset representatives.

Lemma 5.4.1: *Let* $\mathbf{s} = \langle \pi_1, \pi_2, \delta\rangle$ *be a symmetry of* G. *Then* $\mathbf{s}$ *is the minimal symmetry of* w *with respect to* v *iff* $w \in \delta([v]_{\pi_1})$ *and for all symmetries* $\mathbf{s}' = \langle \pi_3, \pi_4, \delta'\rangle$ *for which* $\mathbf{s}' \preceq \mathbf{s}$ *in* L_S, *if* $w \in \delta'([v]_{\pi_3})$ *then* $\mathbf{s}' = \mathbf{s}$.

Proof Let $\mathbf{s}$ be a symmetry having coset representative $f_a\mathcal{H}$ with respect to v. Suppose first that $\mathbf{s}$ is the minimal symmetry of w with respect to v, so that $f_a\mathcal{H}$ is the smallest coset representative of a symmetry of G for which $w \in f_a\mathcal{H}(v)$. Let $\mathbf{s}' \preceq \mathbf{s}$ for $\mathbf{s}' = \langle \pi_3, \pi_4, \delta'\rangle$; let $\mathbf{s}'$ have coset representative $f_b\mathcal{J}$ with respect to v, and suppose that $w \in \delta'([v]_{\pi_3}) = f_b\mathcal{J}(v)$. Since $\mathbf{s}' \preceq \mathbf{s}$ we have $f_b\mathcal{J} \subseteq f_a\mathcal{H}$, by definition. Since $f_a\mathcal{H}$ is the smallest coset representative for which $w \in f_a\mathcal{H}(v)$, we have $f_b\mathcal{J} = f_a\mathcal{H}$ and $\mathbf{s}' = \mathbf{s}$.

Conversely, suppose that for all symmetries $\mathbf{s}' = \langle \pi_3, \pi_4, \delta'\rangle$ for which $\mathbf{s}' \preceq \mathbf{s}$, if $w \in \delta'([v]_{\pi_3})$ then $\mathbf{s}' = \mathbf{s}$. Let $\mathbf{s}'$ be one such symmetry, having coset representative $f_b\mathcal{J}$ with respect to v. Then if $w \in f_b\mathcal{J}(v)$ then $f_b\mathcal{J} = f_a\mathcal{H}$ by hypothesis. Thus $f_a\mathcal{H}$ is the smallest coset representative for which $w \in f_a\mathcal{H}(v)$, and so $\mathbf{s}$ is the minimal symmetry of w with respect to v. □

The next proposition shows that the constraint set, like the lattice of symmetries, is independent of the choice of vertex v.

Proposition 5.4.2: *For any* $v, w \in \mathsf{V}(\mathsf{G})$, *the constraint-set of* G *with respect to* v *equals the constraint-set of* G *with respect to* w.

Proof Let $\mathbf{s} = \langle \pi_1, \pi_2, \delta\rangle$ be the minimal symmetry of a vertex u with respect to v, and let $f_c(v) = w$. We claim that $\mathbf{s}$ is the minimal symmetry of $f_c(u)$ with respect to w. By Lemma 5.3.2, if $f_a\mathcal{H}$ is the coset representative of $\mathbf{s}$ with respect to v then $f_c f_a \mathcal{H} f_c^{-1}$ is the coset representative of $\mathbf{s}$ with respect to w. Now $\mathbf{s}$ being the minimal symmetry of u with respect to v means that there is an element $h \in \mathcal{H}$ for which $f_a h(v) = u$. Then $f_c f_a h f_c^{-1}(w) = f_c(u)$, so $f_c(u) \in f_c f_a \mathcal{H} f_c^{-1}(w)$. It is easy to see that $f_c f_a \mathcal{H} f_c^{-1}$ is the smallest coset representative of $f_c(u)$ with respect to w. Thus $\mathbf{s}$ is the minimal symmetry of $f_c(u)$ with respect to w, as claimed. Since f_c is a bijection on $\mathsf{V}(\mathsf{G})$, it induces a bijection between the constraints of G with respect to v and the constraints of G with respect to w. □

Corollary 5.4.1: *Two networks have identical symmetry-sets iff they have identical constraint-sets. Hence two networks compute the same functions iff they have the same constraint-sets.*

Proof Let G_1 and G_2 be networks. By Corollary 5.3.1, if $\mathbf{s}_a^1 = \mathbf{s}_a^2$ and $\mathbf{s}_b^1 = \mathbf{s}_b^2$ for symmetries $\mathbf{s}_a^1$ and $\mathbf{s}_b^1$ of G_1 and $\mathbf{s}_a^2$ and $\mathbf{s}_b^2$ of G_2, then $\mathbf{s}_a^1 \vee \mathbf{s}_b^1 = \mathbf{s}_a^2 \vee \mathbf{s}_b^2$. Hence if G_1 and G_2 have the same constraint-sets then they generate the same lattice of symmetries under the lattice join.

Conversely, suppose that G_1 and G_2 have identical symmetry-sets. Let $\mathbf{s} = \langle \pi_1, \pi_2, \delta \rangle$ amd $\mathbf{s}' = \langle \pi_3, \pi_4, \delta' \rangle$ be symmetries of both G_1 and G_2. By Proposition 5.3.4, $\mathbf{s}' \preceq \mathbf{s}$ iff $\delta([v]_{\pi_1}) \subseteq \delta'([v]_{\pi_3})$ for all $v \in \mathsf{V}(\mathsf{G}_1) = \mathsf{V}(\mathsf{G}_2)$. Hence $\mathbf{s}' \preceq \mathbf{s}$ for $\mathbf{s}'$ and $\mathbf{s}$ in the symmetry-set of G_1 iff $\mathbf{s}' \preceq \mathbf{s}$ for $\mathbf{s}'$ and $\mathbf{s}$ in the symmetry-set of G_2. By Proposition 5.4.2, $\mathbf{s}$ is the minimal symmetry of w with respect to v in G_1 (respectively, G_2) if for all symmetries $\mathbf{s}'$ of G_1 (respectively, of G_2) for which $\mathbf{s}' \preceq \mathbf{s}$, if $w \in \delta'([v]_{\pi_3})$ then $\mathbf{s}' = \mathbf{s}$. Since G_1 and G_2 have the same symmetries this means that $\mathbf{s}$ is the minimal symmetry of w with respect to v in G_1 iff it is the minimal symmetry of w with respect to v in G_2. □

Remark 5.4.1: As a corollary of the above we can conclude that if two networks compute the same set of functions then they have isomorphic automorphism groups, since the symmetries $\langle \pi_1, \pi_2, \delta \rangle$ in which δ is an automorphism of G are always constraints of G.

5.5 Computing Constraints

In this section we will give an algorithm for computing the constraints of a network. The algorithm runs in time polynomial in the number of edges of a network, so in light of Corollary 5.4.1 above, two networks can be checked for f-equivalence in polynomial time. The algorithm makes use of M. D. Atkinson's algorithm for finding the smallest block of a block-system of a group containing a pair of points. For reference, we give Atkinson's algorithm below. Atkinson's algorithm is polynomial in $|X|$ for a permutation group $\mathcal{G}$ with generator-set X acting on a set $\{1, \ldots, n\}$.

Let $\mathcal{G} = \langle X \rangle$ be a permutation group on $\{1, \ldots, n\}$, and let $\{v, w\} \subseteq \{1, \ldots, n\}$.

Algorithm 5.5.1: (Atkinson's Algorithm) [2] Given a finite permutation group $\mathcal{G}$ on a set $\{1, \ldots, n\}$; a pair $(v, w) : v, w \in \{1, \ldots, n\}$, and a set of generators X for $\mathcal{G}$, find the block-system with the smallest block containing v and w, as follows:

Use the "orbit finding procedure" described below to find a relation A on $\{1, \ldots, n\}$, where $(i, j) \in A$ iff there is an element $g \in \mathcal{G}$ such that $g(v) = i$ and $g(w) = j$. Construct a graph G from A such that $\mathsf{V}(\mathsf{G}) = \{1, \ldots, n\}$ and $\mathsf{E}(\mathsf{G}) = \{(i, j) \in A\}$. Then the connected components of G form a block-system, in fact, the block-system with the smallest block containing $\{v, w\}$.

Procedure for finding A: Given a set X of generators for $\mathcal{G}$ and the pair (v, w), find $A = \{(g(v), g(w) : g \in \mathcal{G}\}$:

Find the orbit of the pair (v, w) under the action of $\mathcal{G}$ on the set of all pairs in $\{1, \ldots, n\}$, as follows:

- $A_1 = \{(v, w)\}$.

[2] Eugene Luks attributes this algorithm to C. C. Sims, in [Luk90]. The version of the algorithm we use is from [Luk90].

- For $i = 2$ until $A_{i-1} = A_i$;
  $A_i = A_{i-1} \cup \{(g(r), g(s)) : (r,s) \in A_{i-1} \text{ and } g \in X\}$.
- $A_i = A$.

□

Theorem 5.5.1: *Fix a vertex $v \in \mathsf{V}(\mathsf{G})$. There is an algorithm polynomial in the number m of edges of G for finding the minimal symmetry in $\mathsf{L}_c v$ of a vertex w, given $\mathsf{V}(\mathsf{G})$ and the set $\{f_a : a \in \mathsf{A}(\mathsf{G})\}$ of generators for $\mathcal{E}(\mathsf{G})$ as input.*

The idea behind the algorithm is fairly simple: We first construct a relation A on $\mathsf{V}(\mathsf{G})$ which is the "closest possible approximation" to an automorphism: $\mathsf{G} \to \mathsf{G}$ taking v to w. A pair (i,j) of vertices is in A if the proposed automorphism maps i to j. Next, we construct block-systems π_1 and π_2 such that A induces a bijection δ between the blocks of $\pi_1$and π_2. Then $\langle \pi_1, \pi_2, \delta \rangle$ is the desired minimal symmetry.

How are A, π_1, and π_2 constructed? If there did exist an automorphism $\delta : \mathsf{G} \to \mathsf{G}$ mapping v to w, it would satisfy: $\delta f_a(v) = f_a \delta(v) = f_a(w)$ for all $f_a \in \mathcal{E}(\mathsf{G})$. Hence A must contain all pairs $\{(f_a(v), f_a(w)) : f_a \in \mathcal{E}(\mathsf{G})\}$. If A is to induce a bijection between $\pi_1$and π_2, it must map each block of π_1 to a block of π_2. For this to hold, π_1 must satisfy: If (i,k) and $(j,k) \in A$ then i and j are in the same block of π_1. In the same way A must "preserve" the blocks of π_2, so if (k,i) and $(k,j) \in A$ then i and j must be in the same block of π_2.

Notice that the relation A consisting of all pairs $\{f_a(v), f_a(w) : f_a \in \mathcal{E}(\mathsf{G})\}$ is precisely the relation A from Atkinson's algorithm. The relations $R_1 = \{(i,j) : (i,k) \text{ and } (j,k) \in A\}$ and $R_2 = \{(i,j) : (k,i) \text{ and } (k,j) \in A\}$ used to construct π_1 and π_2 are easily derived from A.

More formally, we have:

Algorithm 5.5.2: (For finding the minimal symmetry of a vertex w with respect to a vertex v:)

Step 1: Run the procedure in Atkinson's algorithm for finding the relation $A = \{(f_a(v), f_a(w)) : f_a \in \mathcal{E}(\mathsf{G})\}$ given the pair (v,w) and the generator-set $\{f_a : a \in \mathsf{A}(\mathsf{G})\}$ for $\mathcal{E}(\mathsf{G})$.

Step 2: Construct relations R_1 and R_2 from A as follows: $(i,j) \in R_1$ iff (i,k) and $(j,k) \in A$ for some $k \in \mathsf{V}(\mathsf{G})$, and $(i,j) \in R_2$ iff (k,i) and $(k,j) \in A$ for some $k \in \mathsf{V}(\mathsf{G})$.

Step 3: Construct partitions π_1 and π_2 of $\mathsf{V}(\mathsf{G})$ from R_1 and R_2 as follows: If $(i,j) \in R_1$ then i and j are in the same block of π_1; and if $(i,j) \in R_2$ then i and j are in the same block of π_2.

Step 4: Let $\delta : \pi_1 \to \pi_2$ be defined by: $\delta(\mathsf{B}_i) = \mathsf{C}_j$ if there are vertices $i \in \mathsf{B}_i$ and $j \in \mathsf{C}_j$ such that $(i,j) \in A$.

