Introduction to Linear Asynchronous Structures

R. J. Lipton,[1] R. E. Miller,[2] and L. Snyder[3]

Research Report #101

[1] Department of Computer Science, Yale University, 10 Hillhouse Ave., New Haven, Connecticut 06520. Supported in part by Office of Naval Research Grant N00014-75-C-0752 and National Science Foundation Grant DCR-74-12870.

[2] Mathematical Sciences Department, T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598.

[3] Department of Computer Science, Yale University, 10 Hillhouse Ave., New Haven, Connecticut 06520. Supported in part by Office of Naval Research Grant N00014-75-C=0752.

Several different models of parallelism have been discussed so far at this conference -- Petri nets and Vector Addition Systems, to name two. These models have been around for some years and we are all familar with them (though they may not be fully understood as yet). By contrast, we will present a new model that introduces the idea of *bounded asynchronous* delay. This idea provids a useful tool for studying both synchronism and purely event-driven asynchronism by introducing a spectrum of approximating models to fill the gap between these two extremes.

Our goal is two-fold:

(1) to introduce our model of bounded delay, and

(2) to point out some of the subtleties of asynchronism that are elucidated by our model.

Thus, we hope to interest you in bounded asynchronism, generally, and our model specifically as well as to point out some of the difficulties of event-driven asynchronism.

1. Preliminaries

Our model addresses questions of asynchronism with bounded delay and there do not seem to be any theoretical treatments of this phenomenon in the literature. What we have done is pretty much to generalize the cellular array characterization of Smith and others [1], where we consider a linear array

$$\begin{tikzcd}
\boxed{M_1} \arrow[r, leftrightarrow] & \boxed{M_2} \arrow[r, leftrightarrow] & \cdots \arrow[r, leftrightarrow] & \boxed{M_n}
\end{tikzcd}$$

of n finite state machines capable of communicating with their neighbors. In the classic treatment of these devices, time is measured in discrete steps and the evaluation rule is "all machines capable of firing at a given step, do so." We introduce asynchronism by relaxing this rule to "some machine(s) capable of firing at a given step do so." Thus, a device can fire "at will," so to speak, and thus the concept of machines operating at different rates can be characterized.

All of the usual questions asked of the synchronous parallel arrays can be asked about these asynchronous arrays, such as synchronization, recognition capabilities etc. But an immediate observation is that in this asynchronous model some device, capable of firing, may postpone doing so for an unbounded or possible infinite number of steps.

Although exact response times are not always known, it is usually the case that some information is available about the relative response times. In such cases we believe that the information should be used. Hence we introduce the concept of *delay*, an upper bound on the response time of any device. As a consequence, we get sharper and more quantitative results as well as providing a convenient means of comparing synchronous and event-driven asynchronous behavior.

Delay is a fixed, nonnegative integer upper bound on the number of steps a device capable of firing is allowed to *postpone* that action. Hence, when $D = 0$, no postponesment is allowed and the system operates synchronously. Note the difference between *delay* and *frequency of firing*. In particular, a device which is always capable of firing must fire, at worst every $D + 1$ steps. When D tends to infinity the system tends to act totally as an event-driven asynchronous system. Thus, we have a whole spectrum of execution disciplines between the extreme points.

Before actually introducing the model, it may be useful to mention the difference between bounded asynchronism and nondeterminism. The distinction is, of course, between "what" to do and "when" to do it. Hence, there are really four possible models involving these two concepts:

deterministic	synchronous
nondeterministic	synchronous
deterministic	asynchronous
nondeterministic	asynchronous

