Product automata and process algebra

Kamal Lodaya
The Institute of Mathematical Sciences,
C.I.T. Campus, Taramani, Chennai 600 113, India

Abstract

We define a model of labelled product systems of automata and explore its connections with process calculi and trace languages. Bisimilarity of labelled product systems is defined using a new definition of bisimulation with renaming. Concurrent μ-expressions are defined to describe labelled product systems. This leads to complete axiomatizations and algorithms for bisimulation and failure equivalence over labelled product systems, and for equality over recognizable trace languages.

Process algebra is a large enough field of research to have come out with its own handbook [8], but does not have the wide applicability that a field like automata theory does. Valmari [40] points out that ideas from process algebra can usefully be applied to verification using automata theory. Since there are a few textbooks [4, 36, 14], several monographs (with illustrious authors like Hoare [18] and Milner [28]) and a fine Turing award lecture by Milner [29], that this does not commonly happen seems a bit surprising.

One reason could be the lack of comunication between the two communities. Way back in 1984, Milner considered the behaviour of finite automata upto Park bisimilarity [34] rather than the more usual language acceptance, and provided an inference system which was sound and complete [27]. There have been axiomatizations of finite-state processes thereafter (see the recent [3], for example), but it is difficult to see how to use them in applications.

To take an example, it is not obvious that if one extends Milner's axiomatization with the characteristic axioms of failure equivalence [10] (which axiomatize *finite* processes rather than *finite-state* ones), one gets a complete axiomatization for failure equivalence of finite automata. It is certainly possible to extract such a completeness proof if one digs through Roscoe's book [36], but to the not so diligent reader, it isn't clear that the laws in the book, designed for the semantic treatment of deadlocks in general, lead to an algorithm for detecting deadlocks in systems of communicating automata.

The automata theorists are far from blemishless. They have little patience for the elaborate spectrum of equivalences for process calculi [16] and they are more interested in working with a basic formalism and attacking expressiveness and decidability questions. Looking at *The book of traces* [12], it is difficult to figure out that it has something to say about systems of communicating automata. (In our library, this book is classified under physics, which shows how much we fail to tell even professionals in the business of information.)

0.1 Summary

We work in the setting of axiomatizations of automaton behaviour, with the (wistful?) aim of retaining both audiences. We start with a simple communicating automaton model familiar to implementors, labelled product systems. Valmari's systems of co-operating automata [40] are close, but our model arose in the world of Mazurkiewicz traces [25] and is inspired by the asynchronous automata defined by Zielonka [41], and their local presentations by Mohalik and Ramanujam [31]. We define bisimulation with renaming over (ordinary) automata and use this to define system bisimilarity over their labelled products. We give a syntax which is as expressive as the automata. A simple extension of Milner's completeness proof [27] gives an axiomatization for system bisimilarity of finite labelled product systems. Since Milner's proof is a refinement of Salomaa's completeness proof for equality of regular languages, we now work backwards and get an axiomatization for equality of finite-state trace languages. We also have that system bisimilarity and failure equivalence are decidable for finite labelled product systems, and language equivalence is decidable for our expressions. It does not need completeness proofs to see this, but the proofs are appealingly simple.

Acknowledgements. Four SEFM referees looked at this paper. Their comments, critical as well as supportive, were useful in rewriting this paper. We thank them for their detailed reports.

1 Automata, products and labelling

Fix a finite set of locations Loc. Let B be a finite alphabet, and $loc : B \rightarrow \wp(Loc)$ map each action to the locations it is executed in. We wish to define systems of automata at these locations, with synchronization actions (those c with $|loc(c)| > 1$) as the medium of communication, as is common in process calculi. There is a variety of ways to do this; here is what suits us.

Definition 1 *A **rendezvous alphabet** $(B, loc, 0, |)$ over Loc consists of a finite alphabet B containing 0, $loc : B \rightarrow \wp(Loc)$ a function and $| : B \times B \rightarrow B$ a Loc-respecting function, that is, satisfying the properties below.*

- *$|$ is commutative and associative with 0 an absorbing element.*
- *$loc(0) = \emptyset$ and $loc^{-1}(\emptyset) = \{0\}$.*
- *If $r = loc(a)$ and $s = loc(b)$ are not disjoint, then $a|b = 0$.*
- *If $r = loc(a)$ and $s = loc(b)$ are disjoint and $a|b \neq 0$ then $loc(a|b) = r \cup s$.*
- *If $loc(c) = r \cup s$ for disjoint nonempty r and s then there exist a with $loc(a) = r$ and b with $loc(b) = s$ such that $c = a|b$.*

We can think of the actions in B with a single location as the local actions, 0 a dummy action representing lack of communication, and the rest as a rendezvous structure of synchronized actions built up from local ones using the function $|$. We call a nonzero action a global if for every b, $a|b = 0$. Thus a global action is either a local action which does not rendezvous at all, or a "full" synchronization, say $a_1|a_2|a_3|\ldots|a_k$, whose "partial" subsets (such as a_3, $a_1|a_k$, $a_2|a_3|a_k$) are not global since further rendezvous can be carried out. We call these partial synchronizations rendezvous actions.

