Modelling Asynchrony with a Synchronous Model

R. P. Kurshan[1], M. Merritt[1], A. Orda[2] and S. R. Sachs[3]

[1] AT&T Bell Labs, Murray Hill, NJ 07974, USA
[2] Dept. of Elec. Eng., Technion, Haifa 32000, Israel
[3] Dept. of Elec. Eng. and CS, U. C. Berkeley, Berkeley, CA 94720, USA

Abstract. The I/O Automaton paradigm of Lynch and Tuttle models asynchrony through an interleaving parallel composition and generalizes more common interleaving models based upon message-passing, such as Hoare's CSP. It is not generally recognized that such interleaving models in fact can be viewed as a special cases of *synchronous* parallel composition, in which components all move in lock-step. Let $\mathcal{A}$ be any set of finite-state I/O Automata drawing actions from a fixed finite set containing a subset Δ. In this article we establish a translation $T : \mathcal{A} \to \mathcal{P}$ to a class of ω-automata $\mathcal{P}$ closed under a synchronous parallel composition, for which T is monotonic with respect to implementation relative to Δ, and linear with respect to composition. Thus, for $A^1, \ldots, A^m, B^1, \ldots, B^n \in \mathcal{A}$ and $A = A^1 || \cdots || A^m$, $B = B^1 || \cdots || B^n$, if Δ is the set of actions common to both A and B, then A implements B (in the sense of I/O Automata) if and only if the ω-automaton language containment $\mathcal{L}(T(A^1) \otimes \cdots \otimes T(A^m)) \subset \mathcal{L}(T(B^1) \otimes \cdots \otimes T(B^n))$ obtains, where $||$ denotes the interleaving parallel composition on $\mathcal{A}$ and $\otimes$ denotes the synchronous parallel composition on $\mathcal{P}$. For the class $\mathcal{P}$, we use the L-process model of ω-automata. This result enables one to verify systems specified by I/O Automata through model-checkers such as COSPAN or SMV, that operate on models with synchronous parallel composition. The translation technique generalizes to other interleaving models, although in each case, the translation map must match the specific model. Proofs have been eliminated on account of space limitations. A full version (with all proofs) is available upon request.

1 Introduction

The I/O Automaton paradigm is used to model discrete event systems consisting of concurrently operating components [9, 10]. Each system component is modelled as an "I/O Automaton", which is an action-labelled transition structure. An Automaton's actions are classified as "input", "output" or "internal". An Automaton generates output and internal actions autonomously, and transmits output actions instantaneously to all other Automata having the same action designated as an input. All such components synchronize on each given output action, and take a step simultaneously with the output step. Only one output action or internal action may occur at any time. This restriction defines

the "interleaved" executions of component Automata, used to model asynchrony. I/O Automata can be composed to yield another I/O Automaton.

An *execution* of a family of I/O Automata consists of an alternating sequence of states and actions. An execution is *fair* if it satisfies a weak-fairness constraint (expressed in terms of actions). The *behaviors* of an Automaton are the respective subsequences of fair executions consisting of external (*i.e.*, input or output) actions. The language $\mathcal{L}(A)$ of an Automaton A is its set of behaviors. Two I/O Automata A and B are considered *equivalent* if $\mathcal{L}(A) = \mathcal{L}(B)$; A *implements* B, denoted $A \leq B$, if $\Pi\mathcal{L}(A) \subset \Pi\mathcal{L}(B)$, where Π denotes a projection of the language on the set of actions that are common to A and B (this concept will be made precise below).

It is not generally recognized that interleaving models such as I/O Automata in fact can be viewed as special (restricted) cases of *synchronous* parallel composition [6], [7]. Let $\mathcal{A}$ be any set of finite-state I/O Automata drawing actions from a fixed finite set containing a subset Δ. In the following we establish a translation $T : \mathcal{A} \rightarrow \mathcal{P}$ to a class of ω-automata $\mathcal{P}$ closed under a synchronous parallel composition, for which T is monotonic with respect to $\leq$ relative to the set of common actions Δ, and linear with respect to composition. Thus, for I/O Automata $A^1, \ldots, A^m, B^1, \ldots, B^n \in \mathcal{A}$ and $A = A^1 || \cdots || A^m$, $B = B^1 || \cdots || B^n$, if Δ is the set of actions common to A and B, we show that $A \leq B$ if and only if the ω-automaton language containment $\mathcal{L}(T(A^1) \otimes \cdots \otimes T(A^m)) \subset \mathcal{L}(T(B^1) \otimes \cdots \otimes T(B^n))$ obtains. (Hence, T is a function of Δ.) Here $||$ denotes the interleaving parallel composition on $\mathcal{A}$ and $\otimes$ denotes the synchronous parallel composition on $\mathcal{P}$. For the class $\mathcal{P}$, we use the L-process model of ω-automata [2], [7]. This result enables one to verify systems specified by I/O Automata through model-checkers such as COSPAN [4] or SMV [11], that operate on models with synchronous parallel composition. The translation generalizes to other interleaving models, although in each case, the translation map must match the specific model. We have chosen the I/O Automaton model to illustrate this translation principle because it strictly subsumes more common interleaving models based upon message-passing, such as Hoare's CSP [5]. To some extent this translation is applicable to Milner's CCS as well (*cf.* [12]) although CCS on the one hand is based upon branching-time semantics (while ω-automata have a linear-time semantics), and unlike ω-automata, CCS has no notion of fairness, on the other hand.

2 Background

2.1 I/O Automata

Although I/O Automata are defined without restrictions on cardinality [9, 10], we henceforth restrict our consideration to those I/O Automata whose state set and alphabet of actions are finite.

