Petri nets, event structures and algebra

Kamal Lodaya
Institute of Mathematical Sciences
CIT Campus, Chennai 600 113, India
Google: Lodaya

Abstract. We define an algebraic framework for Petri nets, and prove a Myhill-Nerode theorem. As an application, we present a proof of the deterministic case of Thiagarajan's conjecture[1]. This generalizes both the trace event structure case[2] and the conflict-free case[3], for which the conjecture has been verified.

For Rani Siromoney, on the occasion of her 75th birthday.

It is more than 40 years since Petri nets have been around[4], and there have been many algebraic attempts towards understanding them. In particular, let me mention the "Petri nets as monoids" of Meseguer and Montanari[5, 6], the "process algebra with causes" of Baeten and Bergstra (cited in a later survey[7]) and the "network algebra" of Ştefănescu[8].

While I adopt ideas from all of these, what is new in this article is the emphasis on finite nets, following the classical treatment of finite automata on words and trees. I am indebted to Zoltán Ésik for helping to correct several errors in an earlier version of this article. The final article was prepared during a visit to Szeged supported by the Indian National Science Academy and the Hungarian Academy of Sciences.

1 Signatures and terms

The basic transitions of automata over words are from a state to a state. When viewing automata as algebras[9], Büchi models transitions as unary functions. Sequencing of transitions is modelled by function composition. Büchi also suggests the view of tree automata as term algebras with general k-ary functions from a signature Σ, and function composition works as before.

We generalize this slightly. The basic transitions of Petri nets[4] are multivalued functions, which take a set of i elements as argument and return a set of j elements as value.

Definition 1 *A **signature** Σ consists of a finite set (the "alphabet"), with a function assigning to each element a pre-arity and a post-arity.*

If $f^{j \longleftarrow i}$ is an i-to-j-ary symbol, the "assignment command" $\{v_1, \ldots, v_j\} := f\{u_1, \ldots, u_i\}$ is called a transition. Here $u_1, \ldots, u_i$ ("preconditions") and $v_1, \ldots, v_j$ ("postconditions") are distinct variables, which we assume come from a suitable countable set Var.

Our syntactic entities are called "programs". These will get mapped to runs of a Petri net just as words get mapped to runs of an automaton.

Definition 2 *A Σ-**program** is a sequence of transitions, satisfying the following conditions:*

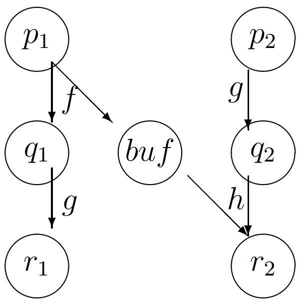
- *All the variables occurring on the right and left hand side of a transition must be distinct. (Sets of variables, not multisets.)*
- *The variables on the left hand side can only occur on the right hand side of later transitions. (A variable cannot be read and then written to.)*
- *A variable can be assigned to only once. (A variable cannot be overwritten.)*
- *A variable can appear only once on the right hand side. (Conflict is ruled out.)*

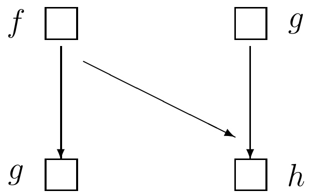
The input variables of the program are those which appear on the right hand side but not on the left hand side of any transition in the program. The output variables are those which appear on the left hand side but not on the right. We will call the rest of the variables internal and identify programs upto renaming of internal variables. The **arity** of a program is $o \leftarrow n$ if it has n input variables and o output variables. Such a program is called an $n\Sigma o$**-program**. We also use $n\Sigma$**-program** if all outputs are allowed. An $n\Sigma$**- (**$n\Sigma o$**-)language** is a set of $n\Sigma$- ($n\Sigma o$-)programs.

Programs of arbitrary length are possible, independent of $|\Sigma|$, because of the unbounded number of variables available. In a sense, every "control point" of a program is named by a unique variable.