To underscore the differences, consider the three element linear array

$$\begin{tikzcd} \boxed{a} \arrow[r, leftrightarrow] & \boxed{b} \arrow[r, leftrightarrow] & \boxed{c} \end{tikzcd}$$

and two types of state diagrams:

deterministic:

$$\begin{tikzcd} a \arrow[r, "\omega"] & r \\ b \arrow[r, "\omega"] & s \\ c \arrow[r, "\omega"] & t \end{tikzcd}$$

nondeterministic:

$$\begin{tikzcd} a \arrow[r, "\omega"] \arrow[dr, "\omega"] \arrow[ddr, "\omega"] & r \\ b \arrow[ur, "\omega"] \arrow[r, "\omega"] \arrow[dr, "\omega"] & s \\ c \arrow[uur, "\omega"] \arrow[ur, "\omega"] \arrow[r, "\omega"] & t \end{tikzcd}$$

where ω means that the transition is legal regardless of the states of the neighbors. Then we can construct a table:

	synchronous (D = 0)	asynchronous (D ≥ 2)
deterministic	#computation seq: 1 #different outputs: 1	#computation seq: 13 #different outputs: 1
nondeterministic	#computation seq: 27 #different outputs: 27	#computation seq: 351 #different outputs: 27

The computation sequences possible are compared with the number of possible outputs and we see that for this simple system, the asynchronism influences how the output is arrived at (computation sequences) but not the output itself. Note that the table is for some $D \geq 2$ and that if $D = 0$ were assumed, both columns would be the same.

This property, which we might call *transparency*, where the asynchronism influences only the way the result is obtained but not the result itself, does not necessarily always hold. In particular, there are deterministic machines which, when run asynchronously, give several different outputs. This is because a device, capable of changing from state a to state b, can postpone that transition and then, as a result of a state change by one of its neighbors, it may be capable of a transition from a to c. Consequently, nondeterminism has apparently been introduced by the asynchronism! This phenomenon, which might be called *surreptitious nondeterminism*, is poorly understood and preliminary indications are that it leads to a different class of systems. Our interest here will be to achieve transparency for the deterministic asynchronous case, since this is a broad and more manageable class of systems.

2. The Model

Since working with finite state machines is cumbersome, we have developed a rewriting system which generalizes the standard finite state machine model.

Definition: An *asynchronous grammar* $G = (\Sigma, P)$ is a finite alphabet Σ and a finite set of productions of the form $\alpha \rightarrow \beta$ where

(i)	$\alpha,\beta \in \Sigma^*$	(symbols represent states)
(ii)	$\|\alpha\| = \|\beta\|$	(length preserving)
(iii)	$\alpha \neq \beta$	(no "idling" productions)

A set of devices in various states and with various interconnections is, thus, represented by a word in Σ^n. A symbol capable of being changed by a production application is said to be *active*. A computation is described by defining $\vdash$.

Definition: Let $x = x_1 \ldots x_n$, $y = y_1 \ldots y_n$ where $x,y \in \Sigma^n$. Then

$$x_1 \ldots x_n \vdash y_1 \ldots y_n$$

if $x \neq y$ and if the ith position of x changes (i.e. $x_i \neq y_i$) then there exists a production $\alpha_1 \ldots \alpha_j \ldots \alpha_k \rightarrow \beta_1 \ldots \beta_j \ldots \beta_k$ such that

(i) $\alpha_1 \ldots \alpha_j \ldots \alpha_k$ matches some context surrounding x_i
(i.e. $\exists j \ni x_{i-j+1} \ldots x_i \ldots x_{i-j+k} = \alpha_1 \ldots \alpha_j \ldots \alpha_k$)

and

(ii) the changes implied by the productions obtain and are consistent (i.e. $\alpha_s \neq \beta_s \Rightarrow y_{i-j+s} = \beta_s$ for $s = 1, 2, \ldots, k$).

The first requirement guarantees that the transitions performed are legal according to the production set and the second requirement makes certain that all changes take place and are not contradictory.

