LOGSPACE HIERARCHIES, POLYNOMIAL TIME AND THE COMPLEXITY OF FAIRNESS PROBLEMS CONCERNING ω-MACHINES

Louis E. Rosier and Hsu-Chun Yen

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

TR-85-08 May 1985

Abstract

In this paper, we define a *restricted logspace oracle hierarchy* which turns out to be equivalent to the *logspace alternation hierarchy* and thus is contained within the second level of the *logspace oracle hierarchy*. We then examine problems concerning various types of "fair" computations with respect to ω-FSM's, and ω-1CM's. For example, we consider the emptiness problem for ω-FSM's and ω-1CM's where acceptance is defined in the usual fashion, but with a fairness constraint imposed on accepting computations. Our results yield problems that are complete not only for NLOGSPACE and PTIME but the second and third level of the restricted logspace hierarchy as well. As far as we know, these are the first natural problems shown to be complete for various levels of the logspace alternation hierarchy. Furthermore, the results can be used to strengthen known results concerning various related fairness problems that involve logic and/or networks of CFSM's.

1. Introduction

One of the more fundamental tools used in categorizing the complexity of decision problems is the notion of completeness. Most notable, of course, is the class of problems which are NP-complete. See e.g. [7, 11, 19]. Other complexity classes for which abundant numbers of complete problems are known include NLOGSPACE, PTIME, and PSPACE. See e.g. [11, 16-18]. More recently certain natural problems have been shown to be complete for certain complexity classes which are "between" NP and PSPACE. Recent advances here include [1, 26, 27], which tend to consider problems which are complete for various levels of the Polynomial-time Hierarchy [32]. One could also consider the analogous complexity classes for space hierarchies which fall "between" NLOGSPACE and PTIME. Appropriate consideration here then should focus on the *logspace oracle hierarchy*, and the *logspace alternation hierarchy* (which is contained in the second level of the logspace oracle hierarchy), which were introduced in [3] and [30].[1] Both of these hierarchies are contained in PTIME $\cap$ DSPACE($\log^2$n). Other complexity classes "below" PTIME for which complete problems are known can be found in [6].

In this paper, we define a *restricted logspace oracle hierarchy* (which as we shall later see turns out to be essentially equivalent to the logspace alternation hierarchy of [3]) which is also contained within the second level of the logspace oracle hierarchy. Our restrictions have to do with fixing the number of oracle calls allowed during a computation. We then examine problems concerning various types of "fair" computations with respect to *ω-Finite State Machines* (ω-FSM's), *ω-One Counter Machines* (ω-1CM's), *ω-Blind Counter Machines* (ω-BCM's) and networks of *Communicating Finite State Machines* (CFSM's). For example, we consider the emptiness problem for ω-FSM's, ω-1CM's and ω-BCM's where acceptance is defined as in [5, 21], but with a fairness constraint imposed on accepting computations. (For a discussion about various types of fairness see e.g. [12, 22].) Our results yield problems that are complete for the following classes:

- NLOGSPACE,
- the second level of the restricted logspace oracle hierarchy,
- the third level of the restricted logspace oracle hierarchy, and
- PTIME.

These results, we feel, are interesting for two reasons:

---

[1] These were referred to, in [30], as the log n space "oracle hierarchy" and log n space "alternation hierarchy", respectively.

i. As far as we know, these are the first natural problems which have been shown to be complete for various levels of the logspace alternation hierarchy of [3].

ii. The results can also be used to show that the canonical form model checking problem of [9] (see also [24]) is PTIME-complete; and that certain liveness questions [4] concerning networks of CFSM's are either PTIME-hard or complete for the third level of the restricted logspace oracle hierarchy.

Related complexity results concerning questions about fair computations can be found in [8-10, 24]; while completeness results concerning complexity classes "between" DLOGSPACE and NLOGSPACE can be found in [23].

In what follows, we assume the reader is familiar with the basic tenets of automata and complexity theory. Relevant sources would include [11, 16]. The basic computational model used in this paper is the *nondeterministic offline multitape Turing machine* (NTM). Since this paper concerns itself with many different problems the relevant definitions are introduced in the text as they are needed. For brevity, we sometimes use the terms DL and NL to denote the complexity classes DLOGSPACE and NLOGSPACE, respectively. All completeness classes mentioned in this paper are with respect to deterministic logspace (many-to-one) reductions.

The remainder of the paper is organized as follows. In section 2, we define the restricted logspace oracle hierarchy and in section 3, we illustrate some useful graph problems that are complete for certain levels within this hierarchy. Section 4 concerns itself with the complexity of various problems with respect to ω-FSM's, ω-1CM's and ω-BCM's. The last section seeks to apply our results to problems considered in [4] and [9].

2. The restricted logspace oracle hierarchy

In [30], the logspace oracle hierarchy which fell between NSPACE(log n) and PTIME $\cap$ DSPACE($\log^2$n) was defined. Our intention, in this section, is to see what happens to this hierarchy when constraints are imposed on the OTM's restricting the number of allowable oracle calls made during the course of a computation. In particular, we focus on the case where at most one oracle call is allowed during the course of a computation. What happens is that we obtain a new hierarchy - which we call the restricted logspace oracle hierarchy. We then show that the restricted logspace oracle hierarchy coincides with the logspace alternation hierarchy [3], which was also discussed in [30]. This characterization is potentially useful because when considering problems related to this hierarchy it allows us to provide the analysis using either computational model. Furthermore, this hierarchy has the following interesting property:

- As pointed out in [30], the restricted logspace oracle hierarchy (or the logspace alternation hierarchy), unlike its polynomial time analog, is clearly contained within the second level of the logspace oracle hierarchy.

In subsequent sections, we show that some natural problems concerning the notion of fair computations are complete for certain levels in this hierarchy. This study, we feel, is of interest because it allows us to have more insight into the question whether the inclusion in "NSPACE(log n) $\subseteq$ PTIME" is proper. Results concerning relativized logspace have received broad attention in recent years [20, 25, 30]. Unlike the case with the polynomial-time hierarchy [1, 26], however, little has been done in actually "pigeonholing" natural problems into the logspace hierarchies. The main contribution of this paper will be exactly that; and thus, in some sense, we examine this portion of sublinear space from the problem-oriented point of view.

Informally, an *oracle Turing machine* (OTM, for short) M is a nondeterministic offline multitape Turing machine with an additional write-only tape called a *query tape* and three distinguished states called *query*, *yes* and *no states*. In addition, a set A, called the *oracle set* is always related to the computation of the OTM M. For convenience, we use $L(M^A)$ to denote the language accepted by the OTM M using A as its oracle set (denoted by M^A). The computation of an OTM is similar to that of an ordinary NTM except when visiting the query states. When in a query state, if the string on the query tape is in the oracle set, then the machine enters the yes state; otherwise, it enters the no state. Moreover, the contents of the query tape will be erased immediately upon entering the yes or no state. The machine can write a symbol on the query tape in every state except the query state. In some sense, it is easier to think of the computation of M^A as follows. Let A' be a machine that accepts the set A. When M reaches a query state, it will "consult" the machine A' (which will be referred as a "query machine"), and A' returns yes if the contents of the query tape is accepted by A', and returns no otherwise. Then M enters the yes or no state depending on the value returned by A'. Note, however, that the complexity of the computation is only concerned with those resources used by M and not the oracle. Using this concept, one may easily generalize to the class of languages X^Y, where X and Y are complexity classes. Formally, we define X^Y to be the complexity class such that for each language L in X^Y, L can be recognized by an OTM M with oracle set A where M operates within the complexity constraints of X and A is in Y.

In order to study the space hierarchy below PTIME, it is natural to define classes like DL^{NL}, NL^{NL},..., etc. Unfortunately, this kind of definition does not seem to be of great interest because when the OTM's are defined as above one obtains $NL^{NL}=NL^{DL}=NP$. That is, the logspace hierarchy jumps into the polynomial-time hierarchy on the second level. As pointed out in [20, 30], this phenomenon occurs because nondeterminism when combined with the ability to write long strings on the query tape, interacts in such a way that the machine's power is boosted dramatically.

To limit the undesired boost in the power of the OTM's, [30] required that only strings of logarithmic length could be written on the query tape. However, the query machine was also, in a sense, allowed read access to the input of the oracle machine. More precisely, the string that appears on the query tape at any instance will be of the form x#y, where x (which cannot be altered by the OTM) is a copy of the OTM's input tape, and |y| is O(log|x|). One may consider the string x# to always be on the query tape (i.e. it will not be erased), and only the y part to be alterable during the course of a computation. In this paper, we consider only OTM's of this sort. Using such OTM's, a hierarchy, similar to the polynomial time hierarchy, was defined in [30] as follows:

$$\Sigma_0^L = \mathrm{DL},$$

$$\Sigma_{k+1}^L = \mathrm{NL}^{\Sigma_k^L},$$

$$\Delta_{k+1}^L = \mathrm{DL}^{\Sigma_k^L}.$$

The following theorem was also shown in [30].

Theorem 2.1: $\cup_k \Sigma_k^L \subseteq \mathrm{PTIME} \cap \mathrm{DSPACE}(\log^2 n)$.

An interesting question now arises when one also restricts the number of oracle calls allowed during the course of a computation. Intuitively, the power of an OTM seems to be enhanced as it is allowed to make more oracle calls. We define the hierarchies $\Sigma_k^{L[d]}$ and $\Delta_k^{L[d]}$ to be the classes where, in each level the OTM, during the course of its computation, is allowed to consult the oracle at most d times. The next theorem shows an interesting result concerning the class $\mathrm{NL}^{\mathrm{NL}[d]}$, where $d \geq 1$ is a fixed constant.

Theorem 2.2: Let $d>1$ be an integer constant and let $A \in \mathrm{NL}$. Furthermore, let M be an OTM that during the course of any computation consults its oracle at most d times. Then there exists an $A' \in \mathrm{NL}$ and an OTM M_1 that during the course of any computation consults its oracle at most once such that:

1. $L(M^A) = L(M_1^{A'})$, and

2. For M_1, the last step of any computation is an oracle call and only when the answer to this oracle call is "no" will the machine accept the input.

Proof. Let $d>1$ be an arbitrary fixed constant. Let M be an OTM that makes at most d oracle calls during the course of any computation. Let T be an NTM which accepts A and runs in O(log n) space. In what follows, we define A' and show how to construct an OTM M_1 to simulate the computation of M. M_1 will have one more work tape than M which we shall refer as T_2. (The additional worktape is not necessary but it allows a much briefer description of M_1.) The basic idea of the simulation is that, M_1 will simulate the computation of M by writing symbols (those that are supposed to be written on M's query tape) on T_2 until a query state is visited. At this time M_1

"guesses" the oracle's response. If M_1 guesses "yes" then it uses T to nondeterministically verify the guess. If T is able to do so then the computation continues; otherwise the input is rejected. If, on the other hand, M_1 guesses "no" then the "query" string on T_2 is appended to M_1's actual query tape - the guess to be verified later in the computation. The computation is then continued. Finally when an accepting state is reached, the oracle will be consulted to determine whether the previous "no guesses" were correct. Then M_1 accepts or rejects accordingly. More precisely, in each query state M_1 will nondeterministically choose one of the following to execute:

1. Call the subroutine T using T_2 as its input (along with a copy of M's original input). If T accepts, then M_1 enters the yes state; otherwise the input is immediately rejected.

2. Enter the no state and at the same time, append the contents of T_2 to the end of M_1's query tape. (A separator "," is also added.)

Furthermore, each time M_1 enters yes or no state the string y on T_2 will be immediately erased. Now define $A'=\{(x_1,x_2,...,x_m)|\ x_i\in\Sigma^*,\ 1\leq m\leq d,\ \text{and}\ \exists i,\ 1\leq i\leq m,\ x_i\in A\}$. Note that λ (the empty string) is not in A'. Since A is in NL, clearly A' is also in NL. During the computation of M_1, when the accepting state is reached, an oracle call, using A' as its oracle set, will be made. If the answer is "no", M_1 will then stop and accept the input; otherwise, M_1 rejects the input. The simulation clearly works, since the function of the oracle call is to verify whether the "no" guesses made during the computation were correct. □

From this theorem we can immediately obtain the following:

Corollary 2.1: $NL^{NL[d]} \equiv NL^{NL[1]}$, for any fixed constant d.

At this time, we do not know if the above corollary can be extended to higher levels of the hierarchy.

In [3], the alternation logspace hierarchy was introduced based on the model of *alternating Turing machines* (ATM's). Basically the concept of alternation is a generalization of nondeterminism in a way that allows existential and universal quantifiers to alternate during the course of a computation. Four kinds of states exist in an ATM; namely existential, universal, accepting and rejecting states. A universal state can lead to acceptance iff all successors lead to acceptance. On the other hand, an existential state leads to acceptance iff there exists a successor that leads to acceptance. Details can be found in [3]. The complexity classes of languages accepted by time (space) bounded ATM's were also defined in [3]. In particular, $A\Sigma_k^L$ was defined to be the set of languages accepted by log n space-bounded ATM's in which the starting state

was an existential state and the machine was constrainted to make at most k-1 alternations during the course of a computation. In what follows, we show that the $A\Sigma_k^L$ hierarchy coincides with the restricted logspace hierarchy $\Sigma_k^{L[1]}$ defined earlier. This characterization proves useful in later sections as both computational models are used in deriving completeness results for various levels of this hierarchy.