Having set our alphabets in order, we are now ready for the basic objects of study in this paper, product systems. We will later generalize these to labelled product systems.

Definition 2 *Let $B_i \overset{\text{def}}{=} \{a \in B \mid i \in loc(a)\}$. A **product system of automata** over the rendezvous alphabet $(B, loc, 0, |)$ is given by automata $M_i = (P_i, \rightarrow_i, p_0^i)$ over the alphabet B_i, for each i in Loc, together with a set of global final states $F \subseteq \Pi_{i \in Loc} P_i$.*

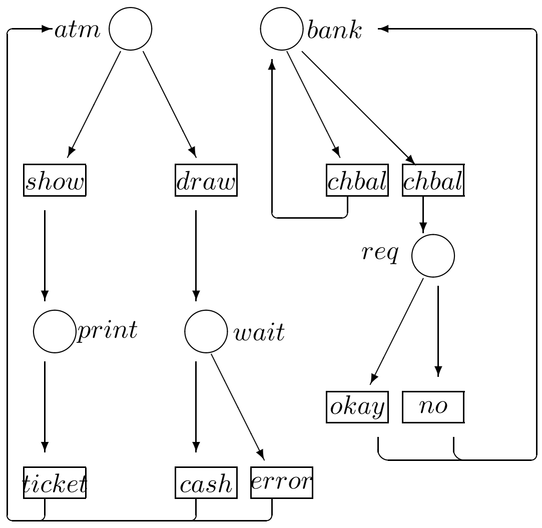

Figure 1. A product system

As usual P_i is a finite set of states, $\rightarrow_i$ the transition relation and p_0^i the initial state. The figure shows a simple ATM-bank example. atm and $bank$ are the local initial states, say at locations 1 and 2. $\{atm, bank\}$ is the only global final state. The nonzero rendezvouses are $s = show|chbal$, $d = draw|chbal$, $c = cash|okay$ and $e = error|no$. $loc(s) = loc(d) = loc(c) = loc(e) = \{1, 2\}$.

Let $G(B)$ be the global actions in B. A product system runs on a word w over $G(B)$, that is, the rendezvous structure used to generate a synchronization letter is seen as invisible detail. A run assigns global states $\Pi_{i \in Loc} P_i$ to prefixes of w: the empty word is assigned the product formed from the local initial states, and for every prefix va of w, if $\Pi_{i \in Loc} p_i$ is the global state reached after v, then the state $\Pi_{i \in Loc} q_i$ reached after va satisfies, for every $j \in loc(a)$, $p_j \xrightarrow{a}_i q_j$ in M_i, and for every other j, $p_j = q_j$. Thus every action transforms the states of the locations it affects, the other states remaining fixed. The run accepts the word if it ends in a final state in F.

The word $s \cdot ticket \cdot d \cdot c \cdot s \cdot ticket \cdot d \cdot e$ has an accepting run in the example.

Since a product system can be simulated by a finite automaton over the alphabet $G(B)$, its accepted language is regular. Our product systems are named "extended product systems" in Mohalik's thesis [30] and they are shown to accept the boolean closure of the regular shuffle languages. We shorten this name and call the class recognizable shuffle languages. The thesis contains a nice overview of properties of this class.

We will use two formal models of concurrent behaviour. The first one is that of Mazurkiewicz trace languages [25], an extension of the classical notion of languages associated with automata. Later we will use the more refined notion of bisimilarity, which is commonly used in process algebra. We will also make some comments about the intermediate failure model of [10].

1.1 Mazurkiewicz traces

Let I be an irreflexive symmetric relation over B called independence defined by aIb if $loc(a)$ and $loc(b)$ are disjoint. Let its reflexive transitive closure on B^* be $\sim_I$, called trace congruence. For instance, if aIb then $wabx \sim_I wbax$ (a and b commute).

Notice that if a product system has a run (or an accepting run) on a word $wabx$ and aIb, then it has a run (resp. an accepting run) on the word $wbax$ as well. Hence a recognizable shuffle language is a recognizable trace language over $(G(B), I)$ in the sense of Mazurkiewicz [25]. Zielonka showed that the converse is not true [41].

Intuitively, the global actions are those that are "visible" at the level of the product automaton, but since CCS has an alternate approach of making the outcome of a rendezvous "invisible" by renaming it to τ [28], we leave visibility of alphabet letters as an orthogonal decision. To implement this idea, we will now further rename the global actions to get the visible ones.