Then $\langle \pi_1, \pi_2, \delta \rangle$ is the desired minimal symmetry. □

EXAMPLE 5.5.1: Referring back to Examples 5.3.1 and 5.4.1 again:

Take $v = 1$ and $w = 2$. Using the procedure for finding A in Atkinson's algorithm, we find that A includes the pairs $(1,2)$, $(2,1) = (f_a(1), f_a(2))$, $(1,3) = (f_b(1), f_b(2))$, $(2,4) = (f_a(1), f_a(3))$, $(3,1) = (f_b(2), f_b(1))$, and so on. This gives us

- $A = \{(1,2),(3,1),(1,3),(2,4),(4,2),(3,4),(4,3)\}$,

- $R_1 = \{(1,1),(2,2),(3,3),(4,4),(1,4),(4,1),(2,3),(3,2)\} = R_2$,
- δ maps $\{1,4\}$ to $\{2,3\}$ and $\{2,3\}$ to $\{1,4\}$.

(See Figure 5.2.)

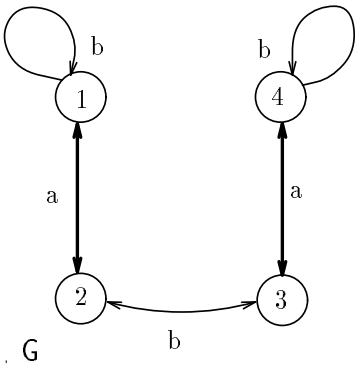


Figure 5.2: The generators for $\mathcal{E}(\mathsf{G})$ are $f_a = (1,2)(3,4)$ and $f_b = (2,3)$.

Proof of Theorem 5.5.1:

I *The Algorithm Is Correct:*

We will show: (1) that π_1 and π_2 are block-systems, (2) that δ is an isomorphism: $\mathsf{G}/\pi_1 \rightarrow \mathsf{G}/\pi_2$ mapping $[v]$ to $[w]$, and (3) that $S = \langle \pi_1, \pi_2, \delta \rangle$ is the minimal symmetry for w with respect to v.

(1) π_1 and π_2 are block-systems: Let R_1^* and R_2^* be the transitive closures of R_1 and R_2, respectively. It is easy to verify that R_1^* and R_2^* are equivalence relations defined on all of $\mathsf{V}(\mathsf{G})$. By construction, π_1 and π_2 are the partitions associated with R_1^* and R_2^*. To show that π_1 is a block-system, it suffices to show that R_1^* is a congruence, i.e., that for all $(i,j) \in R_1^*$ and for all $f_a \in \mathcal{E}(\mathsf{G})$, the pair $(f_a(i), f_a(j))$ is in R_1^*. Now $(i,j) \in R_1$ iff there is a vertex k and elements f_b and f_c in $\mathcal{E}(\mathsf{G})$ such that $f_c(v) = i, f_b(v) = j$ and $f_b(w) = f_c(w) = k$. (Figure 5.3)

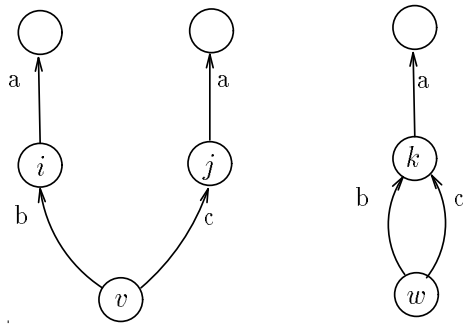


Figure 5.3: R_1^* is a congruence relation

Then
$f_a f_c(v) = f_a(i)$; $f_a f_b(v) = f_a(j)$ and $f_a f_c(w) = f_a f_b(w) = f_a(k)$, so $(f_a(i), f_a(j)) \in R_1$.

If $(i,j) \in R_1^*$ then there are pairs $\{(i,k_1),(k_1,k_2),\ldots,(k_{l-1},k_l),(k_l,j)\} \in R_1$. We showed that $\{(f_a(i),f_a(k_1)),\ldots,(f_a(k_l),f_a(j))\} \in R_1$, and so $(f_a(i),f_a(j)) \in R_1^*$ by transitivity.

The proof that π_2 is a block-system is the same.

(2) $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ is an isomorphism and $\delta([v]) = [w]$ for $[v] \in \pi_1$ and $[w] \in \pi_2$:

(a) $\delta([v]) = [w]$*:* Since $id(v) = v$ and $id(w) = w$ for the identity element id of $\mathcal{E}(\mathsf{G})$, we have $(v,w) \in A$, and so $\delta([v]) = [w]$.

(b) *δ is defined on all blocks of π_1:* Let $i \in \mathsf{V}(\mathsf{G})$ and let $f_a \in \mathcal{E}(\mathsf{G})$ map v to i. Then $(i, f_a(w)) \in A$. Hence δ is defined on $[i] \in \pi_1$.

(c) *δ is well-defined:* Suppose that $\delta(\mathsf{B}) = \mathsf{C}_1$ and $\delta(\mathsf{B}) = \mathsf{C}_2$ for $\mathsf{B} \in \pi_1$ and for C_1 and $\mathsf{C}_2 \in \pi_2$. Then there are pairs (i,k) and $(r,l) \in A$ such that $i, r \in \mathsf{B}$ and $k \in \mathsf{C}_1$, and $l \in \mathsf{C}_2$. Since i and $r \in \mathsf{B}$ there are pairs $\{(i,k_1),(k_1,k_2),\ldots,(k_{h-1},k_h),(k_h,r)\} \in R_1$ and hence elements $\{l_h\} \in \mathsf{V}(\mathsf{G})$ such that A contains all of the following pairs:

$\{(i,k),\ (i,l_1),\ (k_1,l_1),\ (k_1,l_2),\ (k_2,l_2),\ldots,(k_{h-1},l_h),(k_h,l_h),\ (k_h,l_{h+1}),\ (r,l_{h+1}),\ (r,l)\}$.

Then R_2 contains the following pairs:

$\{(k,l_1),\ (l_1,l_2),\ (l_2,l_3),\ldots,(l_h,l_{h+1}),\ (l_{h+1},l)\}$.

By transitivity, $(k,l) \in R_2^*$. That is, k and l are in the same block of π_2, and so $\mathsf{C}_1 = \mathsf{C}_2$.

(d) *δ is a bijection:* A very similar argument to that used in (c) shows that δ is one-to-one. To show that δ is onto, let $[j] \in \pi_2$. Then since G is connected, there is an element $f_a \in \mathcal{E}(\mathsf{G})$ such that $f_a(w) = j$. Hence there is a pair (i,j) in A with $i = f_a(v)$, and δ is onto.

(e) *δ commutes with the elements of $\mathcal{E}(\mathsf{G})$:* Suppose that $\delta([i]) = [j]$ for $[i] \in \pi_1$ and $[j] \in \pi_2$. Let $f_a \in \mathcal{E}(\mathsf{G})$. By the construction of δ there is an element $f_b \in \mathcal{E}(\mathsf{G})$ such that $f_b([v]) = [i]$ and $f_b([w]) = [j]$. Then $f_a f_b([v]) = f_a([i])$ and $f_a f_b([w]) = f_a([j])$, so there exists $i_1 \in f_a([i])$ and $j_1 \in f_a([j])$ such that $(i_1,j_1) \in A$. Then $\delta f_a([i]) = \delta([i_1]) = [j_1] = f_a([j]) = f_a\delta([i])$, and δ commute with the elements of $\mathcal{E}(\mathsf{G})$. (Figure 5.4)

Hence δ is an isomorphism.

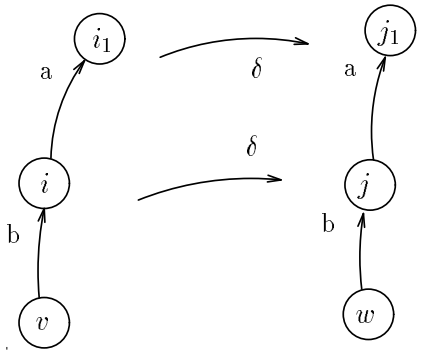


Figure 5.4: δ commutes with the elements of $\mathcal{E}(\mathsf{G})$

(3) π_1, π_2 are the finest partitions having an isomorphism $\delta : [v]_{\pi_1} \to [w]_{\pi_2}$: Suppose that there are partitions $\pi_1{}'$ and $\pi_2{}'$ having blocks D_v and F_w, respectively, with $v \in \mathsf{D}_v \subseteq [v]_{\pi_1}$ and $w \in \mathsf{F}_w \subseteq [w]_{\pi_2}$. If $\mathsf{F}_w \neq [w]_{\pi_2}$ then there is a pair $(i,j) \in R_2$ such that i is in F_w

but $j \in [w]_{\pi_2} \setminus \mathsf{F}_w$. Since $(i,j) \in R_2$ and since $\delta([v]_{\pi_1}) = [w]_{\pi_2}$, there are elements f_a and $f_b \in \mathcal{E}(\mathsf{G})$ and $k \in [v]_{\pi_1}$ such that $f_a(v) = f_b(v) = k$ and $f_a(w) = i$ and $f_b(w) = j$. (Figure 5.5) Since F_w is a block, $f_b\mathsf{F}_w$ is a block not equal to F_w and $f_a\mathsf{F}_w = \mathsf{F}_w$.

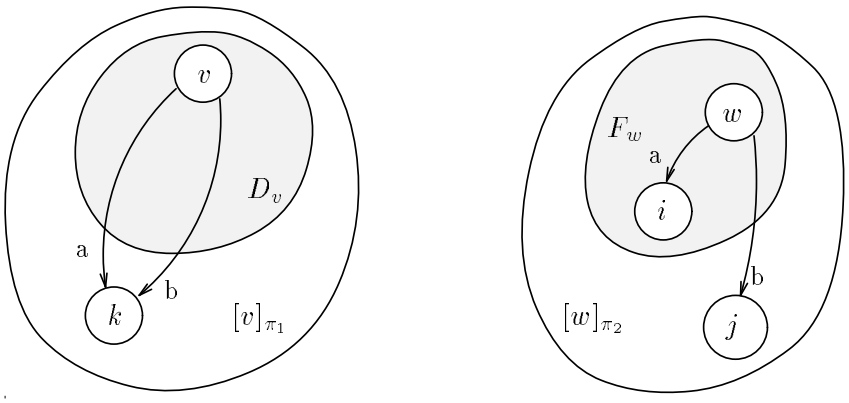


Figure 5.5: π_1 and π_2 are the finest such partitions

If $k \in \mathsf{D}_v$ then there is no isomorphism which maps D_v to F_w, since there is no function δ which both maps D_v to F_w and satisfies $\delta(f_b\mathsf{D}_v) = \delta(\mathsf{D}_v) = f_b\delta(\mathsf{D}_v)$. If $k \in [v]_{\pi_1} \setminus \mathsf{D}_v$ then there is no isomorphism which maps D_v to F_w, since there is no function δ which both maps D_v to F_w and satisfies $\delta(f_a\mathsf{D}_v) = f_a\delta(\mathsf{D}_v) = \delta(\mathsf{D}_v)$. Hence $\mathsf{F}_w = [w]_{\pi_2}$ and $\mathsf{D}_v = [v]_{\pi_1}$.

This concludes the proof of correctness.

II *The algorithm runs in time polynomial in m:* The relation A can be computed using Atkinson's algorithm in time polynomial in m. The relations R_1 and R_2 can be constructed from A in n^3 steps each: There are n^2 pairs (i,j) to check for membership, and each pair can be checked in n steps. The blocks of π_1 are the connected components of an undirected graph ?, where $\mathsf{V}(?) = \{1,\ldots,n\}$ and $\mathsf{E}(?) = R_1$. Since ? has n vertices and does not have parallel edges, these components can be found in $O(n^2)$ steps. (For instance, use a depth-first search algorithm to find a spanning tree of ? in $O(n^2)$ steps, where ? has at most n^2 edges.) The same argument shows that the blocks of π_2 can be computed in $O(n^2)$ steps. The map δ can be computed in $O(n^2)$ steps also: In n steps the algorithm can choose a representative from each block of π_1. For each representative i, the algorithm can find a pair $(i,j) \in A$ in n steps and can identify the block that j belongs to in n steps.

Since $n \leq m$, the algorithm can be executed in $p(m)$ steps for p a polynomial. □

Corollary 5.5.1: *Let G_1 and G_2 be networks with $|\mathsf{V}(\mathsf{G}_1)| = |\mathsf{V}(\mathsf{G}_2)| = n$, and let $m = \max\{|\mathsf{E}(\mathsf{G}_1)|, |\mathsf{E}(\mathsf{G}_2)|\}$, the maximum number of edges in G_1 and G_2. There is an algorithm polynomial in m for determining whether G_1 and G_2 compute the same set of functions.*

Proof By Corollary 5.4.1, two networks have identical constraint-sets iff they have identical symmetry-sets. By Theorem 3.4.2 in Chapter 3, two networks compute the same

functions iff their symmetry-sets are identical. The constraint-sets of G_1 and G_2 can be found in time polynomial in m and compared for equality in time polynomial in m. □

EXAMPLE 5.5.2: In the first chapter the claim was made that the graphs G_1 and G_2 in Figure 5.6 below compute the same set of functions. We are now in a position to prove that claim, by showing that G_1 and G_2 have identical constraint sets.

