The Inductive Inference of Finite Interleaving with Synchronization

Brian J. Ross
Brock University
Department of Computer Science
St. Catharines, Ontario, Canada L2S 3A1
phone: (416) 688-5550 ext. 4284
fax: (416) 688-2789
email: bross@sandbanks.cosc.brocku.ca

Technical Report CS-94-02
June 1994

Abstract

The exact identification of finite interleaving expressions over a fixed alphabet using positive example traces is investigated. A restricted language containing interleaving and synchronization is studied. This language is of interest because synchronized interleaving is a central concept in the semantics of concurrency. The inference technique exploits the fact that the interleaving operator is order-preserving amongst term elements within interleaved expressions. The inference algorithm maintains an acyclic digraph that retains information about terms that can possibly generate the example input behaviour. A number of hypothesis selection criteria are discussed, including the greedy selection of interleaving expressions with maximum-sized terms, expressions with the maximal number of maximum-sized terms, and expressions with the minimal number of terms. The greedy-selection algorithm has a worst-case complexity $O(nk^2)$ in the number of examples n and size of alphabet k, and a complexity $\leq O(k^4)$ if each example is required to convey a minimal amount of new information to the algorithm. Because the language domain is finite, it is also shown that the inference of finite interleaving over a fixed alphabet is PAC-learnable. The paper proves that the inductive inference of basic interleaving is tractable, and suggests that more complex languages used in the theory of concurrency might also be learnable.

Key words: inductive inference, exact identification, PAC learning, interleaving, shuffle, synchronization, process algebra, acyclic digraphs.

1 Introduction

Inductive inference is a machine learning paradigm which attempts to discover a formal language from information about members of the language [5]. A characteristic of inductive inference problems is that the inference of many formal languages is NP-complete, while the inference of others is tractable so long as the language under study is restricted and a suitable representation and inference technique is used [23, 22].

This paper is concerned with the inductive inference of a restricted interleaved language $\mathcal{I}$, which features synchronized interleaving behaviour. This language is interesting from a number of perspectives. Firstly, algebraic interleaving is of central importance in the theory of concurrency [21, 12, 20, 1, 19]. An interleaved expression $P \parallel Q$ denotes the concurrent execution of P and Q. Its computational behaviour is modelled by a *trace* or stream of atomic elements, in which P and Q nondeterministically contribute values into the trace. Secondly, the interleaving operator permits its exponentially-large domain space to be denoted with linear efficiency [18]. In addition, with respect to inductive inference, it is possible to derive a maximally efficient set of input examples for any $\mathcal{I}$ expression.

The main result of this paper is that, given a fixed alphabet, there exists a polynomial algorithm for exact identification of interleaved expressions from a set of positive example traces. The algorithm has a complexity $O(nk^2)$ over the size k of the alphabet and number n of examples. As with most machine learning techniques, the efficiency of the inference is fundamentally dependent upon the quality of examples given, which influences the number of examples needed. In addition, because the language domain itself is finite, the PAC learning of $\mathcal{I}$ expressions is possible.

Section 2 introduces the language studied in the paper. Section 3 presents exact identification algorithms using a variety of hypothesis selection criteria. The PAC learning of $\mathcal{I}$ is discussed in section 4. Conclusions, future work, and related work are discussed in section 5.

2 Language definition and properties

2.1 Definition

We define a restricted interleaving language $\mathcal{I}$. $\mathcal{I}$ expressions are called *shuffles* in combinatorial algebra, and have the same semantics [16]. An alphabet $\Sigma = \{a, b, c, ...\}$ is a finite multiset of atomic actions or elements. Using Σ, we define $\mathcal{I}$ as follows.

Definition 1 Abstract syntax of $\mathcal{I}$:

$$\mathcal{I} \quad ::= \quad \alpha_1.\alpha_2. \cdots \alpha_k.\mathbf{0} \quad | \quad E_1 \parallel_A E_2$$

where $k \geq 0$, $E_1, E_2 \in \mathcal{I}$, *and* $A \subseteq \Sigma$. □

$\mathcal{I}$ uses two basic operators: (i) the prefix operator ".", which contatenates actions sequentially; and (ii) the interleaving or shuffle operator " $\parallel_A$ ", whose semantics will be described shortly. For convenience, the expression α denotes $\alpha.\mathbf{0}$, and $\alpha_1\alpha_2\alpha_3 \cdots \alpha_k$ denotes

$\alpha_1.\alpha_2.\alpha_3.\cdots\alpha_k$. An $\mathcal{I}$ expression composed entirely of actions (such as $\alpha_1.\alpha_2.\ \cdots\ \alpha_k.\mathbf{0}$ above) is called a *term*. For example, in the expression $abc \parallel_A de \parallel_A f$, the three terms are abc, de, and f. The size of a term is the number of elements in it. The term $\mathbf{0}$ has size zero. 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. Elements can be freely repeated in an expression (hence Σ's definition as a multiset). To simplify discussions, it is assumed that all $\alpha \in \Sigma$ are always used in any given $\mathcal{I}$ expression.

A trace or stream is an $\mathcal{I}$ expression composed entirely of actions. If the prefix operator and empty action are dropped from trace expressions, a trace is simply a string of actions. Traces for $\mathcal{I}$ expressions are usually sets of strings, since one interleaving expressions may generate multiple strings. The following trace model is taken from a process algebra in [20], and defines the interleaving behaviour of $\parallel_A$. The semantic function $\mathcal{T}$ generates a trace for the argument expression.

Definition 2 Trace model for $\mathcal{I}$:

$$\begin{array}{ll}
\mathcal{T}(\mathbf{0}) & = \{\epsilon\} \\
\mathcal{T}(\alpha.E) & = \{\alpha t \mid t \in \mathcal{T}(E)\} \\
\mathcal{T}(E_1 \parallel_A E_2) & = \{t \mid \exists t_1 \in E_1, t2 \in E_2 : t \in (t_1 \parallel_A t_2)\} \\
 & \\
where: & \\
\epsilon \parallel_A \epsilon & = \{\epsilon\} \\
\alpha t \parallel_A \epsilon = \epsilon \parallel_A \alpha t & = \left\{ \begin{array}{ll} \emptyset & : if\ \ \alpha \in A \\ \alpha.(t \parallel_A \epsilon) & : if\ \ a \notin A \end{array} \right. \\
\alpha s \parallel_A \beta t = \beta t \parallel_A \alpha s & = \left\{ \begin{array}{ll} \alpha.(s \parallel_A t) & : if\ \ \alpha = \beta\ \ and\ \ \alpha \in A \\ \emptyset & : if\ \ \alpha \neq \beta\ \ and\ \ \alpha, \beta \in A \\ \alpha.(s \parallel_A \beta t) & : if\ \ \alpha \notin A\ \ and\ \ \beta \in A \\ \alpha.(s \parallel_A \beta t) \cup \beta.(\alpha s \parallel_A t) & : if\ \ \alpha \notin A\ \ and\ \ \beta \notin A \end{array} \right.
\end{array}$$

□

In the above, ϵ is the null string. Using this model (which is one particular model of many found in process algebraic theories of concurrency), one first sees that a given $\mathcal{I}$ expression can have many possible traces. The terms contribute their actions to a trace nondeterministically, in the sense that different terms can contribute their actions at any point in a particular trace, so long as they are not prefixed with a synchronizing element. Each term, however, contributes its constituent actions in the sequential order they reside within that term. This implies that $\parallel_A$ preserves the order of elements within single terms, but scrambles or shuffles the order of elements between terms themselves. An analogy is to consider splitting a deck of cards into k piles, placing each pile face down on a table, choosing a single card from the top of some randomly chosen pile, and then recording that card. This is modelled by $E = E_1 \parallel_\emptyset \cdots \parallel_\emptyset E_k$, and the order of cards recorded is simply one $t \in \mathcal{T}(E)$.

When the synchronizing component A is considered, elements from A within terms must coincide or synchronize within the trace. In concurrency theory, this models a synchronizing communication or handshake between two or more agents. For example, given the expression $E = abcd \parallel_A ecfg$, and letting $A = \{c\}$, then the subterms ab and e are nondeterministically contributed to the beginning of the trace $\mathcal{T}(E)$. The c action in each term will then wait and synchronize, after which d and fg are nondeterministically shuffled.

The trace model permits transformations such as the following to be applied to expressions:

$$\begin{aligned} P \parallel \mathbf{0} &= P \\ (P \parallel Q) \parallel R &= P \parallel (Q \parallel R) \\ P \parallel P &= P \end{aligned}$$

Such equivalences permit process algebras to be used as computational calculi of concurrent systems.