Definition 3 *A **renaming** between rendezvous alphabets is a relation which is Loc-respecting and $|$-stable: that is, if $a\rho b$ then $loc(a) = loc(b)$, and if $c\rho d$ as well, $(a|c)\rho(b|d)$.*

*Let ρ be an equivalence relation over a rendezvous alphabet $(B, loc, 0, |)$ which is a renaming. $C = (G(B)/\rho, loc)$ is called a **distributed alphabet** where $loc([a]) \overset{\text{def}}{=} loc(a)$ is well defined by the properties of ρ.*

*A **labelled product system of automata** $M[\rho]$ over the distributed alphabet is a product system M over the rendezvous alphabet together with the labelling function $\rho(a) = [a]$.*

For example, in the automaton in the figure, if we set $(cash|okay)\ \rho\ (error|no)$ and label the equivalence class $null$, the product has become nondeterministic.[1] The word $s{\cdot}ticket{\cdot}d{\cdot}null{\cdot}s{\cdot}ticket{\cdot}d{\cdot}null$ is accepted by the labelled product, but we have lost the information whether cash was provided each time a drawing action was performed.

Let $\rho(w)$ be the homomorphic extension to $w \in C^*$. If a product system M accepts the language $L \subset G(B)^*$, we say the labelled product system $M[\rho]$ accepts the language $\rho(L) \subset C^*$.

Let I be the independence relation over B above extended to C. Using the properties of ρ, the languages accepted by labelled product automata continue to be recognizable trace languages over (C, I). But they need no longer be recognizable shuffle languages.

Theorem 4 *Every recognizable trace language is accepted by a labelled product system.*

Proof. The proof is essentially the same as that of Mohalik and Ramanujam [31]. Every recognizable trace language is accepted by a Zielonka automaton [41]. Break up each synchronizing transition $t \in \rightarrow_a$ of such an automaton into local transitions $t_1, \ldots, t_l$, and put all pairs $t_1|...|t_l$ into the equivalence class a. □

Zielonka defined his automata to characterize the recognizable trace languages [41]. They have distributed transitions, which makes them difficult to implement. Mukund and Sohoni [32] performed a detailed analysis of the communicated information and arrived at the gossip information which is needed to distribute the transitions into local ones. This was used to upper bound the complexity of determinization [21]. Mohalik and Ramanujam [31] made the gossip information static by moving it into an "assumption-commitment" structure in the rendezvous alphabet.

Our notion of labelling further coarsens these analyses. It is well known in formal language theory that non-injective letter-to-letter substitution is a powerful operation. The price to be paid in using it is a weakening of determinism. For example, in the automaton in the figure, if we set $cash|okay = error|no = null$, the product has become nondeterministic.

One can easily make the local automata deterministic, but since the global labelling of the product can collapse two actions into one letter, how is that action to be implemented deterministically in a local manner? We do not have any easy solutions to offer.

2 Bisimulations, renaming and systems

The branching behaviour of an automaton is its unfolding as a (possibly infinite) tree from its initial state. Each path from the initial state ending in a state q (call it a q-path) is a node of the tree, and a transition (q, a, q') contributes an a-labelled edge from this node to the node obtained by extending the path with this transition to the longer q'-path ending in state q'. The empty path yields the root node. For convenience, we assume all states are final.

[1]This example was motivated by real life: my bank's employees were on strike and some ATMs were offering a null action!

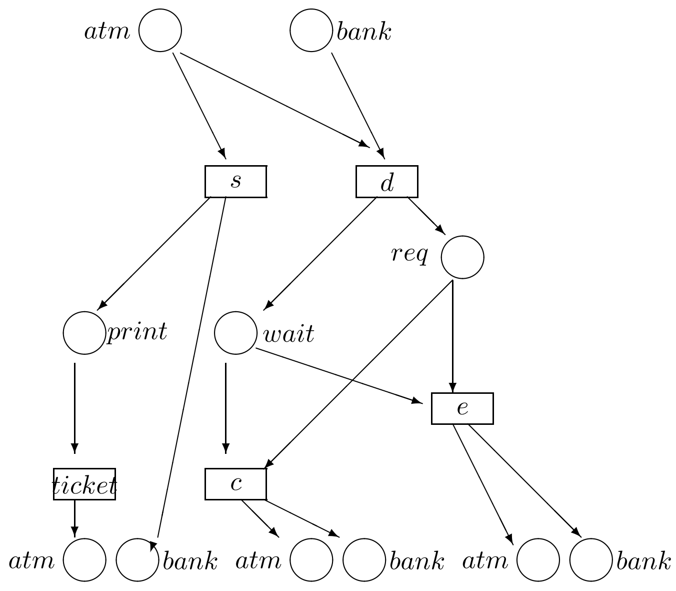

Figure 2. A product tree

Since this is too detailed a description, the unfolding is quotiented by a standard equivalence such as Park bisimilarity [34]. There is a Park bisimulation between an automaton and its unfolding: each state q of the automaton is in bisimulation with the nodes which are q-paths. For a finite automaton, this bisimulation has finite index.