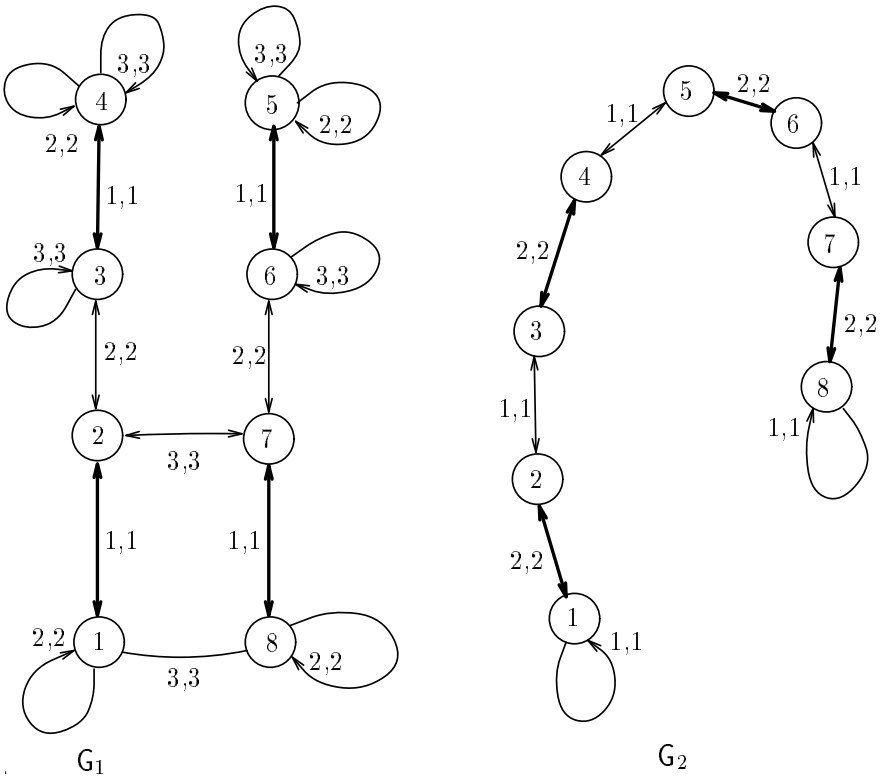


Figure 5.6: Two graphs with the same lattice of symmetries

Using the orbit-finding procedure in Atkinson's algorithm one can show, for instance, that if $v = 1$ and $w = 4$, that the relations A for G_1 and G_2 are equal and consist of the following pairs:

$$\begin{aligned} A \quad = \quad & \{(1,4),(4,1),(1,5),(5,1),(2,6),(6,2),(3,7),(7,3),(4,8), \\ & (8,4),(5,8),(8,5),(6,7),(7,6),(2,3),(3,2)\}. \end{aligned}$$

Hence
$R_1 = R_2 = \{(1,1),(2,2),\ldots,(8,8),(4,5),(5,4),(3,6),(6,3),(2,7),(7,2),(1,8),(8,1)\}$.

This yields partitions $\pi_1 = \pi_2 = 1,8/2,7/3,6/4,5$ and isomorphism $\delta = (\{1,8\},\{4,5\})(\{2,7\},\{3,6\})$. Thus the minimal symmetry for the vertex $w = 4$ is $\langle \pi_1, \pi_1, \delta \rangle$, for this π_1 and δ.

The same sort of calculations give us the following minimal symmetries for G_1 and G_2:

$1 \rightarrow 1$: δ = the identity permutation of the partition $1/2/\ldots/8$.

$1 \rightarrow 2$, $1 \rightarrow 3$, $1 \rightarrow 6$ and $1 \rightarrow 7$: $\delta = (\{1,4,5,8\},\{2,3,6,7\})$.

$1 \rightarrow 4$ and $1 \rightarrow 5$: $\delta = (\{1,8\},\{4,5\})(\{2,7\},\{3,6\})$.

5.6 Networks Differing By A Permutation–Revisited

This last section considers a slightly more general classification scheme for networks than the one previously considered. Call two networks G_1 and G_2 "p-equivalent" if there is a permutation ρ such that the functions computable by G_2 differ from those computable by G_1 by ρ (Section 3.5 In Chapter 3.) Determining whether two networks are p-equivalent under this relation seems to be harder than deciding whether they can compute the same set of functions. In fact, Theorem 5.6.1 will show that it is at least as hard as determining whether two finite permutation groups are isomorphic. As of this writing the best algorithm for the group-isomorphism problem runs in $O(n^{\lg n})$ time for groups of order n.

We will need the following lemma:

Lemma 5.6.1: *If $\mathcal{E}(\mathsf{G})$ is regular on $\mathsf{V}(\mathsf{G})$ then the automorphism group of G is isomorphic to $\mathcal{E}(\mathsf{G})$ and the constraints of G are precisely the automorphisms of G. That is, $\langle \pi_1, \pi_2, \delta \rangle$ is a constraint of G with respect to a vertex v iff $\pi_1 = \pi_2 = 1/2/\ldots/n$ and δ is an automorphism of G.*

Proof If $(\mathcal{E}(\mathsf{G}), \mathsf{V}(\mathsf{G}))$ is regular then by Propositon 4.6.1, the centralizer of $\mathcal{E}(\mathsf{G})$ in $Sym(\mathsf{V}(\mathsf{G}))$ is isomorphic to $\mathcal{E}(\mathsf{G})$. By Proposition 4.6.2 the centralizer is the automorphism group of G. Hence the automorphism group has n elements. Since all automorphisms are constraints (Remark 5.4.1) and since G has at most n constraints, the constraint-set of G equals its automorphism group. □

For instance, if G is a Cayley graph of a group $\mathcal{G}$ then the constraints of G are its automorphisms.

The group isomorphism problem is the following:

Problem: Group Isomorphism

Given: Groups $\mathcal{G}_1$ and $\mathcal{G}_2$ of order n, given as group tables.

Question: Are $\mathcal{G}_1$ and $\mathcal{G}_2$ isomorphic?

Proposition 5.6.1: *(From [Mil78])*

Group isomorphism can be solved in $O(n^{\log n + O(1)})$ steps.

We have:

Theorem 5.6.1: *There is a polynomial-time transformation of the group-isomorphism problem to the problem of determining if two networks are p-equivalent.*

Proof

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be groups of order n given by group-tables. We will construct Cayley graphs G_1 and G_2 for $\mathcal{G}_1$ and $\mathcal{G}_2$ and then rename the vertices and edge-labels so that $\mathsf{V}(\mathsf{G}_1) = \mathsf{V}(\mathsf{G}_2)$ and $\mathsf{A}(\mathsf{G}_1) = \mathsf{A}(\mathsf{G}_2)$:

Define bijections $\rho_1 : \mathcal{G}_1 \to \{1, \ldots, n\}$ and $\rho_2 : \mathcal{G}_2 \to \{1, \ldots, n\}$, where $\rho_1(g) = i$ if g is the ith entry in the group-table for $\mathcal{G}_1$, and ρ_2 is defined similarly. Construct graphs G_1 and G_2 from $\mathcal{G}_1$ and $\mathcal{G}_2$ as follows: $\mathsf{V}(\mathsf{G}_1) = \mathsf{V}(\mathsf{G}_2) = \{1, \ldots, n\}$ and $\mathsf{A}(\mathsf{G}_1) = \mathsf{A}(\mathsf{G}_2) = \{1, \ldots, n\}$. A triple $\langle i, j, k\rangle$ with $i, j, k \in \{1, \ldots, n\}$ is an edge in G_1 if ${\rho_1}^{-1}(j) \cdot {\rho_1}^{-1}(i) = {\rho_1}^{-1}(k)$. G_2 is defined similarly, i.e., $\langle i, j, k\rangle \in \mathsf{E}(\mathsf{G}_2)$ if ${\rho_2}^{-1}(j) \cdot {\rho_2}^{-1}(i) = {\rho_2}^{-1}(k)$. G_1 and G_2 can be constructed in $O(n^2)$ steps each since the group-tables for $\mathcal{G}_1$ and $\mathcal{G}_2$ each have n^2 elements.

Then G_1 and G_2 are the left Cayley graphs for $\mathcal{G}_1$ and $\mathcal{G}_2$, with the vertices and edge-labels renamed. Hence $\mathcal{E}(\mathsf{G}_1)$ and $\mathcal{E}(\mathsf{G}_2)$ are regular permutation groups, and so by Lemma 5.6.1 above, a constraint of G_1 (respectively, of G_2) is a triple $\langle \pi, \pi, \delta\rangle$ where $\pi = 1/2/\ldots/n$ and δ is a graph automorphism of G_1 (respectively, G_2). By Proposition 4.6.4 in Chapter 4, the automorphism group of G_1 is isomorphic to $\mathcal{G}_1$ and the automorphism group of G_2 is isomorphic to $\mathcal{G}_2$. Thus the constraint sets of G_1 and G_2 differ by a permutation iff $\mathcal{G}_1$ and $\mathcal{G}_2$ are isomorphic. □

6. Classifying Monoid Graphs

6.1 Introduction

In this chapter we will look again at the problem of classifying graphs by their symmetry-sets, this time extending our investigation to arbitrary monoid graphs. As before, we seek a small set of graph properties which characterize the set of graphs having the same symmetries. In Section 6.2 we will show that every monoid graph G has associated with it a group-graph G_B whose symmetries partly specify the symmetries of G: Every symmetry of G_B "extends" uniquely to a symmetry of G, and every symmetry of G, "restricted to" G_B, is a symmetry of G_B. This means that the constraints of G_B, together with instructions for extending them, completely specify the symmetries of G. In Section 6.3 we will show that finding G_B and the extensions of its constraints to symmetries of G is not hard. We conclude that classifying monoid graphs by their symmetries is not a hard problem. As in the previous chapter, the algorithms we present here are intended to be transparent, rather than optimally fast.

6.2 Symmetries In Monoid Graphs

The aim of this section is to find conditions under which two monoid graphs have the same symmetries. Suppose that a graph G has unique "coarsest" c-partition Π (cf. Proposition 3.2.2 Chapter 3). We will show the following:

1. If B is a block in Π then the edge-label monoid $\mathcal{E}(\mathsf{G})$, "restricted" to B, is a group, denoted by $\mathcal{E}(\mathsf{G})|_\mathsf{B}$. (This restriction is defined below.) $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ and $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ are isomorphic groups for any blocks B_1 and B_2 in Π.
2. Any isomorphism of quotients of G maps each block of Π to itself. In particular, the only automotphism of G/Π is the identity automorphism.
3. Let G_B be the operator graph of $\mathcal{E}(\mathsf{G})|_\mathsf{B}$ on B with respect to some generator-set. Then the symmetries of G_B are in one-to-one correspondence with the symmetries of G. In fact, each symmetry of G_B extends uniquely to a symmetry of G.

Since G_B is a group-graph, the symmetries of G_B are generated by its constraints, as in Chapter 5. The last theorem of the section uses this fact to give conditions under which two graphs have the same symmetries.

We first define the "restriction of $\mathcal{E}(\mathsf{G})$ to a block":

Definition 6.2.1: Let $\mathcal{E}(\mathsf{G})$ be the edge-label monoid of a graph G and let π be a c-partition of $\mathsf{V}(\mathsf{G})$. If B is a block of π, the *stabilizer submonoid* of B, written $\mathcal{E}(\mathsf{G})_\mathsf{B}$, is the set $\{g \in \mathcal{E}(\mathsf{G}) : g \textit{ is defined on } \mathsf{B} \textit{ and } g\mathsf{B} = \mathsf{B}\}$. It is straightforward to verify that $\mathcal{E}(\mathsf{G})_\mathsf{B}$ is a monoid. The *restriction of* $\mathcal{E}(\mathsf{G})$ *to* B, written $\mathcal{E}(\mathsf{G})|_\mathsf{B}$, is the monoid on B whose elements are the functions in $\mathcal{E}(\mathsf{G})_\mathsf{B}$ restricted to domain B.