2.1.1 Definition An *I/O Automaton* A is a quintuple $A = (\Sigma^A, S^A, I^A, \delta^A, R^A)$ where:

- the *signature* Σ^A is a triple $\Sigma^A = (\Sigma^A_{IN}, \Sigma^A_{OUT}, \Sigma^A_{INT})$, where Σ^A_{IN}, Σ^A_{OUT}, Σ^A_{INT} are pairwise disjoint finite sets of elements, called *input*, *output* and *internal actions*, respectively. We denote by $\Sigma^A_{EXT} = \Sigma^A_{IN} \cup \Sigma^A_{OUT}$ the set of *external* actions, by $\Sigma^A_{LOC} = \Sigma^A_{OUT} \cup \Sigma^A_{INT}$ the set of *local* actions, and we abuse notation, denoting by Σ^A also the set of all actions $\Sigma^A_{LOC} \cup \Sigma^A_{IN}$;
- S^A is a finite set of *states*;
- $I^A \subset S^A$ is a set of *initial states*;
- $\delta^A \subset S^A \times \Sigma^A \times S^A$ is a *transition relation* which is *complete* in the sense that $\forall a \in \Sigma^A_{IN}, s \in S^A$ there exists $s' \in S^A$ with $(s, a, s') \in \delta^A$. For $a \in \Sigma^A_{LOC}$ and $s \in S^A$ such that $(s, a, s') \in \delta^A$, we say that a is *enabled* at s and *enables* the *transition* (s, s');
- R^A is a partition of Σ^A_{LOC}, each element of which is termed a *fairness constraint* of A.

2.1.2 Definition An *execution* of A is a finite string or infinite sequence of state-action pairs $((s_1, a_1), (s_2, a_2), \cdots)$, where $s_1 \in I^A$ and for all i, $s_i \in S^A$, $a_i \in \Sigma^A$ and $(s_i, a_i, s_{i+1}) \in \delta^A$.

2.1.3 Definition An execution $\mathbf{x}$ of A is *fair* if, for all $C \in R^A$:

- if $\mathbf{x}$ is finite then no action in C is enabled in the final state in $\mathbf{x}$;
- if $\mathbf{x}$ is infinite then either some action in C occurs infinitely often in $\mathbf{x}$ or else infinitely many states in $\mathbf{x}$ have no enabled action which is in C.

Thus, an infinite execution $\mathbf{x}$ is fair if and only if whenever some suffix of $\mathbf{x}$ has an action of C enabled in every state, then some action of C in fact occurs infinitely often in $\mathbf{x}$.

2.1.4 Definition Given a set $\Delta \subset \Sigma^A$, the *projection* of an execution $\mathbf{x} = ((s_i, a_i))$ of A or of a sequence $\mathbf{x} = (a_i)$ of actions of A, *onto* Δ, denoted $\Pi_\Delta(\mathbf{x})$, is the sub-sequence of actions obtained by removing from the action sequence (a_i) all actions $a_i \notin \Delta$. The projection Π_Δ is extended to sets of sequences, element-wise.

2.1.5 Definition A *behavior* of A is the projection of a fair execution of A on the set Σ^A_{EXT} (*i.e.*, the fair execution, with states and internal actions removed). The *language* $\mathcal{L}(A)$ of A is the set of behaviors of A.

2.1.6 Definition Of two I/O Automata A and B, we say that A *implements* B (denoted $A \leq B$) if, for $\Delta = \Sigma^A_{EXT} \cap \Sigma^B_{EXT}$, $\Pi_\Delta \mathcal{L}(A) \subset \Pi_\Delta \mathcal{L}(B)$.

2.1.7 Definition For I/O Automata $A^1, A^2, \cdots, A^k$, with respective pairwise disjoint sets of local actions, their *interleaving parallel composition*, denoted $A^1 || A^2 \cdots || A^k$, is an I/O Automaton A defined as follows. The set of internal actions of A is the union of the respective sets of internal actions of the component Automata, and likewise for the output actions; the input actions of A are the remaining actions of the components not thus accounted for. The set of states of A, S^A, is the Cartesian product of the component state sets, and

likewise for the initial states I^A. The transition relation δ^A is defined as follows: for $s = (s_1, \ldots, s_k)$, $s' = (s'_1, \ldots, s'_k)$, and $a \in \Sigma^A$, $(s, a, s') \in \delta^A$ if and only if for all $i = 1, \ldots, k$, $(s_i, a, s'_i) \in \delta^{A^i}$ or $a \notin \Sigma^{A^i}$ and $s'_i = s_i$. R^A is the union of the partition elements of the respective components.

It easily is checked that δ^A is complete, that the elements of R^A are pairwise disjoint and hence that A thus defined indeed is an I/O Automaton. It follows from this definition that component I/O Automata "synchronize" on common actions (changing state together), and only one action is enabled at a time (giving the composition its "interleaving" character).

The following definition (and lemma) are non-standard in the theory of I/O Automata.

2.1.8 Definition A sequence $\mathbf{x} = ((s_1, a_1), (s_2, a_2), \cdots)$ of state-element pairs is a *pseudo-execution* of an I/O Automaton A if $s_1 \in I^A$ and for all i, $s_i \in S^A$, and either $a_i \in \Sigma^A$ and $(s_i, a, s_{i+1}) \in \delta^A$, or $a_i \notin \Sigma^A$ and $s_{i+1} = s_i$. We say $\mathbf{x}$ is *fair* if it satisfies the conditions of Definition 2.1.3.