Milner showed that the syntax of regular expressions is not adequate for describing finite automata upto bisimilarity [27]. Instead, he used Chomsky's type 3 grammars where one works with a system of μ-expressions. If the equations are guarded,[2] the expression $\mu x.e$ denotes the unique solution for the equation $x = e$. Formally, this can be defined using certain idempotent μ-groves [5, 13, 23], over which such solutions were shown to exist by Bergstra and Klop [6]. Here we just give Milner's axiomatization.

Axiom system M for Park bisimilarity

(Assoc)	$(e + f) + g = e + (f + g)$
(Comm)	$e + f = f + e$
(Idem)	$e + e = e$
(Ident)	$e + 0 = e$
(Assoc)	$(ef)g = e(fg)$
(LeftAbs)	$0e = 0$
(RightDistr)	$(e + f)g = eg + fg$
(Guard)	$\mu x.e = \mu x.(x + e)$
(Fixpt)	$\mu x.e = e[\mu x.e/x]$
(GuardInd)	$\dfrac{f = e[f/x]}{f = \mu x.e}$ (provided x guarded in e)

[2]One of the referees suggested using a more modern treatment like Kleene algebra [22] for a universal Horn axiomatization.

2.1 Unfoldings of product systems

Trees are the yardstick for measuring branching behaviour. The definition below is a variant of the more general definition of systems of communicating agents [24].

Definition 5 *A **product tree** $T = (N, dist, \{H_a \mid a \in C\})$ over the distributed alphabet (C, loc) is a bounded-degree directed hypergraph with nodes labelled by $dist : N \to Loc$ and hyperedges H_a labelled by $a \in C$, satisfying the conditions below. Let $T_i \overset{\text{def}}{=} (dist^{-1}(i), H^i_a)$ be the directed graph with the edge relation $uH^i_a v$ iff u is a source and v a target of a hyperedge in H_a.*

- *Each a-hyperedge has exactly one source node and one target node labelled by a location in $loc(a)$.*
- *Each T_i is a tree.*

Basically a product tree is $|Loc|$ trees connected up by "synchronization" hyperedges. It is not difficult to see that the unfolding of a product system will be a product tree U with each U_i having q-paths as nodes for states q of the i'th automaton, a set of such nodes $q_1, \ldots, q_l$ in different component trees being taken by an $[a_1| \ldots |a_l]$-labelled hyperedge to the set of nodes $q'_1, \ldots, q'_l$ when there are transitions (q_i, a_i, q'_i) in the component automata. The figure shows the initial part of the unfolding of our example product system.

While this is sufficient, it is again too detailed. So we need to quotient it by a suitable equivalence.

One of the characteristics of Park bisimilarity is that it preserves language equivalence. Since there are several different alphabets at play, we need a more flexible version here. We use a mild generalization which allows renaming and collapses to the usual definition when the alphabets are the same and the renaming is the identity function. A more sophisticated kind of renaming bisimulation is used by Kiehn and Arun-Kumar [20], in a different context.

Definition 6 *A **renaming bisimulation** (R, ρ) between two automata $(P_i, \to_i, p_i), i = 1, 2$ with edges labelled by alphabets B_1 and B_2 respectively, is a relation $R \subseteq P_1 \times P_2$ with a renaming ρ such that:*

- *The initial state p_1 is related by R to p_2.*
- *If pRq and $p \overset{a}{\to}_1 p'$ in P_1, then there are b, q' such that $a\rho b$ and $q \overset{b}{\to}_2 q'$ in P_2 and $p'Rq'$.*
- *Symmetrically, if pRq and $q \overset{b}{\to}_2 q'$ in P_2, then there are a, p' such that $a\rho b$ and $p \overset{a}{\to}_1 p'$ in P_1 and $p'Rq'$.*

Now we can define our notion of branching behaviour.

Definition 7 *Let $M^1[\rho_1], M^2[\rho_2]$ be product systems over the alphabets $G(B^1), G(B^2)$ respectively, with the labellings ρ_1, ρ_2 taking these alphabets to the distributed alphabet (C, loc). The product systems are said to be **system bisimilar** if there are renaming bisimulations between pairs of local automata M_i^1, M_i^2 over the alphabets B_i^1, B_i^2 respectively, for each i in Loc.*

A product system $M = (P_i, \rightarrow_i, p_i), i \in Loc$, is system bisimilar to its product tree unfolding U (a product tree is also a product system, possibly an infinite state one) given by the renaming bisimulations between the automata P_i and the trees U_i, which take states q to nodes corresponding to q-paths. This sanity check confirms that system bisimilarity nicely generalizes Park bisimilarity over ordinary automata.

2.2 Syntax for labelled product systems

We now turn to syntax. A straightforward extension of Milner's syntax is sufficient for our purposes.