EXAMPLE 6.2.1: For G as in Figure 6.1, $\mathcal{E}(\mathsf{G})$ is generated by $f_a : 1 \to 2; 2 \to 3; 3 \to 1$ and $f_b : 1 \to 4; 2 \to 5$, and $3 \to 6$. The "coarsest partition" is $\Pi = 1,2,3/4,5,6 = \{\mathsf{B}_1, \mathsf{B}_2\}$. Here $\mathcal{E}(\mathsf{G})_{\mathsf{B}_1} = \langle f_a \rangle$ and $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1} = \langle (1,2,3) \rangle$. One can show that $\mathcal{E}(\mathsf{G})_{\mathsf{B}_2} = \langle f_b f_a f_b{}^{-1} \rangle$, where $f_b f_a f_b{}^{-1}$ maps 4 to 5, 5 to 6, and 6 to 4; and that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_2} = \langle (4,5,6) \rangle$.

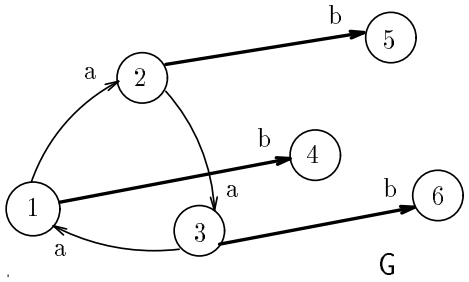


Figure 6.1: The graph for Example 6.2.1

Proposition 6.2.1 (following) shows that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ is a group, and Proposition 6.2.2 shows that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ and $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ are isomorphic for any blocks B_1 and $\mathsf{B}_2 \in \Pi$. We have:

Proposition 6.2.1: *Let* G *be a (connected) graph and let* π *be a c-partition of* $\mathsf{V}(\mathsf{G})$. *Then for any* $\mathsf{B} \in \pi$, $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ *is a transitive permutation group on* B.

Proof By definition of c-partition, if $f_a \in \mathcal{E}(\mathsf{G})$ is defined on any element of B then it is defined on all of B. Since the elements of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ are one-to-one on B, they are permutations of the elements of B, and so $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ is a group. Since G is connected, for every $v, w \in \mathsf{B}$ there is an element $f_a \in \mathcal{E}(\mathsf{G})$ such that $f_a(v) = w$. Hence since B is a block, the restriction of f_a to B is an element of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$. Thus $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ is transitive on B. □

Proposition 6.2.2: *Let* G *be a graph and let* $\pi = \{\mathsf{B}_1, \ldots, \mathsf{B}_k\}$ *be a c-partition of* $\mathsf{V}(\mathsf{G})$. *If* $f_a : \mathsf{B}_1 \to \mathsf{B}_2$ *is an element of* $\mathcal{E}(\mathsf{G})$, *define a map* $\tilde{f}_a : \mathcal{E}(\mathsf{G})|_{\mathsf{B}_1} \to \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ *by* $\tilde{f}_a(g_b) = h_c$ *iff* $f_a g_b(v) = h_c f_a(v)$ *for all* v *in* B_1. *Then* $\tilde{f}_a$ *is a group isomorphism:* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1} \to \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$.

EXAMPLE 6.2.2: In Example 6.2.1. $f_b\mathsf{B}_1 = \mathsf{B}_2$, and so, for instance, $\tilde{f}_b(f_a) = f_b f_a {f_b}^{-1}$ since $f_b f_a = f_b f_a {f_b}^{-1} f_b$.

Proof of Proposition 6.2.2 We will show first that $\tilde{f}_a$ is well-defined. Suppose that $\tilde{f}_a(g_b) = h_c$ and $\tilde{f}_a(g_b) = h_d$; that is, that $f_a g_b(v) = h_c f_a(v)$ and $f_a g_b(v) = h_d f_a(v)$ for all $v \in \mathsf{B}$, for $g_b \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ and for h_c and $h_d \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$. Then $h_c f_a(v) = h_d f_a(v)$ for all $v \in \mathsf{B}_1$. That is, $h_c f_a f_a^{-1}(w) = h_d f_a f_a^{-1}(w)$ for all $w \in \mathsf{B}_2$, so $h_c(v) = h_d(v)$ for all $v \in \mathsf{B}_1$. Thus $h_c = h_d$ and $\tilde{f}_a$ is well-defined. Next, $\tilde{f}_a$ is a homomorphism: For suppose that $\tilde{f}_a(g_b) = h_d$, and $\tilde{f}_a(g_c) = h_e$, so that $\tilde{f}_a(g_b)\tilde{f}_a(g_c) = h_d h_e$. Then $f_a g_b(v) = h_d f_a(v)$ and $f_a g_c(v) = h_e f_a(v)$ for all $v \in \mathsf{B}_1$, and so $f_a g_b(g_c(v)) = h_d f_a(g_c(v)) = h_d h_e f_a(v)$ for all $v \in \mathsf{B}_1$. Hence $\tilde{f}_a(g_b g_c) = h_d h_e$. $\tilde{f}_a$ also preserves inverses; for suppose that $\tilde{f}_a(g_b) = h_c$ and $\tilde{f}_a(g_b^{-1}) = h_d$. Then $f_a g_b(v) = h_c f_a(v)$ and $f_a g_b^{-1}(v) = h_d f_a(v)$ for all $v \in \mathsf{B}_1$. Hence $h_c = f_a g_b {f_a}^{-1}$ and $h_d = f_a g_b^{-1} {f_a}^{-1}$ on B_2, or $h_c h_d = f_a g_b {f_a}^{-1} f_a g_b^{-1} {f_a}^{-1} = id$ on B_2, so $h_d = h_c^{-1}$. Thus $\tilde{f}_a$ is a homomorphism. Finally, $\tilde{f}_a$ is a bijection; for if $g_b \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ then the restriction of $f_a^{-1} g_b f_a$ to B_1 is an element of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$. Then $f_a(f_a^{-1} g_b f_a)(v) = g_b f_a(v)$ for all $v \in \mathsf{B}_1$, and so $\tilde{f}_a({f_a}^{-1} g_b f_a) = g_b$, and $\tilde{f}_a$ is onto. $\tilde{f}_a$ is one-to-one since f_a is. Thus $\tilde{f}_a$ is a group isomorphism, as claimed. □

Let π be a c-partition of G. The next proposition shows that a c-partition of a block of π extends uniquely to a c-partition of G. First we will show, in the following lemma, that a c-partition of one block of π "extends" to a c-partition of another block of π.

Lemma 6.2.1: *Let* G *have c-partition* $\pi = \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$ *and let* $\pi' = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_j\}$ *be a c-partition of* B_1 *with respect to* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$. *Then if* $f_a \in \mathcal{E}(\mathsf{G})$ *maps* B_1 *to* B_2, *then* $\{f_a\mathsf{C}_1, \ldots, f_a\mathsf{C}_j\}$ *is a c-partition of* B_2 *with respect to* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$.

Proof Let $\mathsf{C} \in \pi'$. We show that $f_a\mathsf{C}$ is preserved by all $g_b \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$. Let $g_b \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ and let h_c be the restriction of $f_a^{-1}g_bf_a$ to B_1. Then $h_c \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ and $f_ah_c(v) = g_bf_a(v)$ on all points v in B_1. Since π' is a c-partition of B_1, either $h_c\mathsf{C} = \mathsf{C}$ or $\mathsf{C} \cap h_c\mathsf{C} = \emptyset$. Since $g_b(w) = f_ah_cf_a^{-1}(w)$ for all $w \in \mathsf{B}_2$, we have $g_bf_a\mathsf{C} \cap f_a\mathsf{C} = f_ah_cf_a^{-1}f_a\mathsf{C} \cap f_a\mathsf{C}$ $= f_ah_c\mathsf{C} \cap f_a\mathsf{C} = f_a(h_c\mathsf{C} \cap \mathsf{C})$ because f_a is a bijection: $\mathsf{B}_1 \to \mathsf{B}_2$. Hence $g_bf_a\mathsf{C} \cap f_a\mathsf{C}$ is empty iff $h_c\mathsf{C} \cap \mathsf{C}$ is empty, and equals $f_a\mathsf{C}$ iff $h_c\mathsf{C} \cap \mathsf{C} = \mathsf{C}$. Since $h_c\mathsf{C} \cap \mathsf{C}$ is either empty or equal to C, this implies that $g_bf_a\mathsf{C} \cap f_a\mathsf{C}$ is either empty or equal to $f_a\mathsf{C}$; that is, that $f_a\mathsf{C}$ is a block. □

Proposition 6.2.3: *Let* G *be a graph and let* $\pi = \{\mathsf{B}_1, \ldots, \mathsf{B}_k\}$ *be a c-partition of* $\mathsf{V}(\mathsf{G})$. *If* $\pi' = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_j\}$ *is a c-partition of* B_1 *with respect to* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ *then* π' *extends uniquely to a refinement of* π.

EXAMPLE 6.2.3: For the graph G in Figure 6.2 we have $\Pi = 1, 2, 3, 4/5, 6, 7, 8 = \{\mathsf{B}_1, \mathsf{B}_2\}$. Then $\pi' = 1, 2/3, 4$ is a c-partition of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$, and π' extends uniquely to the c-partition $1, 2/3, 4/5, 6/7, 8$ of G.

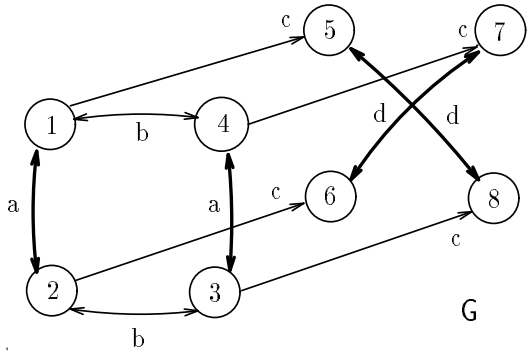


Figure 6.2: Extending a partition

Proof of Proposition 6.2.3 We have seen that any $f_a \in \mathcal{E}(\mathsf{G})$ which maps B_1 to B_2 also maps π' to a c-partition $\pi_2' = \{f_a\mathsf{C}_1, \ldots, f_a\mathsf{C}_j\}$ of B_2. We want to show that any $f_b \in \mathcal{E}(\mathsf{G})$ which maps B_1 to B_2 maps π' to the same partition π_2'. Let $f_a\mathsf{C} \in f_a\pi'$. Since ${f_b}^{-1}f_a$ maps B_1 to B_1, it must preserve π'. In particular, ${f_b}^{-1}f_a\mathsf{C}$ is a block $\mathsf{C}' \in \pi'$, and $f_a\mathsf{C} = f_b\mathsf{C}' \in f_a\pi'$. That is, $f_a\pi'$ and $f_b\pi'$ share a block in common. Since a block-system is determined by a single block this means that $f_a\pi' = f_b\pi'$. □

By Propostion 3.2.2 in Chapter 3, any graph G has a unique "coarsest" c-partition Π such that all other c-partitions of G are refinements of Π. Suppose that B is a block of this partition, and let G_B be an operator graph of $\mathcal{E}(\mathsf{G})|_\mathsf{B}$ with respect to the natural action on B. Proposition 6.2.4 will show that any symmetry of G_B extends uniquely to a symmetry of G, and that any symmetry of G, restricted to B, is a symmetry of G_B. The next two results are preludes to this.

We have:

Lemma 6.2.2: *Let G be a graph such that $\mathcal{E}(\mathsf{G})$ is a monoid but not a group, and let Π be the unique coarsest c-partition of $\mathsf{V}(\mathsf{G})$. For any $\mathsf{B}_1 \neq \mathsf{B}_2 \in \Pi$ there exists $g_a \in \mathcal{E}(\mathsf{G})$ such that $g_a\mathsf{B}_1 = \mathsf{B}_2$ but g_a is undefined on B_2.*

Proof We will need the following fact:

(*) Let G be a connected graph, and let $v, w \in \mathsf{V}(\mathsf{G})$ be such that for all $g_a \in \mathcal{E}(\mathsf{G})$, if g_a is defined on v then g_a is defined on w. Then any $g_a \in \mathcal{E}(\mathsf{G})$ is defined on v iff g_a is defined on w.

Proof of (*): If $v = w$ then (*) is immediate. Suppose that $v \neq w$ and let $g_b \in \mathcal{E}(\mathsf{G})$ map v to w. Then g_b is defined on v and so, by hypothesis, on w; and so g_b^2 is defined on v and so also on w. Continuing this argument, for any $k \geq 1$, g_b^k is defined on v and w. Since G is finite and g_b is one-to-one on its domain there is a number $k \geq 1$ such that $g_b^k(v) = v$ and $g_b^k(w) = w$. Now suppose that $g_c \in \mathcal{E}(\mathsf{G})$ is defined on w. Then $g_c g_b$ is defined on v, and so $g_c g_b$ is defined on w also. Then $g_c g_b^2$ is defined on v and hence w. Repeating this argument, we see that $g_c g_b^{k-1}$ is defined on w and so $g_c g_b^k$ is defined on v. Now $g_c g_b^k(v) = g_c(v)$, so g_c is defined on v if it is defined on w. This proves (*).