2.1.9 Lemma *The sequence* $((s_1, a_1), (s_2, a_2), \cdots)$ *of state-action pairs of the I/O Automaton* $A^1||A^2|| \cdots ||A^k$ *is a (fair) execution if and only if for all* $i = 1, \ldots, k$, $(s^i_1, a_1), (s^i_2, a_2), \cdots)$ *is a (fair) pseudo-execution of* A^i, *where for each* j, $s_j = (s^1_j, \ldots, s^k_j)$.

2.2 L-Processes

Boolean Algebra

A Boolean algebra [3] is a set L with distinguished elements $0, 1 \in L$, closed under the Boolean operations:

$$\begin{aligned} * &- \text{AND} \\ + &- \text{OR} \\ \sim &- \text{NOT} \end{aligned}$$

with universal element 1 and its complement 0. A Boolean algebra $L' \subset L$ is a *subalgebra* of L if L' and L share the same $0, 1$ and their operations agree. Every Boolean algebra contains the trivial 2-element Boolean algebra $\{0, 1\}$ as a subalgebra. For $x, y \in L$, write $x \leq y$ if and only if $x * y = x$. $S(L)$— the *atoms* of L, are the nonzero elements of L, minimal with respect to $\leq$. Every finite Boolean algebra is determined by its atoms, as every element is a sum of atoms (uniquely, up to permutation), and it is easy to see that any finite set is the set of atoms of a Boolean algebra.

2.2.1 Lemma [7, 4.2.20] *Any set of distinct elements of a Boolean algebra whose sum is* 1, *and all pairs of which have product* 0, *is the set of atoms of a subalgebra.*

2.2.2 Definition For subalgebras $L_1, \ldots, L_k \subset L$, their (*interior*) *product* is the subalgebra

$$\prod_{i=1}^{k} L_i = \left\{ \sum_{j \in J} x_{1j} * \cdots * x_{kj} \;\middle|\; x_{ij} \in L_i, \;\; J \text{ finite} \right\} .$$

In [13, §13] it is proved that for any Boolean algebras $L_1, \ldots, L_k$, there exists a Boolean algebra L containing (isomorphic copies of) them as subalgebras, with $\prod_{i=1}^{k} L_i = L$. This L is defined to be the *exterior* product of $L_1, \ldots, L_k$.

Transition Structure

2.2.3 Definition Let V be a nonempty set, and let M be a map

$$M : V^2 \to L \quad (V^2 = V \times V, \text{ the Cartesian product}).$$

Say M is an *L-matrix* with *state-space* $V(M) = V$. M provides the (static) transition function for automata. Note that $M(e) = \sum_{s \in S(L),\, s \le M(e)} s$ (where each s is an "input letter"). For all $v \in V(M)$, define $s_M(v) = \sum_{w \in V(M)} M(v, w)$.

2.2.4 Definition For an L-matrix M, and sequences $\mathbf{x} \in L^\omega$ and $\mathbf{v} \in V(M)^\omega$, we say $\mathbf{v}$ is a *run* of $\mathbf{x}$ *in* M provided for all i, $x_i * M(v_i, v_{i+1}) \neq 0$.

2.2.5 Definition The *tensor product* of L-matrices M and N is the L-matrix $M \otimes N$ with $V(M \otimes N) = V(M) \times V(N)$ and

$$(M \otimes N)((v, v'), (w, w')) = M(v, w) * N(v', w') .$$

Automata

2.2.6 Definition An *L-process* P is a 4-tuple[4]

$$P = (L_P,\ M_P,\ I(P),\ Z(P))$$

where L_P is a subalgebra of L (the *output* subalgebra), M_P is an arbitrary L-matrix, and

$$I(P) \subset V(M_P) \quad (\textit{initial} \text{ states})$$
$$Z(P) \subset 2^{V(M_P)} \quad (\textit{cycle} \text{ sets}) .$$

For an L-process P, write $V(P) \equiv V(M_P)$, $P(v, w) \equiv M_P(v, w)$.

2.2.7 Definition The *selections* of an L-process P at $v \in V(P)$ are the elements of the set

$$S_P(v) = \{ s \in S(L_P) \mid s * s_{M_P}(v) \neq 0 \} .$$

[4] The usual definition of an L-process includes an additional acceptance structure called "recur edges", not needed for our construction, and hence omitted here.

The intended interpretation of "selection" is a set of (nondeterministic) outputs as a function of state.

2.2.8 Definition Let M be an L-matrix and let $\mathbf{v} \in V(M)^\omega$. Set

$$\mu(\mathbf{v}) = \{v \in V(M) \mid v_i = v \text{ infinitely often}\}.$$

2.2.9 Definition Let P be an L-process. For $\mathbf{x} \in L^\omega$, a run $\mathbf{v}$ of $\mathbf{x}$ in M_P with $v_1 \in I(P)$, satisfying

$$\forall C \in Z(P), \ \ \mu(\mathbf{v}) \cap (V(P) \setminus C) \neq \phi$$

is called an *accepting* run of $\mathbf{x}$ *in* P. The *language* of P is the set

$$\mathcal{L}(P) = \{\mathbf{x} \in S(L)^\omega \mid \mathbf{x} \text{ admits of an accepting run in } P\}.$$

2.2.10 Definition Let $P_1, \ldots, P_k$ be L-processes. Then their tensor product is the L-process

$$\bigotimes_{i=1}^{k} P_i = \left(\prod_i L_{P_i}, \bigotimes_i M_{P_i}, \mathbf{X}_i I(P_i), \bigcup_i \Pi_i^{-1} Z(P_i) \right)$$

where the inverse projection $\Pi_i^{-1} Z(P_i) \equiv \{\Pi_i^{-1} C \mid C \in Z(P_i)\}$ and $\mathbf{X}$ is the Cartesian product.

The following lemma is the crucial *language intersection property* [7].