Definition 8 ***Concurrent** μ**-terms** over the distributed alphabet $C = (G(B)/\rho, loc)$ are defined as below by beginning with Milner's μ-expressions, and allowing a single level each of parallel composition and renaming (letter-to-letter substitution). **Concurrent** μ**-expressions** are a little more general.*

$r ::= 0 \mid X \mid a \cdot r_1 \mid r_1 + r_2 \mid \mu X.r_1$
$p ::= r \mid p_1 || p_2$
$t ::= p \mid t[c/b]$
$e ::= t \mid a \cdot t \mid e_1 + e_2$

The μ-expressions r are over the action alphabet $G(B)$. The substitutions rename a letter b in $G(B)$ to a letter c in C. The metavariables t and e define the concurrent μ-terms and concurrent μ-expressions respectively.

Our basic syntax to match labelled product systems are the concurrent μ-terms. The concurrent μ-expressions are intermediate forms which arise in the axiomatization which comes a little later below.

Milner's inductive Kleene-like construction [27] gives a finite automaton $M(r_i)$ for μ-expressions r_i. To add on parallel composition for the term $r_1 || \dots || r_n$ we first set $Loc = \{1, \dots, n\}$ and define a loc function mapping letters of the alphabet according to the expressions in which they occur. For each letter a occurring in more than one expression, we define $a| \dots |a = a$ in the rendezvous structure. Putting together the individual automata gives a product system M. Now for each substitution $[b/a]$ we set $a\rho b$ in a renaming ρ, to get a labelled product system $M[\rho]$.

For the converse direction, we use closed concurrent μ-terms (those with no free variables).

Theorem 9 *Closed concurrent μ-terms describe the behaviour of labelled product systems upto system bisimilarity.*

Proof. Let the automata $M_i = (P_i, \rightarrow_i, p_i), i \in Loc$ form a labelled product system $M[\rho]$ over (C, loc). From Milner's proof [27], we have a closed μ-expression e_i such that $M(e_i)$ is Park bisimilar to M_i for each $i \in Loc$. For each global action $a_1| \dots |a_l$ in the rendezvous structure, we replace the letters $a_1, \dots, a_l$ in the expressions by a fresh letter b to get a new expression e_i' with $M(e_i')$ renaming bisimilar to M_i. The product M is system bisimilar to the parallel composition of the expressions e_i'. For each ρ-equivalence class of a global action $c \in C$ we add the substitution $[c/b]$ for each b in the equivalence class. Let $\sigma_1, \dots, \sigma_m$ be all the substitutions. $(e_1' || \dots || e_n')[\sigma_1] \dots [\sigma_m]$ is the desired concurrent μ-term e with $M(e)$ system bisimilar to $M[\rho]$. $\square$

2.3 System bisimilarity and failures

By adding Milner's expansion axiom [28] to the theory of Park bisimilarity, we reduce parallel composition to sum and axiomatize system bisimilarity over finite product systems. The axiom system below (using the broader syntax of concurrent μ-expressions) can derive associativity and commutativity of parallel composition.

Axioms MS for system bisimilarity

(M)	All valid equalities of M
(Subst)	$a[\sigma] = \sigma(a)$
(Comp)	$e[\sigma_1][\sigma_2] = e[\sigma_1 \circ \sigma_2]$
(Expan)	Let $p = \sum_i a_i p_i + \sum_j c_j p_j'$ and $q = \sum_k b_k q_k + \sum_l c_l q_l'$, with the a_i, b_k local actions which are global, and the c_j, c_l local actions which rendezvous. Then: $(p\|\|q)[\sigma] = \sum_i a_i((p_i\|\|q)[\sigma] + \sum_k b_k((p\|\|q_k)[\sigma]) + \sum_{c_j \| c_l = c_{jl} \neq 0} c_{jl}[\sigma](p_j'\|\|q_l')[\sigma]$

Theorem 10 *The axiom system MS over closed concurrent μ-terms is sound and complete for system bisimilarity of finite labelled product systems.*

Proof. Soundness of the expansion axiom follows from the definition of system bisimilarity. The completeness technique is a well-known extension of [1, 37] to process calculi [27]. We sketch the completeness proof.

Let the closed concurrent μ-terms e and f over (C, loc) denote bisimilar systems $M(e), M(f)$. The syntax does not allow $||$ to occur within μ-terms, so the axioms of μ-groves can be applied to reduce each μ-term in e and f to a sum of concatenation form, where each concatenation is a guarded prefix, that is, the left hand term of the concatenation is a single action and the right hand term (called a derivative) is a closed concurrent μ-term. At the outer level, the composition axiom is used to make one renaming out of all the substitutions. Now the expansion axiom, followed by the substitution axiom if necessary, can be applied to reduce the parallel composition of a guarded sum of prefixes to a guarded sum of prefix form. Hence e and f are both reduced to guarded sum of prefix form which is a closed concurrent μ-expression.

