The Inductive Inference of Cyclic Synchronized Interleaving

Brian J. Ross[1]

Abstract. An inductive inference algorithm for inferring cyclic interleaving expressions from homing sequences is presented. The restricted language studied uses a synchronized interleaving or shuffle operator, which is an important component in algebraic theories of concurrency. Expressions of this language denote concurrent perpetual processes that communicate via synchronized handshaking. The paper introduces an algorithm for inferring an interleaved expression from a homing sequence, which is a contiguous, non-resettable trace describing a desired target behaviour. A main component of the algorithm deals with the determination of language terms that are consistent with the homing sequence. Term consistency is modelled via a labelled acyclic digraph. Various order-preserving properties of interleaving are used to refine this digraph during the inference session. In addition, the digraph permits efficient construction of hypotheses that admit correction by the teacher. The inference algorithm converges to any target expression equivalence class in $O(nk^2)$ time and $O(k^2)$ space for an alphabet of size k and homing sequence of size n. Given the importance of infinitary cyclic interleaving in concurrency theory, this research shows that machine learning can be tractably applied towards concurrent computations. This may lead to new semi-automated methodologies founded in machine learning technology for synthesizing and testing concurrent and distributed systems.

1 Introduction

This paper introduces a polynomial inductive inference technique for a restricted interleaved language $\mathcal{I}^\omega$ that features a synchronized interleaving operator similar to ones in process algebra [?, ?]. Interleaving models the nondeterministic computation of processes, and its synchronizing semantics models the synchronized communication or handshaking between processes. For example, the expression $S \parallel_A T$ denotes the concurrent execution of processes S and T. Both processes nondeterministically contribute their component actions into a visible stream or trace for the expression. In order to avoid deadlock, these processes must synchronize on any actions they generate that occur in a synchronization set A. In addition, S and T are *perpetual processes*, and have the the form $(\alpha_1 ... \alpha_k)^\omega$. In the absence of deadlock, perpetual processes are infinitely cycling ones that never terminate. Such processes are common in practice, and may denote operating system components, distributed network nodes, and other concurrent structures.

This paper discusses an efficient inductive inference algorithm for inferring an $\mathcal{I}^\omega$ expression from an example trace that exhibits the desired behaviour for the target expression. The algorithm uses a *homing sequence* [?], which is a non-resettable sequence of positive behaviour that orients the inference towards a desired result. The learning scenario is as follows. A teacher presents an example trace to the algorithm. The algorithm then infers an $\mathcal{I}^\omega$ expression which is consistent with this trace, and this hypothesis is presented to the teacher. If the teacher is satisfied with it, the algorithm terminates. Otherwise, if the hypothesis is inadequate, the teacher supplies a further extension of the trace or homing sequence. The algorithm refines its hypothesis based on any additional information culled from this new trace extension, and then presents a revised hypothesis to the teacher. This continues until a satisfactory expression is eventually converged upon, or until it is determined that the homing sequence can only be modelled by a degenerate expression. The above implies that a hypothesis expression is constructed that admits correction by the homing sequence. The algorithm has a time and space complexity of $O(nk^2)$ over the size k of the alphabet and size n of homing sequence.

Section ?? introduces the language studied in the paper. Section ?? presents the inductive inference algorithm. Conclusions and related work are discussed in section ??.

2 Language definition and properties

2.1 Definition

A restricted interleaving language $\mathcal{I}^\omega$ will be used. Given a finite set of atomic actions $(\alpha, \beta \in) \Sigma = \{a, b, c, ...\}$, the abstract syntax of $\mathcal{I}^\omega$ is:

$$\begin{array}{lll} \mathcal{I}^\omega \ ::= & (\alpha_1.\alpha_2. \cdots \alpha_k)^\omega \quad | & : k \geq 1, \forall_{i \neq j} : \alpha_i \neq \alpha_j \\ & E_1 \parallel_A E_2 & : \forall \alpha \in E_1, \beta \in E_2 : \\ & & \quad \alpha = \beta \Rightarrow \alpha \in A \end{array}$$