A few more definitions are now given. If $\mathcal{T}(E) = \alpha\mathcal{T}(E')$, then E' is a *derivative* of E. A k-trace is a trace of length k. A 2-trace is called a pair. Given $|\Sigma| = k$, the *trace universe* is the set of size $k!$ of all possible k-trace permutations of elements in Σ. The notation $\alpha \prec \beta$ means that α precedes β in some trace s. A *subtrace* of a k-trace t is a j-trace ($j \leq k$) whose elements are a subset of those in t, and share the same relative order. More precisely, s is a subtrace of t if

$$\forall \alpha, \beta : \alpha \prec \beta \ \ in \ \ s \ \Rightarrow \ \alpha \prec \beta \ \ in \ \ t$$

For example, bde is a subtrace of $abcdef$. We let ϵ be a subtrace of ϵ. Two traces s and t are *inconsistent* if there exists subtraces s' of s and t' of t such that s' and t' are permutations of each other. The terms $xayb$ and $xyba$ are inconsistent because of the inconsistent pairs ab and ba embedded in them. Two traces are consistent if they are not inconsistent.

To summarize some of the above, letting $\Sigma = \{a, b, c\}$, one possible $\mathcal{I}$ expression E is $a \parallel_\emptyset bc$, the trace universe is $\{abc, acb, bac, bca, cab, cba\}$, and $\mathcal{T}(E) = \{abc,\ bac, bca\}$.

2.2 Trace properties of $\mathcal{I}$

Some relevant trace properties of $\mathcal{I}$ are now given. The details of many proofs are omitted, since they use similar basic structural inductive arguments.

Definition 3 *An $\mathcal{I}$ expression E is* deadlocked *if $\mathcal{T}(E) = \emptyset$.* □

Because we assume that all Σ are used in $\mathcal{I}$ expressions, deadlock for any E or derivative of E implies that $\exists t \in \mathcal{T}(E) : |t| < |\Sigma|$. To simplify the paper, we restrict our inductive inference to non-deadlocked expressions, which brings the next definition.

Definition 4 *Given an $\mathcal{I}$ expression E over some Σ with $A \subseteq \Sigma$, we say E is* well-behaved *if it does not deadlock, ie. $\forall t \in \mathcal{T}(E) : |t| = |\Sigma|$.* □

The key to inducing $\mathcal{I}$ expressions from positive traces will be to determine terms inconsistent with those example traces. The following theorems establish some properties relevant to this idea.

Theorem 5 *Two traces s and t are inconsistent if they contain an inconsistent pair between them.*
Proof: *Direct consequence of the definition of inconsistency.* $\square$

Lemma 6 *If t is a subtrace of s, then αt is a subtrace of αs.*
Proof: *Since t is a subtrace of s, then $\alpha t \subseteq \alpha s$ In addition, α precedes all elements in both t and s, and thus the relative order is maintained.*[1] $\square$

Theorem 7 *Given a well-behaved expression $E = E_1 \parallel_A E_2$, then both E_i terms are subtraces of all $t \in \mathcal{T}(E)$.*
Proof: *Induction on size of terms E_1 and E_2. Let $k = |E_1| + |E_2|$. When $k = 0$ then $\mathcal{T}(\mathbf{0} \parallel_A \mathbf{0}) = \{\epsilon\}$ and both E_i are trivially subtraces of ϵ.*

Consider when $|E_1| + |E_2| = k + 1$. Let $E_1 \parallel_A E_2 = \alpha_1 \cdots \alpha_m \parallel_A \beta_1 \cdots \beta_n$, and $m + n = k + 1 \quad (m \geq 0, n \geq 0)$. If $m = 0$, then $E_1 = \mathbf{0}$ and $E_2 = \beta_1 \cdots \beta_n$ are both in $\mathcal{T}(E_1 \parallel_A E_2) = \{\beta_1 \cdots \beta_n\}$. Similarly for $n = 0$. If $m > 0$ and $n > 0$, then

$$\mathcal{T}(E_1 \parallel_A E_2) \;=\; \begin{cases} (i) & \alpha_1 \mathcal{T}(\alpha_2 \cdots \alpha_m \parallel_A \beta_1 \cdots \beta_n) \\ & \cup\, \beta_1 \mathcal{T}(\alpha_1 \cdots \alpha_m \parallel_A \beta_2 \cdots \beta_n) \quad : \alpha_1, \beta_1 \notin A \\ (ii) & \alpha_1 \mathcal{T}(\alpha_2 \cdots \alpha_m \parallel_A \beta_1 \cdots \beta_n) \quad : \alpha_1 \notin A,\ \beta \in A \\ (iii) & \beta_1 \mathcal{T}(\alpha_1 \cdots \alpha_m \parallel_A \beta_2 \cdots \beta_n) \quad : \alpha_1 \in A,\ \beta \notin A \\ (iv) & \alpha_1 \mathcal{T}(\alpha_2 \cdots \alpha_m \parallel_A \beta_2 \cdots \beta_n) \quad : \alpha_1, \beta_1 \in A,\ \alpha_1 = \beta_1 \end{cases}$$

In case (ii), both $\alpha_2 \cdots \alpha_m$ and $\beta_1 \cdots \beta_n$ are subtraces of $\mathcal{T}(\alpha_2 \cdots \alpha_m \parallel_A \beta_1 \cdots \beta_n) = \mathcal{T}(E')$ by the induction hypothesis. Then by lemma 6, since $\alpha_2 \cdots \alpha_m$ is a subtrace of all $t \in \mathcal{T}(E')$, then $\alpha_1 \alpha_2 \cdots \alpha_m$ is a subtrace of $\{\alpha_1 t : t \in \mathcal{T}(E')\}$.

Cases (i) and (iii) are similar to (ii).

For case (iv), both $\alpha_2 \cdots \alpha_m$ and $\beta_2 \cdots \beta_n$ are subtraces of $\mathcal{T}(\alpha_2 \cdots \alpha_m \parallel_A \beta_2 \cdots \beta_n) = \mathcal{T}(E'')$ by the induction hypothesis. So, by lemma 6, $\alpha_1 \alpha_2 \cdots \alpha_m$ is a subtrace of $\mathcal{T}(E'')$, as is $\alpha_1 \beta_2 \cdots \beta_n \ (\alpha_1 = \beta_1)$. $\square$

Corollary 8 . *Terms $E_i (1 \leq i \leq n)$ are subtraces of each $t \in \mathcal{T}(E_1 \parallel_A \cdots \parallel_A E_n)$.*
Proof: *Induction over the number of E_i. When associativity of $\parallel_A$ is considered, then theorem 7 can be applied to any number of terms.* $\square$

Corollary 9 *An $\mathcal{I}$ term cannot be used to generate a set $S \subseteq \mathcal{T}(E)$ if it contains a pair inconsistent with any $t \in S$.*
Proof: *If such a term contains a pair inconsistent with $t \in S$, then it cannot be a subtrace of t, and hence it cannot generate t by corollary 8.* $\square$

[1] When set notation is used with a term t, then t is being treated as the set of elements comprising t.

The following theorems are concerned with the structure of traces when synchronization is considered. The intention is the derivation of a useful equational description of traces that result from well-behaved $\mathcal{I}$ expressions.

Lemma 10 *Given* $s, t, A \subseteq \Sigma$ *where* $s \cap A = t \cap A = \emptyset$, *then* $\mathcal{T}(s \parallel_A t) = \mathcal{T}(s \parallel_\emptyset t)$.
Proof: *From definition of* $\mathcal{T}$. $\square$

Theorem 11 *Given* $E = s_1\alpha \parallel_A s_2\alpha$, *where* $A = \{\alpha\}$, $\alpha \notin s_1 \cup s_2$, *and* $|s_1|, |s_2| \geq 0$, *then* $\mathcal{T}(E) = \{s'\alpha : \ s' \in s_1 \parallel_\emptyset s_2\}$.
Proof: *Induction on* $k = |s_1| + |s_2|$. *When* $k = 0$, *then* $s_1 = s_2 = \epsilon$, *and* $\alpha \parallel_A \alpha = s'\alpha$ *where* $s' = \mathbf{0} \parallel_\emptyset \mathbf{0} = \{\epsilon\}$ *using lemma 10.*