As an example, let $\Sigma = \{a, b, c, d\}$ and P =

$$\begin{aligned}\{ab &\to ac \\ bc &\to bd \\ ab &\to dc\}\end{aligned}$$

then the legal transitions would be, for abc,

1.	$abc \vdash acc$	$ab \to ac$ applied
2.	$abc \vdash abd$	$bc \to bd$ applied
3.	$abc \vdash dcc$	$ab \to dc$ applied
4.	$abc \vdash acd$	$ab \to ac$ and $bc \to bd$ applied
5.	$abc \vdash dcd$	either $ab \to dc$ and $bc \to bd$ applied or all three applied

Note that in the fourth case the two productions do overlap but the changes are made consistently and so this is allowed by the definition. Also, in the final case (and 3 as well) it is ambiguous just exactly what productions applied since the production $ab \to dc$ subsumes the production $ab \to ac$. Another point to note is that multiple changes are allowed ($ab \to dc$). Clearly, very complex behavior can be described by asynchronous grammars.

The reflexive transitive closure ($\overset{*}{\vdash}$) of $\vdash$ is defined in the usual way, but, as the following definition indicates, some sequences may not qualify as acceptable computations. (Notationally, superscripts are used for elements of a sequence and subscripts are used for coordinates.)

Definition: Let $x^0, x^1, \ldots$ be in Σ^n, $G = (\Sigma,P)$ be an asynchronous grammar and $D \geq 0$ be an integer. A *D-computation* is a sequence

$$x^0, x^1, \ldots$$

such that

$$\text{(i)} \ \forall j \geq 0, x^j \vdash x^{j+1}$$

and

(ii) $\nexists\, i,j$ such that $x_i^j = \ldots = x_i^{j+k}$ and coordinate i is active for all $k = 1, 2, \ldots, D+1$.

Thus, a D-computation is a sequence of legal transitions defined by an asynchronous grammar where no postion postpones firing longer than D steps. When $D = 0$, (i.e. no postponement), the D-computation is said to be synchroncus — i.e. when a postion becomes active at the j^{th} step it fires at the $j+1^{st}$ step. A D-computation

$$x^0, \ldots, x^m$$

halts when $\nexists\, x \in \Sigma^n$ such that $x^m \vdash x$, i.e. when no further changes take place.

As an example, consider $\Sigma = \{*,a,b,c\}$ and $P =$

$$\{\begin{array}{lcl} *a & \to & *b \\ a* & \to & c* \\ baa & \to & bba \\ ac & \to & cc \end{array}\}$$

then the 0-computation on $*aaaa*$ is

$$*aaaa* \vdash *baac* \vdash *bbcc*$$

while for $D \geq 3$, the D-computations are

$$\begin{aligned} *aaaa* &\overset{*}{\vdash} *bbcc* \\ *aaaa* &\overset{*}{\vdash} *bbbc* \\ *aaaa* &\overset{*}{\vdash} *bccc* \\ *aaaa* &\overset{*}{\vdash} *cccc* \end{aligned}$$

where each computation has been forced to a halting state. Now, clearly, this particular grammar on *aaaa* is not transparent although it is deterministic in the sense of finite state machines. Note where the surreptitious nondeterminism crept in.

$$\ldots\ b\underset{\substack{\Big\uparrow \\ \text{can go only} \\ \text{to b}}}{a}ac\ \ldots \quad \vdash \quad \ldots\ b\underset{\substack{\Big\uparrow \\ \text{can go only} \\ \text{to c}}}{a}cc\ \ldots$$

(Note that in the degenerate case, the grammar isn't even deterministic since $*a* \vdash *b*$ and $*a* \vdash *c*$ are legal.)

There are several ways to restrict asynchronous grammars to only those which lead to transparent D-computations. One such method to avoid the interference inherent in the previous example will now be considered.

<u>Definition</u>: An asynchronous grammar $G = (\Sigma,P)$ is *interference-free*

if $\forall p, p'$ and t

$$p = \alpha_1 \ldots \alpha_{t+1} \ldots \alpha_k \rightarrow \beta_1 \ldots \beta_{t+1} \ldots \beta_k ,$$

$$p' = \alpha'_1 \ldots \alpha'_\ell \rightarrow \beta'_1 \ldots \beta'_\ell$$

and $\alpha_{t+i} = \alpha'_i \quad i = 1,2,\ldots, \min(\ell, k-t)$ implies

$\alpha_{t+i} = \alpha'_i = \beta_{t+i} = \beta'_i$.