Here is an example program. Below it are a hypergraph and a poset derived from the program, which will be defined later in this article.

$$\begin{array}{l} \{q_1, buf\} := f\{p_1\} \\ \{r_1\} := g\{q_1\} \\ \{q_2\} := g\{p_2\} \\ \{r_2\} := h\{buf, q_2\} \end{array}$$





This seems fairly straightforward, but the difficulty lies in formalizing the *composition* $r; s$ of two programs (assuming the result is a program), which we have done in the example by sequencing s after r.

In earlier work[10], we introduced the notion of a *series* Σ*-algebra* where one Σ-term s can be sequenced after another r. But the semantics there requires that the execution of r be complete before the execution of s begins.

This is inadequate to model the N-shaped execution in the example above. If r has post-conditions $J \cup J_1$ and s takes preconditions J_1 to postconditions K, then their composition has postconditions $J \cup K$. Conversely, if we sequence a program s after r, we get a composition depending on the relationship between the postconditions of r and the preconditions of s. (This was studied by Goltz and Reisig[11].)

Simple projection operations π_k, $k \in \mathbf{N}$, do not work either. The names of the variables in J_1 are crucial to making the definition of composition unambiguous. But it is not *really* the names which are important, they are just "tokens" or "place-holders" !

Earlier authors have grappled with this problem in different ways: Meseguer and Montanari[5] move to categories, Ştefănescu[8] uses operators for permuting a tuple, and Baeten and Bergstra[7] introduce an explicit set of *causes* similar to our variables. The solution adopted here is to go back to Büchi's view as transitions as unary functions[9], and declare that given an $n\Sigma o$-program r and an i-to-j transition f, $i \leq o$, there are $\binom{o}{i}$ different compositions $r;_C f$, with C ranging over the combinations. (Thus the effect of tuple permutation[8] is achieved.) Our programs seemingly use only one composition, but the notation disambiguates the different compositions.

Remark. Suppose r is the transition $\{w_1\} := f_1\{v_1\}$ and s the transition $\{w_2\} := f_2\{v_2\}$. We say the two programs are independent since they work with disjoint sets of variables. This example suggests representing programs by Mazurkiewicz traces[12]. However, note that f_1 and f_2 may or may not be independent in different contexts, so independence is not coded into the action label. The two g actions in the picture above are independent.

2 Algebras and recognizability

We now interpret our signature not just as multiple-valued functions, but as functions from multisets to multisets. We use the notation $\wr \ldots \int$ to denote multisets and the usual $\{\ldots\}$ for sets. $X^{\wr i \int}$ denotes multisets (unordered tuples) with i elements from the set X.

Definition 3 *A Σ-**multialgebra** consists of a nonempty domain X partitioned into sorts $X^k, k \in$ $\boldsymbol{N}$, and for each operation $f^{j\longleftarrow i}$ of Σ, an interpretation, a function $\mathcal{I}(f) : X^i \to X^j$, which extends for each $k \geq i$ and each C in $\binom{k}{i}$, to $\mathcal{I}(f_C) : X^k \to X^{k-i+j}$ by choosing the i variables using C. An $n\Sigma$-**multialgebra** has in addition a distinguished element I of X^n, and an $n\Sigma o$-**multialgebra** further has a distinguished subset O of X^o. For a **symmetric** Σ-multialgebra (possibly with distinguished input and output), we take unordered tuples instead of ordered ones (i.e., $X^{\wr m \int}$ instead of X^m) in the statements above.*

The interpretation $\mathcal{I}$ can be extended in a unique way to programs, disambiguating program composition as explained above.

This definition is related to strictly symmetric monoidal categories[5, 6]. But here we make do with just the composition operation rather than introducing an additional tensor product, and stay within the ambit of algebra rather than venturing into category theory.