where $E_1, E_2 \in \mathcal{I}^\omega$, and $A \subseteq \Sigma$. For convenience, the expression $\alpha_1\alpha_2\alpha_3$ denotes $\alpha_1.\alpha_2.\alpha_3$. The term $(\alpha_1\alpha_2 \ \cdots \ \alpha_k)^\omega$ is called a *cycle*. The initial element α_1 in this cycle *owns* the cycle, and we call the cycle an α_1*-cycle*. The size of a cycle is the number of elements in it. The set A is the *synchronization set* for an expression, and contains the elements that must synchronize between terms in order to be generated. This set A is fixed for an entire expression, and the notation $\parallel$ replaces $\parallel_A$ when there is no confusion about A. Informally, an $\mathcal{I}^\omega$ sentence $S = s_1^\omega \parallel_A \ \cdots \ \parallel_A s_k^\omega$ is one in which (i) no

[1] Department of Computer Science, Brock University, St. Catharines, Ontario, Canada L2S 3A1

ECAI 94. *11th European Conference on Artificial Intelligence* Edited by A. Cohn
Published in 1994 by John Wiley & Sons, Ltd.

element in any cycle s_i is repeated within that cycle s_i, and (ii) an element shared amongst cycles s_i and s_j $(i \neq j)$ is a member of the synchronisation set A. It is possible for A to be empty, in which case no synchronization occurs, and in which all $\alpha \in \Sigma$ are partitioned among the terms of the expression. It is assumed that, given an expression $s_1^\omega \parallel \cdots \parallel s_n^\omega$, then $\Sigma = \cup\{\alpha \in s_i\}$.

A trace or stream is an expression composed entirely of actions. The following trace model is based on one in [?], and defines the interleaving behaviour of $\parallel_A$.

Definition 1 Trace model for $\mathcal{I}^\omega$:

$$\begin{array}{ll} \mathcal{T}((\alpha_1\alpha_2...\alpha_k)^\omega) & = \{\alpha_1.t \mid t \in \mathcal{T}(\alpha_2...\alpha_k\alpha_1)\} \\ \mathcal{T}(E_1 \parallel_A E_2) & = \{t \mid \exists t_1 \in \mathcal{T}(E_1), t_2 \in \mathcal{T}(E_2) \\ & \quad : t \in (t_1 \parallel_A^* t_2)\} \end{array}$$

$where\ \ \alpha s \parallel_A^* \beta t = \beta t \parallel_A^* \alpha t =$

$$\begin{cases} \alpha.(s \parallel_A^* t) & : if\ \ \alpha = \beta\ \ and\ \ \alpha, \beta \in A \\ \emptyset & : if\ \ \alpha \neq \beta\ \ and\ \ \alpha, \beta \in A \\ \alpha.(s \parallel_A^* \beta t) & : if\ \ \alpha \notin A\ \ and\ \ \beta \in A \\ \alpha.(s \parallel_A^* \beta t) \cup \beta.(\alpha s \parallel_A^* t) & : if\ \ \alpha \notin A\ \ and\ \ \beta \notin A \ \Box \end{cases}$$

The trace of a single cycle $\mathcal{T}((abc)^\omega)$ is simply the infinitely cycling stream $(abc)^\omega$ itself. $\parallel_A$ preserves the cyclic order of elements within single terms, but shuffles the order of elements between terms themselves. The subsidiary $\parallel_A^*$ operator defines interleaving based upon actions in the streams generated by component cycles. The synchronizing set A models synchronizing handshakes between two or more agents. Elements from A within cycles must synchronize within the trace, or else deadlock occurs. A set of $\mathcal{I}^\omega$ expressions $\{E_1, ..., E_k\}$ over Σ belongs to the same *equivalence class* if $\mathcal{T}(E_1) = \mathcal{T}(E_2) = ... = \mathcal{T}(E_k)$.