2.2.11 Lemma *Let $P_1, \ldots, P_k$ be L-processes. Then*

$$\mathcal{L}\left(\bigotimes P_i\right) = \bigcap \mathcal{L}(P_i) \ .$$

3 Translation Map

Let $\mathcal{A}$ be any set of finite-state I/O Automata drawing actions from a fixed finite set containing a subset Δ. We will construct a translation map $T : \mathcal{A} \to \mathcal{P}$ into the class $\mathcal{P}$ of L-processes, such that, for $A^1, \ldots A^m, B^1, \ldots, B^n \in \mathcal{A}$, $A = A^1 || \cdots || A^m$ and $B = B^1 || \cdots || B^n$, if Δ is the set of actions common to A and B,

$$A \leq B \Leftrightarrow \mathcal{L}(T(A^1) \otimes \cdots \otimes T(A^m)) \subset \mathcal{L}(T(B^1) \otimes \cdots \otimes T(B^n)).$$

(The map T thus depends upon $\mathcal{A}$ and the set of actions Δ common to A and B.) We will say that such a translation map T is *linear-monotone*.

The translation $A \to T(A)$ is the composition of five steps. Although some of these steps change the language of A, this is done uniformly, preserving linear-monotonicity.

The first step modifies the original I/O Automata, creating new I/O Automata all of whose fair executions are infinite, while preserving the implementation relation. With this transformation, we may restrict our consideration to

infinite executions, consistent with ω-automata. The second step associates with an I/O Automaton A, an L-process P^A that preserves its transition structure; by adding self-loops in P^A, the interleaving parallel composition can be replaced with the "synchronous" parallel composition implemented by the tensor product. Fairness constraints are handled in the third step, where for each constraint C we construct an L-process Q_C, the tensor product over all C of which, denoted Q^A, constrains the behaviors of P^A to correspond to the fair behaviors of A.

The transform $t(A) = P^A \otimes Q^A$ does only "half" the job, however, in the sense that $\mathcal{L}(t(A)) \subset \mathcal{L}(t(B))$ implies $A \leq B$, but not conversely (whereas we need the implication to hold in both directions). The reason for this is that implementation for I/O Automata is up to "stuttering-equivalence": projection onto external actions, whereas for ordinary automata, including L-processes, there is no such intrinsic equivalence relation. We deal with this in the fourth step, by expanding the set of executions of the image of t by taking a closure. This adds all those executions which are equivalent to the original ones from the perspective of the I/O Automaton.

The fifth step abstracts all internal actions, by mapping them homomorphically to a single symbol in a fashion which preserves language containment. This is needed in order to prevent the language containment test from distinguishing among internal actions.

3.1 First Step: removal of finite executions

The first step is a technical modification, which translates all fair executions of an I/O Automaton A into corresponding fair *infinite* executions of an I/O Automaton A'. This is accomplished by extending finite fair executions to infinite fair executions defining the same behavior. These extensions are implemented through the introduction into Σ^A_{INT} of a new internal action for each state of A in which no local action is enabled. Henceforth, we assume that all I/O Automata have been thus modified.

3.2 Second Step: basic conversion into L-processes

Given a finite set $\mathcal{A}$ of I/O Automata, let $L^{\mathcal{A}}$ be the Boolean algebra whose set of atoms $S(L^{\mathcal{A}})$ is the union of all actions of members of $\mathcal{A}$, *i.e.*, $S(L^{\mathcal{A}}) = \bigcup_{A \in \mathcal{A}} \Sigma^A$. For each $A \in \mathcal{A}$, let $\%_A$, the *pause* of A (relative to $\mathcal{A}$), denote $\%_A = \sum_{a \in S(L),\ a \notin \Sigma^A_{LOC}} a$, and let L_A denote the subalgebra of $L^{\mathcal{A}}$ whose atoms are $S(L_A) = \Sigma^A_{LOC} \cup \{\%_A\}$. Note that $L^{\mathcal{A}} = \prod_{A \in \mathcal{A}} L_A$. The Boolean algebra $L^{\mathcal{A}}$ provides the basis to represent *actions* of A. However, in order to deal with the fairness constraints of A, we also represent the states of A in the selections of the associated L-process P^A. For each $A \in \mathcal{A}$, let L_{S^A} be the Boolean algebra whose set of atoms $S(L_{S^A}) = S^A$. Through a representation of the state of P^A in the selection of P^A, another L-process, Q^A, will be able to track a succession of state transitions of P^A, and exclude the unfair ones. We assume without loss of generality that distinct I/O Automata in $\mathcal{A}$ have no states in common. Define the Boolean algebra L to be the exterior product $L = L^{\mathcal{A}} \cdot \prod_{A \in \mathcal{A}} L_{S^A}$.

Given an I/O Automaton $A \in \mathcal{A}$, define the L-process P^A as follows:

$$L_{P^A} = L_A \cdot L_{S^A};\ V(P^A) = S^A;\ I(P^A) = I^A;\ Z(P^A) = \phi;$$

for all $v, w \in S^A$

$$P^A(v,w) = \begin{cases} \sum_{(v,a,w)\in\delta^A} a * w & \text{if } v \neq w \\ (\%_{A^*} \sim \sum_{(v,a,u)\in\delta^A} a + \sum_{(v,a,v)\in\delta^A} a) * v & \text{if } v = w \end{cases}$$

The L-process P^A is obtained by interpreting each local action together with the resulting next state of A, as the respective selections of P^A. The "self-loop" at each state $v \in V(P^A)$ is enabled by its "pause" selection $\%_A * v$, which corresponds to the conjunction with v of the disjunction of all input actions, or local actions of other Automata in $\mathcal{A}$. This definition corresponds to an interpretation in which the L-process P^A "pauses" in its current state whenever another process exercises one of its local actions (unless that local action entails a transition of A). Indeed, $a * b = 0$ for any $a,\ b \in S(L)$, $a \neq b$, and thus simultaneous external selections in distinct L-processes corresponds to a "null" event, enabling no transition, whereas $\%_A * b \neq 0$ for any $b \in \Sigma^B_{LOC}$, $B \neq A$.