For $|s_1| + |s_2| = k+1$, *let* $|s_1| = m$ *and* $|s_2| = k+1-m$, *where* $0 \leq m \leq k+1$. *Then*

$$\begin{array}{rcl}
\mathcal{T}(s_1\alpha \parallel_A s_2\alpha) & = & \mathcal{T}(\beta_1 \cdots \beta_m\alpha \parallel_A \beta_{m+1} \cdots \beta_{k+1-m}\alpha) \\
& = & \left\{ \begin{array}{lll}
(i) & \beta_1\mathcal{T}(\beta_2 \cdots \beta_m\alpha \parallel_A \beta_{m+1} \cdots \beta_{k+1-m}\alpha) & \\
 & \cup\ \beta_{m+1}\mathcal{T}(\beta_1 \cdots \beta_m\alpha \parallel_A \beta_{m+2} \cdots \beta_{k+1-m}\alpha) & : 0 < m < k+1 \\
(ii) & \beta_1\mathcal{T}(\beta_2 \cdots \beta_m\alpha \parallel_A \alpha) & : m = k+1 \\
(iii) & \beta_1\mathcal{T}(\alpha \parallel_A \beta_2 \cdots \beta_{k+1}\alpha) & : m = 0
\end{array} \right.
\end{array}$$

Expanding equation (i):

$$(\cup_i \beta_1\{u_i\alpha\}) \ \cup (\cup_j \beta_{m+1}\{v_j\alpha\})$$

where $\cup_i\{u_i\alpha\} = \mathcal{T}(\beta_2 \cdots \beta_m \parallel_\emptyset \beta_{m+1} \cdots \beta_{k+1-m})$ *and* $\cup_j\{v_j\alpha\} = \mathcal{T}(\beta_2 \cdots \beta_m \parallel_\emptyset \beta_{m+1} \cdots \beta_{k+1-m})$ *by the induction hypothesis. This is further refined:*

$$\begin{array}{rll}
 & [(\cup_i \beta_1\{u_i\}) \ \cup \ (\cup_j \beta_{m+1}\{v_j\})]\alpha & : move \ \ \alpha \\
= & [\beta_1\mathcal{T}(\beta_2 \cdots \beta_m \parallel_\emptyset \beta_{m+1} \cdots \beta_{k+1-m}) & \\
 & \cup \ \ \beta_{m+1}\mathcal{T}(\beta_1 \cdots \beta_m \parallel_\emptyset \beta_{m+2} \cdots \beta_{k+1-m})]\alpha & : subst. \ \mathcal{T} \\
= & \mathcal{T}(\beta_1\beta_2 \cdots \beta_m \parallel_\emptyset \beta_{m+1}\beta_{m+2} \cdots \beta_{k+1-m})\alpha & : definition \ \ \mathcal{T} \\
= & \cup_i\{s_i'\alpha\} & : s_i' \in s_1 \parallel_\emptyset s_2
\end{array}$$

Equations (ii) and (iii) are similar. $\square$

Corollary 12 *Given* $E = \alpha t_1 \parallel_A \alpha t_2$, *where* $A = \{\alpha\}$, $\alpha \notin t_1 \cup t_2$, *and* $|t_1|, |t_2| \geq 0$, *then* $\mathcal{T}(E) = \cup_i\{\alpha t_i'\}$ *where* $t_i' \in t_1 \parallel_\emptyset t_2$.
Proof: *Similar to theorem 11.* $\square$

Theorem 13 *Given* $E = s_1\alpha_1 s_2\alpha_2 \cdots s_{n-1}\alpha_{n-1}s_n \parallel_A t_1\beta_1 t_2\beta_2 \cdots t_{n-1}\beta_{n-1}t_n$, *where* $k = |\Sigma|$, $A = \cup_i\{\alpha_i, \beta_i\}$, *and* $\forall i : \alpha_i, \beta_i \notin s_i \cup t_i$. *Then*

$$\forall t \in \mathcal{T}(E) : |t| = k \ \ \Rightarrow \ \ \forall i : \alpha_i = \beta_i$$

Proof: *Structural induction on the size of* E, *and applying theorem 11. If* $\exists i : \alpha_i \neq \beta_i$, *then deadlock arises, and that term cannot contribute that* α_i *or* β_i *to the trace, nor any of the actions in the subterm following that* α_i *or* β_i. $\square$

Corollary 14 *Given E as in theorem 13, then $\mathcal{T}(E) = \cup\{w_1\alpha_1 w_2\alpha_2 \cdots w_{k-1}\alpha_{k-1}w_k\}$, where $w_i \in \mathcal{T}(s_i \parallel_\emptyset t_i)$.*
Proof: *Structural induction on E, using an argument similar to theorem 11.* □

The next theorem gives a generalized syntactic form of $\mathcal{I}$ expression having equivalent traces, which defines an equivalence class for $\mathcal{I}$ expressions.

Theorem 15 *Given $E = E_1 \parallel_A \cdots \parallel_A E_m$, where $E_i = s_{i,1}\alpha_1 \cdots s_{i,n-1}\alpha_{n-1}s_{i,n}$, $|s_{i,j}| \geq 0$, and $A = \cup_i\{\alpha_i\}$. Then $\mathcal{T}(E) = \cup\{w_1\alpha_1 w_2\alpha_2 \cdots w_{n-1}\alpha_{n-1}w_n\}$, where $w_i \in \mathcal{T}(s_{1,i} \parallel_\emptyset s_{2,i} \parallel_\emptyset \cdots \parallel_\emptyset s_{m,i})$.*
Proof: *Structural induction on number of terms in E, and applying corollary 14.* □

When using a non-empty synchronization set A, it is likely the case that there are multiple hypothesis expressions that exhibit equivalent behaviour. This implies that it is useful to identify the equivalence classes for hypotheses, in order to permit the teacher to ascertain whether the semantics of a generated hypothesis is satisfactory. For example, both $acd \parallel_A bce$ and $ace \parallel_A bcd$ with $A = \{c\}$ reduce to the general form $\{t_1.c.t_2 : t_1 \in \mathcal{T}(a \parallel_A b), t_2 \in \mathcal{T}(d \parallel_A e)\}$, and hence can be regarded as being semantically equivalent.

Definition 16 *A set of $\mathcal{I}$ expressions $\{E_1, ..., E_k\}$ over Σ belongs to the same equivalence class if $\mathcal{T}(E_1) = \mathcal{T}(E_2) = ... = \mathcal{T}(E_k)$.* □

We abuse notation and let the expression $(a \parallel_A b).c.(d \parallel_A e)$ denote the preceding equivalence class. Note that, if $A = \emptyset$, then theorem 15 is inapplicable, and there is a $1:1$ mapping between $\mathcal{I}$ expressions and their traces. One common source of expression equivalence is when elements are repeated, for example,

$$\mathcal{T}(aaa) \;=\; \mathcal{T}(a \parallel aa) \;=\; \mathcal{T}(a \parallel a \parallel a)$$

Thus, because the terms aaa and $a \parallel a \parallel a$ are identical with respect to traces, expressions may use them interchangeably.

2.3 Combinatorial properties

Some combinatorial measures of $\mathcal{I}$ expressions are given. Additional discussion is in section 4.

Theorem 17 *Let $k = |\Sigma|$, $\{\beta_1, ..., \beta_w\} \subseteq \Sigma$ be the set of repeating elements in Σ such that β_i is found r_i times, and $A = \emptyset$. The number of traces generated by $E_1 \parallel_\emptyset \cdots \parallel_\emptyset E_k$ where $|E_i| = n_i$ is given by the formula*

$$\frac{n!}{n_1!\, n_2! \,\cdots\, n_k!\, r_1!\, r_2! \,\cdots\, r_w!}$$

Proof: *There are $n!$ permutations of the n-traces of Σ. However, the order of elements in each E_i is retained in subtraces of each trace. Therefore the number of traces is reduced by $n_i!$ for each E_i (the number of permutations of those n_i elements in E_i). Likewise, each multiset of r_i repeated actions β_i reduces the total by a factor of $r_i!$.* □

Therefore, the cardinality of the trace universe for an $\mathcal{I}$ expression can exceed 2^n for $|\Sigma| = n$. We are working in a combinatorially large example space.

When non-empty A are considered, the number of traces generated is further reduced due to synchronization.