Given a trace s, the notation $\alpha \prec \beta$ means that α *initially precedes* β in s. Similarly, $\alpha \prec \beta$ for a cycle E^ω iff α initially precedes β in E. A *subtrace* of a trace t is some trace s whose elements are taken from t and share the same relative order. More precisely, s is a subtrace of t if $\forall \alpha, \beta \in s : if\ \ \alpha \prec \beta\ \ in\ \ S,\ \ then\ \ \alpha \prec \beta\ \ in\ \ t$. For example, bde is a subtrace of $abcdef\cdots$. Subtraces may be finite or infinite in size. A subtrace of size k is called a k-trace. A 2-trace is called a *pair*. An *embedded cycle* is a subtrace that may have been generated by a particular cyclic term $(\alpha_1...\alpha_k)^\omega$. For example, $(cdc)^\omega$ generates the embedded cycle cdc within the trace $abcdecf$.

To summarize, letting $\Sigma = \{a, b, c, x, y\}$, one possible $\mathcal{I}^\omega$ expression E is $(abc)^\omega \parallel_{\{b\}} (xby)^\omega$, and one $t \in \mathcal{T}(E)$ is $axbycxabcya\cdots$.

2.2 Properties

We first identify the types of $\mathcal{I}^\omega$ expressions we wish to study. A cycle $(\alpha_1...\alpha_k)^\omega$ of some $\mathcal{I}^\omega$ expression E is *deadlocked* if some α_i cannot be contributed to any trace. An $\mathcal{I}^\omega$ expression E is deadlocked if it has a deadlocked cycle. Note that deadlock does not imply the case when greedy cycles preempt others from running. For example, in $a^\omega \parallel b^\omega$, the a^ω term might indefinitely pre-empt b^ω. Rather, deadlock implies that a particular cycle derivative will *never* proceed, due to a lack of synchronization amongst terms. We are interested in the inductive inference of non-deadlocked expressions, which we call *well-behaved*.

The following theorems state that the ordering found within cycles and their traces are consistent with each other. They give a means for identifying and discarding impossible cycles. Omitted proofs invariably use structural induction over traces and expressions.

Theorem 1 *Given $s = \mathcal{T}(E)$, if $\alpha \prec \beta$ in s, then α cannot be in any β-cycle.*
Proof: *Structural induction on definition 1 ($\mathcal{T}$).* □

Theorem 2 *If $\alpha \prec \beta$ in cycle E, then $\alpha \prec \beta$ for $s \in \mathcal{T}(E)$.*
Proof: *Structural induction on definition 1.* □

Theorem 3 *A trace $\mathcal{T}(E) = t_1\alpha t_2\beta t_3\beta\cdots \quad (|t_i| \geq 0)$ is observed, where $\alpha \notin t_3$. Then α and β cannot belong to the same cycle.*
Proof: *Structural induction on cycles. By definition 1, if α and β belong to the same cycle, then they must cycle between one another in the trace. If $\alpha \notin t_3$, then α and β are not cycling, and cannot belong to the same cycle.* □

The following gives a way for identifying largest-sized cycles, which is important for constructing hypotheses that permit corrections, and hence convergence.

Theorem 4 *Given subtraces $\alpha s_1\alpha$ and $\alpha s_2\alpha$ found in a trace $s \in \mathcal{T}(E)$, the largest-sized α-cycle that can generate these subtraces is $(\alpha s')^\omega$, where s' is the largest subtrace common to s_1 and s_2.* □

Corollary 5 *Theorem* **??** *extends to any number of subtraces $\cup\{\alpha s_i \alpha\}(i \geq 2)$, where s' is the largest subtrace common to all the s_i.* □

Finally, the following are concerned with the structure of traces when synchronization is used. They permit an equational characterization of traces for well-behaved $\mathcal{I}^\omega$ expressions.