3.2.1 Lemma *The sequence* $((s_1,a_1),(s_2,a_2),\cdots)$ *is a pseudo-execution of the I/O Automaton* $A \in \mathcal{A}$ *if and only if* $(s_1,s_2,\dots)$ *is an accepting run in the L-process* P^A *of* $(a_1,a_2,\dots)$.

3.2.2 Corollary *The sequence* $((s_1,a_1),(s_2,a_2),\cdots)$ *of state-action pairs of the I/O Automaton* $A = A^1||A^2||\cdots||A^k$ *is an execution of A if and only if* $(s_1,s_2,\dots)$ *is an accepting run of* $(a_1,a_2,\dots)$ *in the L-process* $P = P^{A^1} \otimes P^{A^2} \otimes \cdots \otimes P^{A^k}$.

For any sequence $\mathbf{x} = ((s_1,a_1),(s_2,a_2),\cdots)$ of state-action pairs of A, let $\mathbf{x}^* = (s_1 * a_1, s_2 * a_2, \dots) \in S(L^A \cdot L_{S^A})^\omega$. By Lemma 3.2.1, $\mathbf{x}$ is an execution of A if and only if $\mathbf{x}^* \in \mathcal{L}(P^A)$, for P^A considered as an $L^A \cdot L_{S^A}$-process. Likewise, for A and P as in Corollary 3.2.2, $\mathbf{x}$ is an execution of A if and only if $\mathbf{x}^* \in \mathcal{L}(P)$, for P considered an $L^A \cdot \prod_{i=1}^k L_{S^{A^i}}$-process.

3.3 Third Step: fairness constraints

In this step we incorporate the fairness constraints of each $A \in \mathcal{A}$ into the corresponding L-process P^A, by constraining P^A through a product with an L-process Q^A. The result will be that the *fair* executions of A correspond to the elements of $\mathcal{L}(P^A \otimes Q^A)$.

For each fairness constraint $C \in R^A$, we construct a two-state L-process Q_C, as follows. Denote by σ_C the set of states of the I/O Automaton A in which some action in C is enabled. Let $V(Q_C) = \{\text{fair}, \text{unfair}\}$. The behavior we define in Q_C is that it moves to fair whenever either the process P^A has selected an

action in C, or makes a transition to a state not in σ_C, and Q_C moves to unfair whenever the opposite is the case. Executions of A unfair with respect to C are those which cause Q_C ultimately to remain in its state unfair. These are eliminated in $P^A \otimes Q_C$ by defining {unfair} to be a cycle set of Q_C.

Set $\sigma'_C = V(P^A) \setminus \sigma_C$.

Formally, Q_C is the L-process defined by

$$Q_C(\mathsf{unfair}, \mathsf{fair}) = Q_C(\mathsf{fair}, \mathsf{fair}) = \sum_{a \in C} a + \sum_{v \in \sigma'_C} v,$$

$$Q_C(\mathsf{fair}, \mathsf{unfair}) = Q_C(\mathsf{unfair}, \mathsf{unfair}) = \sim Q_C(\mathsf{fair}, \mathsf{fair});$$

$$L_{Q_C} = \{0, 1\};\ I(Q_C) = \{\mathsf{unfair}\};\ Z(Q_C) = \{\{\mathsf{unfair}\}\}.$$

Define $Q^A = \bigotimes_{C \in R^A} Q_C$, and set $t(A) = P^A \otimes Q^A$. Note that $t(A)$, like P^A, is an $L^{\mathcal{A}} \cdot L_{S^A}$-process.

Once the product $t(A)$ is formed, we remove the state information through an abstraction transformation of a type called a *support map.*

Define $L \to L^{\mathcal{A}}$ by $x \to \widehat{x}$ for $x \in S(L)$, where $\widehat{a * s} = a$ for $x = a * s$, $a \in S(L^{\mathcal{A}})$, $s \in S(\prod_A L_{S^A})$, and extend it linearly to all of L, with $\widehat{0} = 0$.

Any map $x \to \sigma(x)$ of an arbitrary Boolean algebra, such as the above map $L \to L^{\mathcal{A}}$, defined on the atoms of one Boolean algebra to the atoms of another, and extended to the entire Boolean algebra by linearity, with $\widehat{0} = 0$, is called a *support* map. Note that the image of a Boolean algebra under a support map, is a Boolean algebra. (A support map $L \to L'$ is dual to a Boolean algebra homomorphism $L' \to L$ and preserves language containment [7].)

For any string or sequence $\mathbf{x} = (x_i)$ and any support map σ, define $\sigma(\mathbf{x}) = (\sigma(x_i))$. Likewise, for any set $S \subset L$, define $\sigma(S) = \{\sigma(x) \mid x \in S\}$. Thus, $\widehat{L} = L^{\mathcal{A}}$.

For any Boolean algebra L and any support map $\sigma : L \to \sigma(L)$, and for any L-process P, let $\sigma(P)$ be the $\sigma(L)$-process whose output subalgebra $L_{\sigma(P)}$ satisfies $S(L_{\sigma(P)}) = S(\sigma(L_P))$, with $M_{\sigma(P)}(v, w) = \sigma(M_P(v, w))$ for all $v, w \in V(P)$, and $I(\sigma(P)) = I(P)$, $Z(\sigma(P)) = Z(P)$.

