Chapter 2

Constrained Systems

2.1 Labeled graphs and constraints

First, we recall a convenient diagrammatic method used to present a constrained system of sequences. An encoder, in turn, may generate sequences only from this set.

A *labeled graph* (or a *finite labeled directed graph*) $G = (V, E, L)$ consists of —

- a finite set of states $V = V_G$;
- a finite set of edges $E = E_G$ where each edge e has an *initial state* $\sigma_G(e)$ and a *terminal state* $\tau_G(e)$, both in V;
- an edge labeling $L = L_G : E \to \Sigma$ where Σ is a finite alphabet.

We will also use the notation $u \xrightarrow{a} v$ to denote an edge labeled a from state u to state v in G.

Figure 2.1 shows a "typical" labeled graph.

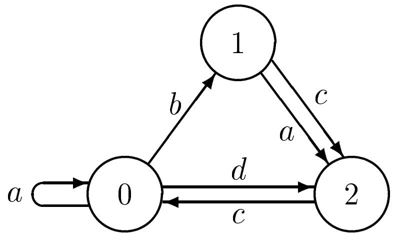

Figure 2.1: Typical labeled graph.

While some of the properties of interest to us do not depend on the labeling L, most do. We will omit the labeling qualifier from the term 'graph' in those cases where the labeling is immaterial.

There are a few features worth highlighting. Since the graph is directed, each edge can be traversed in only one direction, as indicated by the arrow. *Self-loops*, meaning edges that start and terminate in the same state, are allowed. Also, there can be more than one edge connecting a given state to another state; these are called *parallel edges*. However, we assume that distinct edges that share the same initial and terminal states have distinct labels. A graph is called *essential* if every state has at least one outgoing edge and at least one incoming edge; we will sometimes need to assume that graphs are essential, but then we will make this assumption explicitly. The *out-degree* of a state in a graph is the number of edges outgoing from that state. The *minimum out-degree* of a graph is the smallest among all out-degrees of the states in that graph.

A path γ in a graph G is a finite sequence of edges $e_1 e_2 \ldots e_\ell$ such that $\sigma_G(e_{i+1}) = \tau_G(e_i)$ for $i = 1, 2, \ldots, \ell{-}1$. The length of a path γ is the number of edges along the path and is denoted by $\ell(\gamma)$. The *state sequence* of a path $e_1 e_2 \ldots e_\ell$ is the sequence of states $\sigma_G(e_1)\sigma_G(e_2)\ldots\sigma_G(e_\ell)\tau_G(e_\ell)$. A *cycle* in a graph is a path $e_1 e_2 \ldots e_\ell$ where $\tau_G(e_\ell) = \sigma_G(e_1)$. We will also use the term right-infinite path for an infinite sequence of edges $e_1 e_2 \cdots$ in G such that $\sigma_G(e_{i+1}) = \tau_G(e_i)$ for $i \geq 1$. Similarly, a bi-infinite path is a bi-infinite sequence of edges $\cdots e_{-1} e_0 e_1 e_2 \cdots$ with $\sigma_G(e_{i+1}) = \tau_G(e_i)$ for all i.

A labeled graph can be used to generate finite symbol sequences by reading off the labels along paths in the graph. A finite sequence of symbols over a given alphabet will be called a *word* or a *block*. The length of a word $\mathbf{w}$—which is the number of symbols in $\mathbf{w}$—will be denoted by $\ell(\mathbf{w})$. A word of length ℓ will be called an *ℓ-block*. If a path γ in a graph G is labeled by a word $\mathbf{w}$, we say that $\mathbf{w}$ is generated by γ (and G). For example, in Figure 2.1, the 5-block *abccd* is generated by the path

$$0 \xrightarrow{a} 0 \xrightarrow{b} 1 \xrightarrow{c} 2 \xrightarrow{c} 0 \xrightarrow{d} 2 .$$

We also define the *empty word* as a 0-block: it is generated by a *zero-length path* which consists of one state and no edges. The empty word will be denoted by $\boldsymbol{\epsilon}$. A *sub-word* of a word $\mathbf{w} = w_1 w_2 \ldots w_\ell$ is either the empty word or any of the words $w_i w_{i+1} \ldots w_j$, where $1 \leq i \leq j \leq \ell$. Such a sub-word is *proper* if $1 < i \leq j < \ell$. Observe that every word that is generated by an essential graph is a proper sub-word of some other word that is generated by that graph.

Let $G_1 = (V_1, E_1, L_1)$ and $G_2 = (V_2, E_2, L_2)$ be labeled graphs. We say that G_1 and G_2 are *(labeled-graph) isomorphic* if there is a one-to-one mapping ψ from V_1 onto V_2 such that $u \xrightarrow{a} v$ is an edge in G_1 if and only if $\psi(u) \xrightarrow{a} \psi(v)$ is an edge in G_2.

The underlying finite directed graph of a labeled graph is conveniently described by a matrix as follows. Let G be a graph. The *adjacency matrix* $A = A_G = \Big((A_G)_{u,v} \Big)_{u,v \in V_G}$ is the $|V_G| \times |V_G|$ matrix where the entry $(A_G)_{u,v}$ is the number of edges from state u to state v in G. For instance, the adjacency matrix of the graph in Figure 2.1 is

$$A_G = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 2 \\ 1 & 0 & 0 \end{pmatrix} .$$

The adjacency matrix of course has nonnegative, integer entries. It is a useful artifice; for example, the number of paths of length ℓ from state u to state v is simply $(A_G^\ell)_{u,v}$, and the number of cycles of length ℓ is simply the trace of A_G^ℓ.

The fundamental object considered in the theory of constrained coding is the set of words generated by a labeled graph. A *constrained system* (or *constraint*), denoted S, is the set of all words (i.e., finite sequences) obtained from reading the labels of paths in a labeled graph G (although sometimes we will consider *right-infinite* sequences $x_0 x_1 x_2 \cdots$ and sometimes *bi-infinite* sequences $\cdots x_{-2} x_{-1} x_0 x_1 x_2 \cdots$). We say that G *presents* S or is a *presentation* of S, and we write $S = S(G)$. The *alphabet* of S is the set of symbols that actually occur in words of S and is denoted $\Sigma = \Sigma(S)$.

As central examples of constrained systems, we have the *(d, k)-RLL constrained systems*, which are presented by the labeled graph in Figures 1.3, and the *B-charge constrained systems*, which are presented by the labeled graph in Figure 1.14.

A constrained system is equivalent in automata theory to a regular language which is recognized by an automaton, the states of which are all accepting [Hopc79]. A constrained system is called a sofic system (or sofic shift) in symbolic dynamics [LM95]—except that a sofic system usually refers to the bi-infinite symbol sequences generated by a labeled graph. Earlier expositions on various aspects of constrained systems can be found in [Béal93a], [KN90], [LM95], and [MSW92].

A constrained system should not be confused with any particular labeled graph, because a given constrained system can be presented by many different labeled graphs. For example, the $(0, 1)$-RLL constrained system is presented by all labeled graphs in Figures 2.2 through 2.5, which are very different from one another. This is good: one presentation may be preferable because it has a smaller number of states, while another presentation might be preferable because it could be used as an encoder.

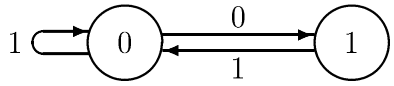

Figure 2.2: Labeled graph for $(0, 1)$-RLL constrained system.

It should be quite clear at this point why we assume that labeled graphs do not contain parallel edges that are labeled the same: the set of words generated by a given graph would not change if such parallel edges were added.

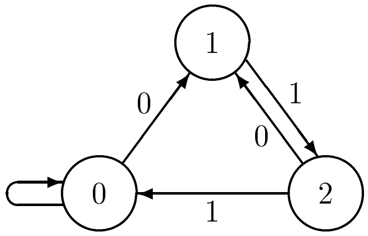

Figure 2.3: Another labeled graph for $(0,1)$-RLL constrained system.

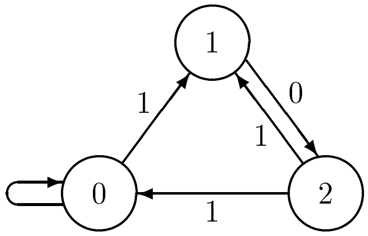

Figure 2.4: Yet another labeled graph for $(0,1)$-RLL constrained system.

2.2 Properties of labelings

2.2.1 Deterministic presentation

For purposes of encoder construction, it will be important to consider labelings with special properties. The most fundamental special property is as follows.

A labeled graph is *deterministic* if at each state the outgoing edges are labeled distinctly. In other words, at each state, any label generated from that state uniquely determines an outgoing edge from that state. The labeled graphs in Figures 1.3, 1.14, 2.2, and 2.3 are deterministic while the labeled graphs in Figures 2.4 and 2.5 are not. Constrained systems in the engineering literature are usually presented by deterministic graphs. In fact, any

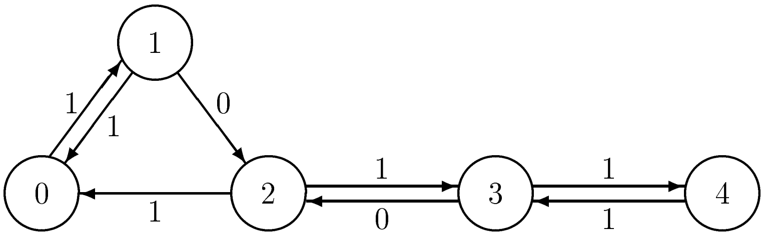

Figure 2.5: One more labeled graph for $(0,1)$-RLL constrained system.

constrained system can be presented in this way, as we show next.