By using additional equations if necessary, the derivative can be replaced by a variable. Since the bisimulation is of finite index, only finitely many new equations need to be added. In fact, the number of equations needed is polynomial in e and f, with the degree of the polynomial the number of parallel operators, which is upper bounded by the alphabet size, a constant. As the pairs of automata are Park bisimilar, the summands with their prefixes must match, and equality is derivable. □

Further, as shown in [16], other weaker equivalences such as failure equivalence [10] can be axiomatized by adding a few characteristic axioms.[3]

Axioms FMS for failure equivalence	
(MS)	All valid MS equalities
(F1)	$a(bx+u)+a(by+v)=a(bx+by+u)+a(by+v)$
(F2)	$ax+a(y+z)=ax+a(x+y)+a(y+z)$

Corollary 11 *The axiom system FMS over concurrent μ-terms is sound and complete for failure equivalence of finite labelled product systems.*

Proof. For completeness, after applying the expansion and renaming axioms and reducing to a sum of prefixes, a graph saturation procedure corresponding to the two failure axioms above is applied [7] to get a stronger normal form from which equality is derivable. □

[3]As pointed out by a referee, this is failure equivalence restricted to processes without any hiding operator or τ actions.

As argued by Kanellakis and Smolka [19], bisimilarity can be checked using a polynomial time partition refinement algorithm. This result extends to system bisimilarity.

Theorem 12 *System bisimilarity of finite labelled product systems can be checked in polynomial time.*

Proof. The partition algorithms can be run for each of the local automata with the renaming. The size of the product system is the sum of the sizes of a constant number of local automata and a labelling which is upper bounded by a constant, both constants depending on the alphabet. The number of global states of the product system is polynomial, with the degree of the polynomial again depending on the alphabet. Hence the algorithms work in polynomial time. □

Corollary 13 *Failure equivalence of finite labelled product systems over a distributed alphabet (C, loc) with $|C| \geq 2$ is complete for polynomial space.*

Proof. To the system bisimilarity algorithm, the graph saturation procedure of [7] can be added to give a polynomial space algorithm. Checking failure equivalence for automata over alphabets of size at least 2 is already hard for polynomial space [19]. □

The gist of what we have done so far is that the entire framework of calculi over finite-state processes extends smoothly from ordinary finite automata to finite labelled product systems. The weakening of determinism that we discussed earlier does not affect the theory of bisimilarity since determinism is not its primary concern. This has been known for some time, but the explicit connection to product systems made here is new.

We now take a step back to language theory, and obtain an equational theory for recognizable trace languages.

3 Mazurkiewicz trace languages

Aanderaa [1] and Salomaa [37] gave the axiomatization below for equality of regular languages using the "no empty word property" (NEWP). This is a syntactically checkable condition equivalent to $a \neq 1 + a$.[4] The main change from our earlier axiom system M is the addition of left distributivity and right absorption. Completeness is proved by reducing to isomorphism over the minimal deterministic finite automaton.

[4]As pointed out by a referee, a more modern treatment uses a universal Horn axiomatization and the theory of Kleene algebras [22], which we do not go into here.

Axiom system R for regular expressions	
(Assoc)	$(e+f)+g = e+(f+g)$
(Ident)	$e+0 = e$
(Comm)	$e+f = f+e$
(Idem)	$e+e = e$
(Assoc)	$(ef)g = e(fg)$
(Ident)	$e1 = 1e = e$
(Absorp)	$e0 = 0e = 0$
(Distr)	$(e+f)g = eg+fg$
(Distr)	$e(f+g) = ef+eg$
(Guard)	$e^* = (1+e)^*$
(Fixpt)	$e^* = 1+ee^*$
(Fixpt)	$e^* = 1+e^*e$
(GuardInd)	Let e have the NEWP. Then: $\dfrac{x = ex+f}{x = e^*f}$; $\dfrac{x = xe+f}{x = fe^*}$

3.1 Syntax for recognizable trace languages

We extend regular expressions to trace languages by adding synchronized shuffle and renaming operators as before; we also need to add union at the outer level since we have to deal with final states, which we had omitted for convenience from our systems so far.

Definition 14 (Grabowski; Garg and Ragunath) ***Concurrent regular terms*** *and* ***concurrent regular expressions*** *over the distributed alphabet $C = (G(B)/\rho, loc)$ are defined as below. The regular expressions r are over the action alphabet $G(B)$. The substitutions substitute a letter c in C for a letter b in $G(B)$. The metavariables t and e define the concurrent regular terms and concurrent regular expressions respectively.*

$r ::= 0 \mid 1 \mid a \mid r_1 \cdot r_2 \mid r_1 + r_2 \mid r_1^*$
$p ::= r \mid p_1 || p_2$
$s ::= p \mid s_1 + s_2$
$t ::= s \mid t_1[c/b]$
$e ::= t \mid a \cdot t \mid e_1 + e_2$

Since regular expressions and finite automata define the regular languages over $G(B)$, the parallel composition, sum and substitution ensure that concurrent regular expressions and finite labelled product systems with final states and renaming define the same trace languages. By Theorem 4 these are the recognizable trace languages. (The theorem does not hold for labelled product systems without global final states.)

Ochmański defined a syntax for recognizable trace languages several years ago [33]. Our new syntax is equivalent, but matches labelled product systems rather directly.