For the remainder of this section, we refer to the specific support map $L \to L^{\mathcal{A}} = \widehat{L}$, as defined above. Let $\widehat{t}$ be the map $\widehat{t} : \mathcal{A} \to \mathcal{P}$ where $\mathcal{P}$ is the class of $L^{\mathcal{A}}$-processes, defined by $\widehat{t}(A) = \widehat{t(A)}$. Note, for example, that for $A^1, \ldots, A^k \in \mathcal{A}$ and $P = \bigotimes t(A^i)$, $\widehat{P} = \bigotimes \widehat{t}(A^i)$.

3.3.1 Lemma *There is a fair pseudo-execution* $((s_1, a_1), (s_2, a_2), \cdots)$ *of* $A \in \mathcal{A}$ *if and only if* $(a_i) \in \mathcal{L}(\widehat{t}(A))$.

3.3.2 Corollary *For any parallel compositions of I/O Automata from* $\mathcal{A}$*:*

$$A = A^1 || A^2 || \cdots || A^m, \quad B = B^1 || B^2 || \cdots || B^n,$$

and the $L^{\mathcal{A}}$*-processes*

$$P = \widehat{t}(A^1) \otimes \widehat{t}(A^2) \otimes \cdots \otimes \widehat{t}(A^m), \quad Q = \widehat{t}(B^1) \otimes \widehat{t}(B^2) \otimes \cdots \otimes \widehat{t}(B^n),$$

$$\mathcal{L}(P) \subset \mathcal{L}(Q) \Rightarrow A \leq B.$$

The converse to (3.3.2) fails, on account of problems illustrated in the following example.

3.3.3 Example The map $\widehat{t}$ is not in general linear-monotone. If A is an I/O Automaton with a single fair execution whose action sequence is ab^ω for an internal action a and external action b, and B is an I/O Automaton with the single fair execution whose action sequence is b^ω, then $\mathcal{L}(A) = \mathcal{L}(B)$ so in particular $A \leq B$. However, obviously $\mathcal{L}(P^A)$ and $\mathcal{L}(P^B)$ are incommensurate, so $\mathcal{L}(t(A)) \not\subset \mathcal{L}(t(B))$. Likewise, if B gave rise to the single action sequence $a'b^\omega$, for and internal action $a' \neq a$, the same conclusion obtains.

3.4 Fourth and Fifth Steps: closure; abstracting internal actions

In the definition of the Boolean algebra L in Section 2.2, redefine each L_{S^A} to be the Boolean algebra whose set of atoms is $S(L_{S^A}) = 2^{(S^A \cup \Sigma^A_{INT})}$, the set of all subsets of states and internal actions of A. We will denote these atoms by minterms, treating the elements of $S^A \cup \Sigma^A_{INT}$ as Boolean variables. In order to distinguish an internal action or state r from the Boolean variable term denoted by the same symbol, the term will be denoted within square brackets, as $[r]$. Thus, for example, the atom consisting of all of the states and none of the internal actions is denoted by the minterm

$$(\prod_{v \in S^A} [v]) * (\prod_{\epsilon \in \Sigma^A_{INT}} \sim [\epsilon]).$$

In fact, we also may denote each such minterm by its characteristic function, defined by χ. Thus, the minterm above also is denoted by $\chi(S^A)$. In this notation, for $v \in S^A$, the term $[v]$ denotes the sum of minterms

$$[v] = \sum_{S \subset S^A \cup \Sigma^A_{INT},\, v \in S} \chi(S)$$

and thus $\chi(\{v\}) \leq [v]$.

Redefine the transition predicate $P^A(v, w)$ in Section 2.2, by replacing the occurrences of the factors v and w in that definition by $\chi(\{v\})$ and $\chi(\{w\})$, respectively. All the results of Section 2.2 trivially continue to hold.

Likewise, in the definition of Q_C in Section 2.3, replace the elements v by the terms $[v]$ and add terms for the elements of C in the (newly defined) subalgebra L_{S^A}. Thus, with the new definition,

$$Q_C(\mathsf{unfair},\, \mathsf{fair}) = Q_C(\mathsf{fair},\, \mathsf{fair}) = \sum_{a \in C}(a + [a]) + \sum_{v \in \sigma'_C} [v]$$

(the $[a]$ terms being for use with the closure). It is trivial as before that all the results of Section 2.3 continue to hold.

Our last redefinition is the support map $L \to \widehat{L} \simeq L^{\mathcal{A}}$ which we redefine accordingly (mapping $\widehat{a * s} = a$ as before, for the new definition of s).

3.4.1 Definition For $\mathbf{x} = (x_i) \in S(L)^\omega$ and strings $\epsilon_i \in S(L)^*$ for all i, we say the sequence $(\epsilon_i x_i)$, formed by the alternating interleaving of a string ϵ_i with an atom x_i, is an *extension* of $\mathbf{x}$ (*by* (ϵ_i)). For $\mathbf{x} = (x_i) \in S(L)^\omega$, $\epsilon \in S(L)$ and a sequence of non-negative integers m_i, let $\mathbf{y} = (\epsilon^{m_i} x_i) \in S(L)^\omega$ denote the extension of $\mathbf{x}$ by (ϵ^{m_i}). In this case, we say $\mathbf{y}$ is an *ϵ-extension* of $\mathbf{x}$. Thus, $\mathbf{y}$ is obtained from $\mathbf{x}$ by inserting m_i ϵ's before x_i, for all i. For notational convenience, if (x_i) is finite, with last element x_n, say, then we infer $(\iota_i x_i) = (\iota_1 x_1 \cdots \iota_n x_n \iota_{n+1} \iota_{n+2} \cdots)$.

We now define the closure.