Let G be a labeled graph. We define the *determinizing graph* H of G in the following manner. For any word $\mathbf{w}$ and state $v \in V_G$, let $T_G(\mathbf{w}, v)$ denote the subset of states in G which are accessible from v by paths in G that generate $\mathbf{w}$. When $\mathbf{w}$ is the empty word $\boldsymbol{\epsilon}$, define $T_G(\boldsymbol{\epsilon}, v) = \{v\}$. The states of H are the distinct nonempty subsets $\{T_G(\mathbf{w}, v)\}_{\mathbf{w},v}$ of V_G. As for the edges of H, for any two states $Z, Z' \in V_H$ we draw an edge $Z \xrightarrow{b} Z'$ in H if and only if there exists a state $v \in V_G$ and a word $\mathbf{w}$ such that $Z = T_G(\mathbf{w}, v)$ and $Z' = T_G(\mathbf{w}b, v)$. In other words, each state of G in Z' is accessible in G from some state in Z by an edge labeled b. By construction, the determinizing graph H is deterministic. We have also the following.

Lemma 2.1 *Let H be the determinizing graph of a labeled graph G. Then $S(H) = S(G)$.*

Proof. If a word $\mathbf{w} = w_1 w_2 \ldots w_\ell$ is generated by paths in G starting at state v, then $\mathbf{w}$ is also generated by the path

$$\{v\} = T_G(\boldsymbol{\epsilon}, v) \xrightarrow{w_1} T_G(w_1, v) \xrightarrow{w_2} T_G(w_1 w_2, v) \xrightarrow{w_3} \cdots \xrightarrow{w_\ell} T_G(w_1 w_2 \ldots w_\ell, v)$$

in H. Conversely, if $\mathbf{w}$ is generated by H starting at a state $Z = T_G(\mathbf{w}', v)$, then, by the construction of H, $\mathbf{w}'\mathbf{w}$ is generated in G by a path that starts at state v. □

By Lemma 2.1 we can conclude the next result.

Proposition 2.2 *Any constrained system can be presented by some deterministic labeled graph.*

We also have the notion of *co-deterministic*, obtained by replacing "outgoing" with "incoming" in the definition.

'Deterministic' is called *right-resolving* in symbolic dynamics [LM95].

2.2.2 Finite anticipation

Encoder synthesis algorithms usually begin with a deterministic presentation and transform it into a presentation which satisfies the following weaker version of the deterministic property.

A labeled graph G has *finite local anticipation* (or, in short, *finite anticipation*) if there is an integer N such that any two paths of length $N{+}1$ with the same initial state and labeling must have the same initial edge. The *(local) anticipation* $\mathcal{A}(G)$ of G is the smallest N for which this holds. Hence, knowledge of the initial state of a path and the first $\mathcal{A}(G){+}1$

symbols that it generates is sufficient information to determine the initial edge of the path. In case G does not have finite anticipation, we define $\mathcal{A}(G) = \infty$.

We also define the *(local) co-anticipation* of a labeled graph G as the anticipation of the labeled graph obtained by reversing the directions of the edges in G.

Note that to say that a labeled graph is deterministic is to say that it has zero anticipation. The labeled graph in Figure 2.4 is a presentation of the $(0, 1)$-RLL constrained system with anticipation 1 but not 0. Figure 2.5 depicts a presentation that does not have finite anticipation.

'Finite anticipation' is also called *right-closing* (in symbolic dynamics [LM95]) or *lossless of finite order* [Huff59], [Even65].

2.2.3 Finite memory

A labeled graph G is said to have *finite memory* if there is an integer N such that the paths in G of length N that generate the same word all terminate in the same state. The smallest N for which this holds is called the *memory* of G and denoted $\mathcal{M}(G)$.

2.2.4 Definite graphs

A labeled graph is (m, a)-*definite* if, given any word $\mathbf{w} = w_{-\mathsf{m}} w_{-\mathsf{m}+1} \ldots w_0 \ldots w_{\mathsf{a}}$, the set of paths $e_{-\mathsf{m}} e_{-\mathsf{m}+1} \ldots e_0 \ldots e_{\mathsf{a}}$ that generate $\mathbf{w}$ all agree in the edge e_0. We say that a labeled graph is definite if it is (m, a)-definite for some finite nonnegative m and a. Definite graphs are referred to in the literature also as graphs with *finite memory-and-anticipation.*

Note the difference between this concept and the concept of finite anticipation: we have replaced knowledge of an initial state with knowledge of a finite amount of memory. Actually, definiteness is a stronger condition, as we show in Proposition 2.3.

Figure 2.3 shows a labeled graph that is $(2, 0)$-definite, while Figure 2.6 shows a labeled graph that has finite anticipation (in fact, is deterministic and co-deterministic) but is not definite.

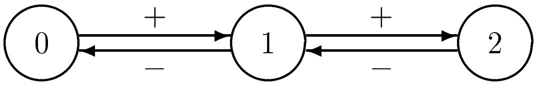

Figure 2.6: Labeled graph for a 2-charge constrained system

Note that, in contrast to the anticipation and the memory, we did not require a and m to be minimal in any sense while talking about (m, a)-definiteness. It would be natural to require that $\mathsf{m}+\mathsf{a}$ be minimal, but even that does not specify m and a uniquely; for instance, the labeled graph in Figure 2.7 is $(1, 0)$-definite and also $(0, 1)$-definite.

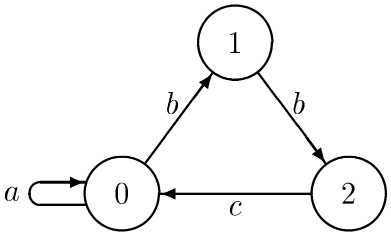

Figure 2.7: Labeled graph which is both $(1, 0)$-definite and $(0, 1)$-definite.

2.2.5 Lossless graphs

A labeled graph is *lossless* if any two distinct paths with the same initial state and terminal state have different labelings. All of the pictures of labeled graphs that we have presented so far are lossless. Figure 2.8 shows a presentation of the $(0, 1)$-RLL constrained system that is not lossless.

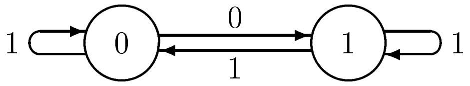

Figure 2.8: Graph which is not lossless.

2.2.6 Summary of terms

The following proposition summarizes the relationships among the labeling properties introduced so far.

Proposition 2.3 *For essential graphs,*

$$\begin{tikzcd}
 & & \textit{Co-deterministic} \arrow[d, Rightarrow] \\
\textit{Finite memory} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \textit{Definite} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \textit{Finite co-anticipation} \arrow[d, Rightarrow] \\
\textit{Deterministic} \arrow[r, Rightarrow] & \textit{Finite anticipation} \arrow[r, Rightarrow] & \textit{Lossless}
\end{tikzcd}$$

Proof. *Finite memory* $\Rightarrow$ *Deterministic* and *Finite memory* $\Rightarrow$ *Definite:* Let G be a labeled graph with finite memory $\mathcal{M}$. All paths, representing the same word, of length $\mathcal{M}+1$ in G must agree in their last two states. Furthermore, since G does not contain parallel edges with the same label, all these paths agree in their last edge. Hence, G is $(\mathcal{M}, 0)$-definite. This also implies that G is deterministic: For a state u in G, let γ be a path of length $\mathcal{M}$ that terminates in u. Then for every edge e outgoing from u, the labeling of γe determines e.

Definite $\Rightarrow$ *Finite anticipation:* Suppose that G is (m, a)-definite for some m and a. Let u be a state in G and let $\gamma = e_0 e_1 \ldots e_{\mathsf{a}}$ and $\gamma' = e'_0 e'_1 \ldots e'_{\mathsf{a}}$ be paths of length $\mathsf{a}+1$ which start at u and generate the same word. We need to show that $e_0 = e'_0$. Let γ'' be any path of length m which terminates in state u. Then, the concatenated paths $\gamma''\gamma$ and $\gamma''\gamma'$ both generate the same word. Since G is (m, a)-definite, we have $e_0 = e'_0$ as desired. A similar proof yields the implication *Definite* $\Rightarrow$ *Finite co-anticipation.*

Deterministic $\Rightarrow$ *Finite anticipation:* As pointed out earlier, a labeled graph is deterministic if and only if it has zero anticipation. The implication *Co-deterministic* $\Rightarrow$ *Finite co-anticipation* is similar.

Finite anticipation $\Rightarrow$ *Lossless:* Let G have anticipation $\mathcal{A}$. Given two paths, γ and γ', with the same initial state u, terminal state v, and the same labeling $\mathbf{w}$, let γ'' be a path of length $\mathcal{A}$ which starts at v. Then $\gamma\gamma''$ and $\gamma'\gamma''$ start at the same state and generate the same word; so, $\gamma = \gamma'$, as desired. The implication *Finite co-anticipation* $\Rightarrow$ *Lossless* is proved in a similar way. ☐

2.2.7 State labeling

In the graph presentations that we have seen so far, the labels are put on the *edges*. However, in the literature, one can find graph presentations where the labels are put on the *states*, and the respective constrained system is defined as the set of words that are obtained by reading off the labels of states along the finite paths in the graph. It is straightforward to see that every such constrained system can be presented by an (edge-)labeled graph, where the incoming edges to each state all have the same label. In fact, every constrained system can be presented by such a labeled graph, as we now show.

Let G be a labeled graph. The *Moore form* of G is a labeled graph H where $V_H = E_G$ and $e_1 \xrightarrow{a} e_2$ is an edge in H if and only if $\tau_G(e_1) = \sigma_G(e_2)$ and $L_G(e_2) = a$. For example, Figure 2.3 shows the Moore form of the labeled graph in Figure 2.2 that presents the $(0, 1)$-RLL constrained system. It can be easily verified that $S(H) = S(G)$ and that the edges incoming to each state in H all have the same labeling. It thus follows that every constrained system can be presented by a state-labeled graph. The anticipation of G is preserved in H and the co-anticipation is increased by 1. In particular, if G is deterministic, so is its Moore form. Also, by construction, there are no parallel edges in a Moore form, so its adjacency matrix is always a 0–1 matrix.