The program Σ-multialgebra has domain the set of programs, and $\mathcal{I}(f)$ for an i-to-j operation f extends its argument program by sequencing the transition $\{v_1, \ldots, v_j\} := f\{u_1, \ldots, u_i\}$ after the program, where the names of the variables on the right are chosen from the output variables according to the combination C and the names on the left are fresh ones.

Definition 4 *A **homomorphism** from a Σ-multialgebra to another is a function h which preserves the operations in the sense that $h(f_C(t)) = f_C(h(t))$. If, in addition, the signature has distinguished elements, h must preserve them: I must map to $h(I)$ and if $t \in O$ in the first algebra, then $h(t) \in O$ in the second.*

Our main interest in this article is homomorphisms which map into finite multialgebras. We require the domain of such a multialgebra to be finite. This is achieved by mapping infinitely many sorts X^k into a finite set. If, in addition, we have that there is an m such that every X^{m+j} maps to a single zero element, we call the multialgebra nilpotent.

Definition 5 *An $n\Sigma(n\Sigma o)$ language L is* ***recognizable*** *if there is a homomorphism h of the $n\Sigma(n\Sigma o)$ programs into a finite $n\Sigma(n\Sigma o)$ multialgebra (with a designated subset F of sort o such that for each term t, $t \in L$ iff $h(t) \in F$).*

From the definition, it is clear that a recognizable $n\Sigma$-language is prefix-closed; but an $n\Sigma o$-language need not be.

3 Nets and algebras

A Σ-program can be represented as a hypergraph $\mathcal{S} = (B, \{E_f | f \in \Sigma\})$, where B is the set of variables in the program, and each transition $\{v_1, \ldots, v_j\} := f\{u_1, \ldots, u_i\}$ is modelled by a directed hyperedge in E_f from the source nodes $u_1, \ldots, u_i$ to the target nodes $v_1, \ldots, v_j$ where f has arity $j \leftarrow i$. The conditions on a Σ-program ensure that the hypergraph is acyclic and unbranched, that is, two different hyperedges do not share source and target vertices. α-conversion yields an isomorphic hypergraph. Hypergraphs whose labelling respects the signature Σ are called Σ-hypergraphs.

Conversely, from an acyclic unbranched Σ-hypergraph we can easily write a Σ-program corresponding to it by using its nodes as names of variables.

We can define the width of a hypergraph as the size of the largest "cut" that can be made through a pictorial representation of its set of nodes without intersecting any hyperedges. A Σ-language is said to be **bounded width** if there is a bound $k \in \mathbf{N}$ such that the width of the hypergraphs corresponding to each program in it is at most k.

A hypergraph can in turn be represented as a bipartite graph $\mathcal{K} = (B, E, F, l)$ where B is as before a "sort" of nodes and E is a "sort" of transitions (hyperedges). The source and target nodes are connected by an incidence relation $F \subseteq (B \times E) \cup (E \times B)$ as follows: for the hyperedge e representing the transition above, there are edges in F from the source nodes $u_1, \ldots, u_i$ of the hyperedge to e, and from e to the target nodes $v_1, \ldots, v_j$ of the hyperedge. l labels nodes of sort E by letters from Σ: the vertex e of the graph above is labelled f.

Such graphs are known in the literature as (labelled finite) causal nets. B is the set of basic conditions of the net, E is the set of events and F is called the flow relation. A causal net is unbranched, that is, all its basic conditions b have at most one predecessor and one successor, and acyclic: the reflexive transitive closure of F is antisymmetric (that is, a partial order). Since $l : E \to \Sigma$ is an arity-respecting labelling, we call them causal Σ-nets. They give a purely relational representation of an acyclic unbranched Σ-hypergraph.

Petri nets

Clearly a (finite) symmetric Σ-multialgebra with domain P and interpretation T can be represented as a (finite) hypergraph $(P, \{T_f | f \in \Sigma\})$, which need be neither acyclic nor unbranched.