3.4.2 Definition For any L-process P, and any $\iota \in L$, define the *closure* $\mathcal{L}^{[\iota]}(P)$ of $\mathcal{L}(P)$ (*relative* to ι) recursively, by

$$\mathcal{L}(P) \subset \mathcal{L}^{[\iota]}(P);$$

and, for any $\epsilon \in S(L)$, $\epsilon \leq \iota$, and any non-negative integers m_i,

$$(\epsilon^{m_i} x_i) \in \mathcal{L}^{[\iota]}(P) \Leftrightarrow (x_i) \in \mathcal{L}^{[\iota]}(P).$$

Thus, $\mathcal{L}^{[\iota]}(P)$ contains all sequences obtained from elements of $\mathcal{L}(P)$ by inserting or deleting atoms $\epsilon \leq \iota$, arbitrarily.

3.4.3 Definition Let P be an L-process, and let $\iota \in L$. An *ι-path* in P, *from* $v \in V(P)$ *to* $w \in V(P)$, is a string $\mathbf{v} = (v_1, \ldots, v_n) \in V(P)^{n+1}$ for $n > 1$ such that $\iota * P(v_i, v_{i+1}) \neq 0$ for $i = 1, \ldots, n-1$, and $v_1 = v$, $v_n = w$. For $v, w \in V(P)$, let $\pi_\iota(v, w)$ be the set of ι-paths from v to w.

3.4.4 Definition For each $A \in \mathcal{A}$, set $\iota_A = \sum_{a \in \Sigma^A_{INT}} a$ and let $\iota = \sum_{A \in \mathcal{A}} \iota_A$. For $A \in \mathcal{A}$ and an ι-path $\mathbf{v} = (v_1, \ldots, v_n)$ in P^A, define for all $i = 1, \ldots, n-1$,

$$\kappa_i(\mathbf{v}) = \{a \in \Sigma^A_{INT} \mid a * P^A(v_i, v_{i+1}) \neq 0\}$$

and set $\kappa(\mathbf{v}) = \mathbf{X}_{1 \leq i < n} \kappa_i(\mathbf{v})$. For each $\alpha = (\alpha_1, \ldots, \alpha_{n-1}) \in \kappa(\mathbf{v})$, define the *content* of $\mathbf{v}$ (*relative* to α) to be

$$\kappa^\alpha(\mathbf{v}) = \{v_2, \ldots, v_n\} \cup \{\alpha_1, \ldots, \alpha_{n-1}\}.$$

3.4.5 Definition For $a \in \mathcal{A}$ define the *closure* of P^A to be the L-process $\widehat{P^A}$ given by

$$L_{\widehat{P^A}} = L_{P^A};\ V(\widehat{P^A}) = V(P^A);\ I(\widehat{P^A}) = I(P^A); Z(P^A) = \phi;$$

$$\widehat{P^A}(v, w) = \begin{cases} P^A(v, w) + \sum\limits_{u \in V(P^A)} \widehat{P^A}(u, w) * \Big(\sum\limits_{\mathbf{v} \in \pi(v,u)} \sum\limits_{\alpha \in \kappa(\mathbf{v})} \chi(\kappa^\alpha(\mathbf{v})) \Big) & \text{if } v \neq w \\ P^A(v, v) + \iota * \chi(\{v\}) & \text{if } v = w \end{cases}$$

Thus, $\widehat{P^A}(v,w)$ is enabled by $P^A(v,w)$ and moreover by $\widehat{P^A}(u,w)$ (the non-L_{S^A} part of $P^A(u,w)$) *together* with the set(s) of internal actions and states along an ι-path from v to u. The self-loops are enabled by ι. As is customary, a sum over the empty set is 0.

3.4.6 Theorem For $A \in \mathcal{A}$,

$$\mathcal{L}^{[\iota]}(P^A) = \mathcal{L}(\widehat{P^A}).$$

Let $\sigma : L^{\mathcal{A}} \to L^{\mathcal{A}}$ be the support map defined on $S(L^{\mathcal{A}})$ by

$$\sigma(x) = \begin{cases} \iota \text{ if } x \in \cup_{A\in\mathcal{A}} \Sigma^A_{INT} \\ x \text{ otherwise} \end{cases}$$

and extended linearly to $L^{\mathcal{A}}$ (with $\sigma(0) = 0$). Now, extend σ to a support map $\sigma : L \to \sigma(L^{\mathcal{A}})$ by defining for $x \in L$, $\sigma(x) = \sigma(\widehat{x})$. Set $L' = \sigma(L) \subset L$.

If at least two I/O Automata in $\mathcal{A}$ have internal actions, then for every $A \in \mathcal{A}$, $\iota * \%_A \neq 0$. There is no loss of generality for our purposes to assume this (two "dummy" elements certainly could be added to $\mathcal{A}$), and it simplifies our constructions. Therefore, we now assume this.

We assume without loss of generality that the internal actions of each $A \in \mathcal{A}$ are distinct from the actions of all other members in $\mathcal{A}$.

3.5 Translation Theorem

The translation map $T : \mathcal{A} \to \mathcal{P}$ now is defined for each $A \in \mathcal{A}$, as

$$T(A) = \sigma(\widehat{P^A} \otimes Q^A).$$

Thus, $T(A)$ is formed from $t(A) = P^A \otimes Q^A$ by replacing P^A with its closure, and associating all internal actions with a single abstract internal action ι.