Lemma 2.1: $A\Sigma_k^L \subseteq \Sigma_k^{L[1]}$

Proof. (By induction on k)

Induction base: When k=1, we easily see that $A\Sigma_1^L \equiv NL \equiv \Sigma_1^{L[1]}$.

Induction hypothesis: Assume that the assertion is true for k=n.

Induction step: Now consider the case when k=n+1. Let L be an arbitrary language in $A\Sigma_{n+1}^L$. Let M be an n+1-alternating ATM that accepts L. Let $u_1, u_2, ..., u_m$ be the universal states in M. Let $M_{u_1}, M_{u_2}, ..., M_{u_m}$ be the n-alternating ATM's constructed from M in such a way that, each M_{u_i} $(0 \leq i \leq m)$ is essentially the machine M except that every universal (accepting) state is made existential (rejecting), and vice versa. Furthermore, each M_{u_i} has u_i as its initial state. Since now u_i is an existential state for the machine M_{u_i}, the language accepted by M_{u_i} is in $A\Sigma_n^L$. Now define A_i to be $\{(x,u_i,p)|$ starting in state u_i with its input head at the p-th position, M_{u_i} accepts x$\}$, which is clearly in $A\Sigma_n^L$, and hence according to the induction hypothesis, A_i is also in $\Sigma_n^{L[1]}$. Let $A=\cup A_i$, $1 \leq i \leq m$. Then A is in $\Sigma_n^{L[1]}$ as well.

In what follows, we want to construct an oracle machine M' to accept L. Basically, M' is going to simulate the computation of M on input x until the first universal state, say u_i, is reached. At this point, M' writes down the current state and input head position (of length $O(\log|x|)$) on the query tape and then consults the oracle set A. If the answer of the oracle call is "no", M' enters the accepting state; otherwise M' rejects the input. See Figure 2.1. Clearly M' accepts the language L which in turn must therefore be in $\Sigma_{n+1}^{L[1]}$. The lemma then follows. □

Lemma 2.2: $\Sigma_k^{L[1]} \subseteq A\Sigma_k^L$

Proof. (By induction on k)

Induction base: When k=1, we easily see that $\Sigma_1^{L[1]} \equiv NL \equiv A\Sigma_k^L$.

Induction hypothesis: Assume that the assertion is true for k=n.

Induction step: Now consider the case when k=n+1. Let L be an arbitrary language in $\Sigma_{n+1}^{L[1]}$. Let $A \in \Sigma_n^{L[1]}$. Let M be an OTM that operates in O(log n) space such that $L(M^A)=L$. We may, as indicated earlier in this section, modify M in such a way that the last step of any computation is the oracle call. According to the induction

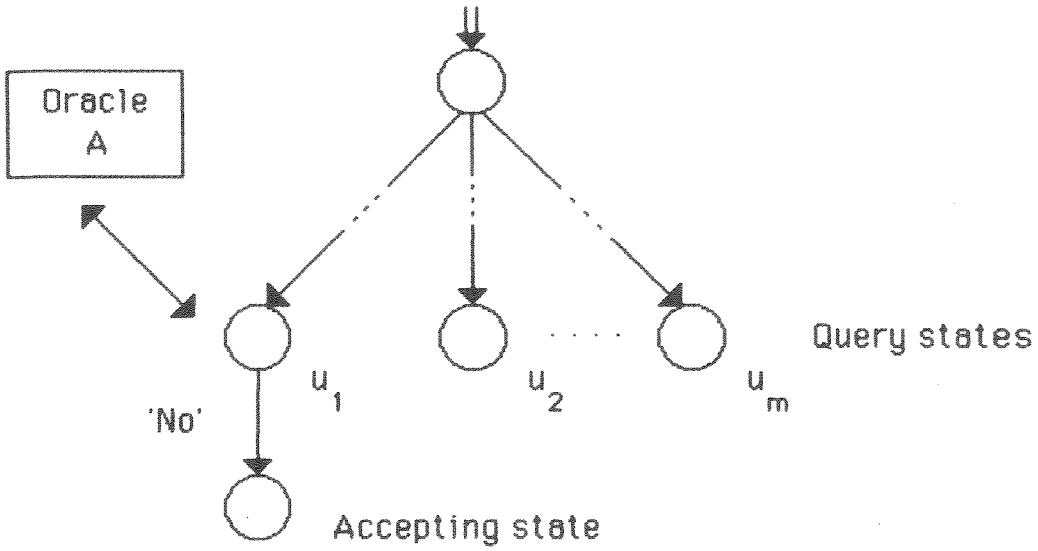


Figure 2.1 The machine M'.

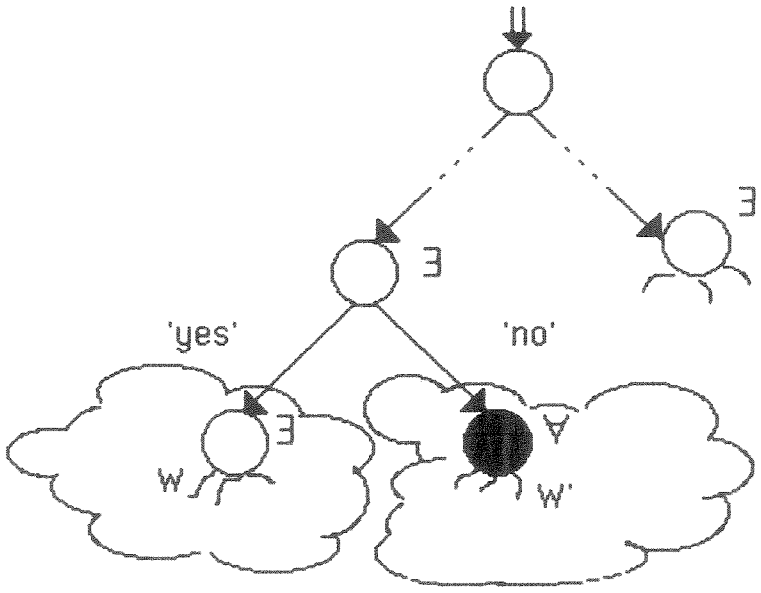


Figure 2.2 The machine M'.

hypothesis, the language (or set) A is in $A\Sigma_n^L$, and therefore can be accepted by an n-alternating ATM W. Let W' be the machine W with universal (accepting) and existential (rejecting) states interchanged. In this way, W', with a universal initial state, will accept the complement of A. Now, we want to construct an n+1-alternating ATM M' to accept L. Simply speaking, M' will simulate the computation of M faithfully except that, the string s that is to be written on the query tape by M, will be written on its working tape instead. Therefore, the working tape of M' will look like \$s#... . Now, when a query state is reached, M' will simulate the oracle call by using the mechanism described in Figure 2.2. More precisely, if both a "yes" or "no" response causes M to accept then M' simply accepts the input. If only a "yes" response causes M to accept then control is passed to W which simply treats the \$s# part as its input and simulates the moves accordingly. If only a "no" response causes M to accept then control is passed to W' instead. Notice that the language accepted by M' is L, which is therefore in $A\Sigma_{n+1}^L$. Hence the lemma follows. □

Consequently, we have the following theorem:

Theorem 2.3: $\Sigma_k^{L[1]} \equiv A\Sigma_k^L$

In [30], it has been shown that $\cup_k A\Sigma_k^L \subseteq DL^{NL}$. Using Theorem 2.3, we then have that the restricted logspace oracle hierarchy is contained in DL^{NL}. Let $D^L=\{L_1 \cap L_2 \mid L_1 \in NL, L_2 \in Co\text{-}NL\}$. This class is the space analog of the class D^P defined in [27], where, in the previous definition, the terms NL and Co-NL are replaced by the terms NP and Co-NP, respectively. Since Σ^* is in Co-NL (NL), we also have that NL (Co-NL) $\subseteq D^L$, since for any language L $\in$ NL (Co-NL), $L \cap \Sigma^* = L$. Furthermore, it should be fairly easy to see that $D^L \subseteq NL^{NL[1]}$. As a result, we now have the following theorem:

Theorem 2.4: $DL \subseteq NL, Co\text{-}NL \subseteq D^L \subseteq NL^{NL[1]} \ldots \subseteq DL^{NL} \subseteq NL^{NL} \ldots \subseteq PTIME \cap DSPACE(\log^2 n)$.

3. Some complete problems

In this section, we introduce some easy graph problems that are complete for $NL^{NL[1]}$. These problems are useful, in the sense, that they will later be used in Sections 4 and 5 as a vehicle to analyze the complexity of problems concerning ω-machines and networks of CFSM's. In what follows, we define the notion of liveness for a graph, and then we show that the problem to decide whether a node in a graph is "not" live is complete for $NL^{NL[1]}$. First, however, the following definitions are required.

An *I-graph* G (or simply a graph if the initial node is clear and understood) consists of a directed graph G=(V,E), where V and E represent the sets of nodes and edges respectively, and there is a unique distinguished node $v_0 \in V$ called the initial node. For

two nodes p and q in V, q is said to be *reachable* from p (denoted by $p \xrightarrow{*} q$) iff there exists a sequence of nodes $s_0,s_1,...,s_k$ such that $s_0=p$, $s_k=q$ and for each i, $0 \leq i \leq k-1$, $s_i \in V$ and the edge (s_i, s_{i+1}) is in E. We use $p \xrightarrow{+} q$ to denote the case when $k \neq 0$. Similarly, an *infinite path* is an infinite sequence of nodes $s_0,s_1,...,s_n,...$ such that $s_0=v_0$ (the initial node) and for every i, $s_i \in V$ and the edge (s_i, s_{i+1}) is in E. An infinite path P is said to be *strongly fair* iff for every node v that occurs infinitely often in P, it is the case that all outgoing edges of v have been traversed infinitely often in P. A node u is said to be *weakly live* iff on every strongly fair path, u occurs infinitely often. A subgraph G'=(V',E') of G is called an *Isolated Reachable Component* (IRC, for short), if G' satisfies the following conditions:

1. $\exists\ v \in V'$, $v_0 \xrightarrow{*} v$ in G.

2. $\forall\ v,w \in V'$, $v \xrightarrow{+} w$ and $w \xrightarrow{*} v$.

3. $\forall\ v \in V'$ and $w \in V$, $v \xrightarrow{*} w$ in G implies $w \in V'$.

The first condition indicates that the IRC G' is reachable from v_0. The second condition guarantees that G' is strongly connected. The third condition indicates that no node in V-V' can be reached from any node in V'. (The reader can therefore think of G' as sort of a black hole.) The following lemma provides a way to test whether a node in a graph is weakly live. The proof is quite easy and hence will be left to the reader.

Lemma 3.1: Given an I-graph G and a node u, u is not weakly live iff there exists an IRC G' that does not contain u.

Theorem 3.1: The language $L_1=\{(G,u)|$G is an I-graph, u is a node in G, and u is not weakly live$\}$ is $NL^{NL[1]}$-complete.

Proof. Let u_0 be the initial node of the I-graph G. First, we show that L_1 is in $NL^{NL[1]}$. To do this, we construct a new graph G' from G by adding an edge from every deadend node (i.e. nodes without outgoing edges) to the node u, and an edge from u to the initial node u_0 of G. It should be reasonable easy to see that, every IRC not containing u in G is also an IRC not containing u in G'. The converse may not be true. This will be the case only if u is not reachable in G, and from u an IRC is reachable in G' that was not reachable in G. In either case, by Lemma 3.1 we have that u is not weakly live in G iff there is an IRC C in G' (which is also reachable in G) such that u is not in C. Note that the transformation from G to G' requires only deterministic logspace.

Since G' has no deadend nodes, u is not weakly live in G iff the following things are true:

1. $\exists\ v \in G$, $u_0 \xrightarrow{*} v$ in G, and

2. $\neg(v \xrightarrow{*} u)$ in G'.

Let $A=\{(G,u,v)|$ u,v are either not nodes in the graph G or $v \not\xrightarrow{*} u$ in the graph $G'\}$, where G' is the graph obtained from G as indicated above. Clearly A is in NL. Then we can construct an OTM M using A as its oracle set such that $L(M^A)=L_1$. Basically, M nondeterministically chooses a node v in G and writes v on the query tape. M then nondeterministically verifies that $u_0 \xrightarrow{*} v$ in G. If M is unable to do so the input is rejected; otherwise, M consults its oracle and accepts the input only if the response is "no". Thus, an input (G,u) will be accepted iff 1 and 2 above are true. Hence, L_1 is in $NL^{NL[1]}$.

In what follows, we show that L_1 is $NL^{NL[1]}$-hard. Let M be an O(log n) space bounded OTM that makes at most one oracle call during the course of any computation. Let A be in NL. Without loss of generality, we assume that $A=\{(G,u,v)|$u and v are nodes in G and $u \xrightarrow{*} v$ in G$\}$ -- the directed graph reachability problem. We also assume that M behaves in the canonical fashion described in Theorem 2.2, i.e., the last step of any computation is the oracle call and M accepts its input only when it receives a negative oracle response. We now show how to construct an I-graph G containing a node u (depending on M, A, and x), in deterministic log space, such that M accepts x iff u is not weakly live in G.