Proof of the lemma Suppose that there are blocks B_1 and B_2 in Π such that for all $g_a \in \mathcal{E}(\mathsf{G})$ which map B_1 to B_2, if g_a is defined on B_1 then g_a is defined on B_2. Pick g_a such that $g_a\mathsf{B}_1 = \mathsf{B}_2$, and suppose that $g_b \in \mathcal{E}(\mathsf{G})$ is defined on B_1. Then $g_a g_b^{-1} g_b$ maps B_1 to B_2, and so $g_a g_b^{-1} g_b$, and hence g_b, is defined on B_2. By (*) this implies that for any $g_b \in \mathcal{E}(\mathsf{G})$, g_b is defined on $v \in \mathsf{B}_1$ iff it is defined on $w \in \mathsf{B}_2$. By Lemma 3.2.3 in Chapter 3, this implies that $\mathrm{U}_v \simeq \mathrm{U}_w$ for any $v \in \mathsf{B}_1$ and $w \in \mathsf{B}_2$. Thus v and w are in the same block of Π, by its construction. That is, $\mathsf{B}_1 = \mathsf{B}_2$. □

Corollary 6.2.1: *Let Π be the coarsest c-partition of a graph G, let π_1 and π_2 be c-partitions of G and let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism. Then for any block $\mathsf{C} \in \pi_1$ there is a block $\mathsf{B} \in \Pi$ such that $\mathsf{C} \subseteq \mathsf{B}$ and $\delta(\mathsf{C}) \subseteq \mathsf{B}$. In particular, the only automorphism of G/Π is the identity automorphism.*

EXAMPLE 6.2.4: In Example 6.2.3, $\Pi = 1, 2, 3, 4/5, 6, 7, 8$ and G/Π is as pictured in Figure 6.3. The only automorphism of the quotient is the identity automorphism.

Proof of Corollary 6.2.1 Let $\delta : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be a map. Suppose that $\delta(\mathsf{C}) = \mathsf{D}$ for blocks $\mathsf{C} \in \pi_1$ and $\mathsf{D} \in \pi_2$, and suppose that C and D are in two different blocks B_1 and B_2 of Π. By Lemma 6.2.2 there is an element $f_a \in \mathcal{E}(\mathsf{G})$ which maps B_1 to B_2 but is undefined on B_2. Then δ cannot be an isomorphism, since f_a is defined on C but not on $\delta(\mathsf{C})$. □

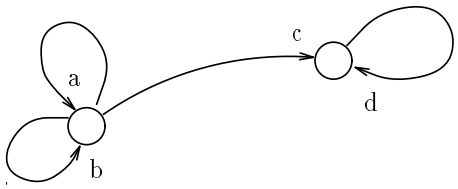


Figure 6.3: G/Π for G in Figure 6.2

The next proposition shows that the symmetries of G are in one-to-one correspondence with the symmetries of any operator graph of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ under its natural action on B. We will use the following notation:

Notation: Let $\mathsf{B} \in \Pi$, and let $X \subseteq \mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ be a set of generators for $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$. Write $\mathsf{G}_{\mathsf{B}}X$ for the operator graph with respect to X of the natural action of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ on B.

Recall that $\mathsf{G}_{\mathsf{B}}X$ and $\mathsf{G}_{\mathsf{B}}Y$ have the same symmetries for any generator-sets X and Y of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ (Remark 4.4.2 in Chapter 4). The symmetries of an operator graph of a group do not depend on the generators chosen for the group, but only on the group and its action on a set. With this in mind, we will omit the specification of a generator-set X of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$, and will write "$\mathsf{G}_{\mathsf{B}}$" for $\mathsf{G}_{\mathsf{B}}X$.

We have the following:

Proposition 6.2.4: *Let* Π *be the coarsest c-partition of* G*, let* $\mathsf{B} \in \Pi$*, and let* G_{B} *be an operator graph of the natural action of* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ *on* B*. Then:*

1. *Any graph isomorphism from one quotient of* G_{B} *to another extends uniquely to an isomorphism of one quotient of* G *to another.*
2. *For any isomorphism* $\tilde{\delta}$ *from one quotient of* G *to another, there is an isomorphism from one quotient of* G_{B} *to another that extends uniquely to* $\tilde{\delta}$*.*

Proof

Proof of (1): Let ${\pi_1}'$ and ${\pi_2}'$ be c-partitions of G_{B}. By Proposition 6.2.3, ${\pi_1}'$ extends uniquely to a c-partition π_1, and ${\pi_2}'$ extends uniquely to a c-partition π_2 of G. Let $\delta : \mathsf{G}_{\mathsf{B}}/{\pi_1}' \to \mathsf{G}_{\mathsf{B}}/{\pi_2}'$ be an isomorphism. We will show that there is a unique isomorphism $\tilde{\delta} : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ such that $\tilde{\delta}$ equals δ on B.

Construct $\tilde{\delta}$ as follows: For each $\mathsf{B}' \in \Pi$, pick an element $f_a \in \mathcal{E}(\mathsf{G})$ which maps B to B'. Define $\tilde{\delta} : \mathsf{V}(\mathsf{G}/\pi_1) \to \mathsf{V}(\mathsf{G}/\pi_2)$ such that for each B', the restriction of $\tilde{\delta}$ to B' equals $f_a \delta {f_a}^{-1}$ on the restriction of π_1 to B'. By the construction of the extensions π_1 and π_2 (Proposition 6.2.3), δ maps each block of $\pi_1|_{\mathsf{B}'}$ to a block of $\pi_2|_{\mathsf{B}'}$. Then:

(i) For each $\mathsf{B}' \in \Pi$*, the definition of* $\tilde{\delta}|_{\mathsf{B}'}$ *is independent of the choice of* $f_a \in \mathcal{E}(\mathsf{G})$ *which maps* B *to* B'*:*

For suppose that $f_b \in \mathcal{E}(\mathsf{G})$ also maps B to B'. Then ${f_a}^{-1} f_b$ maps B to itself. Since δ is an isomorphism, $\delta {f_a}^{-1} f_b = {f_a}^{-1} f_b \delta$ on π_1', or ${f_b}^{-1}(f_a \delta {f_a}^{-1}) f_b = \delta$. That is, $f_b \delta {f_b}^{-1} = f_a \delta {f_a}^{-1}$ on B', so the definition of $\tilde{\delta}$ on B' is independent of the choice of element $f_a : \mathsf{B} \to \mathsf{B}'$.

(ii) $\tilde{\delta}$ *restricted to* B *equals* δ*:*

Let f_a map B to itself. Then since δ is an isomorphism, we have $f_a\delta = \delta f_a$ on B, or $\tilde{\delta}|_{\mathsf{B}} = {f_a}^{-1}\delta f_a = \delta$.

(iii) $\tilde{\delta}$ is an isomorphism: $\mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$:

First note that $\tilde{\delta}$ maps each block $\mathsf{B}' \in \Pi$ to itself, and so:

(a) If f_b is defined on $\mathsf{C} \in \pi_1$ then f_b is defined on the block $\mathsf{B}' \in \Pi$ which contains C. Hence f_b is defined on C iff f_b is defined on $\tilde{\delta}(\mathsf{C})$.

(b) Suppose that f_b is defined on $\mathsf{C} \in \pi_1$ and that $\mathsf{C} \subseteq \mathsf{B}' \in \Pi$. Let $f_a \in \mathcal{E}(\mathsf{G})$ map B to B', so that $\tilde{\delta}|_{\mathsf{B}'} = f_a\delta {f_a}^{-1}$. Then on the block $f_b\mathsf{B}' \in \Pi$, we have $\tilde{\delta} = f_b f_a \delta {f_a}^{-1} {f_b}^{-1}$. Since $f_b(\mathsf{C}) \subseteq f_b\mathsf{B}'$, we have that $\tilde{\delta}(f_b\mathsf{C}) = f_b f_a \delta {f_a}^{-1} {f_b}^{-1} f_b(\mathsf{C}) = f_b f_a \delta {f_a}^{-1}(\mathsf{C}) = f_b\tilde{\delta}(\mathsf{C})$, so $\tilde{\delta}$ commutes with the elements of $\mathcal{E}(\mathsf{G})$. Hence $\tilde{\delta}$ is an isomorphism.

(iv) $\tilde{\delta}$ is the unique extension of δ:

This holds since $\tilde{\delta}|_{\mathsf{B}} = \delta$ and since an isomorphism is determined by its action on a single vertex (or block).

This proves (1).

Proof of (2): To prove the second part of the theorem, let $\tilde{\delta} : \mathsf{G}/\pi_1 \to \mathsf{G}/\pi_2$ be an isomorphism, and write ${\pi_1}'$ for $\pi_1|_{\mathsf{B}}$ and ${\pi_2}'$ for $\pi_2|_{\mathsf{B}}$. By Corollary 6.2.1, $\tilde{\delta}$ maps each block of Π onto itself, and so if we define $\delta = \tilde{\delta}|_{\mathsf{B}}$ then δ is an isomorphism from $\mathsf{G}_{\mathsf{B}}/{\pi_1}'$ to $\mathsf{G}_{\mathsf{B}}/{\pi_2}'$. □

The last theorem in the section gives conditions under which two graphs have the same symmetries. If G is a graph, we will write Π_{G} for the coarsest c-partition of G. We have:

Theorem 6.2.1: *Let G and H be graphs. Then G and H have the same symmetries iff all of the following hold:*

1. $\Pi_{\mathsf{G}} = \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\} = \Pi_{\mathsf{H}}$.
2. $\mathsf{G}_{\mathsf{B}_1}$ *and* $\mathsf{H}_{\mathsf{B}_1}$ *have the same constraints.*
3. *For each* $\mathsf{B}_i \in \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$, *if* $g_i \in \mathcal{E}(\mathsf{G})$ *and* $h_i \in \mathcal{E}(\mathsf{H})$ *both map* B_1 *to* B_i, *then for each constraint* $\langle \pi_1, \pi_2, \delta \rangle$ *of* $\mathsf{G}_{\mathsf{B}_1}$ *and* $\mathsf{H}_{\mathsf{B}_1}$, *we have* $g_i\pi_1 = h_i\pi_1$, $g_i\pi_2 = h_i\pi_2$, *and* $g_i\delta g_i^{-1} = h_i\delta h_i^{-1}$ *on* $g_i\pi_1 = h_i\pi_1$.

Proof (This is a corollary of Proposition 6.2.4.)

Suppose first that G and H have the same symmetries. Then (1) holds since $\langle \Pi_{\mathsf{G}}, \Pi_{\mathsf{G}}, id \rangle$ and $\langle \Pi_{\mathsf{H}}, \Pi_{\mathsf{H}}, id \rangle$ are symmetries, where id is the identity automorphism. (2) holds since the symmetries of G and H, restricted to $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ are symmetries of $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$. Similarly, the symmetries (and hence the block-systems) of $\mathsf{G}_{\mathsf{B}_i}$ must equal the symmetries of $\mathsf{H}_{\mathsf{B}_i}$ for each $i = 1, \ldots, n$. By Proposition 6.2.3, any block system π_1 of $\mathsf{G}_{\mathsf{B}_1}$ extends uniquely to a block-system $\pi = \{\pi_1, g_2\pi_1, \ldots, g_k\pi_1\}$ of G, where $g_i : \mathsf{B}_1 \to \mathsf{B}_i$. Thus we have $g_i\pi_1 = h_i\pi_1$ and $g_i\pi_2 = h_i\pi_2$ for π_1, π_2, g_i and h_i as given in condition (3). By part 1 of Proposition 6.2.4, any constraint δ of $\mathsf{G}_{\mathsf{B}_1}$ (or of $\mathsf{H}_{\mathsf{B}_1}$) extends uniquely to a symmetry of G (respectively, of H), where δ extends to an isomorphism on a block B_i by $g_i\delta g_i^{-1}$ or $h_i\delta h_i^{-1}$, respectively, for $g_i\mathsf{B}_1 = \mathsf{B}_i$ and $h_i\mathsf{B}_1 = \mathsf{B}_i$. Hence (3) must hold.