We have also the notion of a *Moore co-form* of a labeled graph G which is identical to the Moore form except for the labeling: The edge $e_1 \xrightarrow{a} e_2$ in a Moore co-form inherits the labeling of e_1 in G, rather than that of e_2. For example, Figure 2.4 is the Moore co-form of the labeled graph of Figure 2.2. If H is a Moore co-form of a labeled graph G, then $S(H) = S(G)$ and the edges *outgoing* from each state in H all have the same labeling. The co-anticipation of G is preserved in H and the anticipation is increased by 1. Therefore, if G is co-deterministic, so is H.

2.3 Finite-type constraints

In this section, we consider some special classes of constraints. The properties that define these constraints will be useful for encoder construction.

A constrained system S is *finite-type* (in symbolic dynamics, *shift of finite type* [LM95]) if it can be presented by a definite graph. As an example, the (d, k)-RLL constraint is finite-type: the labeled graph in Figure 1.3 is $(k, 0)$-definite—i.e., for any given word $\mathbf{w}$ of length at least $k+1$, all paths that generate $\mathbf{w}$ end with the same edge.

It is important to recognize that there are "bad" presentations of finite-type constrained systems, meaning labeled graphs that are not definite. For example, the labeled graph in Figure 2.5 represents the $(0, 1)$-RLL constrained system, but it is not definite, as can be seen by considering the paths that generate words consisting of all 1's.

Given the existence of bad labeled graphs, one might begin to worry about potential problems in determining whether or not a constrained system is finite-type. However, there is an intrinsic characterization of finite-type constrained systems that resolves this difficulty.

A constrained system S is said to have *finite memory* if there is an integer N such that, for any symbol $b \in \Sigma(S)$ and any word $\mathbf{w} \in S$ of length at least N, we have $\mathbf{w}b \in S$ if and only if $\mathbf{w}'b \in S$ where $\mathbf{w}'$ is the suffix of $\mathbf{w}$ of length N. The smallest such integer N, if any, is called the *memory* of S and is denoted by $\mathcal{M}(S)$.

It is can be readily verified that the (d, k)-RLL constrained system has memory k.

Lemma 2.4 *A constrained system S has finite memory if and only if there is a presentation G of S with finite memory. Furthermore, the memory of S is the smallest memory of any presentation of S with finite memory.*

Proof. Clearly, if a constrained system S has a presentation G with finite memory $\mathcal{M}(G)$, then $\mathcal{M}(S) \leq \mathcal{M}(G)$.

On the other hand, let S be a constrained system with finite memory $\mathcal{M}(S) = \mathcal{M}$. Then all presentations of S have memory which is bounded from below by $\mathcal{M}$. We construct a labeled graph H with $\mathcal{M}(H) = \mathcal{M}$ as follows: For each word $\mathbf{w}$ of length $\mathcal{M}$ in S, we associate a state $u_{\mathbf{w}}$ in H. Given two words, $\mathbf{w} = w_1 w_2 \ldots w_{\mathcal{M}}$ and $\mathbf{z} = z_1 z_2 \ldots z_{\mathcal{M}}$, in S, we draw an edge $u_{\mathbf{w}} \xrightarrow{b} u_{\mathbf{z}}$ in H if and only if the following three conditions hold:

(a) $z_j = w_{j+1}$ for $j = 1, 2, \ldots, \mathcal{M}-1$;

(b) $b = z_{\mathcal{M}}$;

(c) $\mathbf{w}b \in S$.

It is easy to verify that H is a presentation of S (and, so $\mathcal{M}(H) \geq \mathcal{M}$). On the other hand, the paths in H of length $\mathcal{M}$ that generate the word $\mathbf{w}$ all terminate in state $u_{\mathbf{w}}$. Hence, $\mathcal{M}(H) \leq \mathcal{M}$. $\square$

Proposition 2.5 *A constrained system is finite-type if and only if it has finite memory.*

Proof. Suppose that S has finite memory and let G be a presentation of S with finite memory $\mathcal{M}$. As such, G is also $(\mathcal{M}, 0)$-definite and, so, S is finite-type.

Now, suppose that S is finite-type and let G be an (m, a)-definite presentation of S. If $\mathsf{a} = 0$, then G has memory $\leq \mathsf{m}+1$ and we are done. When $\mathsf{a} > 0$, it suffices to find a presentation of S which is $(\mathsf{m}+\mathsf{a}, 0)$-definite.

Such a presentation H can be obtained as follows: The states of H are pairs $[u, \mathbf{w}]$, where $u \in V_G$ and $\mathbf{w}$ is a word of length a that can be generated by a path in G that starts at u. Let u and v be states in G and $\mathbf{w} = w_1 w_2 \ldots w_{\mathsf{a}}$ and $\mathbf{z} = z_1 z_2 \ldots z_{\mathsf{a}}$ be two words that can be generated in G from u and v, respectively. We draw an edge $[u, \mathbf{w}] \xrightarrow{b} [v, \mathbf{z}]$ in H if and only if the following three conditions hold:

(a) $z_j = w_{j+1}$ for $j = 1, 2, \ldots, \mathsf{a}-1$;

(b) $b = z_{\mathsf{a}}$;

(c) there is an edge $u \xrightarrow{w_1} v$ in G.

We now define a mapping from the set of all paths of length $\mathsf{m}+\mathsf{a}+1$ in G onto the set of paths of length $\mathsf{m}+1$ in H as follows. The path

$$\gamma_G = u_0 \xrightarrow{b_1} \cdots \xrightarrow{b_{\mathsf{m}}} u_{\mathsf{m}} \xrightarrow{b_{\mathsf{m}+1}} u_{\mathsf{m}+1} \xrightarrow{b_{\mathsf{m}+2}} \cdots \xrightarrow{b_{\mathsf{m}+\mathsf{a}}} u_{\mathsf{m}+\mathsf{a}} \xrightarrow{b_{\mathsf{m}+\mathsf{a}+1}} u_{\mathsf{m}+\mathsf{a}+1}$$

in G is mapped to the path

$$\gamma_H = [u_0, b_1 b_2 \ldots b_{\mathsf{a}}] \xrightarrow{b_{\mathsf{a}+1}} \cdots \xrightarrow{b_{\mathsf{m}+\mathsf{a}}} [u_{\mathsf{m}}, b_{\mathsf{m}+1} b_{\mathsf{m}+2} \ldots b_{\mathsf{m}+\mathsf{a}}] \xrightarrow{b_{\mathsf{m}+\mathsf{a}+1}} [u_{\mathsf{m}+1}, b_{\mathsf{m}+2} b_{\mathsf{m}+3} \ldots b_{\mathsf{m}+\mathsf{a}+1}]$$

in H. It is easy to verify that this mapping is indeed onto. Since G is (m, a)-definite, the word $b_1 b_2 \ldots b_{\mathsf{m}+\mathsf{a}+1}$ uniquely defines the edge $u_{\mathsf{m}} \xrightarrow{b_{\mathsf{m}+1}} u_{\mathsf{m}+1}$ in the path γ_G in G. It thus follows that the last two states in γ_H are uniquely defined, and so is the last edge of γ_H. Hence, H is $(\mathsf{m}+\mathsf{a}+1, 0)$-definite. □

The following result gives another equivalent formulation of the notion of finite-type systems in terms of lists of forbidden words. This notion was alluded to at the end of Section 1.2 and in Section 1.5.2.

Proposition 2.6 *A constrained system S is finite-type if and only if there is a finite list $\mathcal{L}$ of words such that $\mathbf{w} \in S$ if and only if $\mathbf{w}$ does not contain any word of $\mathcal{L}$ as a sub-word.*

We leave the proof of Proposition 2.6 as an exercise for the reader (Problem 2.7).

Not every constrained system of interest is finite-type. For example, the 2-charge constrained system described by Figure 2.6 is not. This can be seen easily by considering the condition above: the symbol '+' can be appended to the word

$$- + - + - + \cdots - +$$

but not to the word

$$+ + - + - + - + \cdots - + \, .$$

As a second example, consider the $(0, \infty, 2)$-RLL constrained system, which is commonly referred to as the *even* constraint. This constrained system consists of all binary words in which the runs of 0's between successive 1's have even lengths. A graph presentation of this constraint is shown in Figure 2.9. We leave it as an exercise to show that this constraint is not finite-type (Problem 2.26, part 1).

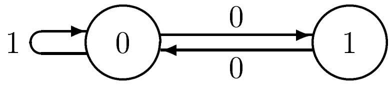

Figure 2.9: Shannon cover of the even constrained system.

However, both the charge constraint and the even constraint fall into a natural broader class of constrained systems, called almost-finite-type systems; these systems should be thought of as "locally finite-type" (perhaps that would have been a better name). A constrained system is *almost-finite-type* if it can be presented by a labeled graph that has both finite anticipation and finite co-anticipation.

By Proposition 2.3, we know that definiteness implies finite anticipation and finite co-anticipation. Thus, every constrained system which is finite-type is also almost-finite-type, and so the almost-finite-type systems do indeed include the finite-type systems. From Figure 2.6, we see that the charge constrained systems are presented by labeled graphs with zero anticipation (i.e., deterministic) and zero co-anticipation (i.e., co-deterministic); thus, these systems are almost-finite-type, but not finite-type. Most constrained systems used in practical applications are in fact almost-finite-type.