Let T be an NTM that accepts A and operates in O(log n) space. A configuration of M (on input x), as defined in Section 2, is a 4-tuple c=(q,i,z,w), where q is the current state of M, i is the input head position, and z and w are contents of the tape and query tape respectively. (z includes the tape head position as well.) A configuration of T (on some input y) is a 3-tuple (q,i,z), where each term is the same as above. We construct a graph $G_{M,x}$ such that each node represents a configuration of M on x, and each edge denotes the corresponding transition. Let the initial configuration of M on x be the initial node of $G_{M,x}$. Similarly, we define $G_{T,y}$ to be the computation graph of T on y. Note that since M operates in O(log n) space, the size of $G_{M,x}$ is polynomially related to the size x. Now, the graph G can be built as follows. G will contain $G_{M,x}$ (as a subgraph) and a new node q'. Furthermore, for each query configuration c=(q,i,z,w), we add an edge from c to the initial node of $G_{T,w}$. In addition, from every accepting node in $G_{T,w}$ we add an edge to the node q', and for every non-accepting node we add an edge to the initial node of $G_{T,w}$. Lastly, we add an edge from every node of $G_{M,x}$ to c_0—the initial node of $G_{M,x}$, from c_0 to q' and from q' to c_0. See Figure 3.1. Note that since the string w is always of length logarithmic in x (the input of M), the size of each $G_{T,w}$ will be polynomially related to the size of x. Therefore, the size of the whole graph G is polynomial in the size of x. This implies that the above construction can be done in deterministic log space.

Now, we claim that M accepts x iff the node q' is not weakly live. To see this, we first note that M accepts x iff for some reachable query configuration (q,i,z,w) in G, an

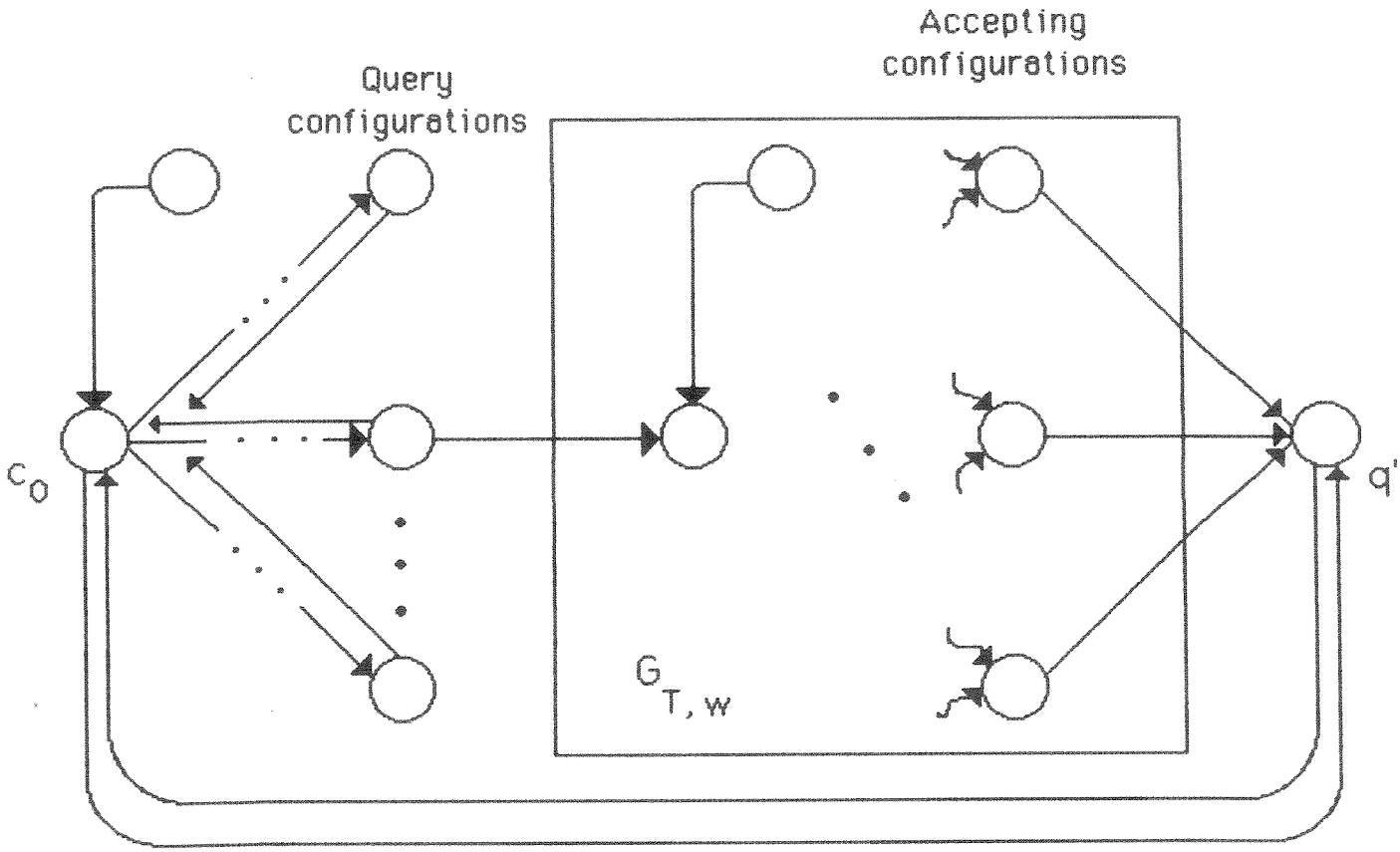


Figure 3.1 The I-graph G.

accepting state is not reachable in $G_{T,w}$. This is true iff the node q' is not reachable from the initial node of $G_{T,w}$, which in turn is true iff there exists some IRC in $G_{T,w}$ not containing q'. Thus, by Lemma 3.1 we have that L_1 is $NL^{NL[1]}$-hard. □

Theorem 3.2: The language L_2={G| G has an IRC} is complete for $NL^{NL[1]}$.

Proof. The fact that L_2 is in $NL^{NL[1]}$ can be derived using a similar argument as was used in the proof of Theorem 3.1. Here we only show that L_2 is $NL^{NL[1]}$-hard. To do this, consider an instance of L_1 (G,u) where G=(V,E) is an I-graph and u is a node in V. We construct a new graph G'=(V',E') by adding a new node u' such that V'= V ∪ {u'} and E'= E ∪ {(u,u')}. In other words, G' is constructed by connecting u to a deadend node u'. In this way, u is not weakly live in G iff G' has an IRC. The theorem follows. □

4. ω-machines with fairness constraints

Our main concern in this section is to define ω-machines where acceptance is defined as in [5, 21] but with the additional criteria that all accepting computations satisfy some fairness constaint. Then we investigate the complexity of the *Non-Emptiness Problem* (NEP) for such machines. It turns out that their complexities fall into one of the following categories:

1. NLOGSPACE-complete,
2. $NL^{NL[1]}$-complete (the second level of the restricted logspace oracle hierarchy),
3. $NL^{NL[1]^{NL[1]}}$-complete (the third level of the restricted logspace oracle hierarchy),
4. PTIME-complete.

Using these results, we are able to derive some complexity bounds for liveness questions concerning logic and systems of CFSM's, which we present in the next section.

4.1 ω-finite state machines

A *nondeterministic finite state machine* (FSM) M is a 4-tuple (Q,Σ,q_0,δ) where

Q is the finite set of states,

Σ is the finite set of input symbols,

q_0 is the initial state, and

δ: $Q \times \Sigma \rightarrow 2^Q$ is the transition function.

Let Σ^ω denote the set of infinite strings over Σ. Let $\sigma=a_1,a_2,...$ be an infinite sequence of input symbols, i.e. an element in Σ^ω. A *run* r on σ in the state q is an infinite sequence of states $q_1,q_2,...$ such that $q_1=q$ and for every $1\leq i$, $q_{i+1}\in\delta(q_i,a_i)$. For a run $r=q_1,q_2,...$, we define $P_r(i)=q_i$, i.e., the i-th state in the sequence. Let q be a state, we use $\#_r(q)$ to denote the number of occurrences of q in the run r. Let $\#_r(q)=\infty$ iff q occurs infinitely often in r. We also define $Inf(r)=\{q \mid \#_r(q)=\infty\}$. In other words, Inf(r) is the set of states that occur infinitely often in r. An ω-FSM is a 5-tuple $M=(Q,\Sigma,q_0,\delta,F)$ where $M_1=(Q,\Sigma,q_0,\delta)$ is a FSM and $F\subseteq 2^Q$ is called the set of *designated state sets*. We now introduce the notion of *"i-acceptance"*, where i=1, 1', 2, 2', or 3, as was defined in [21]. See also [5]. Given an input σ, let r be *any* run on σ in the initial state q_0. We say:

(1) σ is *1-accepted* iff $\exists H\in F, \exists i, P_r(i)\in H$.

(2) σ is *1'-accepted* iff $\exists H\in F, \forall i, P_r(i)\in H$.

(3) σ is *2-accepted* iff $\exists H\in F, Inf(r)\cap H\neq\phi$.

(4) σ is *2'-accepted* iff $\exists H\in F, Inf(r)\subseteq H$.

(5) σ is *3-accepted* iff $Inf(r)\in F$.

We define $L_i(M)=\{\sigma|$ there is a run of M on σ such that σ is i-accepted$\}$ to be the language accepted by M in i-acceptance mode. Informally speaking, σ is 1-accepted if r visits some state which belongs to some designated state set in F. σ is 1'-accepted if there exists a set H in F such that, every state in r is also in H. σ is said to be 2-accepted if the set of states that appear infinitely often in r contains some state from some designated state set in F. σ is 2'-accepted if the set of states that occur infinitely often is a subset of some designated set in F. Finally, σ is 3-accepted if the set of states entered infinitely often in r coincides exactly with some designated state set in F.

The NEP for ω-FSM's is to, given an ω-FSM M, decide whether M accepts any input. The following theorem is easily shown:

Theorem 4.1: For all i-acceptance modes, the NEP is NLOGSPACE-complete.

In what follows, we introduce the notion of *fair acceptance* for ω-FSM's. Let $M=(Q,\Sigma,q_0,\delta,F)$ be an ω-FSM. Let r $(=q_0,q_1,...)$ be a run over the input σ $(=a_0,a_1,...)$. With respect to r and σ, we can define an infinite sequence of transitions E: $e_0,e_1,...$ such that, e_i is the transition $(q_i,a_i)\rightarrow q_{i+1}$. The run r is said to be *strongly fair* iff for every state q such that $\#_r(q)=\infty$, every outgoing transition of q must occur infinitely often in

E. The run r is said to be *fair* if for each e: (q,a)→q', e occurs infinitely often in E, then every transition (q,a)→q", where q"∈δ(q,a), must also occur infinitely often. Note that with comparison to a strongly fair run, a fair run only requires that, those transitions that have been "enabled" infinitely often must be executed infinitely often as well. We then define the language of fair (strongly fair) i-acceptance by an ω-FSM M to be $L_i^f(M)=\{\sigma|$ there exists a fair run r on σ such that, σ is i-accepted$\}$ ($L_i^{sf}(M)=\{\sigma|$ there exists a strongly fair run r on σ such that, σ is i-accepted$\}$). The fair (strongly fair) NEP will be to, given an ω-FSM M, determine whether $L_i^f(M)\neq\phi$ ($L_i^{sf}(M)\neq\phi$).

Theorem 4.2: Given an ω-FSM M, the fair (or strongly fair) NEP, i.e. "Is $L_3^f(M)\neq\phi$ (or $L_3^{sf}(M)\neq\phi$)?", is NLOGSPACE-complete.

Proof sketch. We only show that the problem can be solved in NLOGSPACE. The fact that the problem is NLOGSPACE-hard can be derived using a standard reduction from the directed graph reachability problem [31]. We first consider the strong fairness case. Consider an arbitrary ω-FSM $M=(M_1,\{H_1,H_2,...,H_k\})$, where M_1 is a FSM and $\{H_1,...,H_k\}$ is the set of designated state sets. M accepts some input iff there is a strongly fair run r such that $inf(r)=H_i$, for some i. We now describe an NTM W that accepts M iff $L_3^{sf}(M)\neq\phi$ and runs in nondeterministic log space. First W guesses a value for i, $1\leq i\leq k$. Note that the set H_i is part of the input tape; let $\$h_1,h_2,...,h_m\#$ represent this part. To do the checking, two pointers P_1 and P_2 are used. P_1 is associated with H_i and indicates those states in H_i that W currently recalls having visited. Initially P_1 points to h_1. The checking procedure basically consists of two stages. In the first stage, P_2 will be used as a pointer to traverse the state transition graph of M_1 (initially P_2 points to the initial state of M_1), while P_1 remains unchanged. During the traversing, W nondeterministically "guesses" at some point that the current state is in an IRC that contains exactly those states in H_i; and then proceeds to the second stage. In the second stage, assume that P_1 is currently pointing at h_j, $1\leq j\leq m$. Similar to the first stage, P_2 is used to traverse the graph. However, some extra checking is needed at each step. When visiting a state, say p, W first tests whether every adjacent state q of p (i.e., q∈δ(p,a) for some input symbol a) is in H_i. If not, the checking procedure terminates unsuccessfully; otherwise, one of the adjacent q's, say q', will be nondeterministically chosen to be visited next. Furthermore, if $q'=h_{j+1}$, P_1 will be advanced to the next position. This procedure in stage 2 repeats until P_1 hits the #. Then, the above traversing continues until the state h_1 is again visited. At this time W accepts the input. It is now reasonably easy to see that W accepts such an input iff there is a strongly fair run that visits exactly the states in H_i in this manner. Since the space needed for the pointers is logarithmic, the strongly fair NEP can, therefore, be solved in NLOGSPACE.