Theorem 6 *Given $s^\omega \parallel_A t^\omega$ where $s \cap A = t \cap A = \emptyset$, then $\mathcal{T}(s^\omega \parallel_A t^\omega) = \mathcal{T}(s^\omega \parallel_\emptyset t^\omega)$.*
Proof: *From definition 1.* □

Theorem 7 *Given*

$$E = (s_1\alpha_1 s_2\alpha_2 \cdots s_{n-1}\alpha_{n-1}s_n)^\omega \parallel_A (t_1\alpha_1 t_2\alpha_2 \cdots t_{n-1}\alpha_{n-1}t_n)^\omega$$

in which E is well-behaved, $|s_i| \geq 0$, $|s_i| \geq 0$, $A = \cup_i\{\alpha_i\}$, and $\forall i : \alpha_i, \beta_i \notin s_i \cup t_i$. Then

$$\mathcal{T}(E) = \cup\{w_1\alpha_1 w_2\alpha_2 \cdots w_{k-1}\alpha_{k-1}w_k\},$$

where $w_i = s_i \parallel^*{}_\emptyset\, t_i$.
Proof: *Structural induction on E. Using definition 1, because E is well-behaved, the above schemata holds.* □

Theorem 8 *Theorem* **??** *extends to any number of terms within E. Given $E = E_1 \parallel_A \cdots \parallel_A E_m$, where E is well-behaved,*
$E_i = (s_{i,1}\alpha_1 \cdots s_{i,n-1}\alpha_{n-1}s_{i,n})^\omega \ (n > 1)$, and $A = \cup_i\{\alpha_i\}$. Then

$$\mathcal{T}(E) = \cup\{(w_1\alpha_1 w_2\alpha_2 \cdots w_{n-1}\alpha_{n-1}w_n)^\omega\}$$

where $w_i = s_{1,i} \parallel^*{}_\emptyset\, s_{2,i} \parallel^*{}_\emptyset \cdots \parallel^*{}_\emptyset\, s_{m,i}$. □

Theorem ?? gives a generalized syntactic form of $\mathcal{I}^\omega$ expressions having equivalent traces, which defines an equivalence class for $\mathcal{I}^\omega$ expressions. When using a non-empty synchronization set A, it is likely the case that there are multiple hypotheses that can generate the same stream behaviour. For example, letting $A = \{b\}$, both $(abc)^\omega \parallel_A (xbz)^\omega$ and $(abz)^\omega \parallel_A (xbc)^\omega$ reduce to $\{(t_1.b.t_2)^\omega : t_1 \in \mathcal{T}(a \parallel_A^* x), t_2 \in \mathcal{T}(c \parallel_A^* z)\}$, and hence can be regarded as semantically equivalent. Thus, the equivalence class is relevant when A is non-empty.

3 Inductive Inference of $\mathcal{I}^\omega$

3.1 Algorithm

The major steps of the inference algorithm are: (i) determining what $\mathcal{I}^\omega$ cycles can possibly generate the homing sequence; (ii) determining the synchronization set A consistent with the homing sequence; and (iii) using the information in (i) and (ii), constructing an appropriate hypothesis. These steps are discussed in more detail below.

The input to the inference algorithm takes the form of a homing sequence - an extendable, non-resettable stream of positive behaviour [?]. A homing sequence is well-suited to $\mathcal{I}^\omega$, as its indefinite size corresponds well to the infinite traces that occur in $\mathcal{I}^\omega$. In addition, each extension given by the teacher extends the earlier homing sequence subtrace with improved examples of desired behaviour. This permits interactive correction of erroneous assumptions made by the inference algorithm as evident in hypotheses.

3.1.1 Term inference by digraph refinement

When given a homing sequence for some target expression, the inference algorithm determines what $\mathcal{I}^\omega$ cycles can generate this sequence. It is initially presumed that all terms over Σ are candidates. This set of terms is then refined by identifying terms that are inconsistent with homing sequence extensions as given during the inference session (using theorems ?? through ??) and excluding such inconsistent terms from further consideration. In addition, it is fundamentally important that the teacher is able to correct hypotheses presented by the algorithm. To do this, each hypothesis must contain one maximally-sized cycle consistent with the homing sequence so far. Thus information regarding the size of terms extractable from the digraph is maintained.