Theorem 18 *Let Σ have repeating elements $\{\beta_1, ..., \beta_w\}$ where β_i has frequency r_i, $E = T_1 \parallel_A \cdots \parallel_A T_m$ where $T_i = s_{i,1}\alpha_1 \cdots s_{i,n-1}\alpha_{n-1}s_{i,n}$ and $A = \{\alpha_1, ..., \alpha_{n-1}\}$. Then the number of traces generated by E is given by the formula*

$$(\prod_{i=1}^{m} \frac{B_i!}{C_{i,1}!\ C_{i,2}! \cdots\ C_{i,n}!})(r_1!\ r_2!\ \cdots\ r_w!)^{-1}$$

where $C_{i,j} = |s_{i,j}|$ and $B_i = \sum_{j=1}^{n} C_{i,j}$.
Proof: *Using the equivalence class syntax mentioned earlier, E can be written as:*

$$(E'_1).\alpha_1.(E'_2).\alpha_2.\cdots.(E'_{n-1}).\alpha_{n-1}.(E'_n)$$

where $E'_i = (s_{1,i} \parallel_{\emptyset} s_{2,i} \parallel_{\emptyset} \cdots \parallel_{\emptyset} s_{m,i})$. By theorem 17, E'_i generates

$$\frac{\sum_{j=1}^{n} |s_{i,j}|}{(|s_{1,i}|)! \cdots (|s_{m,i}|)!}$$

elements. The product of these terms models the combinating of all the traces from these intermediate interleaving expressions. The duplicate terms reduce the number of traces as in theorem 17. □

Therefore, the cardinality of a trace set for an expression is dependent upon the number of terms in the expression, the sizes of the terms within the expression, the number of elements in A and their location in the expression, and the number of duplicate elements in Σ. It would be convenient if we could compare the relative trace set sizes from a single scalar property of the expression, such as the size of the largest term, or the number of terms. If this could be done, then it might afford us with a convenient means for generating hypotheses during the learning procedure, since the cardinality of the trace set seems like an intuitively pleasing measure of hypothesis suitability. For example, an expression with a smaller trace set is less specific than one with a larger trace set; the degenerate expression composed of single-element terms is the least specific, as it includes all permuted traces from Σ. Unfortunately, in general, the size of the largest term nor the number of terms in an expression do not directly correlate with the trace set cardinality. For example, letting $A = \emptyset$, then by theorem 17 the expression (i) $abcd \parallel efg$ has 35 traces, (ii) $abcde \parallel f \parallel g$ has 42 traces, and (iii) $abcdef \parallel g$ has 7 traces. Therefore, the size of their largest terms, and the number of terms used in them, by themselves do not yield useful information about their trace set cardinalities.[2] This asymmetry, however, will not cause problems, since the cardinality of trace sets will not be the criteria for choosing hypotheses.

[2] On the other hand, it can be proven that, if the number of terms in two expressions are fixed, then the expression with the largest term will have a smaller trace set[14].

The number of $\mathcal{I}$ expressions that exist over a given Σ is also measurable. Disregarding duplicates and synchronization, counting the number of $\mathcal{I}$ expressions possible over a given Σ is equivalent to the partition problem, which is concerned with counting the number of partitions derivable for a given multiset. The number of ways of partitioning $|\Sigma| = n$ elements is given by the nth Bell number B_n [2, 9]. For example, if $\Sigma = \{a, b, c\}$ and $A = \emptyset$, then $\{abc,\ ab \parallel c,\ ac \parallel b,\ ba \parallel c,\ bc \parallel a,\ ca \parallel b,\ cb \parallel a,\ a \parallel b \parallel c\}$ are the $B_3 = 8$ distinct $\mathcal{I}$ expressions over Σ, and each partition of Σ corresponds to a $\mathcal{I}$ expression. For large n, the Bell number B_n approaches

$$e^{-1} \sum_{k \geq 0} \frac{k^n}{k!}$$

This is greater than 2^n (eg. set $k = 2$ and $n = 1000$). Section 4 contains more discussion about Bell numbers, where this measure will be important in deriving a probabilistic model of inference.

3 Exact identification from positive examples

3.1 The problem

This section will focus on the exact identification of an $\mathcal{I}$ expression for a fixed Σ from positive example traces. Given the definition of $\mathcal{I}$ in section 2, it is important to identify a problem specification in which computational learning is tractable. The most basic strategy is to enumerate the entire set of $\mathcal{I}$ expressions over a fixed Σ, and then discard expressions that are inconsistent with the example set. Using theorem 5, one could determine from each example E_i all the expressions inconsistent with E_i: any expression which produces a pair of elements inconsistent with the ordering in E_i cannot be a solution expression. This strategy will identify in the limit the $\mathcal{I}$ expressions that produce a given set of positive example traces. However, it is not efficient. As stated in section 2.3, the Bell number measurement of the $\mathcal{I}$ expression space is exponential over the size of $|\Sigma|$, and so there are too many expressions to enumerate.

A better approach is to enumerate the possible legal $\mathcal{I}$ terms consistent with the example set, and then construct a valid hypothesis with these terms. This is the general approach taken in this paper. The modelling of consistent terms is made tractable by choosing a *directed acyclic graph* or *DAG* representation of the space of consistent terms. Nodes will denote elements of Σ, and arcs denote consistent orderings of elements within terms. Traversing the DAG permits terms to be extracted. For example, the DAG

$$\begin{tikzcd}
a \arrow[r] \arrow[rr, bend left] \arrow[rrr, bend left] & b \arrow[rr, bend right] & c \arrow[r] & d
\end{tikzcd}$$

represents all the terms $\{abd, acd, ad, bd, cd, a, b, c, d\}$ (each node is trivially a single-element term). Using this DAG model, an exponential number of terms over $|\Sigma| = n$ can be denoted in polynomial space $O(n^2)$. In addition, from corollary 8, if we are given an example trace, then all terms of the hypothesis expression must be consistent with this example. Therefore, E_1 can be used to construct the initial enumeration of possible terms for the learning session. This is preferable to initially enumerating all the possible terms over Σ, since

$$\sum_{i=1}^{n} \left(\frac{n!}{i!} - \binom{n}{i} \right)$$

of these terms will be found to be inconsistent with E_1 anyway. Details of this DAG representation are in the next section.

Another important issue is to determine a suitable criteria for hypothesis selection. Firstly, we are only considering well-behaved expressions, and so deadlocked $\mathcal{I}$ expressions will be ignored. Secondly, the ranking of hypotheses will vary according to the goals of the particular inference setting. Different suitability orderings on expressions are possible, and they are based on syntactic complexity measurements such as the size of terms, the number of terms, and the cardinality of the synchronization set A. Initially, we will use a greedy approach for hypothesis selection, in which an expression is constructed that has at least one maximum-sized term as determinable from the DAG. Section 3.3 will discuss other possible strategies for hypothesis selection. The criteria for synchronization set determination uses the following partial ordering.

Definition 19 Partial Ordering $\supseteq$ over A: *Given two synchronization sets A_1 and A_2 over Σ, then $A_1 \supseteq A_2$ if A_2 is a subset of A_1. Otherwise, the ordering between A_1 and A_2 is undefined.* □

We therefore use standard subset ordering for choosing candidate synchronization sets for hypotheses, in which larger A's are preferred over smaller ones. In practice, the undefined case will never occur in the algorithm.

3.2 Greedy algorithm

The greedy algorithm performs the following to exactly identify an $\mathcal{I}$ expression from example traces S:

1. Record the $\mathcal{I}$ terms consistent with S.

2. Record the largest possible synchronization set A.

3. Build a hypothesis expression from the information collected in the previous two steps, such that it has at least one maximum-sized term.

Step 3 above characterizes the algorithm as "greedy", as the selection of a maximum-sized term is done quickly and directly from the DAG representation of consistent terms. The greedy algorithm is efficient, but at the expense of hypothesis quality (this will

1. Initialize: Read example E_1, and index duplicate elements.
   Initialize graph G using E_1.
   Set $A \leftarrow E_1$, label all $\alpha \in A$ with *yes*.

2. Loop: Construct a hypothesis from graph (fig 6), and present to teacher:
   if teacher satisfied or graph is degenerate (no arcs left) then quit.
   else: a) Read next counterexample E_i and index duplicates.
   b) $G \leftarrow$ ConsistentTerms(G, E_i)
   c) $A \leftarrow$ Synchronization(A, E_i)

   End Loop

Figure 1: Main loop of algorithm

Initialize(trace $\alpha_1 ... \alpha_k$):
1. Create nodes labelled $\alpha_1, ..., \alpha_k$.
2. For i=1,...,k-1:
   For j=i+1,...,k:
   (a) Add arc $\alpha_i \rightarrow \alpha_j$.
   (b) Label this arc with size value k-j+2.
3. Return initialized DAG.

Figure 2: DAG initialization

be remedied in section 3.3). Despite this weakness, the algorithm is well-suited for use within interactive dialogues with a teacher or oracle. The use of maximum-sized terms in an hypothesis means that it permits correction by counterexamples, which is important for effective interactive inference dialogues.

Figure 1 shows pseudocode for the main loop of the greedy algorithm. Specific details are found in the following sections.