Informally, an asynchronous grammar is interference-free if whenever two productions overlap, the overlapping portion is unchanged in both productions. Thus

$$\begin{array}{lll} p: & a \ldots bxyz & \rightarrow e \ldots fxyz \\ p': & \quad xyzc \ldots d & \rightarrow \quad xyzg \ldots h \end{array}$$

could be productions of an interference-free grammar.

Now returning to the grammar that suggested this restriction, we observe that there is an interference-free asynchronous grammar that is transparent and equivalent to the former grammar for 0-computations on $*a^k*$ for all $k \geq 2$. Specifically,

$$\Sigma = \{*, a, b, c, b', c', \bar{b}, \bar{c}\}$$

and

$$\begin{aligned} P = \{ *a &\rightarrow *c' \\ c'a &\rightarrow cb' \\ b'a &\rightarrow bc' \\ c'* &\rightarrow c* \\ b'* &\rightarrow b* \\ cb &\rightarrow \overline{cb} \\ \overline{cb} &\rightarrow bc \}. \end{aligned}$$

This grammar is clearly interference free since the only overlaps leave the overlapping portions unchanged:

$$\begin{array}{l} c'{*} \rightarrow c{*} \\ \quad {*}a \rightarrow {*}c' \end{array}$$

and

$$\begin{array}{l} b'{*} \rightarrow b{*} \\ \quad {*}a \rightarrow {*}c' \;. \end{array}$$

The grammar is also substantially more complicated than the previous one as can be seen in the following 0-computation:

$$\begin{array}{rl} {*}aaaa{*} & \vdash {*}c'aaa{*} \\ & \vdash {*}cb'aa{*} \\ & \vdash {*}cbc'a{*} \\ & \vdash {*}\bar{c}\bar{b}cb'{*} \\ & \vdash {*}bccb{*} \\ & \vdash {*}bc\bar{c}\bar{b}{*} \\ & \vdash {*}bcbc{*} \\ & \vdash {*}b\bar{c}\bar{b}c{*} \\ & \vdash {*}bbcc{*} \end{array}$$

Note that the strategy is to first initialize the string with b's and c's (half of each) and then interchange adjacent pairs until they propagate to their respective ends.

Rather than trying to argue directly that this interference-free grammar is in fact transparent, we appeal to the following theorem.

<u>Theorem</u> (Transparency for interference-free grammars)

Let $G = (\Sigma,P)$ be an interference free asynchronous grammar such that $x^0, \ldots, x^m$ is a halting 0-computation, then $\forall D > 0$, for every D-computation

(i) $\exists q$ such that $x^0, \ldots, x^q$ is a halting D-computation, and

(ii) $x^q = x^m$.

This can be proved in several ways. A direct proof would argue that for each position i in x^0, the sequence of 0-computation state changes that take place also occur, in the same order, for any D-computation.

Thus we have a characterization of one class of asynchronous grammars which is transparent. Note that the interference-free grammar just presented allowed multiple changes. This is not really faithful to our original objective where the machines are to fire individually. An alternative characterization has been developed where the single change property does hold. This alternative model and results concerning its timing characteristics, recognition capabilities and synchronization capabilities is presented in the proceedings of the *IEEE* symposium on the Foundations of Computer Science [2].

3. Decidability

Next we consider some decidability questions. Clearly, since the productions are length preserving and thus there is a bound on the number of states from any initial input x^0, it follows that the halting problem for any given input x^0 is decidable, for a given delay D. Moreover, because Smith [1] has embedded a Turing machine in his model (which is contained in ours as a special case), it follows that it is not decidable whether, for a given D, an asynchronous grammar halts for all inputs. Also, the reachability problem is decidable for a given asynchronous grammar G, delay D, input x^0 and reachability configuration x^r.