The modelling of consistent cycles is done with a labelled acyclic digraph, and inference decisions cause graph reduction and expansion during the learning session. Nodes denote members of Σ as seen in the homing sequence. Nodes representing actions that have been seen repetitively are marked as *cyclic* ("*c*"), and nodes for actions seen only once so far are marked *extendable* ("*x*"). Directed arcs between nodes denote orderings of actions that are consistent with the homing sequence. If an arc is used by an α-cycle, then the arc is labelled with α, along with any other cycle owners that use it. Arcs used by extendable nodes are labelled with x. We call these labels *arc owner labels*. Owner labels have associated with them a *size marking*, which is an integer $i \geq 2$ specifying the maximum size of the term that can be constructed by traversing the rest of the graph using that particular arc and owner. The size marking will enable the quick identification and extraction of maximum-sized terms.

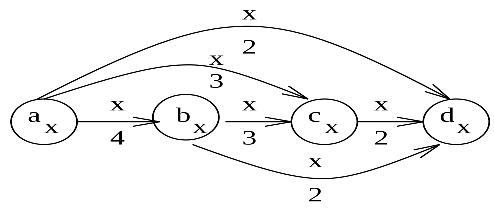
The digraph is extended whenever a new element is seen in the homing sequence, which can happen at any point in the learning session. When this occurs, a new node is created, arcs are directed to it from each extendable node, and extendable arc length labels are incremented. On the other hand, the digraph is refined when inconsistent pairs of elements are detected within the homing sequence. When the element ordering as it exists in the graph is violated by some combination of actions in the homing sequence, the inconsistent arcs in the digraph are removed, and arcs are relabelled if necessary. Should the digraph reduce into a set of unconnected nodes, then the degenerate expression $\alpha_1^\omega \parallel \cdots \parallel \alpha_k^\omega$ is the only expression currently feasible.

Letting s be the homing sequence for the session, the general steps for term consistency maintenance are as follows:

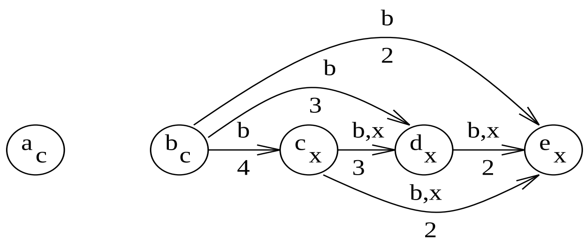
1. *Read homing subsequence $s' = \alpha_1 ... \alpha_k$. Set $s \leftarrow s.s'$.*
2. *For each $\alpha_i \in s' (1 \leq i \leq k)$:*
   (a) *If first time α_i encountered $\Rightarrow$ new action*
   – Add α_i as extended node to G. Add arcs from all other extendable nodes to α_i (theorem ??).
   (b) *If second time α_i encountered $\Rightarrow$ new cycle*
   – Convert α_i node to cyclic. Add α_i owner labels to all ext arcs that can be traversed from node α_i in G.
   (c) *If α_i is already cyclic $\Rightarrow$ refine cycles if possible*
   – Find most recent α_i-cycle in s having form "$\cdots \beta t \alpha_i \cdots$", where $\beta = \alpha_i$, $\alpha_i \notin t$, $|t| \geq 0$.
   – Find set of nodes E in G visited by all α_i cycles. Find intersection $F \leftarrow E \cap t$.
      i. *Update G by removing α_i as an owner of arcs refencing nodes not in F (corollary ??).*
      ii. *For every α-cycle in G such that $\alpha \notin F$ and $\alpha \neq \alpha_i$, remove α as an owner of arcs that reference node α_i (theorem ??).*

In the above, whenever graph arcs are created, altered, or deleted, size labels may need to be changed throughout the graph. In addition, when an arc is relabelled to the point where it has no owner label, it is deleted.

An example of digraph representation and refinement follows. Arc owner labels are placed above arcs, and size labels below arcs. (In this particular example, one size happens to be shared by all the owners of an arc; however, this is not always the case.) Given an initial homing subsequence *abcd*, the digraph constructed is:



Because no actions are repeated so far, all nodes and arcs are extendable. If the subtrace *eaab* is next read, then the total homing sequence is *abcdeaab*, and the graph becomes:



The arcs emanating from a were deleted, since by theorem ?? the repeated aa in the subsequence prevents any other element from being in a's cycle. The repeated b means that the largest b-cycle is now $bcde$.