A similar argument can be applied for the fair case. The main modification will be

that, for each state, we only have to test those states with the same label (i.e., the input symbol) to see if they are in H_i. Further details are left to the reader. □

This theorem tells us that, type 3 acceptance is, in some sense, easier to analyze than other types of acceptance. In the remainder of this section, we only consider type 1, 1', 2, and 2'-acceptance modes.

Lemma 4.1: Given an ω-FSM M, the fair NEP, i.e. "Is $L_i^f(M)\neq\phi$, where i=1, 1', 2, or 2'", is PTIME-hard.

Proof. To show this, we reduce the path system problem, which is well-known to be PTIME-complete [17], to the fair NEP for ω-FSM's. Recall that a Path System is a 4-tuple $\mathcal{P}=(X,R,S,T)$ where X is a finite set of nodes, S ($\subseteq X$) is a set of starting nodes, T ($\subseteq X$) is the set of terminal nodes, and R ($\subseteq X\times X\times X$) is the set of rules. A node x in X is said to be *admissible* iff either $x\in T$ or $\exists y,z\in X$ such that $(x,y,z)\in R$ and both y and z are admissible. The Path System $\mathcal{P}=(X,R,S,T)$ is said to have a solution iff there is an admissible node in S.

Now, given the Path System $\mathcal{P}$ we want to, for each i=1, 1', 2, or 2', construct an ω-FSM M_i to, in some sense, "simulate" $\mathcal{P}$ such that $L_i^f(M_i)\neq\phi$ iff $\mathcal{P}$ has a solution. To do this, we define $M_i=(Q,\Sigma,q_0,\delta,F)$ to be:

$Q=X\cup\{q_0,q'\}$,

$\Sigma=\{a,b\}$,

δ: (Let $S=\{s_1,s_2,...,s_k\}$, $T=\{t_1,t_2,...,t_h\}$ and let a/b denote either a or b.)

(1) $s_1\in\delta(q_0,a/b)$,

(2) $\forall\ 1\leq i\leq k\text{-}1$, $s_{i+1}\in\delta(s_i,a/b)$ and
$s_1\in\delta(s_k,a/b)$,

(3) $\forall\ 1\leq i\leq h$, $q'\in\delta(t_i,a/b)$,

(4) $\forall\ x\in(X\text{-}S\text{-}T)$, $q_0\in\delta(x,a/b)$,

(5) $\forall\ x\in(X\text{-}S\text{-}T)$, if $(x,y,z)\in R$, $y\in\delta(x,a)$ and $z\in\delta(x,b)$.

$F=\{X\text{-}T\}$.

See Figure 4.1. We claim that $\mathcal{P}$ has no solution iff M_i has a fair run, i.e. $L_i^f(M_i)\neq\phi$. To show this, first assume that $\mathcal{P}$ has no solution. Let A and B the sets of nonadmissible and admissible nodes, respectively. Clearly, $S\subseteq A$. According to the construction of M, for every node x in A there exists at most one edge that connects x to some node in

B. (Otherwise, x would be admissible.) Furthermore, for each state in A ($\subseteq$X-T), there are two edges back to q_0. Consequently, there exists a fair run, say r, that only visits states in A ($\subseteq$X-T). Since we chose F=$\{$X-T$\}$, the input associated with r will be accepted by M_i, for i=1, 1', 2, and 2'. On the other hand, consider the case when $\mathcal{P}$ has a solution, i.e., some state, say s', in S is admissible. Note that by construction a fair run on M_i must have all states in S appear infinitely often. Since s' is admissible, there must exist a rule (s',x,y)$\in$R where both x and y are admissible. Therefore, we can conclude that either x or y must occur infinitely often in the run. Inductively then there must exist a node t in T that also occurs infinitely often in the run. This implies that the deadend state q' will occur infinitely often, which is certainly impossible. As a result, no fair run exists for the case when $\mathcal{P}$ has a solution. M_i (i=1, 1', 2, or 2'), therefore, cannot fairly accept any input. Thus, we have $L_i^f(M_i)\neq\phi$ iff $\mathcal{P}$ has no solution. The lemma then follows. □

In Section 5, we will further see that the model of fair ω-machies can be modelled by using the so-called Fair Computation Tree Logic. Furthermore, the fair NEP will be reduced to the canonical form model checking problem for that kind of logic, which is known to be solvable in PTIME [9]. Consequently, we have the following result:

Corollary 4.1: The fair NEP for ω-FSM's (for acceptance modes 1, 1', 2 and 2') is PTIME-complete.

Notice that the proof of the above lemma required that the constructed ω-FSM have two distinct input symbols. An interesting question arises if we restrict the problem to ω-FSM's over a single letter input alphabet. In this case, it turns out that the aforementional problems are $NL^{NL[1]}$-complete.

Theorem 4.3: For ω-FSM's over a 1-letter alphabet, the fair (for acceptance modes 1, 1' and 2') NEP is $NL^{NL[1]}$-complete.

Proof. Notice that a fair run for an ω-FSM over a 1-letter alphabet is a strongly fair run, and vice versa. Based on this observation, we first show that the fair NEP is $NL^{NL[1]}$-hard. To show this, we reduce the problem L_2 mentioned in Section 3 (the problem, given an I-graph G=(V,E), to determine if G contains an IRC), to the fair NEP for ω-FSM's. The reduction simply involves constructing an ω-FSM M where the states (transitions) of M correspond to the nodes (edges) of G, and F (the designated state set) contains the set of all nodes in G. In this way, for all fair accepting modes (i.e. 1, 1', 2, and 2'), M accepts some input iff G has an IRC.

Next, we show that the fair NEP is doable in $NL^{NL[1]}$. Consider an ω-FSM $M=(Q,\{a\},q_0,\delta,\{H_1,H_2,...,H_k\})$. We first construct M' from M by adding a new state q' and transitions from every deadend state (a state with no outgoing transitions) to q'. (This is similar to the construction used in the proof of Theorem 3.1.) In the subsequent discussion, we consider the FSM's M and M' as I-graphs with q_0 as the

initial node. The edges, of course, are defined by the transitions. In what follows, we construct an OTM M^i and an oracle set A^i, for each i, such that M^i will accept exactly those machines M where $L_i^f(M)\neq\phi$. Each M^i is similar to the OTM constructed in Theorem 3.1. However, for different accepting modes, additional checking steps are required. Let

$A^1=\{(M,i,u)|$ one of the following is true:

- M is not an ω-FSM,
- i does not represent the index of one of M's designated state sets,
- u is not a state in M, or
- $u \stackrel{*}{\rightarrow} q'$ in M'}.

Let

$A^{1'}=A^{2'}=\{(M,i,u)|$ one of the following is true:

- M is not an ω-FSM,
- i does not represent the index of one of M's designated state sets,
- u is not a state in M, or
- $\exists\ w\notin H_i\ u \stackrel{*}{\rightarrow} w$ in M'}.

These sets can easily be seen to be in NL. Intuitively speaking, (M,i,u) is not in A^1 if u can not reach the deadend node q' in M', and hence u can reach some IRC of M. Similarly, (M,i,u) is not in $A^{1'}$ ($A^{2'}$) if u can reach some IRC of M and the set of states in that IRC is a subset of H_i. Now, M^n (n=1, 1', 2') on input M operates as follows:

1. M^n guesses a value i, $1\leq i\leq k$.
2. M^n nondeterministically traverses the state graph of M until some state say u (chosen nondeterministically) is visited. If n=1 (1') some (each) state visited must be in H_i; if n=2' u must be in H_i.
3. M^n then accepts iff $(M,i,u)\notin A^n$.

The reader can now verify that M^n accepts M iff $L_n^f(M)\neq\phi$. Notice that each OTM mentioned above requires only log space for its working tape, writes only strings of logarithmic length on the query tape, and consults its oracle only once. The theorem then follows. □

Corollary 4.2: The strongly fair NEP for ω-FSM's (for acceptance modes 1, 1' and 2') is $NL^{NL[1]}$-complete.

Proof. In a strongly fair run, it must be the case that for every state occurring infinitely often that all of its outgoing transitions are traversed infinitely often. This is essentially the same problem then as the fair NEP over a 1-letter alphabet, which has just been shown to be $NL^{NL[1]}$-complete. □

We do not know whether Theorem 4.3 (and Corollary 4.2) hold for 2-acceptance. The hardness part holds as indicated in the proof above; however, the best upper bound we can provide at this time is $NL^{NL[1]^{NL[1]}}$. (The upper bound follows from Lemma 4.3 in the following subsection.) Lastly, similar ideas as those used in Theorem 4.2 and [13] can be combined to show that the ω-1CM NEP for 3-acceptance is NLOGSPACE-complete.

4.2 ω-one counter machines

An ω-1CM is a 7-tuple $M=(Q,\Sigma,\Upsilon,q_0,\delta,Z_0,F)$, where Q is the set of states, Σ is the input alphabet, $\Upsilon=\{Z_0,B\}$ is the stack alphabet, q_0 is the initial state, δ: $Q\times\Sigma\times\Upsilon\rightarrow 2^{Q\times\{-1,0,+1\}}$ is the transition function, Z_0 is the bottom-of-stack marker (Z_0 can neither be written nor erased), and F ($\subseteq 2^Q$) is the set of designated state sets. Roughly speaking, ω-1CM's operate in a similar way as conventional 1CM's except that, we are interested in those inputs over Σ^ω, instead of Σ^*. The actions of the ω-1CM depend on the current input symbol, the internal state of the machine, and the status of the counter: positive or zero (i.e. whether there are zero or a positive number of B's currently on the stack). The definition of strongly fair i-acceptance for ω-1CM's is exactly the same as that for ω-FSM's. However, for the definition of fair acceptance, the notion of when a transition is "enabled" is essential. We define a configuration of an ω-1CM to be a 3-tuple (q,i,h) where q is the current state of the machine, i is the current input head position, and h is the value of the counter (i.e. the number of B's currently on the stack). Let c_1 and c_2 be two configurations. We use $c_1\rightarrow c_2$ to denote that c_1 can lead to c_2 in one computational step. (The input is not important here, and hence is not explicitly shown.) Let $c_1\stackrel{*}{\rightarrow}c_2$ represent the reflexive transitive closure of the "$\rightarrow$" relation. Let $c_0=(q_0,0,0)$ denote the initial configuration. A transition is said to be enabled in a configuration iff the move defined by the transition can be taken. An infinite computation path (i.e. an infinite sequence of configurations beginning with c_0 such that each subsequent configuration follows from its predecessor) is said to be a fair run iff every transition that is enabled infinitely often is executed infinitely often. Based on this notion, fair i-acceptance can then be defined similarly to that for ω-FSM's.