3.2.1 Consist term modelling

Corollary 8 states that the terms of an expression E are subtraces of all $t \in \mathcal{T}(E)$. This gives a criterion for establishing possible terms for E. On the other hand, theorem 5 says that inconsistent pairs are all that need be checked to see if a term is inconsistent with a trace. These facts suggest a methodology for determining terms: start with a set of possible legal terms, and for each new example trace read, delete terms that contain inconsistent pairs.

To record valid terms, the algorithm maintains a DAG that models consistent terms. Nodes of the graph denote elements from Σ, and arcs between nodes denote the ordered sequence of those elements within consistent terms. To make hypothesis

ConsistentTerms(graph G, trace E):

1. Pairs $\leftarrow$ OrderedPairs(E).
2. For each pair $\alpha_i\beta_i$ in Pairs:

   If $\exists$ arc $\beta_i \rightarrow \alpha_i$ in G:

   (a) Delete arc $\beta_i \rightarrow \alpha_i$.

   (b) Resize arc size labels in subgraph leading to β_i.
3. Return refined G.

Figure 3: DAG refinement

OrderedPairs(trace $\alpha_1 ... \alpha_k$):

1. Pairs $\leftarrow \emptyset$.
2. For i=1,...,k-1:

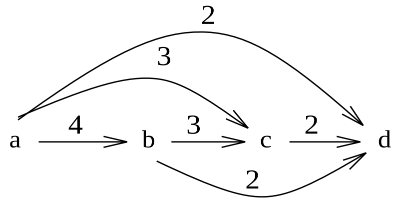
   For j=i+1,...,k:

   Pairs $\leftarrow$ Pairs $\cup \{\alpha_i \alpha_j\}$
3. Return Pairs.

Figure 4: Ordered pair extraction

construction efficient, each arc is labelled with an integer indicating the length of the maximum-sized path that can be traversed from that arc. A node with no arcs connected to it denotes a single element term.

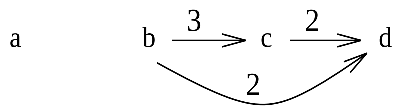
The graph is initialized after the first example E_1 is read (figures 1 and 2). The initial graph denotes all possible subtraces embedded within E_1. For example, if E_1 is *abcd*, then the initial graph is:



Various paths in this graph give possible subtraces within E_1. Note that the largest subtrace $a \rightarrow b \rightarrow c \rightarrow d$ is E_1 itself. Arc lengths are initialized at this time as well, which is done directly from the position of elements within E_1.

Duplicate elements in Σ are handled as follows. Σ is a fixed multiset, and the set of legal traces are simply permutations of Σ. Since a repeated element will have the same frequency throughout all traces, each repeated element is indexed with its relative position with respect to its other duplicates in the trace. For example, $E_i = abacdca$ is relabelled as $E_i' = a_1ba_2c_1dc_2a_3$. If all duplicates are indexed accordingly, then each element in the trace is uniquely identifiable, and all elements of Σ can be treated as being unique.

Figure 3 contains pseudocode for consistent term maintenance. When the next example E_2 is read in figure 1, all the inconsistent subtraces introduced by E_2 are determined by calling $ConsistentTerms$. Its ordered pairs are first obtained (figure 4), and then the pairs inconsistent with the ordering in the DAG are determined. Deleting an inconsistent arc means that the elements joined by that arc cannot reside in the same term, since the those elements have inconsistent orderings within the examples. For example, if $E_2 = bcda$, then its pairs are $\{bc, bd, ba, cd, ca, da\}$, and the pairs inconsistent with the above DAG are $\{ba, ca, da\}$. These arcs are then removed, and arc path lengths are altered appropriately:



Updating arc lengths involves a simple recursive check of paths leading to a node α whose outgoing arc is being deleted. If this deleted arc represented a maximum-length path for any paths that lead to α, then arcs in those paths have their lengths updated with the next largest path that exists for them in the DAG. This is directly determined from looking at path lengths in the DAG – no graph searching is necessary. Note that the deletion of a single arc affects a possibly exponential number of potential terms, due to the fact that there can be an exponential number of traversible paths leading to the two nodes whose connecting arc is removed. Also note that the whole graph might eventually disintegrate. This would indicate that the example set only permits a degenerate solution, eg. $a \parallel b \parallel c \parallel d$.

3.2.2 Synchronization set A inference

Theorem 15 gives a generalized form for $\mathcal{I}$ expressions and their traces. This equation suggests a simple way to determine legal A sets for well-behaved expressions. Firstly, since expressions are well-behaved, each $\alpha_i \in A$ must reside in each term of the hypothesis. So long as these terms are consistent with the example traces (corollary 8), they are permittable. Secondly, these $\alpha_i \in A$ must partition the non-synchronizing elements within the expression. For example, in

$$abcde \parallel_A wxcyz \quad \equiv \quad (ab \parallel_\emptyset wx).c.(de \parallel_\emptyset yz)$$

with $A = \{c\}$, then c partitions $\{a, b, w, x\}$ and $\{d, e, y, z\}$. Hence, as long as the elements from Σ partition nonsynchronizing elements consistently, and maintain their own order with respect to one another, they are candidate elements of A.

Figure 5 shows how the synchronization set is inferred. An ordered list of A actions is maintained. During the inference, each action in this list will be labelled with *yes* or *no*, meaning that it is either a member or nonmember of A. The list is initialized in figure 1 to be equivalent to the first example E_1 with all elements labelled as *yes*. For each subsequent example, the legality of each action in this sequence is checked by testing whether the partitions have changed since the previous example. If they have changed, then that element cannot be in A, and it is labelled *no* in the sequence.

Synchronization(ordered list A, trace E):

1. For each $\alpha_j \in A$ labelled *yes* ($j = 1, 2, ...$):
   (a) Construct set L_A for left-partition in A:
   $L_A = \{\alpha_1, ..., \alpha_{j-1}\}$
   (b) Construct set L_E for left-partition in E_i:
   $L_E = \{\beta_1, ..., \beta_n\} \quad if \quad E_i = \beta_1 ... \beta_n \alpha_j ... \beta_m$
   (c) If $L_A \neq L_E$ then label α_j in A with *no*.
2. Return refined A.

Figure 5: Synchronization set inference

Hypothesis(local graph G, local list A):

1. If no $\alpha_j \in A$ labelled *yes* then $H \leftarrow Interleave(Graph)$
   else: (create hypothesis with synchronization component)
   (a) Find first $\alpha_j \in A$ labelled *yes*.
   (b) Set $L \leftarrow \{\alpha_1, ..., \alpha_{j-1}\}$.
   (c) $G' \leftarrow G \cap L$.
   (d) If $G' \neq empty$ then $H_1 \leftarrow Interleave(G')$
   else $H_1 \leftarrow \epsilon$
   (e) Mark α_j to *no* in A (only affects local copy of A).
   (f) $G'' \leftarrow G - (L \cup \{\alpha_j\})$.
   (g) $H_2 \leftarrow Hypothesis(G'', A)$.
   (h) $H \leftarrow (H_1) \,.\, \alpha_j \,.\, (H_2)$
2. Return H.

Figure 6: Hypothesis construction

3.2.3 Hypothesis construction

The final step of the inference algorithm is to construct an hypothesis expression from the DAG and A. The key to the success of the inductive inference is to (i) follow a partial ordering of expressions so that all expressions have a reasonable chance of being inferred; and (ii) in the case of an interactive algorithm, present the teacher with hypotheses that permit correction in the form of counterexamples. The means for accomplishing both these goals is to always include at least *one* maximal-sized consistent term in each hypothesis. This permits hypotheses to follow a partial ordering over term sizes, and thus allows the inference to converge through all possible equivalence classes consistent with the examples.

Traversing the DAG to generate an interleaved expression that covers some subset of Σ is equivalent to finding a *node-disjoint path cover* of the graph. Hypothesis construction is detailed in figures 6 and 7. The procedure in figure 6 constructs an

Interleave(local graph G):
1. $H \leftarrow \epsilon$
2. Repeat until G is empty:
   (a) Select an arc with maximum path size label k, and traverse graph to extract term t_k of size k.
   (b) $G \leftarrow G - t_k$
   (c) Delete any indices from elements in t_k
   (d) $H \leftarrow$"$H \quad \|_A \quad t_k$"
3. Return H

Figure 7: Term extraction

equivalence-class expression. Note that, as discussed after theorem 15, when synchronization is being used, different $\mathcal{I}$ expressions may in fact be within the same equivalence class with respect to the traces they yield. This implies that a hypothesis, although not syntactically identical to a desired target expression, can be semantically equivalent to it. For a graph G, the notation $G' \leftarrow G \cap L$ means that the nodes of G that are in L, along with their associated arcs, are copied into a new graph G'. Similarly, $G \leftarrow G - L$ means that all the nodes and associated arcs in G that are in set L are deleted, and path size labels are suitably updated.