3.1.2 *Synchronization set maintenance*

The largest A set consistent with the homing sequence is determined in linear time. Using the schema of theorem ??, each action in the homing sequence is identified as being either a member of A, or an element interleaved between members of A. A circular queue recording this information is maintained. The elements in this queue are either ordered synchronizing actions, or are sets of interleaving actions found between the synchronizing ones. The queue is initialized with homing sequence elements up until the time that the first cycle is detected. At that point, the elements in the queue are matched with subsequent actions in the homing sequence. If synchronization is occurring, the cyclic behaviour as described in theorem ?? will be seen. Elements in the queue are matched with homing sequence actions, are marked as being seen, and are moved to the back of the queue in anticipation of the next cycle. Synchronizing actions may be demoted to interleaving actions during this processing if their ordering violates theorem ??. If the A set in the queue reduces to empty, then A will never exist, and synchronization set maintenance will be skipped for the remainder of the session.

3.1.3 *Hypothesis construction*

The results from the previous two steps – the consistency graph and A set – are used to construct an $\mathcal{I}^\omega$ hypothesis. The form of this expression is important if the inference algorithm is to converge in the limit to a satisfactory target expression. Hypotheses are constructed that (i) use the maximum legal synchronization set A, (ii) include at least one maximum-sized cycle, (iii) are well-behaved, and (iv) use the entire set of actions from Σ seen thus far. A partial ordering of hypotheses expressions is used, which orders expressions by the inferred sizes of the set A and the sizes of candidate hypothesis terms. The ordering specifies that largest-sized candidate A and terms E_i are to be sought within a hypothesis, in order for it to admit correction by the teacher. A characteristic of $\mathcal{I}^\omega$ is that, for $\Sigma = \cup_{i=1}^{k}\{\alpha_i\}$, the expression $\alpha_1^\omega \parallel_A \cdots \parallel_A \alpha_k^\omega (A = \emptyset)$ will generate all possible ω-streams over Σ. This expression is the least expression in the partial ordering (all terms have a size of one, and $A = \emptyset$), and so it will be used as a degenerate result. On the other hand, the expression $(\alpha_1 ... \alpha_k)^\omega \parallel_A (\alpha_1 ... \alpha_k)^\omega$ with $A = \Sigma = \cup_i\{\alpha_i\}$ is the greatest expression in the ordering (term size $= |\Sigma|$).

Let G and A be argument copies of the digraph and synchronization set respectively, and let the hypothesis expression *HypExpr* be initialized to null. The hypothesis construction procedure *Hypothesis(HypExpr, G, A)* follows. The equivalence class expression of theorem ?? is constructed by the algorithm (the actual $\mathcal{I}^\omega$ expression is easily obtained from it).

1. If $A = \emptyset \Rightarrow$ Repeat until G is null:

(a) Find arc with highest size marking n.
(b) Traverse G using this initial arc and its owner label to extract the cycle t of size n.
(c) $G \leftarrow G - t$.
(d) $HypExpr \leftarrow HypExpr \parallel t^\omega$.

2. If $A = [\beta_1, ..., \beta_k]$ $(k \geq 1)$:

(a) Find β_1 in G. Let $\alpha_1, ..., \alpha_i$ be nodes preceding β_1 in G, and $\alpha_{i+1}, ..., \alpha_n$ be nodes following β_1.
(b) $G^* \leftarrow G - \{\alpha_{i+1}, ..., \alpha_n\}$.
(c) Apply step 1 to G^* to create $w = (s_1 \parallel^* \cdots \parallel^* s_n)$ (see theorem ??).
(d) $HypExpr \leftarrow HypExpr.w.\beta_1$
$G' \leftarrow G - G^*$
$A' \leftarrow [\beta_2, ..., \beta_k]$
(e) Recurse: Hypothesis(G', A').

A graph deconstruction function "$-$" is used above, where $G - X$ denotes the graph obtained by removing the nodes and arcs in the subgraph, term, or action set X from graph G. This involves recomputing arc size labels in the resulting graph.