Conversely, suppose that conditions 1-3 hold. Let $\mathbf{s}$ be a symmetry of G. We will show that it is a symmetry of H. Note first that the symmetry-set of $\mathsf{G}_{\mathsf{B}_1}$ equals the symmetry-set of $\mathsf{H}_{\mathsf{B}_1}$ since both graphs have the same constraint-set (Corollary 5.4.1 in Chapter 5). Secondly, by part 2 of Propositon 6.2.4, the restriction $\mathbf{s}'$ of $\mathbf{s}$ to $\mathsf{G}_{\mathsf{B}_1}$ is a symmetry of $\mathsf{G}_{\mathsf{B}_1}$ and hence of $\mathsf{H}_{\mathsf{B}_1}$. Since (3) holds, $\mathbf{s}'$ extends to $\mathbf{s}$ in both G and H, by the proof of part 1 of Proposition 6.2.4. □

6.3 Complexity

The last task of the chapter is to show that the symmetry-sets of two monoid graphs can be checked for equality in time polynomial in the number of edges of the graphs. In light of Theorem 6.2.1, this involves finding polynomial-time algorithms for each of the following:

(1) Constructing the coarsest c-partitions Π_{G} and Π_{H} of graphs G and H, and checking Π_{G} and Π_{H} for equality.

(2) Finding generators for $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ for a block $\mathsf{B} \in \Pi_{\mathsf{G}}$, and generators for $\mathcal{E}(\mathsf{H})|_{\mathsf{B}}$ for $\mathsf{B} \in \Pi_{\mathsf{H}}$.

(3) Finding the constraints of G_{B} and H_{B} (that is, of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ and $\mathcal{E}(\mathsf{H})|_{\mathsf{B}}$), and and

(4) Checking whether conditions $1-3$ in Theorem 6.2.1 hold.

Theorem 5.5.1 in Chapter 5 shows that the constraints of G_{B} and H_{B} can be found in polynomial time, given generators for $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ and $\mathcal{E}(\mathsf{H})|_{\mathsf{B}}$. Proposition 6.3.1 (next) gives an algorithm for finding the coarsest partition Π of a graph G.

Propostion 6.3.1 uses a modification of a DFA minimization algorithm (Page 68 in [HU79]) to find Π. Recall that vertices v and w are in the same block of Π iff $\mathrm{U}_v \simeq \mathrm{U}_w$ (Proposition 3.2.2 in Chapter 3). Recall also that $\mathrm{U}_v \simeq \mathrm{U}_w$ iff $\mathrm{U}_v^{n-1} \simeq \mathrm{U}_2^{n-1}$, where $n = |\mathsf{V}(\mathsf{G})|$ (Proposition 2.4.2 in Chapter 2). The algorithm for finding Π will put sucessive equivalence-relations $\sim_k$ on $\mathsf{V}(\mathsf{G})$, where $v \sim_k w$ iff $\mathrm{U}_v^k \simeq \mathrm{U}_w^k$. It halts for the first k such that $\sim_k = \sim_{k-1}$. By the proof of Proposition 2.4.1, $\mathrm{U}_v \simeq \mathrm{U}_w$ iff $\mathrm{U}_v^k \simeq \mathrm{U}_w^k$, for this k.

Proposition 6.3.1: *There is an algorithm polynomial in* $m = |\mathsf{E}(\mathsf{G})|$ *for finding the coarsest c-partition* Π *of* $\mathsf{V}(\mathsf{G})$.

Algorithm 6.3.1: for finding Π**:**

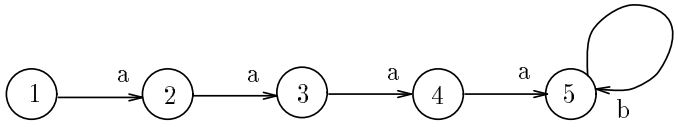
- Put an equivalence-relation $\sim_1$ on $\mathsf{V}(\mathsf{G})$ by: $v \sim_1 w$ iff for all $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$ we have f_a defined on v iff it is defined on w.
- For $k = 2$ until $\sim_{k-1} = \sim_k$: Construct an equivalence-relation $\sim_k$ on $\mathsf{V}(\mathsf{G})$ by $v \sim_k w$ iff $v \sim_{k-1} w$, and for each $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$, if f_a is defined on v then $f_a(v) \sim_{k-1} f_a(w)$.
- Let j be the smallest integer such that $\sim_j = \sim_{j-1}$. Let π_j be the partition of $\mathsf{V}(\mathsf{G})$ associated with $\sim_j$. Then $\pi_j = \Pi$.

□

EXAMPLE 6.3.1: For the graph G in Figure 6.4, the partitions associated with the equivalence-relations $\sim_i$ are as follows:

$\pi_1 = 1, 2, 3, 4/5$

$\pi_2 = 1, 2, 3/4/5$
$\pi_3 = 1, 2/3/4/5$, and
$\Pi = \pi_4 = \pi_5 = 1/2/3/4/5$.



. G

Figure 6.4: Finding the Coarsest Partition

Proof of correctness:

By Lemma 2.4.3 in Chapter 2, for each positive k we have $v \sim_k w$ iff $\mathrm{U}_v^k \simeq \mathrm{U}_w^k$. The proof of Proposition 2.4.2 in Chapter 2 shows that as k increases, the partition π_k associated with $\sim_k$ becomes strictly finer, up to a point. That is, if $\pi_{j-1} = \pi_j$ for some j then $\pi_j = \pi_{j+l}$ for all positive l. Hence if $\sim_{j-1}=\sim_j$, then $\sim_j=\sim_{j+l}$ for all positive l; and so $v \sim_j w$ iff $\mathrm{U}_v \simeq \mathrm{U}_w$. Since $[v] = [w]$ in Π iff $\mathrm{U}_v \simeq \mathrm{U}_w$ (Proposition 3.2.2), we have $\pi_j = \Pi$. □

Time to compute Π*:* Computing each equivalence-relation $\sim_k$ takes $O(n^2m)$ steps: There are $O(n^2)$ unordered pairs $\{v, w\}$ of vertices and $|\mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-| \leq m$ elements $\{f_a \in \mathcal{E}(\mathsf{G})\}$ to check for definition on each pair. By Proposition 2.4.2, $j \leq n - 1$, and so finding the equivalence-relations $\sim_1, \sim_2, \ldots, \sim_j$ takes $O(n^3m)$ steps.

More sophisticated algorithms can be designed by transforming G into a deterministic finite automaton and using a DFA-minimization algorithm, which runs in $O(n^2m)$ steps. ([HU79]) □

Our next task is to find an algorithm for constructing $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$, given a block $\mathsf{B} \in \Pi$. Once we have $\mathcal{E}(\mathsf{G})_{\mathsf{B}}$ (the stabilizer submonoid of $\mathcal{E}(\mathsf{G})$ on B), it will be easy to find $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$, since the elements of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ are just the elements of $\mathcal{E}(\mathsf{G})_{\mathsf{B}}$ restricted to B. By definition, $\mathcal{E}(\mathsf{G})_{\mathsf{B}}$ consists of all the elements of $\mathcal{E}(\mathsf{G})$ which map B to itself. That is, $\mathcal{E}(\mathsf{G})_{\mathsf{B}} = \{f_w :$ *w is the word of a closed path through* B *in* $\mathsf{G}/\Pi\}$.

Since every element in $\mathcal{E}(\mathsf{G})_{\mathsf{B}}$ is defined on all vertices of B, the relation $f_{a^{-1}} f_a(v) = v$ for $f_a \in \mathcal{E}(\mathsf{G})_{\mathsf{B}}$ holds for all $a \in \mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$ and for all $v \in \mathsf{B}$. That is, if a word $w \in \mathsf{A}(\mathsf{G})^*$ has reduced form w' then $f_w = f_{w'}$ on B. From this we can conclude that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}} = \{f_w|_{\mathsf{B}} : f_w \in \mathcal{E}(\mathsf{G})$ *and w is the word of a reduced closed path through* B *in* $\mathsf{G}/\Pi\}$.

We will look for the following:

(1) An algorithm for finding G/Π

(2) A subset $\mathcal{P}$ of the set of reduced closed paths through a block $\mathsf{B} \in \Pi$ which generates the set under path concatenation with cancellation (to be defined later). We will give a method for computing $\mathcal{P}$ and show that $\mathcal{P}$ indeed generates the whole set of paths.

G/Π is easy to construct. We have:

Lemma 6.3.1: *There is an algorithm polynomial in $m = |\mathsf{E}(\mathsf{G})|$ for finding G/Π, given G and Π.*

Algorithm 6.3.2: for finding G/Π For each edge $\langle v\, a\, w\rangle \in \mathsf{E}(\mathsf{G})$:

- Find the blocks of Π containing v and w.
- Add $\langle [v]a[w]\rangle$ to a set E (the edges of G/Π) if $\langle [v]a[w]\rangle$ is not already in E.

□

G/Π can be computed in time polynomial in m: The construction of Π (Algorithm 6.3.1) can include the following: (1) Assigning a name to each block; and (2) constructing a map which takes each element of $\mathsf{V}(\mathsf{G})$ to the name of its block. If this map is available then it takes $O(1)$ steps to construct $\langle [v]\, a\, [w]\rangle$ from $\langle v\, a\, w\rangle$. Thus G/Π can be constructed in time polynomial in m. □

To construct the set $\mathcal{P}$ defined above we will make use of a result which is presented in the literature on combinatorial group theory as follows:

Proposition 6.3.2: *(Corollary 1, page 128 in [Coh89]. See also Theorem 5.2 in [Mas67].)*

Let G be a connected graph and let T be a spanning tree of G with root v. Then the fundamental group $\pi(\mathsf{G}, v)$ of G at v is free, with one basis element for each edge $\mathbf{e} = \langle u\, a\, w\rangle$ not in T. The basis element is the path class of the path $p_{\mathbf{e}} = p_1 \mathbf{e} p_2^{-1}$, where p_1 and p_2 are paths in T having initial vertex v and terminal vertices u and w, respectively.

We will restate this result for graphs and give an elementary proof. First we will need to define the inverse of a path and the operation "path concatenation with cancellation":

Definition 6.3.1:

- Let $p_1 = v_0 a_1 v_1 \ldots v_{k-1} a_k v_k$ and $p_2 = v_k a_{k+1} \ldots v_r$ be paths in a graph. Write $p_1 p_2$ for the concatenation $v_o a_1 \ldots v_{k-1} a_k v_k a_{k+1} \ldots v_r$ of p_1 and p_2.
- Let $p = v_0 a_1 v_1 a_2 \ldots v_{k-1} a_k v_k$ be a path. The *inverse* of p, written p^{-1}, is the path $v_k a_k^{-1} v_{k-1} \ldots a_2^{-1} v_1 a_1^{-1} v_0$.
- Let P' be the set of all closed paths through a vertex v in G. Define an operation — "concatenation with cancellation", on P' as follows: If $p_1 \in P'$ has word w_1 and $p_2 \in P'$ has word w_2 then $p_1 \cdot p_2$ is the path through v with word w, where w is the reduced form of $w_1 w_2$.

Let G be a graph and let T be a spanning tree of G with root v. For each edge $\mathbf{e} = \langle u\, a\, w\rangle$ not in T, define a reduced closed path $p_{\mathbf{e}}$ through v in G and containing $\mathbf{e}$, by $p_{\mathbf{e}} = p_1 \mathbf{e} p_2^{-1}$, where p_1 and p_2 are paths from the root in T having terminal vertices u and w, respectively. Let $\mathcal{P} = \{p_{\mathbf{e}}, p_{\mathbf{e}}^{-1} : \mathbf{e} \notin T\}$.

We have the following:

Proposition 6.3.3:

1. *For any graph G, the set $\mathcal{P}$ defined above can be computed in time polynomial in $m = |\mathsf{E}(\mathsf{G})|$.*
2. *$\mathcal{P}$ generates the set of reduced closed paths through v in G under concatenation with cancellation.*

Proof of (1): Use a graph traversal algorithm (e.g., a depth-first tree algorithm as in [AHU74]) to find T in $O(m)$ steps. For each edge $\mathbf{e} = \langle u\, a\, w\rangle$ not in T, use a graph traversal algorithm to find the (unique) reduced paths p_1 and p_2 in T from v to u and from v to w, in $O(m)$ steps. □

To prove part 2 of the propostion we will make use of the following lemma:

Lemma 6.3.2: *Let G be a graph and $v \in \mathsf{V}(\mathsf{G})$, and let P be a set of reduced closed paths through v which generates the set of reduced closed paths through v under concatenation with cancellation. Construct a (connected) graph G' from G by adding an edge $\mathbf{e} = \langle u\, a\, w\rangle$ (where u or w may or may not be vertices of G). Let p' be a reduced closed path through v in G' containing one instance of $\mathbf{e}$. Then the set $P \cup \{p', (p')^{-1}\}$ generates the set of reduced closed paths through v in G'.*

Proof We will prove the lemma by induction on the number of instances of $\mathbf{e}_i \in \{\mathbf{e}, \mathbf{e}^{-1}\}$ in a reduced closed path p through v.