Recall that every constrained system has a deterministic presentation (and hence finite anticipation); likewise, every constrained system has a co-deterministic presentation (and hence finite co-anticipation). So, the essential feature of the almost-finite-type definition is that there is a presentation that simultaneously has finite anticipation and finite co-anticipation.

As with finite-type systems, we have the problem that a given constrained system may have some presentation that satisfies the finite anticipation and co-anticipation conditions and another presentation that does not. There is an intrinsic condition that defines almost-finite-type, but it is a bit harder to state [Will88]. We will give an example of a constrained system which is not almost-finite-type at the end of Section 2.6.

2.4 Some operations on graphs

In this section, we introduce three graph constructions that create new constraints from old.

2.4.1 Power of a graph

As mentioned in Chapter 1, a rate $p : q$ finite-state encoder will generate a word, composed of q-codewords (q-blocks) that when hooked together belong to the desired constrained system S. For a constrained system S presented by a labeled graph G, it will be very useful to have an explicit description of the words in S, decomposed into such non-overlapping “chunks” of length q.

Let G be a labeled graph. The *qth power of* G, denoted G^q, is the labeled graph with the same set of states as G, but one edge for each path of length q in G, labeled by the q-block generated by that path. For a constrained system S presented by a labeled graph G, the *qth power of* S, denoted S^q, is the constrained system presented by G^q. So, S^q is the constrained system obtained from S by grouping the symbols in each word into non-overlapping “chunks” of length q (in particular, the definition of S^q does not depend on which presentation G of S is used).

For example, Figure 2.10 shows the third power G^3 of the labeled graph G in Figure 2.2 that presents the $(0, 1)$-RLL constrained system.

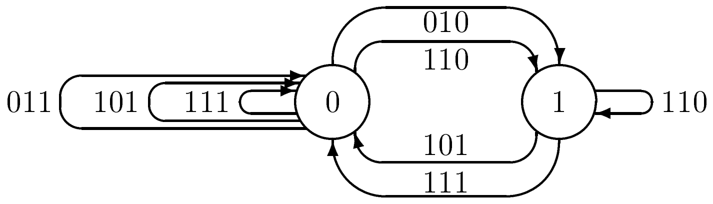

Figure 2.10: Third power of labeled graph in Figure 2.2.

2.4.2 Higher edge graph

The qth *higher edge graph* $G^{[q]}$ is the labeled graph whose states are paths in G of length $q-1$ with an edge for each path of length q in G: the edge $e_1e_2\ldots e_q$ has initial state $e_1e_2\ldots e_{q-1}$, terminal state $e_2\ldots e_q$, and inherits the labeling of $e_1e_2\ldots e_q$. For a constrained system S presented by a labeled graph G, the qth *higher order system of* S, denoted $S^{[q]}$, is the constrained system presented by $G^{[q]}$.

Observe that $S^{[q]}$ is the constrained system whose alphabet is the set of q-blocks of S, obtained from S by replacing each word $w_1w_2\ldots w_\ell$ by the word

$$(w_1w_2\ldots w_q)(w_2w_3\ldots w_{q+1})\ldots(w_{\ell-q+1}w_{\ell-q+2}\ldots w_\ell)\;.$$

Note how S^q differs from $S^{[q]}$: the former divides words into non-overlapping blocks; the latter divides words into blocks which overlap by $q-1$ symbols.

Figure 2.11 shows the edge graph $G^{[2]}$ for the $(0,1)$-RLL labeled graph G in Figure 2.2, and $G^{[3]}$ is shown in Figure 2.12. The reader should contrast this with the third power G^3 in Figure 2.10.

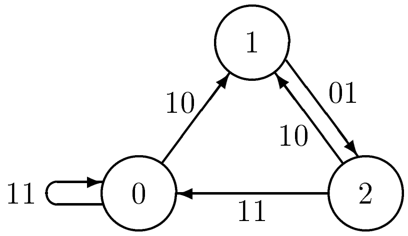

Figure 2.11: Second higher edge graph of labeled graph in Figure 2.2.

The Moore form and co-form of G which were introduced in Section 2.2.7 are almost identical to $G^{[2]}$: To obtain the Moore form (respectively, the Moore co-form), just delete the first (respectively, the second) symbol in each edge label of $G^{[2]}$.

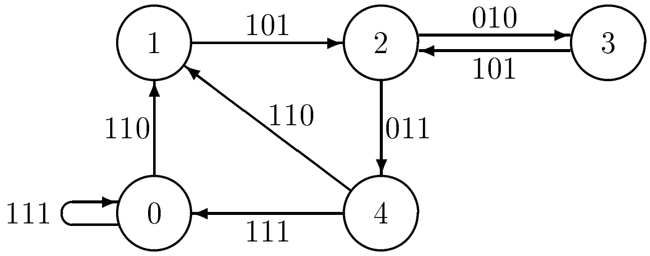

Figure 2.12: Third higher edge graph of labeled graph in Figure 2.2.

2.4.3 Fiber product of graphs

Let G and H be two labeled graphs. We define the *fiber product* of G and H as the labeled graph $G * H$, where

$$V_{G*H} = V_G \times V_H = \{\langle u, u' \rangle \mid u \in V_G,\ u' \in V_H\} ,$$

and $\langle u, u' \rangle \xrightarrow{a} \langle v, v' \rangle$ is in E_{G*H} if and only if $u \xrightarrow{a} v \in E_G$ and $u' \xrightarrow{a} v' \in E_H$. It is easy to verify that the fiber product presents the intersection of the constraints defined by G and H, namely, $S(G * H) = S(G) \cap S(H)$.

Finally, we state a result which asserts that the operations introduced in this section all preserve the properties of labelings introduced in Section 2.2. We leave the proof to the reader.

Proposition 2.7 *The power of a graph, higher edge graph, and fiber product graph all preserve the deterministic, finite anticipation (co-anticipation), and definiteness properties.*

2.5 Irreducibility

2.5.1 Irreducible graphs

A graph G is *irreducible* (or *strongly-connected*) if, for any ordered pair of states u, v, there is a path from u to v in G. A graph is *reducible* if it is not irreducible. Note our use of the term 'ordered': for a given pair of states u, v, we must be able to travel from u to v and from v to u.

All of the graphs in Figures 2.2 through 2.5 are irreducible, while Figure 2.13 shows a reducible graph which presents the system of unconstrained binary words.

Observe that the property of being irreducible does not depend on the labeling and can

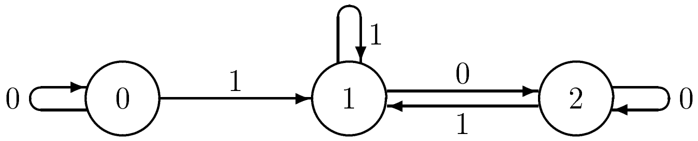

Figure 2.13: Reducible labeled graph for unconstrained binary words.

be described in terms of the adjacency matrix: namely, for every (ordered) pair of states u, v, there exists some ℓ such that $(A_G^\ell)_{u,v} > 0$.

It will be useful later to know that any reducible graph can, in some sense, be broken down into "maximal" irreducible pieces. To make this more precise we introduce the concept of an irreducible component. An *irreducible component* of a graph G is a maximal (with respect to inclusion) irreducible subgraph of G. The irreducible components of a graph are simply the subgraphs consisting of all edges whose initial and terminal states both belong to an equivalence class of the following relation: $u \sim v$ if there is a path from u to v and a path from v to u (we allow paths to be empty so that $u \sim u$).

An *irreducible sink* is an irreducible component H such that any edge which originates in H must also terminate in H. An *irreducible source* is an irreducible component H such that any edge which terminates in H must also originate in H.

Any graph can be broken down into irreducible components with 'transient' connections between the components. The irreducible sinks can have transient connections entering but not exiting. Every graph has at least one irreducible sink (and, similarly, at least one irreducible source). To see this, we argue as follows. Pick an irreducible component and check if it is an irreducible sink. If so, stop. If not, there must be a path leading to another irreducible component. Repeat the procedure on the latter component. The process must eventually terminate in an irreducible sink H; otherwise, the original decomposition into irreducible components would be contradicted. The picture of the irreducible components and their connections is perhaps best illustrated via the adjacency matrix: by reordering the states, $A = A_G$ can be written in block upper triangular form with the adjacency matrices of the irreducible components as block diagonals, as shown in Figure 2.14.

$$A = \begin{pmatrix} A_1 & B_{1,2} & B_{1,3} & \cdots & B_{1,k} \\ & A_2 & B_{2,3} & \cdots & B_{2,k} \\ & & A_3 & \ddots & \vdots \\ & & & \ddots & B_{k-1,k} \\ & & & & A_k \end{pmatrix} .$$

Figure 2.14: Writing matrix in upper triangular form.

Figure 2.15 shows the irreducible components of the graph in Figure 2.13; one is an irreducible sink and the other is an irreducible source.

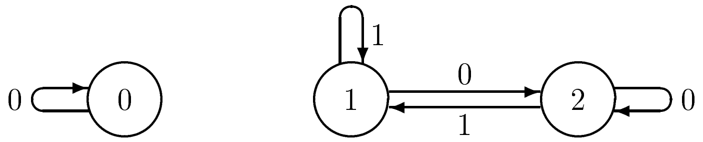

Figure 2.15: Irreducible components of labeled graph in Figure 2.13.

From the point-of-view of finite-state encoder construction, it turns out that, by passing to irreducible components, we can concern ourselves primarily with irreducible labeled graphs; we explain why in Section 4.1.

There are times when the qth power of a graph G will not be irreducible, even when G is. For example, Figure 2.6 shows a labeled graph describing a 2-charge constrained system. Its second power G^2, shown in Figure 2.16, has two irreducible components, G_0 and G_1 (note that in these graphs, the label $+-$ is different from $-+$). This example illustrates the general situation: it can be shown that, if G is an irreducible graph, then any power G^q is either irreducible or decomposes into isolated, irreducible components (see also Figures 2.2 and 2.10). We elaborate upon this in Section 3.3.2.