Since ω-FSM's are just special cases of ω-1CM's, the fair NEP for ω-1CM's is at

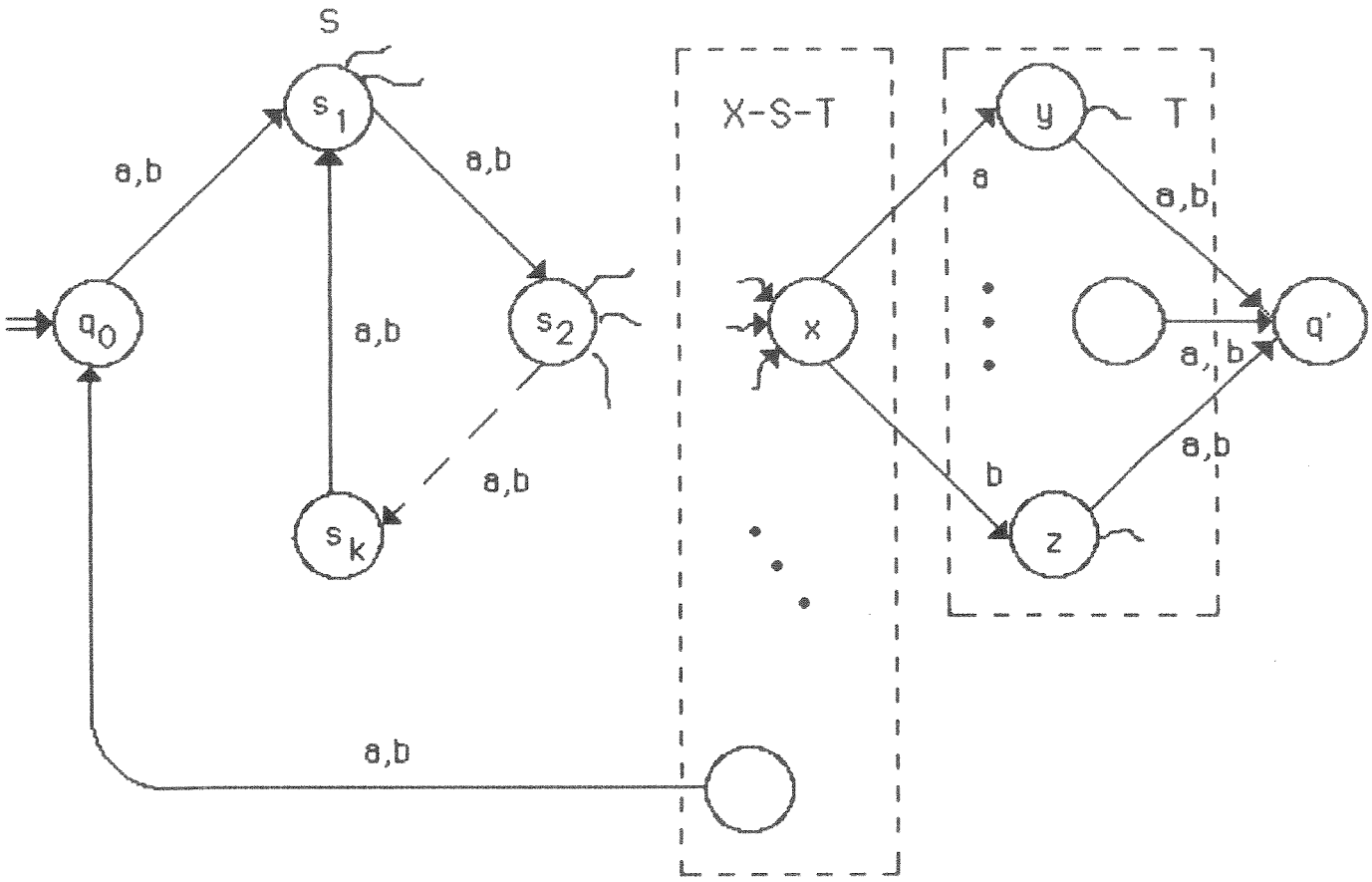
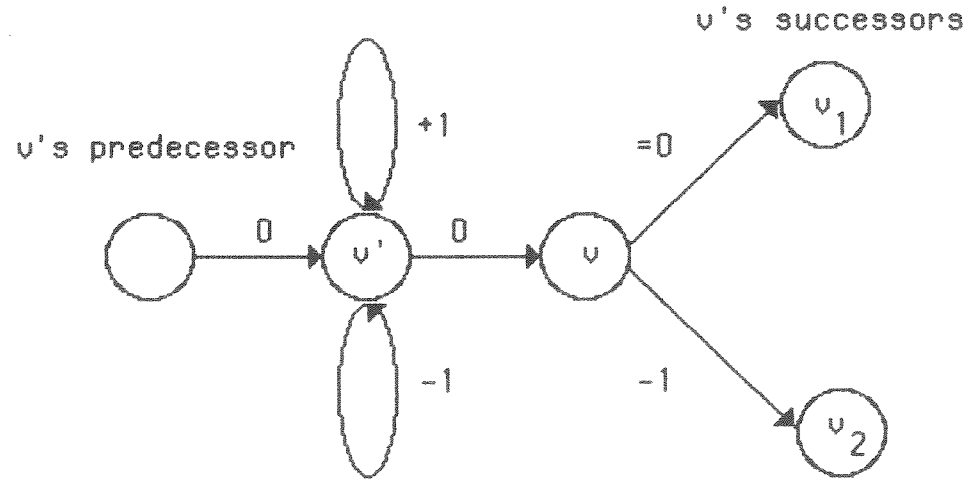


Figure 4.1 The w-FSM M_1.



d(=0,1,or -1): add d to the counter
=0: test for zero

Figure 4.2 A mechanism to enable one and only one of v's outgoing edges.

least PTIME-hard. In what follows, we focus on those ω-1CM's over 1-letter input alphabets.

Lemma 4.2: Given an ω-1CM M over a 1-letter alphabet, the fair (for acceptance modes 1, 1', 2 and 2') NEP is PTIME-hard.

Proof sketch. For a state v, we introduce the mechanism (as shown in Figure 4.2) that will allow the enabling of one (and only one) of a state's outgoing edges. In this way, we can think of the transition from v to v_1 to be the one which reads an input, say a; while the other edge (the transition from v to v_2) is the one that reads an input b. Consequently, we can then emulate the proof for ω-FSM's over 2-letter alphabets, which was shown to be PTIME-hard in Lemma 4.1. The lemma then follows. □

Lemma 4.3: For ω-1CM's, the strongly fair (for acceptance modes 1, 1' 2 and 2') NEP is solvable in $NL^{NL[1]^{NL[1]}}$.

Proof. Let M be a 1CM. In what follows we show how to construct OTM's M^n (that make at most one oracle call during the course of a computation) and oracle sets A^n (in $NL^{NL[1]}$), n=1, 1', 2, 2', such that M^n using the oracle A^n accepts M iff $L_i^{sf}(M)\neq\phi$. First, we show how to determine if there is a strongly fair accepting run in M. In order to show this, we first claim that there exists a strongly fair path in G iff the following conditions are satisfied:

1. $\exists\ c_1=(v,i,h)$, where h is polynomial in |M| (the size of M), and $c_0 \xrightarrow{*} c_1$, and

2. either

   $\forall$ state w in M, either

   a. $\neg(v \xrightarrow{*} w)$, or

   b. there is a computation $(v,i,h) \xrightarrow{*} (w,i',h') \xrightarrow{*} (v,i'',h)$, for some i', i'' (i''>i'>i) and h'.

   or

   $\forall$ state w in M, either

   a. $\neg(v \xrightarrow{*} w)$, or

   b. $\exists$ h''>h such that there is a computation $(v,i,h) \xrightarrow{*} (w,i',h') \xrightarrow{*} (v,i'',h'')$, for some i', i'', (i''>i'>i) h', h'' (h''>h), where no transition requiring a zero counter is used in the path.

Furthermore, using techniques from [13], it can be shown for any 1CM that, if there exists a path satisfying either condition (b) above that there must also exist a short such path whose length is less than a fixed polynomial P in |M|. Hence, the b part can be verified in log space. Now, we want to show how the above procedure can be done in $NL^{NL[1]^{NL[1]}}$. For n=1, 1', 2, 2', define:

B^n={(M,v,i,h,m,t,w)| ∃ i',i" (i">i'>i), h', h" (h">h)
such that either
t=1 and
$(v,i,h)\xrightarrow{*}(w,i',h')\xrightarrow{*}(v,i",h)$
(If n=1' or 2', it is also required that every state visited on this path be in H_m.) or

t=2 and
$(v,i,h)\xrightarrow{*}(w,i',h')\xrightarrow{*}(v,i",h")$
where no transition requiring a zero counter is used in the path.
(If n=1', or 2', it is also required that every state visited on this path be in H_m.)}

From the above discussion we have that B^n is in NL. Now, let

A^n={(M,v,i,h,m,t)| ∃ a state w in M such that

$v\xrightarrow{*}w$ and $(M,v,i,h,m,t,w)\notin B^n$}

Using B^n as an oracle, A^n can easily be recognized in $NL^{NL[1]}$. (This is done by writing w on the query tape, and consulting the oracle B^n.) Now M^n on input M operates as follows:

1. M^n guesses a value m, $1\leq m\leq k$.

2. M^n guesses a computation (step by step) of length less than P(|M|) ending in say configuration (v,i,h). (If n=1 (1'), then one (each) state visited in the above computation must be in H_m.) M^n also guesses a value of 1 or 2 for t.

3. M^n accepts M if $(M,v,i,h,m,t)\notin A^n$ (If n=2 or 2' also require v to be in H_m.)

The reader can now verify that M^n accepts the input M iff $L_n^{sf}(M)\neq\phi$. Notice that for different accepting modes, different conditions are imposed on the checking procedures, as was the case in the proof in Theorem 4.3. This proves the lemma. □

Lemma 4.4: For ω-1CM's over a 1-letter alphabet, the strongly fair (for acceptance modes 1, 1' 2 and 2') NEP is $NL^{NL[1]^{NL[1]}}$-hard.

Proof. As we know from Theorem 3.3, the restricted logspace oracle hierarchy is the same as the logspace alternation hierarchy. In what follows, we show that the problem is hard for $A\Sigma_3^L$, the class of logspace ATM's using at most two alternations. Let M be such a machine, and let x be the input string. Since M uses logarithmic space for its working tape, the number of distinct configurations is polynomial in |x| (the length of x). (A configuration contains the current state, the contents of the working tape, the input head position, and the number of alternations made in the computation thus far

($\leq$2).) Let G be the graph representing the computation of M on x, where each node represents a configuration and (c_1,c_2) is an edge iff the configuration c_2 is a successor of the configuration c_1. Furthermore, without loss of generality, we may assume that G is acyclic; otherwise, we could modify M in such a way that a counter is attached to the machine that will be incremented by one each time a move is made. The new configurations would then include the value of the counter and clearly, no two configurations can have the same value if one is reachable from the other.

Our goal then is to construct an ω-1CM W to simulate the computation of M on x in such a way that, M accepts x iff W accepts some input. By definition, M accepts x iff there is a path that for every existential (universal) state, some (all) of its successor states will lead to accepting states. Let G=(V,E) be the I-graph representing the computation of M on x. For every universal node (configuration) u in V, we define a new I-graph $G_u=(V_u,E_u)$, where $V_u=\{v_u|\ v\in V\}\cup\{s_u\}$, u_u is the initial node and $E_u=\{(v_u,w_u)|\ (v,w)\in E\}$. Let V^i (V_u^i), $0\leq i\leq 2$, denote those nodes (configurations) in V (V_u) where the number of alternations in the computation thus far is exactly i. Let $W=(Q,\{a\},\{Z_0,B\},q_0,\delta,Z_0,F)$, where $q_0=c_0$ (the initial configuration of M), $Q=V^0\cup V^1\cup[\cup_{u\in V^1} V_u^1\cup V_u^2]$ and δ is defined as follows:

1. For every $v,w \in V^0\cup V^1$ we have $(w,0)\in\delta(v,a,Z_0)$ whenever $(v,w)\in E$.

2. For every $u\in V^1$ and every $v,w \in V_u^1\cup V_u^2$ we have $(w,0)\in\delta(v,a,Z_0/B)$ whenever $(v,w)\in E_u$.

3. For every $u\in V^1$ we have $(u_u,0)\in\delta(s_u,a,Z_0)$.

4. For every $v\in V_u^2$ we have $(u_u,0)\in\delta(v,a,Z_0)$.

5. For every $v\in V_u^1$, $w\in V_u^2$ where $(v,w)\in E_u$ we have $(w,0)\in\delta(v,a,Z_0/B)$ and $(w,+1)\in\delta(v,a,Z_0/B)$.

6. For every rejecting node v in V_u^2 we have $(s_u,0)\in\delta(v,a,Z_0/B)$.

7. For every accepting node v in V_u^2 we have $(s_u,-1)\in\delta(v,a,B)$.

The state graph of W is pictorially described in Figure 4.3, where the label (a,Z,i) on edge (v,w) means that $(w,i)\in\delta(v,a,Z)$. We now claim that M accepts x iff $L_2^{sf}(W)\neq\phi$. The reason is that, the subportion of the state graph corresponding to G_u contains an IRC rooted at u; thus every reachable universal node in G_u has to be visited infinitely often. Let v, w be two nodes such that w follows v and v, w are universal, existential nodes respectively. The fact that v occurs infinitely often implies that the transitions from v to w labelled $(a,Z_0/B,0)$ and $(a,Z_0/B,+1)$ must also occur infinitely often. However,

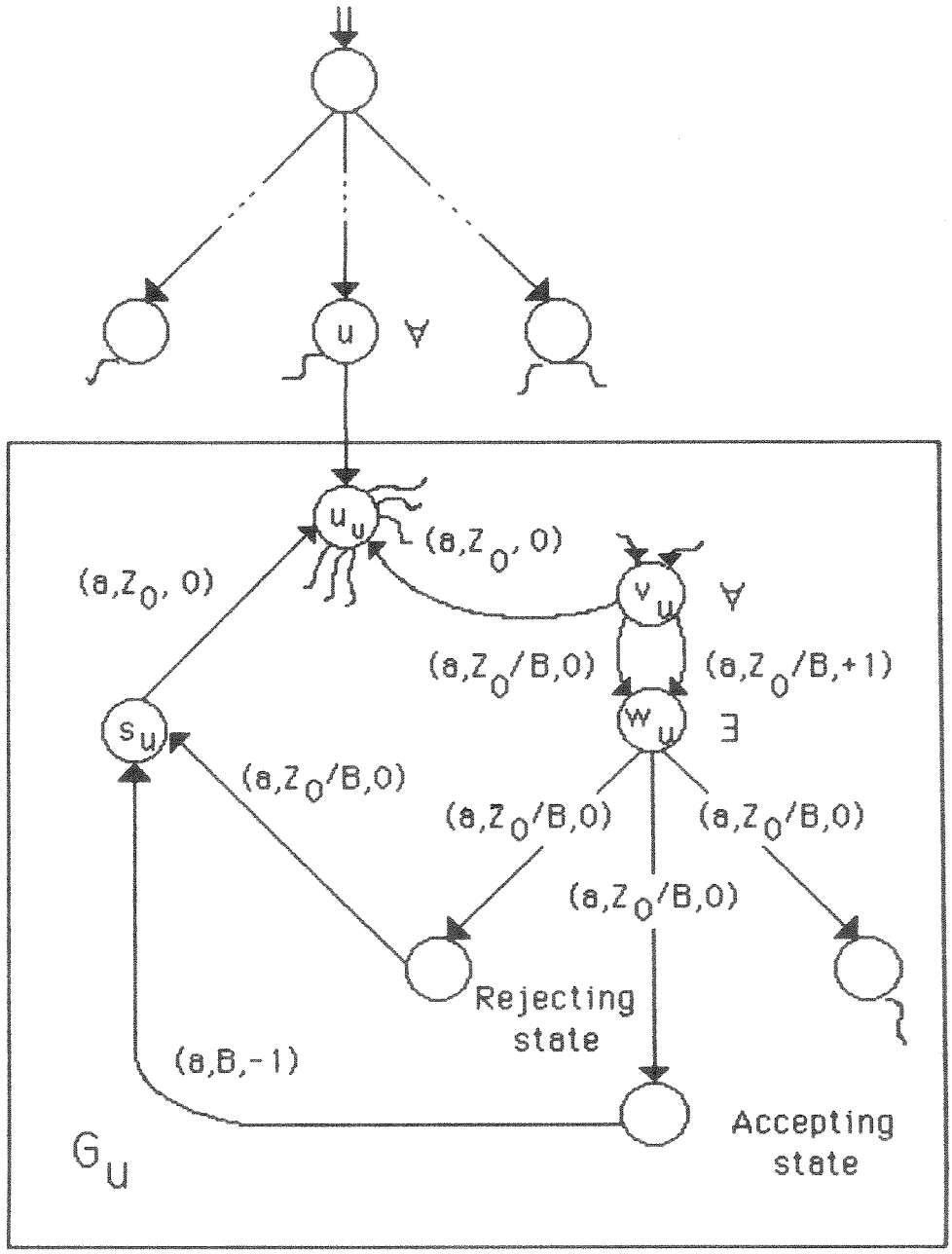


Figure 4.3 The w - 1CM W.