3.2 Example inference

The following learning session infers an $\mathcal{I}^\omega$ formula from a homing sequence. The target expression is

$$(abcd)^\omega \parallel_A (ebfc)^\omega \parallel_A (bcgh)^\omega$$

with $A = \{b, c\}$. By theorem ??, the target equivalence class is (in abused notation) $((a \parallel e).b.(f).c.(d \parallel gh))^\omega$. A simplified form of this is $((a \parallel e).bfc.(d \parallel gh))^\omega$, in which f, although not intended to be a member of A, can be effectively treated as such. In the following inference, "Del arcs" refers to inconsistent arcs pruned from the graph. The equivalence class expression is included when it conveys useful information.

Step	*Homing*	*Del arcs*	*A*
1	$aebf$	–	$\{a, e, b, f\}$
2	$cghd$	–	$\{a, e, b, f, c, g, h, d\}$
3	$aebfcdgh$	gd, hd	$\{a, e, b, f, c\}$
4	$eabfc$	ae	$\{b, f, c\}$

Step	*Hypothesis*	*Equiv. class*
1	$(aebf)^\omega \parallel_A (aebf)^\omega$	–
2	$(aebfcghd)^\omega \parallel_A (aebfcghd)^\omega$	–
3	$(aebfcgh)^\omega \parallel_A (aebfcd)^\omega$	$(aebfc.(gh \parallel d))^\omega$
4	$(abfcgh)^\omega \parallel_A (ebfcd)^\omega$	$((a \parallel e).bfc.(gh \parallel d))^\omega$

After 4 steps, the inference algorithm converged to the desired equivalence class. Typically, when synchronization is to be used within an $\mathcal{I}^\omega$ expression, the nondeterminism usually seen with interleaving is constrained to a great extent, and convergence is rapid.

Consider the addition of one more homing sequence subtrace to the above session:

Step	*Homing*	*Del arcs*	*A*
5	cij	$ac, ec, bc, fc, cg, ch, cd$	$\emptyset$

Step	*Hypothesis*	*Equiv. class*
5	$(ebfgh)^\omega \parallel (ad)^\omega \parallel (ij)^\omega$	$-$

The repeated c element in the homing sequence suffix $...eabfccij$ destroys the synchronization set (as dictated by theorem **??**), and all hypotheses henceforth will have no synchronizing elements. Note how it is always possible to introduce new members from Σ at any arbitrary point in the inference.

3.3 Analysis

The use of maximal-sized cycles in hypotheses permits the algorithm to converge in the limit to the $\mathcal{I}^\omega$ equivalence class of the teachers target formula. If a maximal-sized cycle is a solution term, it is retained in further hypothesis until the end of the session. If it is erroneous, there exists a homing subsequence that can break it down, thereby permitting the teacher to supply corrections to hypotheses.

The algorithm has a worst-case time complexity of $O(nk^2)$ and space complexity $O(k^2)$ over $|\Sigma| = k$ and homing sequence size n. This polynomial time and space complexity arises from the polynomial number of arcs in the term digraph that must be processed (a maximum of $k(k-1)/2$ of them). Note that if the digraph model were discarded in favour of a naive representation in which sets of consistent terms were explicitly constructed, the algorithm would become exponential, since there are an exponential number of consistent terms derivable for interleaving behaviours.

An interesting feature of $\mathcal{I}^\omega$ is the ability to identify and quantify an *optimal learning strategy* for any $\mathcal{I}^\omega$ target formula. Ignoring synchronization, the expression $s^\omega \parallel t^\omega$ can be learned from the homing sequence sst. The cycling s in the trace distinguishes the s and t cycles. Of course, this does not imply that $\mathcal{I}^\omega$ is trivial to learn, since one cannot assume the teacher knows the structure of the target expression. Rather, the teacher knows a *desired behaviour*, and the inference algorithm synthesizes a formula or program that generates that behaviour.