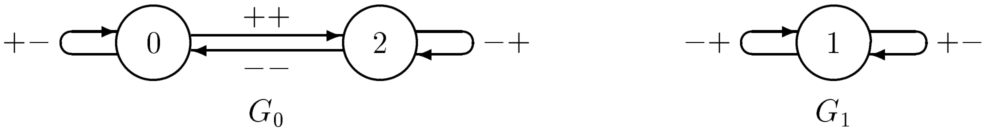

Figure 2.16: Second power of labeled graph in Figure 2.6.

2.5.2 Irreducible constrained systems

A constrained system S is *irreducible*, if for every pair of words $\mathbf{w}, \mathbf{w}'$ in S, there is a word $\mathbf{z}$ such that $\mathbf{wzw}'$ is in S. A constrained system that is not irreducible is called *reducible*.

The following shows that irreducibility of a constrained system can be reformulated in terms of irreducible labeled graphs.

Lemma 2.8 *Let S be a constrained system. The following are equivalent:*

(a) S is irreducible;

(b) S is presented by some irreducible (and in fact, deterministic) labeled graph.

Proof. For (b) $\Rightarrow$ (a), simply connect the terminal state of a path that generates $\mathbf{w}$ to the initial state of a path that generates $\mathbf{w}'$. For (a) $\Rightarrow$ (b), replace inclusion with equality in the stronger statement of the next lemma. $\square$

Lemma 2.9 *Let S be an irreducible constrained system and let G be a labeled graph such that $S \subseteq S(G)$. Then for some irreducible component G' of G, $S \subseteq S(G')$.*

Proof. Let $G_1, G_2, \ldots, G_k$ denote the irreducible components of G. We prove the lemma by contradiction. Suppose that for each $i = 1, 2, \ldots, k$, there is a word $\mathbf{w}_i$ in S but not in $S(G_i)$. Since S is irreducible, there is a word $\mathbf{w}$ that contains a copy of each $\mathbf{w}_i$; moreover, there is a word $\mathbf{z}$ that contains $|V_G|+1$ non-overlapping copies of $\mathbf{w}$. Let γ be a path in G that generates $\mathbf{z}$. Then γ can be written as $\gamma = \gamma_1\gamma_2 \ldots \gamma_{|V_G|+1}$, where each γ_j has a sub-path which generates $\mathbf{w}$. For some $r < t$, the initial states of γ_r and γ_t coincide and, therefore, $\gamma_r\gamma_{r+1} \ldots \gamma_{t-1}$ is a cycle and has a sub-path that generates $\mathbf{w}$. Now, by definition, any cycle in a graph must belong to some irreducible component, say G_i, and thus $\mathbf{w}_i$ is in $S(G_i)$, contrary to the definition of $\mathbf{w}_i$. $\square$

All of the constrained systems that we have considered so far are irreducible, while Figure 2.17 presents a reducible constrained system.

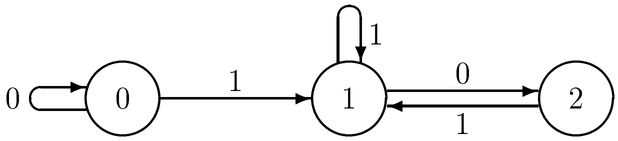

Figure 2.17: Reducible constrained system.

2.6 Minimal presentations

When treating constrained systems, it is useful to present them in a standard manner. Among the various possible presentations of a given constrained system S, the Shannon cover is usually chosen as the canonical presentation of S.

A *Shannon cover* of a constrained system S is a deterministic presentation of S with a smallest number of states.

In general, the Shannon cover is not unique [Jon95], [LM95, Section 3.3]. However, it is unique, up to labeled graph isomorphism, for irreducible constrained systems. We show this in Theorem 2.12 below.

2.6.1 Follower sets and reduced labeled graphs

Let u be a state in a labeled graph G. The *follower set* of u in G, denoted $\mathcal{F}_G(u)$, is the set of all (finite) words that can be generated from u in G. Two states u and u' in a labeled graph G are said to be *follower-set equivalent*, denoted $u \simeq u'$, if they have the same follower set. It is easy to verify that follower-set equivalence satisfies the properties of an equivalence relation.

A labeled graph G is called *reduced* if no two states in G are follower-set equivalent. If a labeled graph G presents a constrained system S, we can construct a reduced labeled graph H from G that presents the same constrained system S by merging states in G which are follower-set equivalent. More precisely, each equivalence class C of follower-set equivalent states becomes a state in H, and we draw an edge $C \xrightarrow{a} C'$ in H if and only if there exists an edge $u \xrightarrow{a} u'$ in G for states $u \in C$ and $u' \in C'$. It is easy to verify that, indeed, $S(H) = S(G)$, and, if G is deterministic, so is H; see [LM95].

2.6.2 The Moore algorithm

The Moore algorithm is an efficient procedure for finding the equivalence classes of the follower-set equivalence relation of states in a deterministic graph. The algorithm is described in [Huff54], [Koh78, Ch. 10], [Moore56].

Let G be a deterministic graph. For a state u in G, let $\mathcal{F}_G^{\ell}(u)$ denote the set of all words of length ℓ that can generated from u in G. Two states u and v in G are said to be *ℓ-(follower-set-)equivalent* in G, if $\mathcal{F}_G^m(u) = \mathcal{F}_G^m(v)$ for $m = 1, 2, \ldots, \ell$. Indeed, ℓ-equivalence is an equivalence relation, and we denote by Π_ℓ the partition of V_G which is induced by the classes of this relation. Also, we denote by $|\Pi_\ell|$ the number of classes in Π_ℓ.

The Moore algorithm iteratively finds the partitions Π_ℓ for increasing values of ℓ, until we reach the partition induced by the follower-set equivalence relation. The partition Π_0 contains one class, namely, V_G. As for 1-equivalence, two states u and v belong to the same equivalence class if and only if the sets of labels of the edges outgoing from state u and from state v are the same. Therefore, the partition Π_1 is easily found from G.

The following is a typical iteration of the Moore algorithm. Assume we have found Π_ℓ for some $\ell \geq 1$. Now, every two $(\ell{+}1)$-equivalent states in G must also be ℓ-equivalent. Hence, $\Pi_{\ell+1}$ is a *refinement* of Π_ℓ (and so $|\Pi_{\ell+1}| \geq |\Pi_\ell|$). More specifically, we put two states u and v in the same class in $\Pi_{\ell+1}$ if and only if (i) u and v belong to the same class in Π_ℓ, and (ii) for each pair of edges, $u \xrightarrow{a} u'$ and $v \xrightarrow{a} v'$, in G (with the same label a), the states u' and v' belong to the same class in Π_ℓ.

Example 2.1 Consider the deterministic graph G in Figure 2.18. We start with the

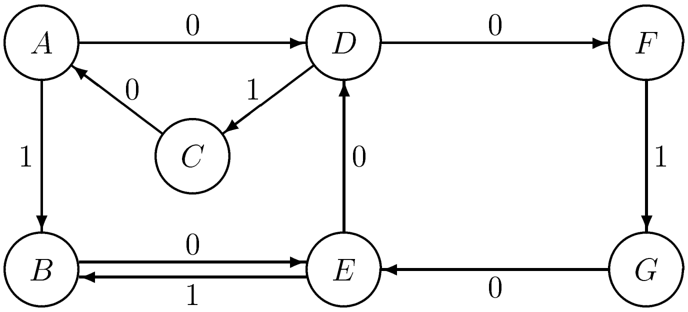

Figure 2.18: Graph G for Example 2.1.

trivial partition, Π_0, which consists of one equivalence class that contains all states, namely,

$$\Pi_0 = \{A, B, C, D, E, F, G\} .$$

The partition Π_1 is obtained by looking at the set of labels of the outgoing edges from each state. That set is $\{0\}$ for states B, C, and G; it is $\{1\}$ for state F; and it is $\{0, 1\}$ for states A, D, and E. Hence,

$$\Pi_1 = \{B, C, G\}\{F\}\{A, D, E\} .$$

To obtain the partition Π_2, we see that the outgoing edges (labeled 0) from states B, C, and G terminate in E, A, and E, respectively, and these terminal states belong to the same equivalence class in Π_1. Hence, $\{B, C, G\}$ will form an equivalence class also in the partition Π_2. On the other hand, the outgoing edges labeled 0 from A, D, and E terminate in D, F, and D, respectively, thus implying that state D should be separated from states A and E in Π_2. Since the outgoing edges labeled 1 from A and E both terminate in state B, we conclude that A and E are 2-equivalent and, so,

$$\Pi_2 = \{B, C, G\}\{F\}\{A, E\}\{D\} .$$

The next iteration will generate no refinement i.e., we end up with $\Pi_3 = \Pi_2$. □

In general, the algorithm finds the partitions Π_ℓ for increasing values of ℓ until $\Pi_{\ell+1} = \Pi_\ell$. Denote by $\ell_{\max}$ the smallest ℓ for which this equality is met. For $\ell > \ell_{\max}$ we will have $\Pi_\ell = \Pi_{\ell_{\max}}$ and, so, the follower-set equivalence relation is identical to the $\ell_{\max}$-equivalence relation. Furthermore,

$$1 = |\Pi_0| < |\Pi_1| < \cdots < |\Pi_{\ell_{\max}-1}| < |\Pi_{\ell_{\max}}| = |\Pi_{\ell_{\max}+1}| \le |V_G| .$$

Therefore, $\ell_{\max} \le |V_G| - 1$, which thus bounds from above the number of iterations in the Moore algorithm. In fact, we have also the following.