Those are the usual questions one asks about parallel systems. But because these are asynchronous systems too, we can ask questions about delay. For example, since increasing the value of D often increases the reachable states, we can ask the D-reachability question.

Problem: (D-reachability) Given an asynchronous grammar G, initial input x^0 and reachability configuration x^r, what is the least delay D, if any, such that $x^0, \ldots, x^r$ is a D-computation?

Thus, if G = {*a → *b
a* → c*
baa → bba
ac → cc}

and x^0 = *aaaa* and x^r = *cccc*, the D-reachability number is 2. That is, for $D \leq 1$, this configuration cannot be reached from *aaaa*. Note that the D-reachability number for x^r = *bbbb* is ∞, i.e. no matter how large D is this configuarion cannot be obtained (due to the third production).

The D-reachability problem is decidable.

The argument here is based on an enumeration af all reachable states for successively larger D. The enumeration stops when D is large enough so that no transition fires due to the expiration of the delay. This value is essentially the longest acyclic sequence of single production applications that can take place independent of some pending transition.

Another variant on the reachability question suggested by delay is based on the concept of duration.

Definition. Let $x^0, x^1, \ldots$ be a D-computation for an asynchronous grammar G on input x^0. The duration $\delta(i,j)$ of position j in configuration i is

$$\delta(i,j) = \begin{cases} 0 & \text{if } x_j^i \text{ is not active} \\ k+1 & \text{if } x_j^i \text{ is active and } k \text{ is the largest value such that } x_j^{i-k} = \ldots = x_j^i \text{ and } x_j^{i-\ell} \text{ is active for all values } \ell = 0, \ldots, k. \end{cases}$$

Hence, the duration $\delta(i,j)$ gives the number of consequtive steps that the jth position has been waiting to fire as of the ith step in the D-computation.

Problem: (Duration reachability) Given an asynchronous grammar G, initial configuration x^0, reachability configuration x^r and n integers $k_1, \ldots, k_n$, is it decidable whether or not there exists a D such that

$$x^0, \ldots, x^r$$

is a D-computation with $\delta(r,j) = k_j$?

Hence, we not only want to know if we can reach a particular configuration, but also do the positions have a particular duration value.

The duration-reachability problem is decidable. The argument uses the same enumeration as for the D-reachability problem but with the added constraint of a complex search to determine if the proper duration values hold.

4. Timings

Finally, note should be made of the fact that although asynchronous grammars operating asynchronously are weaker in certain respects than their synchronous counterparts, the reason is not due to their timings. (The reason is in fact their inability to solve problems such as the "firing squad synchronization problem," see [2].) Indeed, one gets the impression from our discussion that as D increases and the length of the computation sequences increase, the execution time of the parallel system degrades. Our experience, of course, tells us that asynchronism should probably improve rather than degrade the performance. Reconciling these two observations is easy: the quantity $D + 1$ always represents one unit of physical time and consequently an increase in D merely means that time is being partitioned into finer and finer units. Hence, given timings for the various transitions, the synchronous case runs at a rate equal to the time of the longest transition. The asynchronous transitions will vary in cost; each being charged according to the actual usage. Hence even though the computation sequences are longer in terms of the *number* of transitions in the asynchronous case, the actual cost need not be larger.

In conclusion, we emphasize that the important property of this model is that delay is treated parametrically. Thus, a single system may be studied with a synchronous as well as an asynchronous exectuion behavior within a single model merely by changing the quantity of the delay. Comparison of the two types of execution is, therefore, quite convenient. It is hoped that this introduction has presented sufficient motivation to interest others in taking a similar approach with other models of parallel computation.

References

1. A. R. Smith. Real-time language recognition by one-dimensional cellular automata. *JCSS* 6:233-253, 1972.

2. R. J. Lipton, R. E. Miller, and L. Snyder. Synchronization and computing capabilities of linear asynchronous structures. Proceedings of the *IEEE* Symposium on the Foundations of Computer Science, 1975, 19-28.