Base case: Let $\mathbf{e} = \langle u\, a\, w\rangle$ and let $p = p_1\mathbf{e}p_2$ be a reduced closed path through v, where v is the initial vertex of p_1 and the terminal vertex of p_2, and p_1 and p_2 do not contain $\mathbf{e}$. We will show that p is a product of paths in $P \cup \{p', (p')^{-1}\}$. Let $p' = q_1\mathbf{e}q_2$. Since $\mathbf{e}$ is not an edge in q_1 or q_2 we have by hypothesis that the reduced forms of the paths $p_1q_1^{-1}$ and $p_2q_2^{-1}$ are products of paths in P. Then $p = p_1q_1^{-1}q_1\mathbf{e}q_2q_2^{-1}p_2$ is a product of paths in $P \cup \{p', (p')^{-1}\}$. If instead p is a path containing one instance of $\mathbf{e}^{-1}$ (and no instances of $\mathbf{e}$) then p^{-1} contains one instance of $\mathbf{e}$, and the above argument shows that p is a product of paths in $P \cup \{p, (p')^{-1}\}$. This proves the base case.

Induction case: Let $p = p_1\mathbf{e}_1p_2\mathbf{e}_2 \ldots p_k\mathbf{e}_kp_{k+1}$ be a reduced closed path through v in G, where the p_i are paths not containing $\mathbf{e}$ or $\mathbf{e}^{-1}$; the initial vertex of p_1 and the terminal vertex of p_{k+1} is v, and $\mathbf{e}_i \in \{\mathbf{e}, \mathbf{e}^{-1}\}$ for $i = 1, \ldots, k$. Suppose first that $\mathbf{e}_1 = \mathbf{e}$, and let $p' = q_1\mathbf{e}q_2$, as above. By the induction hypothesis, the reduced forms of the paths $p_a = p_1\mathbf{e}q_2$ and $p_b = q_2^{-1}p_2 \ldots \mathbf{e}_kp_{k+1}$ are products of paths in $P \cup \{p', (p')\}$, and so $p = p_ap_b$ is, also. If $\mathbf{e}_1 = \mathbf{e}^{-1}$ then let $p_a = p_1\mathbf{e}^{-1}q_1^{-1}$ and $p_b = q_1p_2 \ldots p_{k+1}$, and argue as above. □

Proof of part 2 of Proposition 6.3.3 We prove the proposition by induction on the number of edges in G.

Base case: If G has one edge $\mathbf{e} = \langle v\, a\, v\rangle$ then $\mathcal{P} = \{\mathbf{e}, \mathbf{e}^{-1}\}$ and the result is clear. If $\mathbf{e} = \langle v\, a\, w\rangle$ with $v \neq w$ then $\mathcal{P}$ is empty and G has no nontrivial reduced closed paths through v.

Induction case: Suppose that the result holds for a graph G: $\mathcal{P}$ generates all reduced closed paths through v in G. Form a connected graph G' from G by adding an edge $\mathbf{e} = \langle u\, a\, w\rangle$. If either u or w is not a vertex of G then there is no reduced path through v containing $\mathbf{e}$ in G'. Otherwise, let $p' = p_{\mathbf{e}}$, where $p_{\mathbf{e}}$ is as given in the definition of $\mathcal{P}$. Let p be any reduced closed path through v in G'. If p does not contain $\mathbf{e}$ or $\mathbf{e}^{-1}$ then it is a reduced closed path through v in G, and so by the induction hypothesis, is a product of paths in $\mathcal{P}$. If p contains k edges from the set $\{\mathbf{e}, \mathbf{e}^{-1}\}$, then by Lemma 6.3.2 above, p is a product of paths in the set $\mathcal{P} \cup \{p', (p')^{-1}\}$. □

Corollary 6.3.1: *A set of generators for* $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ *can be computed in time polynomial in* $m = |\mathsf{E}(\mathsf{G})|$.

Proof Let $\mathcal{P}$ be a set of paths through B in G/Π, defined as above. By Proposition 6.3.3 $\mathcal{P}$ generates all educed closed paths through B under concatenation-with-cancellation. Hence the set $\{f_w \in \mathcal{E}(\mathsf{G})|_{\mathsf{B}} : w \textit{ is a word of a path in } \mathcal{P}\}$ generates $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$. □

We can now show that arbitrary monoid graphs can be checked for f-equivalence in polynomial time. We have the following:

Theorem 6.3.1: *Let* G *and* H *be graphs having the same number of vertices and having* m_1 *and* m_2 *edges, respectively. Then there is an algorithm polynomial in* $m = \max\{m_1, m_2\}$ *for determining whether* G *and* H *have the same symmetry-sets.*

Proof Referring back to Theorem 6.2.1, we need to check three things:

1. $\Pi_{\mathsf{G}} = \Pi_{\mathsf{H}} = \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$.
2. $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ have the same constraints, and
3. For each block $\mathsf{B}_i \in \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$, for each $f_a \in \mathcal{E}(\mathsf{G})$ and $h_b \in \mathcal{E}(\mathsf{H})$ which map B_1 to B_i, and for each constraint $\langle \pi_1, \pi_2, \delta \rangle$ of $\mathsf{G}_{\mathsf{B}_1}$, we have $f_a\pi_1 = h_b\pi_1$ and $f_a\pi_2 = h_b\pi_2$, and $f_a\delta {f_a}^{-1} = h_b\delta h_b^{-1}$ on B_2.

Note that if f_{a_1} and f_{a_2} in $\mathcal{E}(\mathsf{G})$ both map B_1 to B_i, then by the proof of Propostition 6.2.4, $f_{a_1}\delta f_{a_1}^{-1} = f_{a_2}\delta f_{a_2}^{-1}$, where defined, for each constraint δ of $\mathsf{G}_{\mathsf{B}_1}$. This lets us restate condition (3) as follows:

(3') Let $F_1 = \{f_{a_1}, \ldots, f_{a_k}\}$ and $F_2 = \{h_{b_1}, \ldots, h_{b_k}\}$ be sets of elements of $\mathcal{E}(\mathsf{G})$ and of $\mathcal{E}(\mathsf{H})$ such that for each block $\mathsf{B}_i \in \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$, $f_{a_i}\mathsf{B}_1 = \mathsf{B}_i$ and $h_{b_i}\mathsf{B}_1 = \mathsf{B}_i$. Then for all constraints $\delta = \langle \pi_1, \pi_2, \delta \rangle$ of $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ we have $f_{a_i}\pi_1 = h_{b_i}\pi_1$; $f_{a_i}\pi_2 = h_{b_i}\pi_2$, and $f_{a_i}\delta f_{a_i}^{-1} = h_{b_i}\delta h_{b_i}^{-1}$, where defined, for $i = 1, \ldots, k$.

By Proposition 6.3.1 we can find Π_{G} and Π_{H} in time polynomial in m. By Corollary 6.3.1 we can find generator-sets for $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ in time polynomial in m. Theorem 5.5.1 in Chapter 5 shows that we can check the equality of the constraint-sets of $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ in time polynomial in m.

The sets F_1 and F_2 can be found in time polynomial in m as follows: From each block $\mathsf{B}_i \in \{\mathsf{B}_1, \mathsf{B}_2, \ldots, \mathsf{B}_k\}$ pick an element v_i (e.g., the smallest element in the block), and find a path p in G from v_1 to v_i, by using a graph traversal algorithm ([AHU74]). Then $f_{a_i} = f_w \in \mathcal{E}(\mathsf{G})$, where w is the word of p. F_2 is constructed in the same way. $\mathsf{G}_{\mathsf{B}_1}$ and $\mathsf{H}_{\mathsf{B}_1}$ have at most $n \leq m$ constraints, and F_1 and F_2 each has at most n elements, and so for each constraint $\langle \pi_1, \pi_2, \delta \rangle$ the sets $\{f_{a_i}\delta f_{a_i}^{-1} : i = 1, \ldots, n\}$ and $\{h_{b_i}\delta h_{b_i}^{-1} : i = 1, \ldots, n\}$ can be computed in time polynomial in m. Furthermore, the action of each function $f_{a_i}\delta f_{a_i}^{-1}$ on $f_{a_i}\pi_1$ can be computed in time polynomial in m steps, and so in time polynomial in m it can be determined whether $f_{a_i}\delta f_{a_i}^{-1}$ equals $h_{b_i}\delta h_{b_i}^{-1}$, for $i = 1, \ldots, n$. □

7. Conclusion

7.1 Summary Of The Paper

This paper addressed three questions: (1) What characterizes the set of vector-valued functions a given network can compute? (2) How hard is it to tell whether two networks can compute the same set of functions?, and (3) How hard is it to tell if two networks can compute the same functions "up to a permutation"? The second and third questions pose graph classification problems: We called two graphs "f-equivalent" if the set of functions each can compute is the same, and "p-equivalent" if the set of functions each can compute is the same up to a permutation. We sought graph features which would correctly classify graphs into their f- and p-equivalence-classes, and found that the desired graph features are the quotient-graph isomorphisms: Two graphs are f-equivalent iff their sets of quotient-graph isomorphisms are identical, and p-equivalent iff their sets of quotient-graph isomorphisms differ by a permutation. The second and third questions mentioned above were then couched as questions in graph theory; namely, "How hard is it to determine whether two graphs have the same set of quotient-graph isomorphisms? How hard is it to determine if they have the same set of quotient-graph isomorphisms, up to a permutation?"

Chapter-By-Chapter Summary: In Chapter 2 we found a characterization of the functions a given network can compute. We saw that a network computes precisely those functions which "respect" its rooted universal covers. More precisely, G computes f iff for all $\vec{x}$ and $\vec{y}$ in $\mathbf{I^n}$; $\mathrm{U}_i\vec{x} \simeq \mathrm{U}_j\vec{y}$ implies that $f(\vec{x})_i = f(\vec{y})_j$. In Chapter 3 we showed that a network G computes exactly those functions which "satisfy the symmetries" of the network, where a symmetry of G is an isomorphism of quotients of G. We saw that two networks compute the same functions iff they have identical symmetry sets, and that two networks compute the same functions "up to a permutation" iff their symmetry-sets are identical "up to a permutation". In Chapter 4 we developed the notion of an "operator graph", whose vertices are the elements of a set and whose edges correspond to elements of a group acting on the set. We also found a one-to-one map from the set of cosets of subgroups of the edge-label group of a graph to the set of quotient-graph isomorphisms of the graph.

Chapters 4 and 5 examined "group-graphs"; graphs whose edge-label monoids are groups, and Chapter 6 extended the results in Chapter 5 to graphs with arbitrary edge-label monoids. In Chapter 5 we showed that the set of quotient-graph isomorphisms ("symmetries") of a group-graph form a lattice. We found a small subset—the "constraints"—of this lattice which generates the lattice under lattice-join, and concluded that two graphs have the same lattice of symmetries iff they have identical "constraints". Hence two networks compute the same functions iff they have the same constraint-set. We showed that finding the constraint-set of a graph is easy, and concluded that classifying graphs by their symmetries is not a hard problem. Chapter 6 extended these results to arbitrary "monoid graphs". We showed that every graph G has a unique coarsest c-partition Π of its vertices, such that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$, the edge-label monoid of G restricted to a block B of Π, is a group. We saw that $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1} \simeq \mathcal{E}(\mathsf{G})|_{\mathsf{B}_2}$ for any blocks B_1 and $\mathsf{B}_2 \in \Pi$, and that any symmetry of the operator graph of $\mathcal{E}(\mathsf{G})|_{\mathsf{B}_1}$ on B_1 extends uniquely to a symmetry of G. We found algorithms for computing Π and $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$, and concluded that classifying arbitrary monoid

graphs by their symmetries is also easy. Finally, we showed that determining whether two graphs compute the same functions "up to a permutation" is at least as hard as determining whether two groups, given as group tables, are isomorphic.

7.2 Open Questions

There are several extensions of this work which might be worth investigating. We could, for instance, drop the requirement that links be two-way. In this case, processors may not be able to compute their rooted universal covers, but each processor v can compute a subgraph T_v of U_v induced by the path in U_v directed towards the root. The characterization of the functions computable by a network (Theorem 2.5.1) probably extends to directed networks if the following conjecture holds:

Conjecture 7.2.1: *Let* G *be a strongly connected, edge-labeled digraph, let* $v \in \mathsf{V}(\mathsf{G})$*, and let* T_v *be the subraph of* U_v *induced by the set of edges in* U_v *directed towards the root. If* $T_v \simeq T_w$*, then* $\mathrm{U}_v \simeq \mathrm{U}_w$*.*

More generally we could ask: What subgraphs of U_v determine U_v, if G is, say, strongly connected; not strongly connected; edge-labeled; not edge-labeled, and so on?