Proposition 2.10 *Let G be a deterministic essential graph. Then, for every pair of states u and v in G,*

$$\mathcal{F}_G(u) = \mathcal{F}_G(v) \quad \textit{if and only if} \quad \mathcal{F}_G^{|V_G|-1}(u) = \mathcal{F}_G^{|V_G|-1}(v) \ .$$

.

Having found the partition $\Pi_{\ell_{\max}}$ induced by the follower-set equivalence relation, we can construct a reduced deterministic graph H from G that presents the same constrained system $S(G)$, as was described in Section 2.6.1.

If we apply the reduction to the deterministic graph in Example 2.1, we obtain the graph presentation of the (1, 3)-RLL constraint as shown in Figure 1.16, with the following equivalence relation between the states in that figure and the partition elements of Π_2:

$$0 \longleftrightarrow \{B, C, G\} \ , \quad 1 \longleftrightarrow \{A, E\} \ , \quad 2 \longleftrightarrow \{D\} \ , \quad \text{and} \quad 3 \longleftrightarrow \{F\} \ .$$

2.6.3 Homing words

A *homing word* for a state v in a labeled graph G is a word $\mathbf{w}$ such that all paths in G that generate $\mathbf{w}$ terminate in v.

We will make use of the following lemma.

Lemma 2.11 *Let G be a reduced deterministic graph. Then there is a homing word for at least one state in G. Furthermore, if G is also irreducible, then there is a homing word for every state in G.*

Proof. Since G is reduced, for any two distinct states $u, v \in V_G$, there is a *distinguishing word* $\mathbf{w}$ that can be generated in G from one of these states but not from the other. Start with the set $V_0 = V_G$ and assume that $|V_0| > 1$. Let $\mathbf{w}_1$ be a distinguishing word for some pair of states in V_0. Clearly, $\mathbf{w}_1$ can be generated in G by at most $|V_0| - 1$ paths starting in V_0. Let V_1 denote the set of terminal states of these paths. We can now apply the same argument on V_1 with a distinguishing word $\mathbf{w}_2$ for two states in V_1 and continue this way until we end up with a set $V_m = \{v\}$. The word $\mathbf{w}_1\mathbf{w}_2 \ldots \mathbf{w}_m$ is a homing word for v in G.

If G is also irreducible, then we can extend the homing word for v to a homing word for every state in G. □

We can use the notion of homing words to obtain the following equivalent definition for labeled graphs with finite memory: a labeled graph G has finite memory if there is an integer N such that all words of length N in $S(G)$ are homing (and the memory of G is then the smallest such N).

2.6.4 Shannon cover of irreducible constrained systems

The following result summarizes the main properties of the Shannon cover of irreducible constrained systems. See [Fi75a], [Fi75b], [KN90].

Theorem 2.12 *Let S be an irreducible constrained system.*

(a) The Shannon cover of S is unique, up to labeled graph isomorphism. In fact, the Shannon cover is the unique presentation of S which is irreducible, deterministic, and reduced.

(b) For any irreducible deterministic presentation G of S, the follower sets of G coincide with the follower sets of the Shannon cover.

Proof. First, note that any minimal deterministic presentation of S must be reduced. Otherwise we could merge all states with the same follower sets, as was described in Section 2.6.1. Furthermore, by Lemma 2.9, any minimal deterministic presentation must be irreducible.

For the proof of (a), suppose that H and H' are irreducible reduced deterministic graphs that present S. Let u be an arbitrary state in H. By Lemma 2.11, there is a homing word $\mathbf{w}$ for u in H and a homing word $\mathbf{w}'$ for some state in H'. Since S is irreducible, there exists a word $\mathbf{z}$ such that $\mathbf{w}'' = \mathbf{w}'\mathbf{z}\mathbf{w} \in S$. Clearly, $\mathbf{w}''$ is a homing word for the state u in H and for some state u' in H'. Since $S(H) = S(H') = S$, we must have $\mathcal{F}_H(u) = \mathcal{F}_{H'}(u')$. Furthermore, since both H and H' are deterministic, the 'outgoing picture' from state u in H must be the same as that from state u' in H'. Hence, if $u \xrightarrow{a} v \in E_H$, then we must have $u' \xrightarrow{a} v' \in E_{H'}$ and $\mathcal{F}_H(v) = \mathcal{F}_{H'}(v')$. Continuing this way, it follows that for every state $u \in V_H$ there is a state $u' \in V_{H'}$ with the same follower set, and vice versa. Since both H and H' are reduced, we must have $H = H'$, up to labeled graph isomorphism.

For the proof of (b), we form a new graph H from G by merging all states of the latter with the same follower sets. It is not hard to see that H is irreducible, deterministic, and reduced. Then, apply part (a) to see that H is isomorphic to the Shannon cover. Clearly, G and H have the same follower sets. $\square$

The following lemma generalizes part (b) of Theorem 2.12 to the situation where H presents a subsystem of $S(G)$.

Lemma 2.13 *Let G and H be two irreducible deterministic graphs. Then $S(H) \subseteq S(G)$ if and only if for every $v \in V_H$ there exists $u \in V_G$ such that $\mathcal{F}_H(v) \subseteq \mathcal{F}_G(u)$.*

Proof. The sufficiency is immediate. As for the necessity, by Lemma 2.8, $S(H)$ is irreducible. Thus, by Lemma 2.9, there is an irreducible component G' of the fiber product

$G * H$ such that $S(G') = S(G * H) = S(G) \cap S(H) = S(H)$. By Theorem 2.12 (part (b)), for every state $v \in V_H$ there exists a state $\langle u, u' \rangle \in V_{G'} \subseteq V_{G*H}$ such that

$$\mathcal{F}_H(v) = \mathcal{F}_{G'}\Big(\langle u, u' \rangle\Big) \subseteq \mathcal{F}_{G*H}\Big(\langle u, u' \rangle\Big) \subseteq \mathcal{F}_G(u),$$

as desired. □

Given some presentation of an irreducible constrained system S, the Shannon cover can be constructed as follows: First, use the determinizing construction of Section 2.2.1 to find a deterministic presentation G of S. By Lemma 2.9, S is presented by one of the irreducible components, say H, of G. Using Lemma 2.13, we can identify H among the irreducible components of G: for every other irreducible component H' of G, we must have $S(H') \subseteq S(H)$. Next, use the Moore algorithm of Section 2.6.2 (in particular, Proposition 2.10) to merge follower-set equivalent states in H to obtain an irreducible reduced deterministic graph. The latter is, by Theorem 2.12(a), the Shannon cover of S.

As an example, the labeled graph in Figure 2.3 is a deterministic presentation of the $(0, 1)$-RLL constrained system, but it is not the Shannon cover because states 0 and 2 have the same follower set. Indeed, the labeled graph in Figure 2.2 is the Shannon cover of the $(0, 1)$-RLL constrained system because it is deterministic, irreducible, and 0 is the label of an outgoing edge from state 0, but not from state 1. Note that if we merge states 0 and 2 in the labeled graph of Figure 2.3, we get the Shannon cover in Figure 2.2. The reader can verify that the Shannon cover of an RLL constrained system is the labeled graph depicted in Figure 1.3 in Chapter 1 and that Figure 1.14 displays the Shannon cover of a charge constrained system.

We end this section by pointing out the intrinsic nature of the follower sets of the states in the Shannon cover of an irreducible constrained system.

For a constrained system S and a word $\mathbf{w} \in S$, the *tail set* $\mathcal{F}_S(\mathbf{w})$ is the set of all words $\mathbf{z}$ such that $\mathbf{wz} \in S$. A word $\mathbf{w} \in S$ is a *magic word* if, whenever $\mathbf{zw}$ and $\mathbf{wz}'$ are in S, so is $\mathbf{zwz}'$.

Proposition 2.14 *Let S be an irreducible constrained system. The homing words of the Shannon cover of S coincide with the magic words of S, and the follower sets of the states of the Shannon cover coincide with the tail sets of the magic words of S.*

The proof of this proposition is left to the reader (Problem 2.22).

2.6.5 Shannon cover of finite-type constrained systems

The Shannon cover can be used to detect the finite-type and almost-finite-type properties.

Proposition 2.15 *An irreducible constrained system is finite-type (respectively, almost-finite-type) if and only if its Shannon cover has finite memory (respectively, finite co-anticipation).*

Proof. We first prove this for finite-type systems. Let S be an irreducible finite-type constrained system. By Proposition 2.5 and Lemma 2.4, there is a presentation G of S with finite memory. Also, by Lemma 2.9, there is an irreducible component G' of G such that $S = S(G')$. Since G' is an irreducible graph with finite memory, then it must be deterministic. By merging states in G', we obtain the Shannon cover of S. Therefore, the Shannon cover of an irreducible finite-type constrained system S must have finite memory.

Now, assume that S is an irreducible almost-finite-type constrained system. There is a presentation H of S which has finite co-anticipation and finite anticipation. It is not hard to see that the determinizing construction, introduced in Section 2.2.1, preserves finite co-anticipation. Thus, by Lemma 2.1, we may assume that H is actually deterministic. By Lemma 2.9, we may assume that H is irreducible. Then, by Theorem 2.12, the reduced labeled graph obtained from H is the Shannon cover G of S. In particular, this gives a graph homomorphism from H to G; that is, there is a mapping f^* from states of H to states of G and a mapping f from edges of H to edges of G_S such that f is label-preserving and the initial (respectively, terminal) state of $f(e)$ is $f^*(\sigma_H(e))$ (respectively, $f^*(\tau_H(e))$).