4 Conclusion

Theoretical work in process algebra and other theories of concurrency shows that concurrent computations are inherently complicated to analyze. Applying machine learning techniques as given in this paper may offer new means for semi-automating the verification and synthesis of concurrent and distributed systems. An inference system that monitors a network bus as its homing sequence would be able to automatically reconstruct the network topology as denoted in some process algebraic target language.

As with other work in inductive inference [?], this algorithm benefits from the internal representation model used, that being the labelled acyclic digraph modelling consistent terms. As is often the case in computational learning theory, the language $\mathcal{I}^\omega$ is very restricted, and could be argued as being an unrealistic language. By disallowing choice and action repetition within terms, $\mathcal{I}^\omega$ expressions are much weaker than real process algebras. On the other hand, $\mathcal{I}^\omega$ is a non-trivial language. It inherits combinatorial properties due to its interleaving nature [?], and its cycling makes it non-enumerable as well. Despite its restrictions, the main value in using $\mathcal{I}^\omega$ is that it captures essential characteristics of cyclic interleaving in an experimental setting, which should prove useful as a platform for inferring more complex and practical concurrent languages. Since most languages that permit efficient inference from positive examples are typically restricted to a fair degree [?], any interleaving language will likely be restricted in order to admit NP-complete inference.

This paper is a variation of work in [?], which uses a similar digraph representation and hypotheses partial ordering for inferring non-cyclic interleaving expressions. These algorithms differ due to the different nature of the languages inferred. The non-cycling inference scheme uses fixed-sized positive examples, whereas the cycling case in this paper uses homing sequences. Graph refinement and hypothesis extraction are more involved in the cyclic language.

One can improve the information content of hypotheses by more carefully extracting terms from the digraph. For example, by using a polynomial-time Maxflow analysis [?], the maximum number of maximal-sized terms can be found. Arbitrarily picking a maximum term as was done here may exclude the generation of other combinations of terms, some combinations of which might yield more opportunity for correction by the teacher, thereby speeding up inference sessions.

The application of PAC principles to the inference of interleaving expressions is currently underway. Using PAC learning and ideas from [?], the automated generation of homing sequences by the algorithm can be performed. The inductive inference of ω-regular languages in [?] is closely related to $\mathcal{I}^\omega$, as their domain also involves a language with non-enumerable members. It is worth investigating whether their approach may lend insight into inductive techniques for $\mathcal{I}^\omega$ extended with nondeterministic choice ("+").

ACKNOWLEDGEMENTS

I thank the referees for their helpful comments, and Tom Jenkyns for interesting discussions. The author is supported by NSERC Operating Grant 0138467.

REFERENCES

[1] D. Angluin and C.H. Smith, 'Inductive Inference: Theory and Methods', *Computing Surveys*, **15**(3), 237–269, (1983).

[2] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice–Hall, 1985.

[3] O. Maler and Amir Pnueli, 'Learning ω-Regular Languages from Queries and Counter-Examples', in *Workshop on Analogical and Inductive Inference*, ed., K.P. Jantke, pp. 161–170. Springer-Verlag LNAI 397, (1989).

[4] A.J. Mayer and L.J. Stockmeyer, 'The complexity of word problems - this time with interleaving', Technical Report RJ 8947, IBM Research Div., San Jose, CA, (1992).

[5] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[6] E.-R. Olderog and C.A.R. Hoare, 'Specification-Oriented Semantics for Communicating Processes', *Acta Informatica*, **23**, 9–66, (1986).

[7] L. Pitt, 'Inductive Inference, DFAs, and Computational Complexity', in *Workshop on Analogical and Inductive Inference*, ed., K. P. Jantke, pp. 18–44. Springer-Verlag LNAI 397, (1989).

[8] R.L. Rivest and R.E. Schapire, 'Inference of Finite Automata Using Homing Sequences', in *Proc. 21st Symposium on the Theory of Computing*, pp. 411–420, Seattle, (1989). ACM Press.

[9] B.J. Ross. PAC inference of finitary synchronized interleaving, 1994. In progress.

[10] R. Sedgewick, *Algorithms*, Addison-Wesley, 1984.