If the graph had been modelled in a naive manner, extracting maximal-sized terms would require exhaustive searching of the graph, which is unacceptably inefficient. By labelling arcs with maximum path sizes, however, maximal-sized terms are easily extracted: simply find the arc in the graph with the highest path length, and traverse the graph from that point to extract the term having that size. Once the term is obtained, the nodes and arcs referencing elements in that term are ignored when subsequent terms are extracted. When all the nodes have been used, a complete expression has been obtained.

The *Interleave* procedure in figure 7 creates a set of terms that covers its argument DAG. This is done via traversing the DAG, and collecting the actions of the nodes visited. Using a greedy strategy, a maximum-sized term is immediately extracted. This is done by starting the traversal along one of the arcs whose path sizes is maximal in the DAG. This initial path is arbitrarily selected. To obtain a legal maximum-sized term, if the currently traversed arc has size $k > 2$, then the next arc to be traversed must have size $k - 1$.

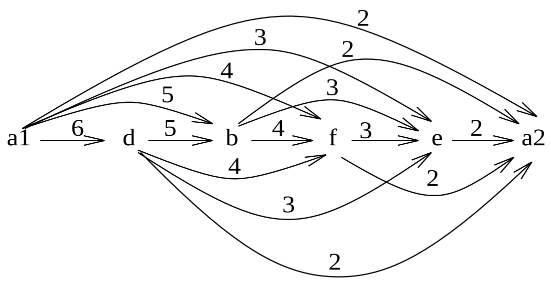
3.2.4 Example inference

An example session is as follows. Let $\Sigma = \{a, a, b, d, e, f\}$, and let the intended target expression be $aba \|_A dbe \|_A bf$ with $A = \{b\}$, and the initial example $E_1 = adbfea$. A possible learning session is the following ("X pairs" are inconsistent pairs):

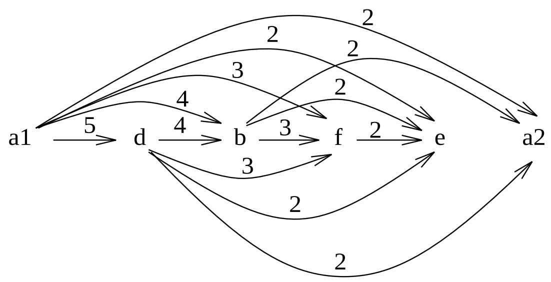
$\underline{step}$	$\underline{example}$	$\underline{X\ \ pairs}$	$\underline{Hypothesis(\equiv)}$	$\underline{A}$
1	$adbfea$	$-$	$adbfea$	$\{a_1, d, b, f, e, a_2\}$
2	$adbafe$	a_2f, a_2e	$adb.(fe \parallel a)$	$\{a_1, d, b\}$
3	$dabafe$	da_1	$(a \parallel d).b.(a \parallel fe)$	$\{b\}$
4	$dabaef$	ef	$(a \parallel d).b.(a \parallel e \parallel f)$	$\{b\}$
5	$(satisfied)$			

Here, the final hypothesis in step 4 is not syntactically identical to the target expression. However, using the equivalence class expression, it can be seen that its trace is identical to the target, and hence it is equivalent.

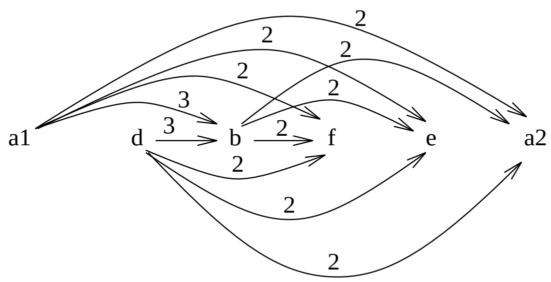
The effect of this session on the term graph is as follows. The initial graph built from the first example has the form:



When the second example is read, the $e \rightarrow c$ arc is deleted, and the graph reduces to:



Eventually, the final form of the graph at step 5 is:



3.2.5 Algorithm analysis

Some algorithm properties are now given.

Theorem 20 Correctness of term modelling. *The terms denoted by the DAG are always consistent with the example set S.*
Proof: *As stated in corollary 8, all the terms E_i for any $\mathcal{I}$ expression $E = E_1 \parallel \cdots \parallel E_j$ $(j \leq |\Sigma|)$ are subtraces of each $t \in \mathcal{T}(\mathcal{E})$. Consistent term maintenance in the algorithm ensures the terms denoted within the DAG model to adhere to this condition.*

Given the first example $t_1 \in \mathcal{T}(E)$, all E_i are initially subtraces of t_1. The DAG initialized in figure 2 denotes all these terms. The DAG created is maximally connected to denote all the subterms embedded in E_1. There can be no term in any permissible hypothesis that is not denoted within this DAG; if this weren't the case, it would not be a subtrace of E_1.

For each subsequent example $t_i (i \geq 2)$ processed in figure 3, if an inconsistent pair is found, it is deleted from the graph. Consistent terms (pairs) in the DAG are left alone. In addition, no arc is ever added to the DAG after its initialization. Hence the graph is always consistent with the example set seen thus far. □

Theorem 21 Correctness of synchronization set inference*: The synchronization set A inferred by the algorithm is correct.*
Proof: *The procedure in figure 5 attempts to determine the largest synchronization set consistent with the example set. Recall from theorem 15 the schema $w_1 \alpha_1 w_2 \alpha_2 \cdots w_{n-1} \alpha_{n-1} w_n$, where $w_i \in \mathcal{T}(s_{1,i} \parallel_{\emptyset} s_{2,i} \parallel_{\emptyset} \cdots \parallel_{\emptyset} s_{m,i})$. Note that each α_i in the schema must partition the elements on each side of it. The creation of partitions in figure 5 is an efficient way of determining the α_i of theorem 15. If such a partition cannot be constructed for some α_i, then it can no longer be a synchronizing element. Otherwise, the largest ordered set of elements that create such a partition is recorded for the example set so far.* □

Theorem 22 *The algorithm permits convergence.*
Proof: *Induction over the number of terms in a hypotheses. The greedy term construction is assured of selecting at least one term from the DAG having maximum size. If this maximum-sized term is not a solution term of the target expression, there must exist an inconsistent pair which prohibits its inclusion in the solution expression, and hence a counterexample exists for it. Otherwise, if it is a solution term, then by the induction hypothesis the greedy algorithm has constructed solution terms for the remainder of the expression. In addition, the set of synchronizing elements is initialized to be E_1, and iteratively refines towards the empty set if no synchronization occurs. Hence the algorithm will converge to any $\mathcal{I}$ expression consistent with the examples given.* □

Theorem 23 *The complexity is $O(nk^2)$ for n examples and $k = |\Sigma|$.*
Proof: *With respect to graph and synchronization set maintenance in figure 1, the extraction of pairs in steps 1 (initializing graph) and 2(b) require the retrieval of $k!/(2!(k-2)!) = (k^2-k)/2$ pairs. In step 2(c), the maximum number of arcs that can be deleted or updated in the graph is likewise $(k^2-k)/2$ arcs. Maintaining the synchronization set*

in 2(d) is $O(k)$. Hypothesis construction in figures 6 and 7 is $O(k)$, as maximal terms can be directly extracted using path length information on arcs.

All the above steps are sequentially executed, giving a complexity of $O(k^2)$ per example. An overall complexity for n examples is therefore $O(nk^2)$. □

Theorem 23 confirms that the algorithm is indeed practical, since it is easily illustrated that naive approaches to solving the problem are very inefficient. For example, disregarding synchronization, consider a set S of different $\mathcal{I}$ terms and an alphabet Σ. Then determining the minimal number of terms that covers Σ is NP-complete. The Minimal Cover problem transforms to it when the terms are considered to be sets [8]. In addition, a naive representation of the enumerated terms uses an exponential amount of space, as there are

$$(\sum_{i=1}^{k} \frac{k!}{i!}) > 2^k$$

terms constructable for $|\Sigma| = k$. Of course, the algorithm's DAG model, as well as the restricted nature of $\mathcal{I}$, contributes to its polynomial time and space complexity.

Theorem 23 implies that the complexity might grow exponential if n is exponential. For example, the user might give the same example repeatedly. A more realistic measure is to consider learning situations in which each example given is required to be useful, or in other words, yields at least one inconsistent pair with respect to the graph. Should one arc be deleted per example, there is a ceiling of $(k^2 - k)/2$ examples required until the DAG degenerates to an arc-less graph. This gives a worst-case complexity of $O(k^2k^2) = O(k^4)$ for "minimally productive" learning sessions.