Suppose, for the moment, that H is also co-deterministic. Create a new labeled graph $\bar{H}$ from H by replacing the edge label of an edge e by $f(e)$. Then $\bar{H}$ presents the constrained system $\bar{S}$ defined by labeling the edges of G distinctly. Moreover, $\bar{H}$ is co-deterministic since H is. When we merge H to form G, we are also merging $\bar{H}$ to form the Shannon cover $\bar{G}$ of $\bar{S}$, and by reversing the arrows, we see that any two states in $\bar{H}$ that are merged via f^* have the exact same set of incoming f-labels. So, whenever $f^*(v') = v$, the set of f-labels of the incoming edges to v' is precisely the set of incoming edges to v.

We claim by Lemma 2.1, we may assume that H is actually deterministic. By

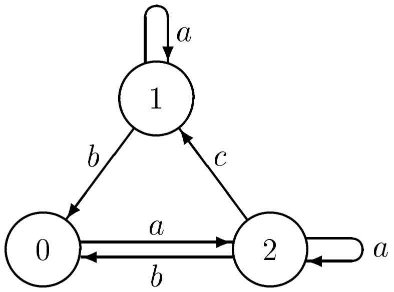

Figure 2.19: Constrained system which is not almost-finite-type.

2.7 Testing algorithms

In this section, we outline efficient algorithms for testing losslessness, finite anticipation, finite memory, and definiteness of a given labeled graph.

2.7.1 Testing for losslessness

The algorithm for testing losslessness of a given labeled graph is due to Even [Even65] and is based on the following proposition (see also [Huff59] and [Koh78, Ch. 14]).

Proposition 2.16 *A labeled graph G is lossless if and only if for every $u, u' \in V_G$, there is no path in the fiber product $G * G$ from state $\langle u, u \rangle$ to state $\langle u', u' \rangle$ that passes through a state of the form $\langle v, v' \rangle$, $v \neq v'$.*

(Recall that we assume that labeled graphs do not contain parallel edges that are labeled the same; such graphs would necessarily be lossy and Proposition 2.16 would not apply to them.)

Proposition 2.16 implies the following algorithm for testing the losslessness of a given labeled graph G. We start by constructing the fiber product $G * G$. Let U denote the states in $G * G$ of the form $\langle u, u \rangle$ for some $u \in V_G$, and let W be the set of all states $\langle v, v' \rangle$ in $G * G$, $v \neq v'$, with an incoming edge from a state in U. To verify that no path which starts in W terminates in U, we proceed as follows: For $\ell = 0, 1, \ldots, |V_G|^2 - 1$, we mark iteratively the states in $G * G$ that can be reached from W by a path of length $\leq \ell$ (this is known as the breadth-first-search (BFS) procedure [Even79]). Then check whether any of the states in U has been marked.

2.7.2 Testing for finite anticipation

A similar algorithm, also due to Even [Even65], allows us to find the anticipation of a given labeled graph. The algorithm is based on the following.

Proposition 2.17 *Let G be a labeled graph and denote by W the set of all states $\langle v, v' \rangle$ in $G * G$, $v \neq v'$, with an incoming edge from a state of the form $\langle u, u \rangle$. Then, G has finite anticipation if and only if no path in $G * G$ that starts at any state in W contains a cycle. If W is empty, then G is deterministic and $\mathcal{A}(G) = 0$. Otherwise, the length of the longest path from W equals $\mathcal{A}(G) - 1$.*

The anticipation of G can therefore be efficiently computed by constructing a sequence of graphs $H_0, H_1, H_2, \ldots, H_t$, where $H_0 = G * G$ and H_i is obtained from H_{i-1} by deleting all states in H_{i-1} with no outgoing edges. The procedure terminates when $H_{i-1} = H_i$ or when H_i contains no states that belong to the set W. In the latter case, the number of iterations, t, equals the anticipation of G. Otherwise, if H_t does contain states of W, then G has infinite anticipation.

Noting that $\langle v, v' \rangle$ and $\langle v', v \rangle$ are follower-set equivalent states in $G * G$, we can construct a reduced labeled graph G' out of $G * G$, where each such pair of states merges into one state of G'. The labeled graph G' will contain at most $|V_G|$ states of the form $\langle u, u \rangle$, $u \in V$, and at most $|V_G|(|V_G|-1)/2$ states of the form $\langle v, v' \rangle$, $v \neq v'$. Now, Proposition 2.17 applies also to the paths in G'. The longest path in G' that neither visits the same state twice, nor visits states of the form $\langle u, u \rangle$, is of length $|V_G|(|V_G|-1)/2 - 1$. Hence, we have the following.

Corollary 2.18 *Let G be a labeled graph. If G has finite anticipation, then $\mathcal{A}(G) \leq |V_G|(|V_G|-1)/2$.*

There are constructions of labeled graphs G that attain the bound of Corollary 2.18 for every value of $|V_G|$ [Koh78, Appendix 14.1].

2.7.3 Testing for finite memory

The following is basically contained in [PRS63] and [Koh78, Ch. 14] (see Problem 2.24).

Proposition 2.19 *Let G be a labeled graph. Then, G has finite memory if and only if there exists an integer N such that all paths in $G * G$ of length N terminate in states of the form $\langle u, u \rangle$, $u \in V_G$. The smallest such N, if any, equals $\mathcal{M}(G)$.*

In particular, in view of Proposition 2.15, given an irreducible constrained system S, we can apply Proposition 2.19 to the Shannon cover of S to check whether S has finite memory.

The following corollary is the counterpart of Corollary 2.18 for the memory of a labeled graph.

Corollary 2.20 *Let G be a labeled graph. If G has finite memory, then $\mathcal{M}(G) \le |V_G|(|V_G|-1)/2$.*

The bound of Corollary 2.20 is tight [Koh78, Ch. 14, Problems].

2.7.4 Testing for definiteness

Next we outline an efficient test for determining whether a given labeled graph G is (m,a)-definite. (Note that we could use a test for definiteness also for testing finite memory. However, to this end, Proposition 2.19 provides a faster algorithm.)

Let G be a labeled graph and let A_{G*G} be the adjacency matrix of $G*G$. Denote by B_{G*G} the $|V_G|^2 \times |V_G|^2$ matrix whose rows and columns are indexed by the states of $G*G$ and for every $u, u', v, v' \in V_G$, the entry $(B_{G*G})_{\langle u,u'\rangle,\langle v,v'\rangle}$ equals the number of pairs of *distinct* edges $u \to v$ and $u' \to v'$ in G that have the same label. In other words,

$$(B_{G*G})_{\langle u,u'\rangle,\langle v,v'\rangle} = \begin{cases} (A_{G*G})_{\langle u,u'\rangle,\langle v,v'\rangle} & \text{if } u \neq u' \text{ or } v \neq v' \\ (A_{G*G})_{\langle u,u'\rangle,\langle v,v'\rangle} - (A_G)_{u,v} & \text{if } u = u' \text{ and } v = v' \end{cases} .$$

Proposition 2.21 *A labeled graph G is (m,a)-definite if and only if*

$$A_{G*G}^{\mathsf{m}} \, B_{G*G} \, A_{G*G}^{\mathsf{a}} = 0 .$$

The proof is left as an exercise (Problem 2.25).

Problems

Problem 2.1 Let G be a labeled graph.

1. Show that G has a unique maximal essential subgraph H.
2. Let S be the constrained system defined by G, and let S' be the subset of S consisting of all words $\mathbf{w}$ such that for any integer ℓ there are words $\mathbf{z}$ and $\mathbf{z}'$ of length ℓ such that $\mathbf{zwz}'$ belongs to S. Show that S' is the constrained system presented by H.

Problem 2.2 Let G be a graph and let G' and G'' be the Moore form and Moore co-form of G, respectively. Prove the following claims:

1. Edges with the same terminal state in G' have the same labels, and edges with the same initial state in G' have the same labels.
2. If the out-degrees of the states in G are all equal to n, then so are the out-degrees of the states in G' and G''.
3. $\mathcal{A}(G') = \mathcal{A}(G)$.
4. $\mathcal{A}(G'') \le \mathcal{A}(G) + 1$. When is this inequality strict?

Problem 2.3 Let G be the graph in Figure 2.20 with labels over the alphabet $\Sigma = \{a, b, c\}$. Show that the anticipation of G is 3.

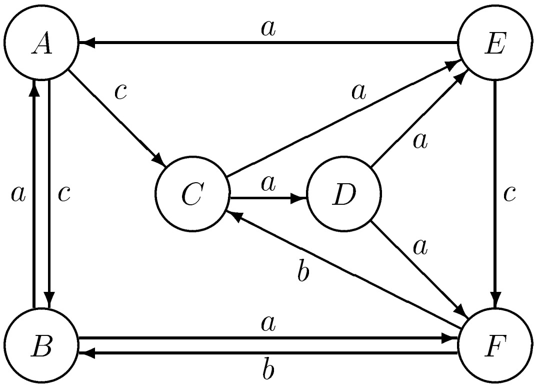

Figure 2.20: Graph G for Problem 2.3.

Problem 2.4 Let G be the graph in Figure 2.21 with labels over the alphabet $\Sigma = \{a, b, c\}$. Show that the anticipation of G is 4.

Problem 2.5 Find the memory of the graph G in Figure 2.22.

Problem 2.6 Let S be a constrained system with finite memory $\mathcal{M} \ge 1$ over an alphabet Σ.

1. Show that for every constrained system S' over Σ,
$$S' \subseteq S \quad \text{if and only if} \quad (S' \cap \Sigma^i) \subseteq (S \cap \Sigma^i) \quad \text{for every} \quad i = 1, 2, \ldots, \mathcal{M}{+}1 \ .$$
2. Show that there exists a constrained system S' that is *not* contained in S, yet satisfies the containment
$$(S' \cap \Sigma^i) \subseteq (S \cap \Sigma^i) \quad \text{for every} \quad i = 1, 2, \ldots, \mathcal{M} \ .$$

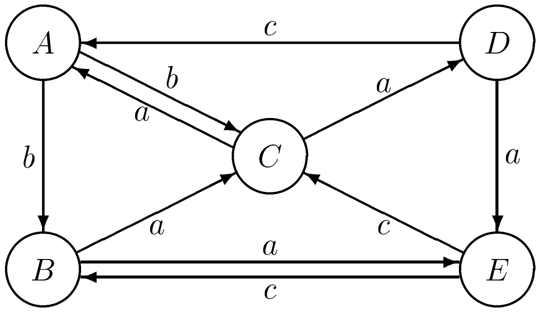

Figure 2.21: Graph G for Problem 2.4.