once the counter becomes nonzero, the only way the edge from s to u can be traversed later is by first traversing an edge from an accepting configuration to s. Thus, M accepts x iff W has a strongly fair run. Since the number of possible configurations of M on input x is polynomial in |x|, the above construction can be done in deterministic logspace. This completes the proof. □

We then have the following theorem:

Theorem 4.4: The strongly fair (for acceptance modes 1, 1', 2 and 2') NEP for ω-1CM's over 1-letter alphabet is $NL^{NL[1]^{NL[1]}}$-complete.

4.3 ω-blind counter machines

An ω-BCM with k counters (ω-BCM_k) is a FSM with k counters attached to it. In each step, the machine can increment or decrement each counter by one if the result is nonnegative. However, the machine can not test any counter for zero. (So it is called *blind*.) Essentially, a BCM with k counters [14] is a k-dimensional vector addition system with states [15] which has been augmented by an input tape. A configuration is a 3-tuple (p,i,x), where p is the current state, i is the input head position; and $x \in N^k$ represents the current contents of counters. Now similar ideas as those used in Theorem 4.2 and [29] can be used to show that the ω-BCM_k NEP for 3-acceptance is NLOGSPACE-complete when k is a fixed constant. Although we are not explicit in what follows, in the remainder of this subsection we only concern ourselves with 1, 1', 2 and 2' acceptance modes.

Lemma 4.5: The fair NEP for ω-BCM_k's, for a fixed constant $k \geq 2$, is PTIME-hard.

Proof. Similar to the proof of Lemma 4.2. However, substitute Figure 4.4 in place of Figure 4.2. □

Lemma 4.6: For an ω-BCM_k with n states, if there exists a computation $c_0 \xrightarrow{*} c_1 \xrightarrow{*} c_2$, where c_0 is the initial configuration, $c_1=(p,i,x_1)$, $c_2=(p,i',x_2)$ and $x_2 \geq x_1$, then there is a computation of the above form whose length is less than $n^{2^{c*k*\log k}}$.

Proof. Note that an n-state k-counter BCM can be simulated by a k-dimensional Vector Addition System with n states. Therefore, similar arguments as those used in [29] can be applied here. See also [2, 28]. □

Lemma 4.7: The strongly fair NEP for ω-BCM_k's over a 1-letter alphabet, where k is a fixed constant, can be solved in $NL^{NL[1]^{NL[1]}}$.

Proof. Similar to the proof of Lemma 4.2. (Lemma 4.5 will be used for part (b), however.) □

Lemma 4.8: The strongly fair NEP for ω-BCM_k's over 1-letter input alphabets is $NL^{NL[1]^{NL[1]}}$-hard.

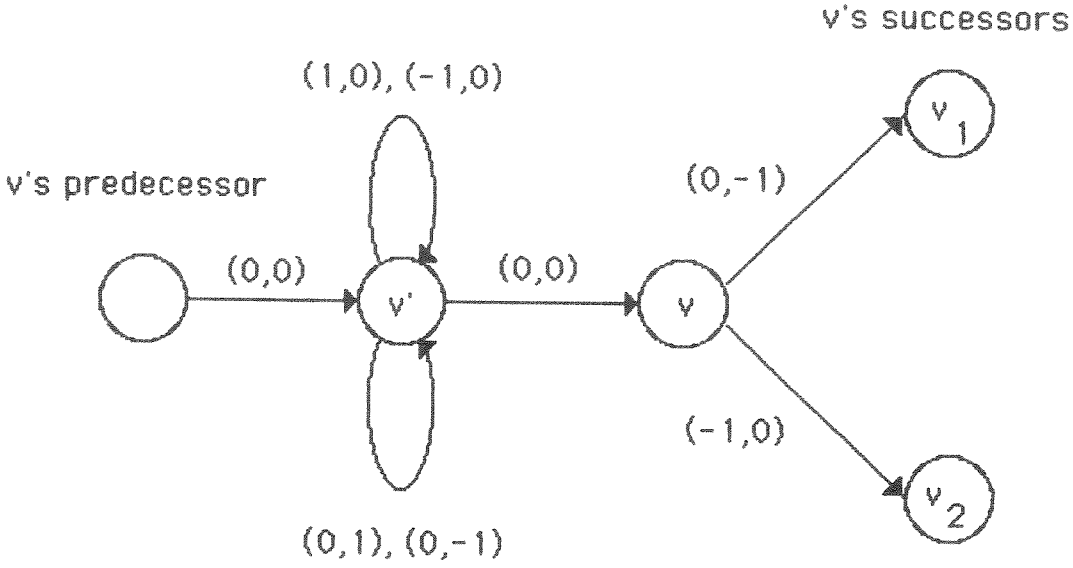


Figure 4.4 A mechanism to enable one and only of v's outgoing edges.

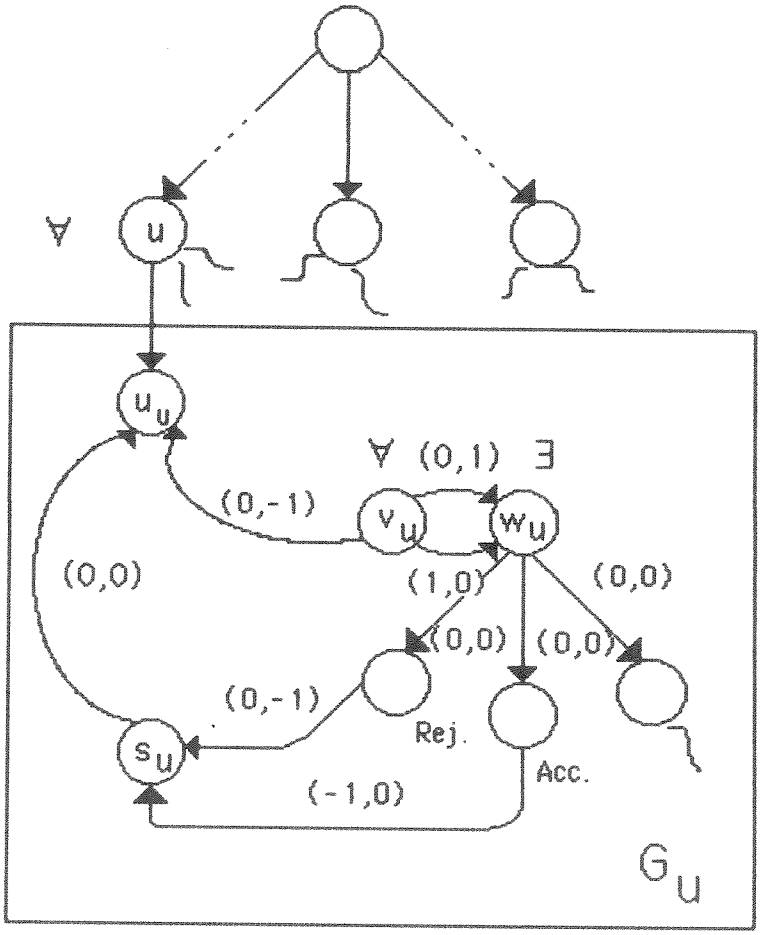


Figure 4.5 An $w\text{-}BCM_2$.

Proof. The proof is much the same as the one in Lemma 4.4 except that the graph in Figure 4.5 will be used in place of the one in Figure 4.3. Note that we only require k (the number of counters) to be two. □

Thus, we obtain:

Theorem 4.5: The strongly fair NEP for ω-BCM_k's over 1-letter input alphabets, where k is a fixed constant, is $NL^{NL[1]^{NL[1]}}$-complete.

5. Applications to related problems

In [4], liveness problems concerning networks of CFSM's were considered. A CFSM M is a directed labelled graph with two types of edges, namely *sending* and *receiving* edges. A sending (receiving) edge is labelled *send(g)* (*receive(g)*), for some message g in a finite set G of messages. One of the nodes in M is called the *initial node.* A *network* of CFSM's consists of two or more CFSM's where each pair of machines communicate by sending (or receiving) messages via one-directional, error-free, unbounded, FIFO channels. Let M and N be two CFSM's over the same message set G. We use (M,N) to denote the network of M and N. A *state* of (M,N) is a 4-tuple [v,w,x,y] where v and w are nodes in M and N respectively, and x and y are two strings over the message set G. Informally, the state [v,w,x,y] means that the executions of M and N have reached nodes v and w respectively, while the input channels of M and N contain the message strings x and y respectively. We define the *initial state* of (M,N) to be $[v_0,w_0,E,E]$, where v_0 and w_0 are initial nodes of M and N, respectively, and E is the null string. See [4] for more detailed definitions.

A *computation path* (or *path*, for short) of (M,N) is an infinite sequence of states *l*: $s_0,s_1,\ldots$ such that s_0 is the initial state of (M,N) and each s_{i+1} follows s_i by executing one move (i.e. traversing one edge) of M or N. (See [4] for the precise definition of "follow".) A path *l* of a network (M,N) is called *strongly fair* iff, for any node u in M or N, if u occurs infinitely often in *l* then all of its outgoing edges must be executed infinitely often in *l*. A path *l* is called *fair* iff for any node u in M (or N) that occurs infinitely often in *l*, the following two conditions are satisfied:

1. Each outgoing sending edge of u must be executed infinitely often in *l*.

2. If a state of the form $[u,w_i,x_i,y_i]$ ($[v_i,u,x_i,y_i]$) occurs infinitely often in *l*, and g is the head message in x_i (y_i), and if u has an outgoing edge e with label receive(g), then the edge e must be executed infinitely often in *l*.

A node u in M or N is said to be *live* (*weakly live*) iff u occurs infinitely often in every *fair* (*strongly fair*) path of (M,N). Given a network (M,N) and a node u in M or N, the liveness (weak liveness) problem is to determine if u is live (weakly live). For

certain restricted classes of CFSM networks, the liveness (weak liveness) problem becomes decidable. Among them, the following two classes of networks:

1. C_1: Networks of two CFSM's whose communication is known to be bounded by some constant k in one channel, and

2. C_2: Networks of two CFSM's in which one machine sends only one type of message,

were considered in [4]. There it was shown for both C_1 and C_2 that the problem of deciding whether a node was not live (weakly live) is NLOGSPACE-hard. Also, weak liveness was shown to be decidable in PTIME. In what follows, we derive some sharper results for these network classes. For example, we show:

Theorem 5.1: Given a network (M,N) in class C_1 (C_2) and a node u in M or N, the question "Is u not live?" is PTIME-hard.

Proof. The proof involves a reduction from the NEP for ω-FSM's. Given an ω-FSM W, we show how to construct in deterministic logspace, a network (M,N) in C_1 (C_2) to simulate the computation of W such that, a certain node in (M,N) is not live iff W accepts some input. As shown in Lemma 4.1 the fair NEP for ω-FSM's is PTIME-complete, hence we have that the liveness problem for C_1 (C_2) is PTIME-hard.

Given an ω-FSM $W=(Q,\Sigma,q_0,\delta,\{H_1,...,H_k\})$, without loss of generality, we assume that W is over a 2-letter alphabet, i.e., $\Sigma=\{a,b\}$. Furthermore, 1-acceptance is assumed. The network (M,N) is constructed as follows. The set of states in M consists of the set Q and a new isolated state u (a state without incoming or outgoing edges). The set of edges in M is constructed from those in W by replacing every read(a) (read(b)) transition by a receive(a) (receive(b)) edge. In addition, a self-loop labelled receive(#) is associated with each state in H_i, $1\leq i\leq k$. See Figure 5.1. Basically, the machine M acts as a "simulator" that simulates the computation of W in such a way that, each read(a) (read(b)) operation in W is simulated by a receive(a) (receive(b)) move in M. The machine N, on the other hand, is used to supply input symbols to M; while the channel from N to M represents the input tape of W. If W accepts any input, say l in $\{a,b\}^\omega$, then either:

1. l contains a finite number of b's and an infinite number of a's, or

2. l contains an infinite number of a's and b's, or

3. l contains a finite number of a's and an infinite number of b's.