The axiom system RS for regular expressions with shuffle and renaming is defined by adding axioms to the system R above exactly as the system MS was defined by adding axioms to the system M. Hence we obtain an axiomatization of the equational theory of recognizable trace languages, which was not known earlier.

Theorem 15 *The axiom system RS over concurrent regular terms is sound and complete for equality of recognizable trace languages.*

Proof. Left-distributivity, right-zero and the star axioms can be seen to be sound for regular languages. For completeness, as in the completeness proof of MS (Theorem 10), the substitution and expansion axioms are applied to reduce terms to guarded sum of prefix form. Because of left-distributivity we have a stronger normal form with the additional property that there is at most one summand for every prefix action. As in the completeness of equality for regular languages [1, 37], again variables can be used for derivatives to get a finite system of equations, with the guardedness ensuring the NEWP for each equation, and the induction rule of R applied to yield a derivation of equality. □

The equational theory of the larger class of rational trace languages is undecidable under fairly mild conditions (see [9] for a survey). It is not even known to be computably enumerable [23]. The recognizable trace languages accepted by our finite labelled product systems are included in the unambiguous rational trace languages for which equivalence is decidable [9]. This result is strengthened below by suitably modifying the algorithm for system bisimulation to provide one for trace language equivalence.

Theorem 16 *Trace language equivalence of finite labelled product systems over a distributed alphabet (C, loc) with $|C| \geq 2$ is complete for polynomial space.*

Proof. As argued in the proof for bisimulation equivalence (Theorem 12), the number of global states of the product system is polynomial in its size. From the fact that there is a polynomial space algorithm for equality of regular languages, we can put together a nondeterministic polynomial space algorithm, which is polynomial space by Savitch's theorem. For the lower bound, that the language equivalence of NFA for an alphabet of size 2 is hard for polynomial space is a classical result [39]. □

By treating the number of components as a constant, the state space of a labelled product system is polynomial in its size. So the upper bound in the theorem above could have been easily obtained, but we have not seen it anywhere earlier.

4 Remarks

We believe the key idea of separating out bisimulation at the component and system levels has applications to formal methods. Current verification technology seems to manage quite well with polynomial space algorithms.

There are a few immediate directions for future work. We have not considered silent (epsilon- or tau-) actions at all. Technically, the extension to weak bisimilarity is not difficult. By allowing a rendezvous structure which allows renaming to τ we can model the hiding operator of CSP [18] as well as the semantics of CCS synchronization [28]. But these are two different ideas: one hiding the local actions and the other hiding the global ones.

Failure equivalence, and the corresponding refinement order, are especially interesting in this setting. The FDR model checker [35] uses a two-level approach which combines both these ideas. As one of the referees pointed out, much of the power of the failures model in analyzing deadlocks is derived from abstracting out τ actions [36]. So should there be one kind of weak system bisimulation or should one allow for more?

A translation of our ideas to CSP syntax [18] seems worthwhile. A distinguishing feature of CSP is its separation of local and global choice. In our formalism, this is somewhat trivialized by the location function. If a CSP expression describes a finite-state process, our work suggests that a static analysis might be useful to connect it to a labelled product system with an appropriate rendezvous structure.

This brings us to the principal weakness of labelled product systems: they have a "flat" structure. We would like to extend our automaton-based approach to richer system structure, matching regular behaviour over a calculus like [3]. The study of process calculi with localities [11], with their corresponding enrichment of bisimulation, could be a starting point.

On the theoretical side, Sénizergues has given a formidable proof of DPDA equivalence, and Stirling has demonstrated that it can be viewed more simply through the prism of process algebra [38]. Since no models of products of DPDA exist, it seems ambitious to extend the current work to pushdown processes, but it might be worthwhile to take on the simpler model of input-driven DPDA (more recently called visibly pushdown automata), for which efficient algorithms are known [26, 2].

References

[1] S. Aanderaa. On the algebra of regular expressions, in *Appl. Math. (course notes)*, Harvard:1–18, Jan 1965.

[2] R. Alur and P. Madhusudan. Visibly pushdown languages, *Proc. STOC*:202–211, Chicago, 2004.

[3] J.C.M. Baeten and M. Bravetti. A ground-complete axiomatization of finite state processes in process algebra, *Proc. Concur*, San Francisco (M. Abadi, L. de Alfaro, eds.) *LNCS* 3653:248–262, 2005.

[4] J.C.M. Baeten and W.P. Weijland. *Process algebra*, CUP, 1990.

[5] D.B. Benson and J. Tiuryn. Fixed points in free process algebras I, *TCS* 63(3):275–294, 1989.

[6] J. Bergstra and J.W. Klop. Fixed point semantics in process algebra, *Report* IW 206/82, Centre for Mathematics and Computer Science, Amsterdam, 1982.

[7] J. Bergstra, J.W. Klop and E.-R. Olderog. Readies and failures in the algebra of communicating processes, *SIAM J. Comput.* 17(6):1134–1177, 1988.