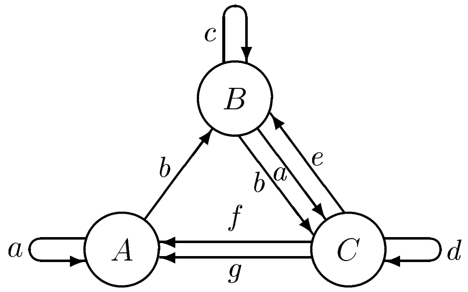

Figure 2.22: Graph G for Problem 2.5.

Problem 2.7 Prove Proposition 2.6.

Problem 2.8 Show that if S_1 and S_2 are constrained systems that are almost-finite-type, then so is $S_1 \cap S_2$.

Problem 2.9 Let G_1 and G_2 be graphs and $G_1 * G_2$ be the fiber product of G_1 and G_2.

1. Show that $S(G_1 * G_2) = S(G_1) \cap S(G_2)$.
2. Prove or disprove the following:
   (a) If G_1 and G_2 are both lossless, then so is $G_1 * G_2$.
   (b) If $G_1 * G_2$ is lossless, then so are both G_1 and G_2.

Problem 2.10 Let G_1 and G_2 be graphs with finite memory. Show that

$$\mathcal{M}(G_1 * G_2) \leq \max\{\mathcal{M}(G_1), \mathcal{M}(G_2)\} .$$

Problem 2.11 Let G be the graph presentation of a 4-charge constrained system in Figure 2.23.

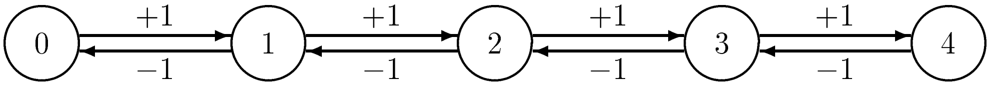

Figure 2.23: Graph G for Problem 2.11.

1. Find a shortest homing word for every state in G.
2. What can be said about the memory of G?

Problem 2.12 Let G be the graph in Figure 2.24.

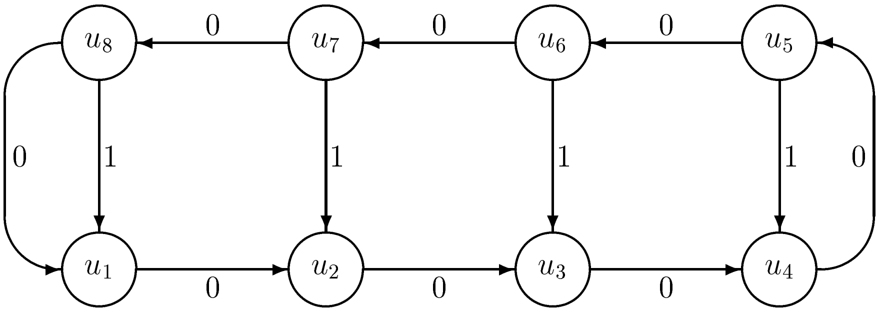

Figure 2.24: Graph G for Problem 2.12.

1. Find a shortest homing word in G.
2. What can be said about the memory of G?

Problem 2.13 Let S be the constrained system presented by the graph G in Figure 2.25.

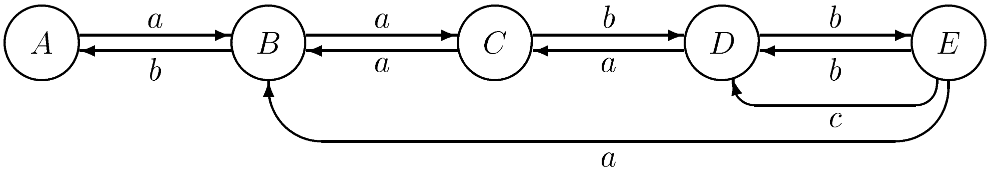

Figure 2.25: Graph G for Problem 2.13.

1. Find a shortest homing word for every state in G.
2. Construct the graph G^2.

3. Find the memory of each irreducible component of G^2.
4. What can be said about the memory of G^2?

Problem 2.14 Let G_1 and G_2 be graphs. Show that $\mathbf{w}$ is a homing word of $G_1 * G_2$ if and only if $\mathbf{w}$ is a homing word of both G_1 and G_2.

Problem 2.15 Let G be an irreducible graph and let G' and G'' be the Moore form and Moore co-form of G, respectively. Show that both G' and G'' are irreducible.

Problem 2.16 Let G be the fiber product of the graphs in Figure 1.15 and Figure 1.16; note that G presents the 6–$(1,3)$-CRLL constrained system.

1. Draw the graph G.
2. Show that one of the irreducible components of G^2 can be reduced to the graph H in Figure 2.26.

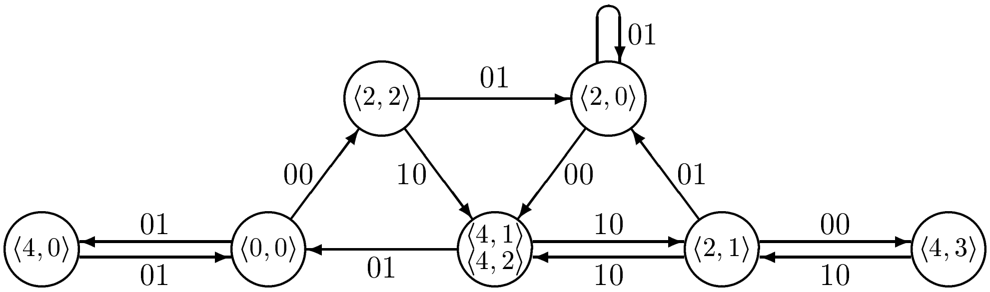

Figure 2.26: Graph H for Problem 2.16.

Problem 2.17 Let S_0 be an irreducible constrained system and let G be a graph such that $S_0 \subseteq S(G)$. Show that there is an irreducible component H of G such that $S_0 \subseteq S(H)$.

Problem 2.18 Let S_1 and S_2 be irreducible constrained systems such that $S_1 \subseteq S_2$.

1. Show that there is an irreducible deterministic presentation H_1 of S_1 that is a subgraph of an irreducible deterministic presentation H_2 of S_2.

   Hint: Let G_1 and G_2 be the Shannon covers of S_1 and S_2, respectively, and, as in the proof of Lemma 2.13, consider an irreducible component H_1 of $G_1 * G_2$ that presents S_1. Show how H_1 can be extended to an irreducible deterministic graph H_2 such that $S(H_2) = S_2$.

2. Show by example that not always can H_1 be taken as the Shannon cover of S_1; that is, for the provided example, the Shannon cover of S_1 is not a subgraph of any *irreducible* deterministic presentation of S_2.
3. Show by example that not always can H_2 be taken as the Shannon cover of S_2.

Problem 2.19 Let G and H be deterministic graphs where H is irreducible. Suggest an efficient algorithm for determining whether $S(H) \subseteq S(G)$.

Problem 2.20 Let G be a deterministic graph that presents an irreducible constrained system S (but G is not necessarily irreducible). It follows from Lemmas 2.8 and 2.9 that G contains an irreducible component G' such that $S(G) = S(G')$. Suggest an efficient algorithm for finding G'.

Hint: See Problem 2.19

Problem 2.21 Let S be an irreducible constrained system with finite memory. Show that the memory of S equals the memory of the Shannon cover of S.

Problem 2.22 Prove Proposition 2.14.

Problem 2.23 Let G be a labeled graph and let W be the subset of states of $G * G$ as defined in Proposition 2.17. Denote by $\mathbf{x} = (x_{\langle v,v' \rangle})_{\langle v,v' \rangle}$ the characteristic vector of W as a subset of the states of $G * G$; that is, the entries of $\mathbf{x}$ are indexed by the states of $G * G$, and

$$x_{\langle v,v' \rangle} = \begin{cases} 1 & \text{if } \langle v,v' \rangle \in W \\ 0 & \text{otherwise} \end{cases} .$$

1. Show that G has finite anticipation if and only if there is a nonnegative integer ℓ such that

$$\mathbf{x} A_{G*G}^{\ell} = \mathbf{0} .$$

2. Show that if G has finite anticipation, then its anticipation is the smallest nonnegative integer ℓ that satisfies the equality in 1.

Problem 2.24 Let $G = (V, E, L)$ be a deterministic graph and let H be the graph obtained from $G * G$ by deleting the set of states $\{\langle v,v \rangle : v \in V\}$, with their incident edges.

1. Show that G has finite memory if and only if H has no cycles.
2. Show that if G has finite memory, then the memory is bounded from above by $|V|(|V|-1)/2$.

Problem 2.25 Prove Proposition 2.21.

Problem 2.26 Let U be a set of positive integers (U may be either finite or infinite). The *U-gap system* is defined as the set of all sub-words of all binary words in which each runlength of 0's belongs to U.

1. Let U be the set of even integers. Show that the U-gap system is a constrained system but has no finite memory.
2. Let U be the set of prime integers. Show that the U-gap system is not a constrained system.
3. Formulate complete necessary and sufficient conditions on U for the U-gap system to be—
   - (a) a constrained system;
   - (b) a constrained system with finite memory.