For the *only if* part of the Translation Theorem that follows, we need to deal with the projection (2.1.6) onto common actions. The T defined above works in this part of the theorem only when A and B share the same external actions. In general, we must define T as a function of $\Delta = \Sigma^A_{EXT} \cap \Sigma^B_{EXT}$: different Δ's require different T's. For simplicity of presentation, instead of defining T (actually, the L-processes P^A of Section 2.2) as a function of Δ, we redefine all actions not common to both I/O Automata A and B to be internal. This redefinition causes all such actions to be abstracted by ι, giving the required T as defined above. Having made this transformation, $A \leq B \Leftrightarrow \mathcal{L}(A) \subset \mathcal{L}(B)$.

3.5.1 Theorem *Let $\mathcal{A}$ be a finite set of I/O Automata drawing actions from a fixed finite set containing a subset Δ, let $L = L^{\mathcal{A}} \cdot \prod_{A\in\mathcal{A}} L_{S^A}$ and let T be the translation map T on $\mathcal{A}$ relative to Δ. For any parallel compositions of I/O Automata $A^1, \ldots, A^m, B^1, \ldots, B^n \in \mathcal{A}$:*

$$A = A^1||A^2||\cdots||A^m,\ B = B^1||B^2||\cdots||B^n,$$

and the corresponding L'-processes

$$P = T(A^1) \otimes T(A^2) \otimes \cdots \otimes T(A^m),\ Q = T(B^1) \otimes T(B^2) \otimes \cdots \otimes T(B^n),$$

if Δ is the set of actions common to A and B, then

$$A \leq B \Leftrightarrow \mathcal{L}(P) \subset \mathcal{L}(Q).$$

3.5.2 Corollary

$$A \leq B \Leftrightarrow \text{for all } i\ \mathcal{L}(P) \subset \mathcal{L}(T(B^i))$$

4 Conclusion

We have demonstrated an instance of a general transformation technique whereby an *interleaving* composition may be viewed as a special case of a *synchronous* (*i.e.*, ordinary automaton) product, over conventional automata. The key to this transformation is the simple observation that in a synchronous product, in each step, all parallel components but one can be constrained to self-loop, remaining in their respective component states, thereby simulating interleaving. The choice of the one exception is nondeterministic. Fairness constraints, if any, can be added in terms of automata acceptance conditions. The main difficulties of the transformation are associated with side conditions required by idiosyncrasies of the specific model. In the case of I/O Automata, the main difficulty arises from its notion of "implementation", Definition 2.1.6, based upon projection and stuttering-equivalence. Unlike the generic basic transformation $A \rightarrow t(A)$, the side conditions require the more *ad hoc* $A \rightarrow T(A)$ which, unlike t, is dependent upon the projection image Δ.

Thus, application of the transformation technique depends in its details on the particular interleaving model. We have chosen for demonstration the I/O Automaton model of Lynch and Tuttle [9, 10], since it strictly subsumes more common interleaving models based upon message-passing, such as Hoare's CSP [5].

The meaning of the Translation Theorem, in terms of verification, is that, in order to check *implementation* for I/O Automata, the question can be transformed component-wise to a *language containment* check for L-processes. This gives a decision procedure with known complexity, for which existing tools (such as COSPAN and SMV) can be used to test implementation between I/O Automata. The complexity of the language containment check $\mathcal{L}(P) \subset \mathcal{L}(Q)$ of the Translation Theorem 3.5.1, is linear in the number of transitions of A [7], and in view of Corollary 3.5.2, it also is linear in the number n of components of B, although exponential in the number of fairness constraints of A and each B^i. Furthermore, the support mapping of $\iota_A \rightarrow \iota$ for each $A \in \mathcal{A}$ introduces nondeterminism in P and Q, as does taking the closure. In the worst case, this makes the implementation check exponential in the size of the largest component of B^i (essentially, each $T(B^i)$ must be determinized).

Application of this approach to the verification of systems and algorithms originally specified with I/O Automata has been successful [8]. That paper reports on the formal verification of a fault-tolerant algorithm for shared memory. The original algorithm as well as the task that it has to perform (and a hand-written argument for its correctness) had been specified with I/O Automata [1]. Through the above translation it was possible for the first time to formally verify the algorithm's correctness, in this case, using COSPAN.

References

1. Y. Afek, D. S. Greenberg, M. Merritt, and G. Taubenfeld. Computing with Faulty Shared Memory. In *Proc. 11th ACM Symp. on Principles of Distributed Computing*, 1992.
2. S. Aggarwal, R. P. Kurshan, and K. Sabnani. A Calculus for Protocol Specification and Validation. In *Protocol Specification, Testing and Verification III*, pages 19–34. North-Holland, 1983.
3. P. Halmos. *Lectures on Boolean Algebras.* Springer-Verlag, 1974.
4. Z. Har'El and R. P. Kurshan. Modelling concurrent processes. In *Proceedings of Internat. Conf. Syst. Sci. Eng.*, pages 382–385, 1988.
5. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
6. R. P. Kurshan. *Automata-Theoretic Verification.* UC Berkeley Lecture Notes, 1992.
7. R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach.* Princeton University Press, 1994.
8. R. P. Kurshan, M. Merritt, A. Orda, and S. R. Sachs. Formal Verification of a Distributed Algorithm for Accessing Faulty Shared Memory. In *Proc. of SBT/IEEE International Telecom. Symp., Rio de Janeiro, Brazil*, 1994.
9. N. Lynch and M. Tuttle. Hierarchical Correctness Proofs for Distributed Algorithms. In *Proc. 6th ACM Symp. on Principles of Distributed Computing*, pages 137–151, 1987.
10. N. Lynch and M. Tuttle. An Introduction to Input/Output Automata. *CWI-Quarterly*, 2(3):219–246, September 1989.
11. K. L. McMillan. *Symbolic Model Checking.* Kluwer, 1993.
12. R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Sci.*, 25:267–340, (1983).
13. E. Sikorski. *Boolean Algebras.* Springer Verlag, 1969.