[8] J. Bergstra, A. Ponse and S.A. Smolka, eds. *Handbook of process algebra*, Elsevier, 2001.

[9] A. Bertoni, M. Goldwurm, G. Mauri and N. Sabadini. Counting techniques for inclusion, equivalence and membership problems, in [12]:131–163.

[10] S.D. Brookes, C.A.R. Hoare and A.W. Roscoe. A theory of communicating sequential processes, *J. ACM* 31(3):560–599, 1984.

[11] I. Castellani. Process algebras with localities, in [8]:945–1045.

[12] V. Diekert and G. Rozenberg, eds. *The book of traces*, World Scientific, 1995.

[13] Z. Ésik and H. Leiß. Algebraically complete semirings and Greibach normal form, *Ann. Pure Appl. Logic* 133:173–203, 2005.

[14] W.J. Fokkink. *Introduction to process algebra*, Springer, 2000.

[15] V.K. Garg and M.T. Ragunath. Concurrent regular expressions and their relationship to Petri nets, *TCS* 96(2):285–304, 1992.

[16] R.J. van Glabbeek. The linear-time branching-time spectrum I, in [8]:3–99.

[17] J. Grabowski. On partial languages, *Fund. Inform.* IV(2):427–498, 1981.

[18] C.A.R. Hoare. *Communicating sequential processes*, Prentice-Hall, 1985.

[19] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence, *Inf. Comput.* 86(1):43–68, 1990.

[20] A. Kiehn and S. Arun-Kumar. Amortised bisimulations, *Proc. FORTE*, Taipei (F. Wang, ed.), *LNCS* 3731:320–334, 2005.

[21] N. Klarlund, M. Mukund and M. Sohoni. Determinizing asynchronous automata, *Proc. ICALP*, Jerusalem (S. Abiteboul and E. Shamir, eds.), *LNCS* 820:130–141, 1994.

[22] D. Kozen. A completeness theorem for Kleene algebras and the algebra of regular events, *Inform. Comput.* 110(2):366–390, 1994.

[23] K. Lodaya. A regular viewpoint on processes and algebra, to appear in *Acta Cybernetica*.

[24] K. Lodaya, R. Ramanujam and P.S. Thiagarajan. Temporal logics for communicating sequential agents: I, *Int. J. Found. Comp. Sci.* 3(2):117–159, 1992.

[25] A. Mazurkiewicz. Concurrent program schemes and their interpretations, Report DAIMI PB-78, Aarhus University, 1977.

[26] K. Mehlhorn. Pebbling mountain ranges and its application of DCFL-recognition, *Proc. Icalp*, Noordweijkerhout (J. de Bakker and J. van Leeuwen, eds.), *LNCS* 85:422–435, 1980.

[27] R. Milner. A complete inference system for a class of regular behaviours, *J. Comput. Syst. Sci.* 28(3):439–466, 1984.

[28] R. Milner. *Communication and concurrency*, Prentice-Hall, 1989.

[29] R. Milner. Elements of interaction, *CACM* 36(1):78–89, 1993.

[30] S.K. Mohalik. *Local presentations for finite state distributed systems*, PhD thesis, University of Madras, 1999.

[31] S.K. Mohalik and R. Ramanujam. Assumption-commitment in automata, *Proc. FSTTCS*, Kharagpur (G. Sivakumar and S. Ramesh, eds.), *LNCS* 1346:153–168, 1997.

[32] M. Mukund and M. Sohoni. Keeping track of the latest gossip in a distributed system, *Distr. Comp.* 10(3):117–127, 1997.

[33] E. Ochmański. Regular behaviour of concurrent systems, *Bull. EATCS* 27:56–67, 1985.

[34] D. Park. Concurrency and automata on infinite sequences, *Proc. 5th GI conference*, Karlsruhe (P. Deussen, ed.), *LNCS* 104:167–183, 1981.

[35] A.W. Roscoe. Model-checking CSP, in *A classical mind: Essays in honour of C A R Hoare* (A.W. Roscoe, ed.), Prentice-Hall:353–378, 1994.

[36] A.W. Roscoe. *The theory and practice of concurrency*, Prentice-Hall, 1998.

[37] A. Salomaa. Two complete axiom systems for the algebra of regular events, *J. ACM* 13(1):158–169, 1966.

[38] C. Stirling. An introduction to decidability of DPDA equivalence, *Proc. FSTTCS*, Bangalore (R. Hariharan, M. Mukund and V. Vinay, eds.), *LNCS* 2245:42–56, 2001.

[39] L.J. Stockmeyer and A.R. Meyer. Word problems requiring exponential time, *Proc. STOC*, Austin:1–9, 1973.

[40] A. Valmari. A modern mathematical theory of co-operating state machines, *Proc. Algorithmic information theory*, Vaasa, *Reports* 124:201–214, Univ. Vaasa, 2005.

[41] W. Zielonka. Notes on finite asynchronous automata, *RAIRO Inf. Th. Appl.* 21(2):99–135, 1987.