An interesting idea is to identify an optimal learning strategy, as the following theorem discusses.

Theorem 24 *Any $\mathcal{I}$ expression can be learned using a maximum of 2 examples.*
Proof: *Firstly, the expression*

$$\alpha_1...\alpha_m \parallel_\emptyset \beta_1...\beta_n \parallel_\emptyset \cdots \parallel_\emptyset \gamma_1...\gamma_p$$

can be learned with the two examples

$$\begin{aligned} E_1 &= \alpha_1...\alpha_m\beta_1...\beta_n...\gamma_1...\gamma_p \equiv \overline{\alpha}\overline{\beta}...\overline{\gamma} \\ E_2 &= \gamma_1...\gamma_p...\beta_1...\beta_n\alpha_1...\alpha_m \equiv \overline{\gamma}...\overline{\beta}\overline{\alpha} \end{aligned}$$

The two examples permute the $\overline{\alpha}$, $\overline{\beta}$,..., $\overline{\gamma}$ subtraces with respect to one another, while maintaining the order within each of them. Recalling that any expression E can be rewritten into a generic equivalence class using theorem 15, in which

$$\mathcal{T}(E) = \cup\{w_1\alpha_1w_2\alpha_2...w_{n-1}\alpha_{n-1}w_n\}$$

where $w_i = s_{1,i} \parallel_\emptyset s_{2,i} \parallel_\emptyset \cdots \parallel_\emptyset s_{m,i}$. Each of these w_i can be learned using 2 examples: $E_{i,1} = s_{1,i}s_{2,i}...s_{m,i}$ and $E_{i,2} = s_{m,i}...s_{2,i}s_{1,i}$. The whole expression can be learned in 2 examples: $E_1 = E_{1,1}\alpha_1E_{2,1}\alpha_2...\alpha_{k-1}E_{n,1}$ and $E_2 = E_{1,2}\alpha_1E_{2,2}\alpha_2...\alpha_{k-1}E_{n,2}$. □

A result of theorem 24 is that minimally productive learning sessions have a polynomial complexity anywhere between $O(k^2)$ to $O(k^4)$, depending on the quality of examples given. However, it is unreasonable to expect a teacher to know an optimal learning strategy (although it might be useful to know one in many machine learning instances). Theorem 24 presumes a knowledge of the syntax of the target expression, and it is unreasonable to assume that a teacher knows that. Rather, the teacher might only know the behaviour of the expression, and wants to use inductive inference to synthesize an $\mathcal{I}$ expression adhering to that behaviour. This is particularly relevant to $\mathcal{I}$ expressions, as many expressions can produce the same trace behaviour, and the inference can in essence generate one valid expression from amongst the many that satisfy a user's specifications.

3.3 Other strategies of hypothesis selection

The greedy algorithm discussed above has the advantage of fast hypothesis construction (linear in $|\Sigma|$), while at the same time producing hypotheses which permit counterexamples, and therefore convergence to any intended target expression. Greedy selection arbitrarily selects one maximum term from amongst potentially many. This strategy is acceptable for an interactive learning setting, since each hypothesis produced can be refuted as necessary. One effect of this selection scheme, however, is that the selection of an arbitrary term can drastically affect the rest of the terms extracted in the remaining uncovered subgraph: selecting one particular term of size k may preclude the selection of n other terms of the same size. For example, in

$$\begin{tikzcd} a \arrow[r, "3"] \arrow[dr, "3"] & b \arrow[r, "2"] & c \\ d \arrow[r, "3"] & e \arrow[r, "2"] & f \end{tikzcd}$$

selecting the term aef means only 1 term of size 3 is possible, whereas selecting abc permits def to be chosen as well.

Rather than selecting a hypothesis based upon whether there exists at least one maximum-sized term within it, a better criteria is to select one that has the maximum number of maximum-sized terms. There are a number of advantages to this. Firstly, expressions with larger-sized terms *generally* have smaller trace universes than those with smaller-sized terms (although this is not strictly true, as discussed in section 2.3). Secondly, a partial ordering based on the maximum number of maximum-sized terms is more discriminating than the ordering used by the greedy algorithm, which has only $|\Sigma|$ equivalence classes. Lastly, and most importantly, a hypotheses having more maximum-sized terms permits richer counterexamples than those with fewer such terms: each maximum-sized term in the expression permits refutation in the next counterexample. In other words, generating as many maximum-sized terms as possible conveys more useful information to the teacher.

Finding the maximum number of maximum-sized disjoint terms is equivalent to finding the maximum number of disjoint paths having a maximum size of k. The means

MaxFlowInterleave(graph G):

1. $H \leftarrow \epsilon$
   $k \leftarrow$ maximum path size label in G
2. If G is empty then return H
   Else: (apply maxflow to term selection)
   (a) Prepare G for MaxFlow: Add a source node by connecting arcs from it to all G nodes having outgoing arcs of size k. Add a sink node by connecting arcs from all sink nodes in G to it. Set capacity of all arcs to 1, and flow to 0.
   (b) $MaxPaths \leftarrow MaxFlow(G)$
   (c) $H \leftarrow$ "$H \ \|_A \ e_1 \|_a \cdots \|_a e_n$" ($e_i \in$ MaxPaths)
   (d) Delete from G the source, sink, and nodes & arcs referencing values in $MaxPaths$, and recurse on $MaxFlowInterleave$.

Figure 8: Term extraction

for doing this is to refine DAG traversal by introducing a "maxflow" analysis into the graph traversal process [7, 15]. Maxflow is a well-known network flow algorithm that attempts to find the maximum output of a network or graph. The routine in figure 8 replaces the *Interleave* routine in figure 7. Maxflow is applied iteratively to find the maximum number of disjoint paths having the current maximal size in the DAG being processed. Single source and sink nodes must be attached to the DAG at each stage. The source is attached by running arcs from it to each DAG node that has an outgoing arc with a maximum size value. The sink is attached by running arcs from each terminating arc to it. Next, all the arcs in the graph are labelled with a *capacity* value of 1 and a *flow* value of 0. The standard maxflow procedure can now be applied to the graph to determine the maximum number of maximum disjoint paths. When these are known, their nodes and arcs are removed from the graph, and the algorithm is applied recursively to the remaining uncovered subgraph. There are many efficient implementations of maxflow less than $O(k^2)$, so adding this to hypothesis extraction does not affect the overall efficiency of the inference algorithm.

Another possible hypothesis selection strategy is to construct a hypothesis with the minimal number of terms. For example $a \parallel bcd$ has fewer terms than $a \parallel b \parallel cd$, and could therefore be said to be simpler. The minimal number of terms extractable from the DAG is determinable by using a polynomial minimal path cover algorithm for selecting paths from the DAG. For example, the one in [6] is $O(k^3)$ for a DAG with k nodes. However, there are some disadvantages to this hypothesis ordering scheme. Firstly, unlike the previous selection schemes that generated maximum-sized term(s), hypotheses produced in the minimal-term scheme do not always permit refutation by counterexamples. Although the terms of such an expression may be consistent with the intended target behaviour, there is no means for the teacher to provide a useful counterexample to orient the inference towards another candidate expression. Hence this strategy is better applied in a batch setting, rather than an interactive one. Secondly, disregarding synchronization, the use of the number of terms as a criterion for hypothesis selection means that, as in the greedy selection case, there are only $|\Sigma|$ equivalence classes possible, which is a very coarse measure of hypothesis suitability.

4 PAC learning of $\mathcal{I}$

Because the universe of possible $\mathcal{I}$ expressions over a given Σ is finite, it is possible to derive a PAC model of learning [25, 10]. Some definitions from [10] are first given. See these sources for a more detailed study.

Definition 25 *Consider the universe U of hypotheses derivable in some domain D, and the set of examples S of some target concept in D. The* version space *of S with respect to E is subset of hypotheses in U that are consistent with all the examples in S.* □

Definition 26 *Consider the selection of examples from some example space that conforms to some probabilistic distribution. The* error *of a hypothesis with respect to a target concept relative to this distribution is the probability that the hypothesis and target concept disagree.* □

Definition 27 *Given a hypothesis space H, a target concept E, a sequence of examples S of E, and an* error tolerance *ϵ, where $0 \leq \epsilon \leq 1$, the version space of S is* ϵ-exhausted *if it does not contain any hypothesis that has error more than ϵ with respect to E.* □

Theorem 28 *Let H be a finite hypothesis space. Let Q be a sequence of m independent random examples of some target concept E in H drawn under some probability distribution on the example space. Then for any δ, $0 < \delta < 1$, if*

$$m \geq \epsilon^{-1}(ln(1/\delta) + ln|H|)$$

then the version space of Q is ϵ-exhausted with probability at least $1 - \delta$.
Proof: *See [25, 10].* □

We can now derive the PAC model for $\mathcal{I}$ inference. Since synchronization reduces the size of the version spaces for hypotheses, the PAC model of expressions involving synchronization will be a combinatorial simplification of the case without it. As discussed in section 2.3, the number $\mathcal{I}$ expressions derivable for a fixed Σ of size n is equivalent to the number of partitions of the n elements of Σ. This is given by the nth Bell number β_n [2, 9], where

$$B_0 = B_1 = 1$$
$$B_n = \sum_{k=0}^{n} \binom{n}{k} B_{n-1}$$

This simple recursive definition of Bell numbers is difficult to use in theorem 28. Fortunately, the following will help.