Note that the nodes w_1, w_2 and w_3 in N are used to simulate the above three cases, respectively; while the node w_0 is used to supply a finite prefix. According to the construction of N, every fair path must include the states w_4 and w_5, and hence the

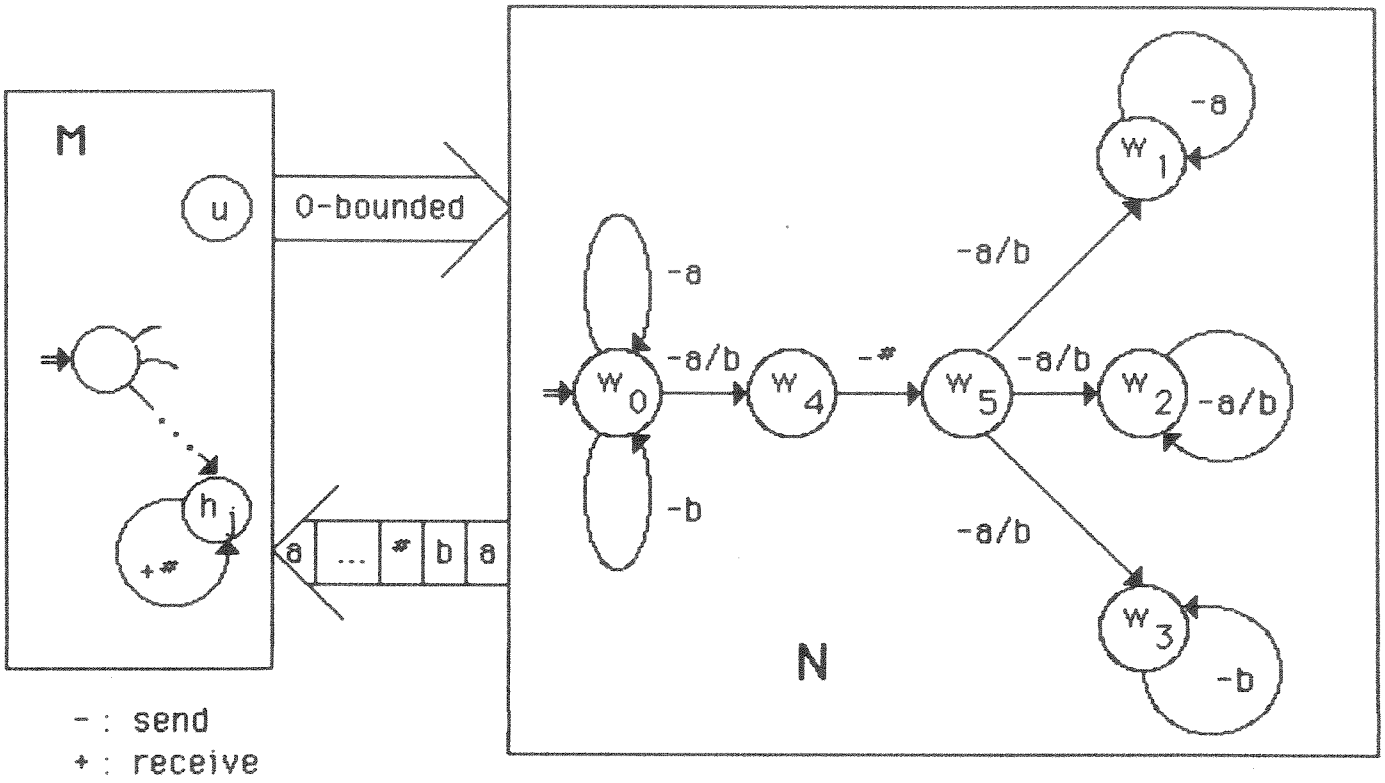


Figure 5.1 A network (M,N) in $\mathcal{C}_1$ and $\mathcal{C}_2$.

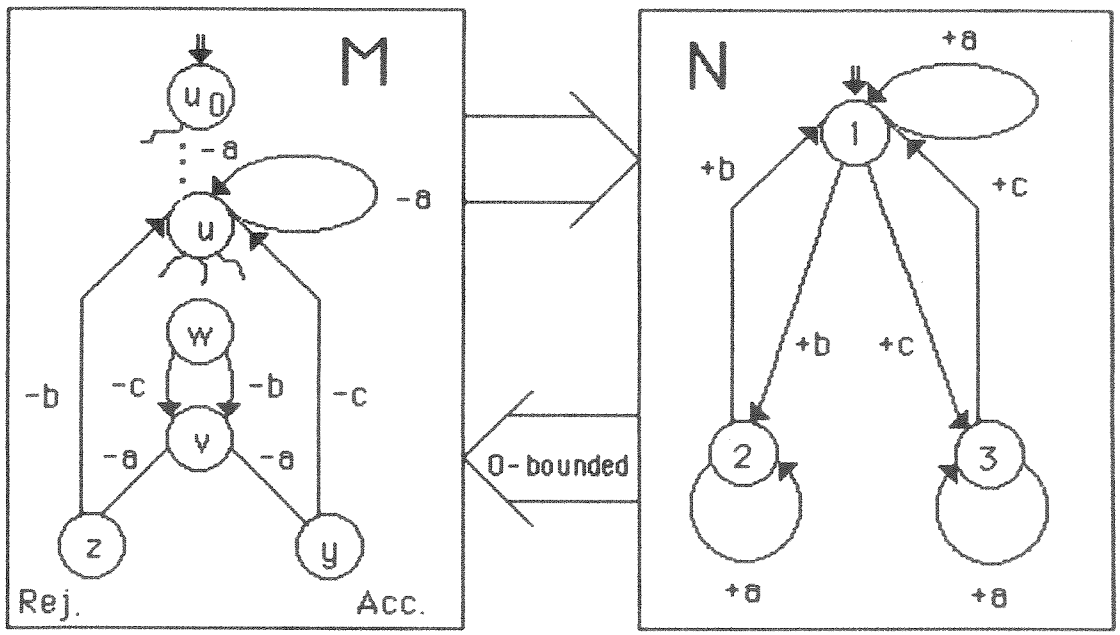


Figure 5.2 A network (M,N) in $\mathcal{C}_1$ and $\mathcal{C}_2$.

action send(#) must be executed. This implies that for every fair path, a receive(#) must be executed in M as well. Correspondingly, some state in H_i, for some i, must be visited. It should then be clear that, W accepts an input iff the node u is not live in (M,N). Since N's input channel is always empty, the network in Figure 5.1 is clearly in C_1 and C_2. Hence, the liveness problem for such networks is PTIME-hard. □

Lemma 5.1: Given a network (M,N) in class C_1 (C_2) and a node u in M or N, the question "Is u not weakly live?" is $NL^{NL[1]^{NL[1]}}$-hard.

Proof. Let W be an arbitrary 2-alternating ATM. Now given an input string x, we show how to construct a network (M,N) in C_1 (C_2) that will simulate the computation of W in such a way that, W accepts x iff a node u in M is not weakly live. See Figure 5.2. Basically M has the same structure as the machine M used in the proof of Lemma 4.4. (See Figure 4.3.) Let v be an existential configuration which immediately follows some universal configuration, say w. We add two edges from w to v; labelled "send(b)" and "send(c)", respectively. In addition, for each accepting configuration y, we add the edge from y to u labelled "send(c)"; while for each rejecting configuration z, we add the edge from z to u labelled "send(b)". Now, once the send(c) at v is executed, another send(c) must be executed later, which ensures that an accepting configuration is subsequently reached. Otherwise, machine N will get blocked in state 3. The reader, then, can easily see that u_0, the initial state of M, is not weakly live iff W accepts x. Furthermore, since M's input channel is always empty, (M,N) is in both C_1 and C_2. The lemma now follows. □

In what follows, without loss of generality, we assume that the channel from N to M is either bounded by a fixed constant k (for C_1 networks) or contains a single type of message (for C_2 networks). Let R' be the set of all computations of (M,N) such that, for every state [v,w,x,y] on the path, $|y| \leq 1$. The notation $c_1 \stackrel{*}{\rightarrow} c_2$ will be used to indicate that configuration c_2 is reachable from c_1 via a path in R'. The following two lemmas concerning necessary and sufficient conditions for the existence of a strongly fair path were given in [4]:

Lemma 5.2: For a network (M,N) in C_1, a node in M or N is weakly live iff it occurs infinitely often in every strongly fair path in R'.

Lemma 5.3: For C_1 networks, if there exists a strongly fair path in R' in which a node u occurs at most finite number of times, there must exist another such path q such that the sequence of moves (i.e. edges) in q is of the form $t_1(t_2)^\omega$, where $|t_1|$ and $|t_2|$ are bounded by a polynomial in 2^k, |M| and |N|. (Note that 2^k is a constant here.)

Using Lemmas 5.2 and 5.3, we can prove the following result:

Lemma 5.4: Given a network (M,N) in class C_1 (C_2) and a node u in M or N, the question "Is u not weakly live?" can be solved in $NL^{NL[1]^{NL[1]}}$.

Proof. Here we only consider networks in C_1. The basic idea is to, as in the proof of Lemma 4.3, find IRC's in M and N that avoid the node u and check whether they can be traversed infinitely often in some computation path. Note that since k is a fixed constant, the number of distinct states of (M,N) in R' is polynomial in terms of |M|, |N|, and |G|. Let P be the polynomial mentioned in Lemma 5.3. According to Lemmas 5.2 and 5.3, u is not weakly live in (M,N) iff

1. $\exists$ v_m and v_n in M and N, respectively, and strings x and y in G^* ($|x| \leq k$ and $|y| \leq 1$), such that $[v_0,w_0,E,E] \xrightarrow{*} [v_m,v_n,x,y]$,

2. $\forall$ w_m and w_n in M and N respectively, either

   a. $\neg(v_m \xrightarrow{*} w_m \wedge v_n \xrightarrow{*} w_n)$ (w_m (w_n) is not reachable from v_m (v_n) on the graph of M (N)), or

   b. $\exists$ strings r and s in G^*, $|r| \leq k$ and $|s| \leq 1$, such that $[v_m,v_n,x,y] \xrightarrow{*} [w_m,w_n,r,s] \xrightarrow{*} [v_m,v_n,x,y]$.

Consider the language

$$A^2 = \{(M,v_m,w_m,x,N,v_n,w_n,y) |$$
$$\exists \text{ strings r and s, } |r| \leq k \text{ and } |s| \leq 1, \text{ such that}$$
$$[v_m,v_n,x,y] \xrightarrow{*} [w_m,w_n,r,s] \xrightarrow{*} [v_m,v_n,x,y]\}.$$

Clearly A^2 is in NLOGSPACE. Now consider

$$A^1 = \{(M,v_m,x,N,v_n,y) | \text{ condition (2) does not hold for } [v_m,v_n,x,y]\}.$$

Given a network state $[v_m,v_n,x,y]$, condition (2) does not hold iff there exist states w_m (reachable from v_m) and w_n (reachable from v_n) such that (b) is not true. Thus, using A^2 as an NL oracle, we have that A^1 is in $NL^{NL[1]}$. Similarly, using A^1 as an oracle, we have that the previous algorithm (composed of checking conditions 1 and 2 above) is doable in $NL^{NL[1]^{NL[1]}}$. Thus, the lemma is proved.

A similar argument can be used for the case of C_2 networks. □

The following theorem follows immediately from Lemmas 5.1 and 5.4:

Theorem 5.2: For classes C_1 and C_2 networks, given a node u to determine whether u is not weakly live is $NL^{NL[1]^{NL[1]}}$-complete.

The second part of this section is devoted to applying our previous results to the Fair Model Checking Problem (FMCP) for Fair Computation Tree Logic (FCTL), which

was considered in [9]. In [9], the authors show that the FMCP is, in general, NP-complete. But the canonical form FMCP is shown to be doable in PTIME (actually linear time). However, no lower bound was given for this restricted class. In what follows, we show that the canonical form FMCP is PTIME-hard, and thus PTIME-complete. Before proceeding, some preliminaries are required. Further details can be found in [9].

A structure M=(S,R,L) is a labelled transition graph where S is a finite set of states, R ($\subseteq S \times S$) is a binary relation representing the possible transitions between states, and L is a labelling which assigns each state to a set of atomic transitions (i.e. those propositions which are true at that state). A fairness constraint Φ_0 is constructed from the propositions $\overset{\infty}{F}\, p$ ("infinitely often p") and $\overset{\infty}{G}p$ ("almost always p") using the standard boolean operators. Here p can be any boolean formula constructed from the atomic propositions. For a structure M and an infinite path $x = x_0, x_1, \ldots$, the $\models$ relation is defined inductively as follows: (Let $x^i = x_i, x_{i+1} \ldots$)

1. $M,x \models P$ iff the atomic proposition P is true at x_0.

2. $M,x \models \neg p$ iff not$(M,x \models p)$.

3. $M,x \models p \wedge q$ iff $M,x \models p$ and $M,x \models q$.

4. $M,x \models \overset{\infty}{F}\, p$ iff there exists infinite many $i \geq 0$ such that $M,x^i \models p$.