For another extension of this work, we could drop the edge-label condition we imposed and allow two edges directed towards a vertex or two edges directed away from a vertex to have the same edge-label. It is not unreasonable to allow two edges directed towards a vertex in a computer network to have the same label. If the edge-label condition were dropped, edge-label functions would become edge-label relations. Graphs with edge-label relations are well-studied in algebraic automata theory (e.g., see [Hol82]), and some of the results of that field might transfer to this case. In particular, the definition of "covering map" found in algebraic automata theory extends the definition we gave to graphs with edge-label relations.

Finally, we would like to know if there is polynomial-time transformation of the problem: "Computes the same functions up to a permutation" (Section 5.6 in Chapter 5) to the problem of group isomorphism. Are these two problems polynomially equivalent?

List Of Symbols

Chapter 2

Section 2.2

- $\mathcal{N} = \langle \mathsf{V}_{\mathcal{N}}, \mathsf{E}_{\mathcal{N}}, \rho_{\mathcal{N}} \rangle$ denotes a *network*, where
  $\mathsf{V}_{\mathcal{N}}$ denotes the *processor set* of $\mathcal{N}$,
  $\rho_{\mathcal{N}}$ denotes the *processor labeling map* of $\mathcal{N}$, and
  $\mathsf{E}_{\mathcal{N}}$ denotes the set of *links* of $\mathcal{N}$.
- $\mathcal{L} = \{(v,i),(w,j)\}$ is a *link*.
- *deg(v)* is the number of links incident with v.

Subsection 2.2.1

- **I** and **O** denote the *input alphabet* and the *output alphabet*, respectively.

Subsection 2.2.2

- $\mathsf{G} = \langle \mathsf{V}(\mathsf{G}), \mathsf{E}(\mathsf{G}), \mathsf{A}(\mathsf{G}) \rangle$ denotes a *directed, edge-labeled graph*, where
  $\mathsf{V}(\mathsf{G})$ denotes the set of *vertices* of G,
  $\mathsf{E}(\mathsf{G})$ denotes the set of *edges* of G, and
  $\mathsf{A}(\mathsf{G})$ denotes the set of *edge-labels* of G.
- G and H are graphs.
- u, v, w, r, s, i, j, k all represent vertices.
- a, b, c, d, e all represent edge-labels.
- $\langle v\, a\, w \rangle$ is an edge with vertices v and w and edge-label a.
- $(\mathsf{G}, \vec{x})$, denotes a graph G with input $\vec{x}$.

Section 2.3

- $\mathcal{E}(\mathsf{G})$ denotes the *edge-label monoid* of a graph G.
- $\mathsf{A}(\mathsf{G})^- = \{a^{-1} : a \in \mathsf{A}(\mathsf{G})\}$.
- $\mathsf{A}(\mathsf{G})^*$ denotes the set of words over $\mathsf{A}(\mathsf{G}) \cup \mathsf{A}(\mathsf{G})^-$.
- λ denotes the *empty word*.
- un is the "undefined vertex".
- $f_a, f_b, f_c, \tilde{f}_a, g_a, g_{a^{-1}}$, and so on, are elements of $\mathcal{E}(\mathsf{G})$.

Subsection 2.3.2

- $\delta, \tilde{\delta}, \alpha, \beta$ denote covering maps.

Section 2.4

- U denotes the *universal cover* of a graph.
- U_v denotes the *rooted universal cover* of a graph.
- $\mathrm{U}_v \simeq \mathrm{U}_w$: "$\mathrm{U}_v$ and U_w are *isomorphic*".
- $\mathrm{U}_v\vec{x}$ denotes a rooted universal cover with input $\vec{x}$.
- U_v^k denotes a rooted universal cover *truncated at depth* k.

Chapter 3

Section 3.2

- $f_a\mathsf{B}$ denotes the set $\{f_a(i_k) : i_k \in \mathsf{B}\}$.

- π denotes a *c-partition*.

  $i_1, i_2, \ldots, i_k / i_{k+1}, \ldots, i_j / \ldots / i_l, i_{l+1}, \ldots, i_m$ denotes a c-partition of the set $\{i_1, i_2, \ldots, i_m\}$, with blocks $\{i_1, \ldots, i_k\}$, $\{i_{k+1}, \ldots, i_j\}$, ..., $\{i_l, \ldots, i_m\}$.
- $[i]$ denotes the block of a partition containing the vertex i.

  $[i]_\pi$ denotes the block of π containing i.
- $\pi_1 \preceq \pi_2$: " π_1 is a *refinement* of π_2".

Section 3.3

- Π denotes the "coarsest c-partition" of a graph.
- G/π denotes a *quotient graph* of G by π.
- $\sim_{\vec{x}}$ is an equivalence-relation on $\mathsf{V}(\mathsf{G})$.
- $x_{[i]}$ denotes x_i, where $i \in [i] \in \pi$.
- $\delta(\vec{x})$ is the vector $(x_{\delta([1])}, \ldots, x_{\delta([n])})$.

Section 3.4

- $\langle \pi_1, \pi_2, \delta \rangle$ denotes a *symmetry* of a network.

Section 3.5

- $\rho\mathsf{G}$, denotes a graph G with its vertices relabeled by a permutation ρ.
- $\rho(\pi)$ denotes the "permutation" of π by ρ.

Chapter 4

Section 4.2

- $(\mathcal{G}, S)$ denotes a permutation group $\mathcal{G}$ on a set S.
- $Sym(S)$ denotes the symmetric group on S.
- $\mathcal{G}_v$ denotes the stabilizer subgroup of $v \in S$.
- $\mathcal{H}(v)$ denotes the set $\{h(v) : h \in \mathcal{H}\}$.

Section 4.3

- $\rho_T(\mathcal{G})$ denotes the *permutation group corresponding to the action* T.

Section 3.5

- R denotes the left Cayley graph of $\mathcal{E}(\mathsf{G})$ with respect to the generator-set $\{g_a : a \in \mathsf{A}(\mathsf{G})\}$.

Section 4.5

- $\mathsf{L}_\mathsf{B}v$ or L_B denote the lattice of block-systems.
- $\wedge$ and $\vee$ denote the lattice meet- and join-operations.

Section 4.6

- $\mathsf{C}_{Sym(S)}\mathcal{G}$ denotes the *centralizer* of a group $\mathcal{G}$.

Chapter 5

Section 5.3

- $\mathsf{L}_c v$ denotes the lattice of coset-representatives of a graph with respect to a vertex v.

  $\mathsf{L}_S v$ denotes the lattice of symmetries with respect to v, and

  L_S denotes the lattice of symmetries.

Chapter 6

Section 6.2

- $\mathcal{E}(\mathsf{G})_{\mathsf{B}}$ deontes the *stabilizer submonoid* of a block.
- $\mathcal{E}(\mathsf{G})|_{\mathsf{B}}$ denotes the *restriction of* $\mathcal{E}(\mathsf{G})$ *to* B.
- $\mathsf{G}_{\mathsf{B}}X$ and G_{B} denote the operator graph of $\mathcal{E}(\mathsf{G})$ on X.

References

[ABR87] Fred Annextein, Marc Baumslag, and Arnold Rosenberg. Group action graphs and parallel architectures. Technical Report COINS Technical Report 87-133, Department of Computer and Information Science, University of Massachusetts at Amherst, 1987.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

[Ang80] D. Angluin. Local and global properties in networks of processors. In *ACM Symposium on the Theory of Computing*, pages 82–93, 1980.

[ASW88] H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *J. ACM*, 35:845–875, 1988.

[B$\ddot{8}$9] J. Richard Büchi. *Finite Automata, Their Algebras and Grammars*. Springer Verlag, 1989.

[BB89] Paul Beame and Hans L. Bodlaender. Distributed computing on transitive networks: The torus. In *Proceedings of The Sixth Annual Symposium on Theoretical Aspects of Computer Science, Springer Verlag Lecture Notes in Computer Science*, pages 294–303, Heidelberg, 1989.

[BMW93] H. L. Bodlaender, S. Moran, and M. Warmuth. The distributed bit-complexity of the ring: From the anonymous to the non-anonymous case. To appear in *Journal of Information and Computation*, 1993.

[Bol79] Béla Bollobás. *Graph Theory, An Introductory Course*. Springer Verlag, 1979.

[BR91] Marc Baumslag and Arnold Rosenberg. Processor-time tradeoffs for Cayley graph interconection networks. In *Proc. of the Sixth Distributed Memory Computing Conference*, pages 630–636, 1991.

[BS81] Stanley Burris and H. P. Sankappanavar. *Algebraic Topology: An Introduction*. Springer Verlag, 1981.

[Coh89] Daniel E. Cohen. *Combinatorial Group Theory: A Topological Approach*. London Mathematical Society, Cambridge University Press, 1989.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[FLM85] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossiblilty proofs for distributed consensus problems. In *Proceedings of The Fourth Annual ACM Symp. on Principles of Distributed Computing*, pages 59–70, Minaki, Ontario, 1985.

[FLP85] M. J. Fischer, N. A. Lynch, and M. S. Patterson. Impossiblity of distributed consensus with one faulty processor. *J. ACM*, 32:374–382, 1985.

[GT87] J. Gross and T. Tucker. *Topological Graph Theory*. Wiley Interscience, 1987.

[Hol82] W. M. L. Holcombe. *Algebraic Automata Theory*. Cambridge Studies in Advanced Mathematics, Cambridge University Press, 1982.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[Jac74] Nathan Jacobson. *Basic Algebra I.* W. H. Freeman, 1974.

[KKvdB90] E. Kranakis, D. Krizanc, and J. van den Berg. Computing boolean functions on anonymous networks. In *Proceedings of The International Conference on Algorithms, Languages and Programming*, pages 254–267, 1990.

[Koc70] R. Kochendorffer. *Group Theory.* McGraw-Hill, 1970.

[Lei82] F. Thomas Leighton. Finite covers of graphs. *Journal of Combinatorial Theory b*, 33:231–238, 1982.

[Luk90] Eugene Luks. Lectures in polynomial-time computation in groups. Technical Report CIS-TR-90-21, Department of Computer and Information Science, University of Oregon, Eugene, OR, 1990.

[MA89] Y. Matias and Y. Afek. Simple and efficient election algorithms for anonymous networks. In J. C. Bermond and M. Raynal, editors, *Third International Workshop on Distributed Algorithms, Springer Verlag Lecture Notes in Computer Science*, volume 392, pages 183–194, Heidelberg, 1989.

[Mas67] W. S. Massey. *Algebraic Topology: An Introduction.* Springer Verlag, 1967.

[Mil78] Gary Miller. On the $n^{\lg n}$ isomorphism technique. In *Proceedings of the Tenth ACM Symposium on the Theory of Computing*, pages 51–58, 1978.

[Moo56] E. F. Moore. Gedanken-experiments on sequential machines. *Annals of Mathematics Studies*, 34:129–153, 1956.

[MW93] S. Moran and M. Warmuth. Gap theorems for distributed computation. To appear in *SIAM Journal on Computing*, 1993.

[Nor93] Nancy Norris. Universal covers of graphs: Isomorphism to depth $n-1$ implies isomorphism to all depths. To appear in *Discrete Applied Mathematics*, 1993.

[RFH72] P. Rosenstiehl, J. R. Fiskel, and A. Hollinger. Intelligent graphs: Networks of finite automata capable of solving graph problems. In Ronald Read, editor, *Graph Theory and Computing*, pages 219 – 265. Academic Press, 1972.

[Rob82] Derek J. S. Robinson. *A Course in the Theory of Groups.* Springer Verlag, 1982.

[Sco87] W. R. Scott. *Group Theory.* Dover Publications, 1987.

[SS89] B. Schieber and M. Snir. Calling names on nameless networks. In *Eighth Annual ACM Symposium on the Principles of Distributed Computation*, pages 319–328, 1989.

[Tch87] Maurice Tchuente. Computation on a finite network of automata. In C. Choffrut, editor, *Automata Networks*, pages 53 – 67. Springer Verlag Lecture Notes In Computer Science, 1987.

[Wei64] Helmut Weilandt. *Finite Permutation Groups.* Academic Press, 1964.

[YK87a] M. Yamashita and T. Kameda. Computing functions on an anonymous network. Technical Report LCCR 87-16, Simon Fraser University, Vancouver, Vancouver, British Columbia, 1987.

[YK87b] M. Yamashita and T. Kameda. Computing on an anonymous network. Technical Report LCCR 87-15, Simon Fraser University, Vancouver, Vancouver, British Columbia, 1987.

[YK88] M. Yamashita and T. Kameda. Computing on anonymous networks. In *Proc. 7th ACM Symp. on Principles of Distributed Computing*, pages 117–131, Ontario, 1988.