Theorem 29

$$\lim_{n \to \infty} B_n = m(n)^n e^{m(n)-n-1/2} / \sqrt{ln(n)}$$

where $m(n) ln(m(n)) = n - 1/2$*, with a relative error of* $O(log\ log\ n / log\ n)^3$.
Proof: *A derivable approximation for* B_n *is*

$$B_n \approx e^{-1} \sum_{k \geq 0} \frac{k^n}{k!}$$

The result follows by solving for B_n *(see [9] for details).* □

We can now derive a PAC result for $\mathcal{I}$ without synchronization.

Theorem 30 *Consider the universe* H *of* $\mathcal{I}$ *expressions (without synchronization) for some fixed* Σ*. Let* Q *be a sequence of* $m \geq 1$ *independent random traces from the trace of a target* $\mathcal{I}$ *expression* E *drawn under some probability distribution on the example space. Then, for large* $n = |\Sigma|$*, and for any* $0 < \epsilon < 1$*, the version space for* Q *will be* ϵ*-exhausted with probability* $1 - \delta$ *after approximately*

$$\epsilon^{-1}(ln(1/\delta) + n\ ln(m(n)) + m(n) - n - 1/2 - ln(\sqrt{ln(n)}))$$

independent random examples using the given probability distribution, where $m(n) ln(m(n)) = n - 1/2$.
Proof: *Use the limit defined in theorem 29 within theorem 28, and simplify. The final expression is obviously polynomial in* n. □

5 Conclusion

This paper has shown that the exact identification and PAC learning of interleaving expressions with synchronization over a fixed alphabet is tractable. The nature of concurrency is such that combinatorial explosions readily occur in many analyses of concurrent computations. This paper suggests that machine learning techniques might aid in the synthesis and testing of concurrent software. For example, this paper's inference algorithm could be used to read the stream of communications resident on an network line, and then automatically ascertain the $\mathcal{I}$ configuration of client systems communicating on it.

As is often the case in inductive inference experiments, the tractability of the algorithm is due to the representation technique used. Our use of a DAG with path-length labels made the algorithm polynomial. This DAG representation is related to dependence graphs found in the theory of traces, in which the shuffle operator is an essential concept [1]. Similar graph models of concurrent systems are used in process algebras [20, 19] and Petri nets [21]. There are a number of ways in which the inference

[3] ln is the natural log, and log is the common log.

algorithm could be made faster than $O(nk^2)$. The number of pairs to be processed in figure 3 can be reduced, for example, by ignoring pairs that refer to solitary nodes (ie. single-element terms), and ignoring pairs that begin with an element that refers to a node without outgoing arcs.

This work began as an investigation whether inductive inference techniques could be applied to a subset of a process algebraic language such as CCS or CSP, or a Petri net system. Although the $\mathcal{I}$ language used in the paper is admittedly a very constrained subset of these other languages, it does incorporate the essential characteristics of interleaving with synchronization, and thus forms a basis from which to investigate more complex systems. An important concept that should be considered in concert with interleaving is deterministic and/or nondeterministic choice ("+"). Adding such an operator, however, would make the system as powerful as a regular language, and regular languages cannot be tractably identified from positive examples alone [23]. It is safe to assume that, unless interleaved languages are restricted in some fashion, their inference will often be NP-complete. Hence any interleaving language must be enhanced with caution.

A language variant of $\mathcal{I}$ whose inference has been shown to be tractable is perpetual cyclic interleaving expressions with synchronization [24]. The cyclic term $(abc)^\omega$ generates the infinite trace $abcabcabca$.... The $\mathcal{I}^*$ expression

$$(abcd)^\omega \parallel_c (ecfg)^\omega$$

denotes an interleaving expression in which the terms cyclicly contribute to the trace while synchronizing with c. The inference of $\mathcal{I}^*$ expressions uses a DAG model similar to this paper. However, a complication with $\mathcal{I}^*$ is the fact that examples take the form of an indefinite-size homing sequence, in which new actions can occur at any time in the inference. Σ is defined by all the elements seen up to one particular moment in the session, which can grow in following input subsequences. The overall complexity of this inference scenario is $O(nk^2)$ for $k = |\Sigma|$ and homing sequence length n.

Other hypothesis selection criteria are also worth considering in future extensions of this work. For example, can a minimum-term hypothesis with maximum-sized terms be found efficiently? Or can a maximum number of maximum-sized terms be found, in which the non-maximum terms are minimized in the expression? Or, ideally, can the hypothesis with the smallest trace set be found? These are variations of path-cover problems.

Inductive inference using equivalence queries was first suggested in [4], in which a target DFA is inferred from counterexamples in response to intermediate inferred machines. [17] extends this technique to ω-regular languages – those with infinite members. This has relationships with process-theoretic computation theory, since processes often generate indeterminately large computational sequences. Other restricted languages used for inference have included k-reversible regular languages [3], 0-reversible commutative regular languages [11], and k-bounded regular languages [13].

Acknowledgements: The author thanks Tom Jenkyns for his interesting analyses of some combinatorial issues. The author is supported by NSERC Operating Grant 0138467.

References

[1] I.K. Aalbersberg and G. Rozenberg. Theory of Traces. *Theoretical Computer Science*, 60:1–82, 1988.

[2] G.E. Andrews. *The Theory of Partitions*. Addison-Wesley, Massachusetts, 1976.

[3] D. Angluin. Inference of Reversible Languages. *Journal of the ACM*, 29(3):741–765, July 1982.

[4] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1987.

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

[6] F.T. Boesch and J.F. Gimpel. Covering the Points of a Digraph with Point-Disjoint Paths and Its Application to Code Optimization. *Journal of the ACM*, 24(2):192–198, April 1977.

[7] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Wadsworth and Brooks/Cole, Monterey, CA, second edition, 1986.

[8] M.R. Garey and D.S. Johnson. *Computers and Intractability*. W.H. Freeman, New York, 1979.

[9] R.L. Graham, D.E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison Wesley, New York, 1989.

[10] D. Haussler. Quantifying Inductive Bias: AI Learning Algorithms and Valiant's Learning Framework. *Artificial Intelligence*, 36:177–221, 1988.

[11] D. Helmbold, R. Sloan, and M.K. Warmuth. Learning Submodules and Reversible Commutative Regular Languages. Technical report, Department of Computer and Information Sciences, University of California at Santa Cruz, 1989.

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

[13] O.H. Ibarra and T. Jiang. Learning regular languages from counterexamples. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, San Mateo, CA, 1988. Morgan Kaufmann.

[14] T. Jenkyns, 1993. Personal communication.

[15] R. Johnsonbaugh. *Discrete Mathematics*. Macmillan, New York, third edition, 1990.

[16] M. Lothaire. *Combinatorics on Words*. Addison-Wesley, Massachusetts, 1983.

[17] O. Maler and Amir Pnueli. Learning ω-Regular Languages from Queries and Counter-Examples. In K.P. Jantke, editor, *Workshop on Analogical and Inductive Inference*, pages 161–170. Springer-Verlag LNAI 397, 1989.

[18] 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.

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

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

[21] J.L. Peterson. Petri Nets. *Computing Surveys*, 9(3), September 1977.

[22] L. Pitt. Inductive Inference, DFAs, and Computational Complexity. In K. P. Jantke, editor, *Workshop on Analogical and Inductive Inference*, pages 18–44. Springer-Verlag LNAI 397, 1989.

[23] L. Pitt and L.G. Valiant. Computational Limitations on Learning from Examples. *Journal of the ACM*, 35(4):965–984, October 1988.

[24] B.J. Ross. The Inductive Inference of Cyclic Synchronized Interleaving. In *Proceedings of the 11th European Conference on Artificial Intelligence*, Amsterdam, 1994. John Wiley and Sons.

[25] L.G. Valiant. A Theory of the Learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.