5. $M,x \models \overset{\infty}{G}p$ iff $\exists i \geq 0 (\forall j \geq i,\ M,x^j \models p)$.

Given a structure M=(S,R,L) and a fairness constraint Φ_0, the Fair State Problem (FSP) is to determine for each state s in S, whether there exists a path x (in M) starting at s such that $M,x \models \Phi_0$. Note that the FSP is a special case of the FMCP. In [9], it has been shown that the FMCP can be efficiently reduced to the FSP. There it is also shown that the FSP is, in general, NP-complete. However, when Φ_0 is in the canonical form $\bigwedge_{i=1}^{k} (\overset{\infty}{G} p_i \vee \overset{\infty}{F}\, q_i)$, the FSP can be solved in linear time. In what follows, we show that we can reduce the fair NEP for ω-FSM's under 2-acceptance to the canonical form FSP. Since the former problem is known to be PTIME-complete, the canonical form FSP is PTIME-hard (and hence so is the canonical form FMCP).

Lemma 5.5: The fair NEP for ω-FSM's under 2-acceptance can be reduced to the canonical form FSP by a deterministic log space reduction.

Proof. The basic idea of the proof is that, given an ω-FSM W, we construct a model M=(S,R,L) and a fairness constraint Φ_0 (of the form indicated) in deterministic log space such that, W accepts some input iff the FSP has a solution with respect to M and Φ_0.

Let $W=\{Q,\{a,b\},q_0,\delta,\{H_1,H_2,...,H_t\}\}$. Let $Q=\{q_0,q_1,...,q_r\}$. We define $S=Q\cup\{a_{i,j}|\ q_j\in\delta(q_i,a)\}\cup\{b_{i,j}|\ q_j\in\delta(q_i,a)\}\cup\{q'\}$. The relation R of M is constructed from the graph of W by replacing each edge (q_i,q_j) labelled read(a) (read(b)) by two edges $(q_i,a_{i,j})$ and $(a_{i,j},q_j)$ ($(q_i,b_{i,j})$ and $(b_{i,j},q_j)$). In other words, the state $a_{i,j}$ ($b_{i,j}$) is inserted between q_i and q_j, if $q_i\rightarrow q_j$. In addition, for every state q in S-$\{q_0\}$, the edges $q\rightarrow q'$ and $q'\rightarrow q$ are added. Finally, the edge $q'\rightarrow q_0$ is added. Note that R is strongly connected. See Figure 5.3. We define L as follows:

1. $Q_{q'}$ is true only at state q'.

2. Q_i $(0\leq i\leq r)$ is true at state s_j iff $s_j=q_i$.

3. H is true at state s_j iff $s_j\in \cup_{i=1}^{t} H_i$.

4. $A_{i,j}$ $(B_{i,j})$ $(0\leq i,j\leq r)$ is true at state s_d iff $s_d=a_{i,j}$ $(b_{i,j})$.

Now we define the fairness constraint to be the conjunct of the following:

(1) $\overset{\infty}{G}(\neg Q_{q'}) \vee \overset{\infty}{F}$ **false**

* This ensures the state q' occurs only finitely often. *\

(2) $\overset{\infty}{G}H \vee \overset{\infty}{F}$ **false**

* This ensures that a state in some H_i, $1\leq i\leq t$, occurs infinitely often. *\

(3) For every i, $0\leq i\leq r$, $\overset{\infty}{G}(\neg Q_i) \vee \overset{\infty}{F}((\vee_{j=0}^{r} A_{i,j}) \vee (\vee_{j=0}^{r} B_{i,j}))$

* For each q_i, either q_i is visited only finitely often or one of its outgoing edges must be traversed infinitely often. *\

(4) For every i,j, $0\leq i,j\leq r$, $\overset{\infty}{G}(\neg A_{i,j}) \vee \overset{\infty}{F}(\wedge_{k\neq j} A_{i,k})$

* For each state, if one of its outgoing edges labelled "read(a)" occurs infinitely often, then all of its outgoing edges with the same label must occur infinitely often. *\

(5) Replace A with B in case (4).

Let $\Phi_0=(1)\wedge(2)\wedge(3)\wedge(4)\wedge(5)$. It should be clear that W accepts something iff the FSP with respect to M and Φ_0 has a solution. This completes the proof. □

Theorem 5.3: The canonical form FMCP is PTIME-complete.

In what follows, we generalize the result in lemma 5.5 to ω-1CM's. To show this, we need the following easily seen facts: (Let $(p,i)\overset{*}{\rightarrow}(q,j)$ denote that in state p with

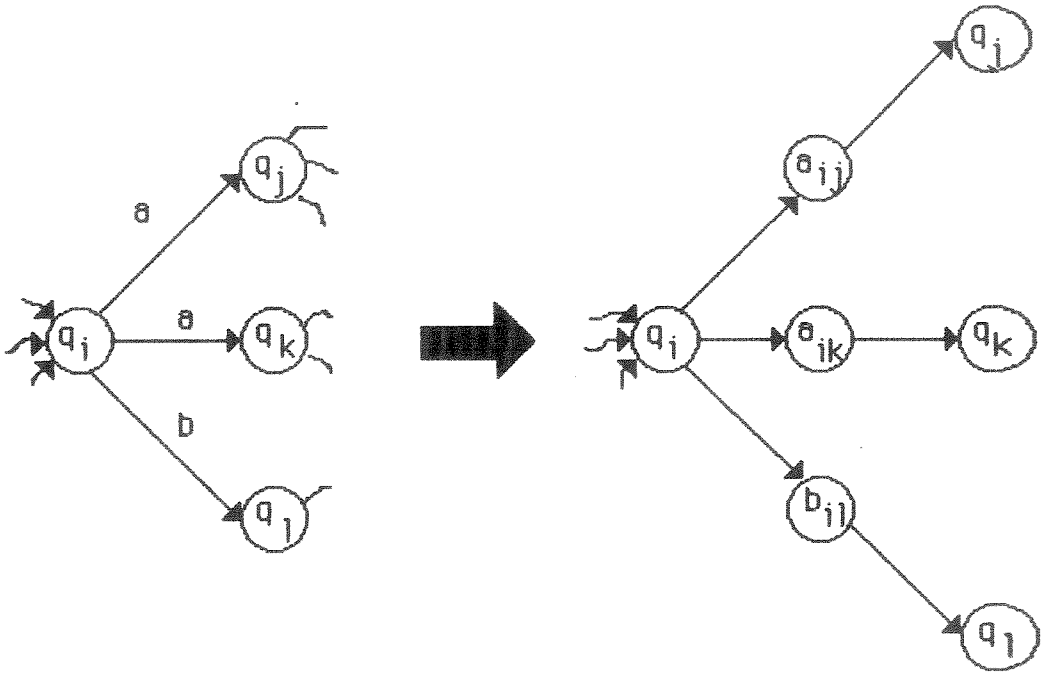


Figure 5.3 The construction of M from W.

counter value i, W can reach the state q with counter value j.) Now, there exists a polynomial P such that for every ω-1CM W, if there exists a fair run, then there exists a path $(q_0,0)\stackrel{*}{\rightarrow}(q,h)\stackrel{*}{\rightarrow}(q,h+d)$ such that

1. $0\leq h,d\leq P(|W|)$,

2. the length of the path is no more than $2*P(|W|)$, and

3. either $d=0$, or $d>0$ and no move in the path depends on the counter being zero.

The significance of this fact is that, we are able to capture the computations of some infinite paths using finite models. Therefore, using similar methods as were used in the proof of Lemma 5.5, we can derive the following result:

Lemma 5.6: The fair NEP for ω-1CM's can be reduced to canonical form FSP.

Hence, we also have:

Theorem 5.4: The fair NEP (for acceptance modes 1, 1', 2 and 2') for ω-1CM's is PTIME-complete.

References

[1] Bentley, J., Ottmann, T. and Widmayer, P., The complexity of manipulating hierarchically defined sets of rectangles, *Advances in Computing Research*, JAI Press Inc., Vol. 1, 1983, pp. 127-158.

[2] Borosh, I. and Treybis, L., Bounds on positive integral solutions of linear Diophantine equations, *Proc. AMS*, Vol. 55, No. 2, March 1976, pp. 299-304.

[3] Chandra, A., Kozen, D. and Stockmeyer, L., Alternation, *JACM*, Vol. 28, No. 1, January 1981, pp. 114-133.

[4] Chang, C., Gouda, M. and Rosier, L., Deciding liveness for special classes of communicating finite state machines, *Proc. of the 22nd Annual Allerton Conf. on Communication, Control, and Computing*, 1984, pp. 931-939.

[5] Cohen, R. and Gold, A., Theory of ω-languages. I: Characterizations of ω-Context-Free languages, *J. of Computer and System Sciences*, 15, 1977, pp. 169-184.

[6] Cook, S., The classification of problems which have fast parallel algorithms, *Fundamentals of Computation Theory*, LNCS 158, 1983, pp. 78-93.

[7] Cook, S., The complexity of theorem proving procedures, *Proc. of the 3rd*

Annual ACM Symp. on Theory of Computing, 1971, pp. 151-158.

[8] Emerson, E. and Lei, C., Modalities for model checking: branching time strikes back, *Proc. of the 12th Annual ACM Symp. on Principles of Programming Languages*, 1985, pp. 84-95.

[9] Emerson, E. and Lei, C., Temporal model checking under generalized fairness constraints, *Proc. of the 18th Annual Hawaii Int. Conf. on System Sciences*, 1985, pp. 277-288.

[10] Fischer, M. and Paterson, M., Storage requirements for fair scheduling, *Information Processing Letters*, 17, 1983, pp. 249-250.

[11] Garey, M. and Johnson, D., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H.Freeman and Company, San Francisco, 1979.

[12] Gouda, M. and Chang, C., A technique for proving liveness of communicating finite state machines with examples, *Proc. of the 3rd Annual ACM Symp. on Principles of Distributed Computing*, 1984, pp. 38-49.

[13] Gouda, M. and Rosier, L., On deciding progress for a class of communication protocols, *Proc. of the 18th Annual Conf. on Information Sciences and Systems*, Princeton Univ., 1984, pp. 663-667.

[14] Greibach, S., Remarks on blind and partially blind one-way multicounter machines, *Theoretical Computer Science*, 7, 1978, pp. 311-324.

[15] Hopcroft, J. and Pansiot, J., On the reachability problem for 5-dimensional vector addition systems, *Theoretical Computer Science*, 8, 1979, pp. 135-159.

[16] Hopcroft, J. and Ullman, J., "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley, Reading, Mass., 1979.

[17] Jones, N. and Laaser, W., Complete problems for deterministic polynomial time, *Theoretical Computer Science*, 3, 1977, pp. 105-117.

[18] Jones, N., Lien, E. and Laaser, W., New problems complete for nondeterministic log space, *Mathematical Systems Theory*, 10, 1976, pp. 1-17.

[19] Karp, R., Reducibility among combinatorial problems, in *Complexity of Computer Computations*, edited by R. E. Miller and J. Thatcher, Plenum Press, New York, 1972, pp. 85-104.

[20] Ladner, R. and Lynch, N., Relativization of questions about log space computability, *Mathematical Systems Theory*, 10, 1976, pp. 19-32.

[21] Landweber, L., Decision Problems for ω-automata, *Mathematical Systems Theory*, 3, 1969, pp. 376-384.

[22] Lehmann, D., Pnueli, A and Stavi, J., Impartiality, justice and fairness: The ethics of concurrent termination, *Automata, Languages and Programming*, LNCS 115, 1981, pp. 264-277.

[23] Lewis, H. and Papadimitriou, C., Symmetric space-bounded computation, *Automata, Languages and Programming*, LNCS 85, 1980, pp. 374-384.

[24] Lichtenstein, O. and Pnueli, A., Checking that finite state concurrent programs satisfy their linear specification, *Proc. of the 12th Annual ACM Symp. on Principles of Programming Languages*, 1985, pp. 97-107.

[25] Lynch, N., Log space machines with multiple oracle tapes, *Theoretical Computer Science*, 6, 1978, pp. 25-39.

[26] Papadimitriou, C., On the complexity of unique solutions, *JACM*, Vol. 31, No. 2, April 1984, pp. 392-400.

[27] Papadimitriou, C. and Yannakakis, M., The complexity of facets (and some facets of complexity), *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, 1982, pp. 255-259.

[28] Rackoff, C., The covering and boundedness problems for vector addition systems, *Theoretical Computer Science*, 6, 1978, pp. 223-231.

[29] Rosier, L. and Yen, H., A multiparameter analysis of the boundedness problem for vector addition systems, Univ. of Texas at Austin, Dept. of Computer Science, Tech. Report No. 85-03, 1985. (Also, to be presented at the *Fifth International Conference on the Fundamentals of Computation Theory*, September 1985.)

[30] Ruzzo, W., Simon, J. and Tompa, M., Space-bounded hierarchies and probabilistic computations, *Proc. of the 14th Annual ACM Symp. on Theory of Computing*, 1982, pp. 215-223.

[31] Savitch, W., Relationships between nondeterministic and deterministic tape complexities, *J. of Computer and System Sciences*, Vol. 4, No. 2, 1970, pp. 177-192.

[32] Stockmeyer, L., The polynomial-time hierarchy, *Theoretical Computer Science*, 3, 1977, pp. 1-22.