A relational representation of such a hypergraph is more elaborate: a bipartite weighted graph $\mathcal{N} = (P, T, W, \ell)$ where P is a set of places, T a (disjoint) set of transitions, ℓ an arity-respecting labelling of the transitions and $W : (P \times T) \cup (T \times P) \to \mathbf{N}$ defines a weighted flow relation which models hyperedges.

$\mathcal{N}$ is traditionally called a (labelled finite) Petri net[4]. For a place or transition y, its pre-set $\{x \mid W(x, y) > 0\}$ is conventionally denoted ${}^\bullet y$ and its post-set $\{z \mid W(y, z) > 0\}$ is denoted $x^\bullet$. W satisfies the condition that for each transition t, ${}^\bullet t$ and $t^\bullet$ are nonempty, and for each place p, either ${}^\bullet p$ or $p^\bullet$ is nonempty. For all transitions t, the arity of $\ell(t)$ is from the indegree of t to its outdegree $(\sum_q W(t, q) \leftarrow \sum_p W(p, t))$. If a net satisfies this condition, we call it a Σ-net.

A marking is a multiset of places. A net system is a net with a set of initial markings (and possibly a set of final markings). A finite $n\Sigma$ ($n\Sigma o$) multialgebra can be represented as a net with a single initial marking (and a set of final markings).

A "run" of a net system is described by a "token game" from an initial marking (to a final marking)[4]. A net system is said to be k-safe if for all markings M reachable from the initial markings and for all places p, p occurs at most k times in M. As a consequence, the weight of the flow relation of a 1-safe system can only be 0 or 1 and we can redefine flow as a subset of $(P \times T) \cup (T \times P)$. We adopt a more abstract definition[13].

Definition 6 *A net system $\mathcal{N}$ **accepts** an $n\Sigma$-program r if there is a homomorphism from r to the net system, seen as an $n\Sigma$-multialgebra.*

Conversely, any (finite) Σ-net is a hypergraph and a (finite) *deterministic* Σ-net is a symmetric partial Σ-multialgebra with domain $\bigcup_k P^k$ and a partial interpretation given by the transitions T. As for automata, a net is said to be deterministic if at any marking, at most one transition with the same label can be "fired". A net system is deterministic if, in addition, it has a single initial marking. If the net is k-safe, we can turn it into a nilpotent Σ-multialgebra by adding a zero element.

Remark. We can also model unlabelled nets; equip them with a labelling $\ell : T \to T$ which is the identity function. To make T into a signature, each transition is given an arity from its indegree to its outdegree. Then any (finite) unlabelled net is deterministic and is a (finite) symmetric partial T-algebra.

Subset algebras

Thus a net might be thought of as a multialgebra which allows nondeterministic operations. More formally, we represent a net as a **subset multialgebra**. Let $\mathcal{N} = (P, T, W, \ell)$ be a finite $n\Sigma$-net system, with set of initial markings $I \subseteq P^n$. The required algebra has as the domain of sort k all subsets $U \subseteq P^k$ of markings of $\mathcal{N}$ of size k. The interpretation of an operation $f^{j \leftarrow i}$ is given by $h(f)$, defined to take $U \subseteq P^i$ to the set V of markings M_2 such that there is a marking $M_1 \in U$ and an f-labelled transition of $\mathcal{N}$ from the marking M_1 to the marking M_2. Since Σ is a signature, f can only take the marking M_1 to a marking of size j, hence $V \subseteq P^j$. For every Σ-program r, $h(r)$

is the unique extension of h under composition starting from the subset I. h will map a program r to the set of markings reachable after executing r. It will map an $n\Sigma o$-program accepted by $\mathcal{N}$ to a subset of $O \subseteq P^o$.

The proof that the subset multialgebra accepts the same language as the net follows that of the subset construction for automata.

The subset algebra will be symmetric since the markings of a net are just multisets. The elements of this algebra (that is, subsets of markings) which consist of only reachable markings we will call **accessible**.

4 Congruences

An equivalence relation $\equiv$ on an algebra is said to be a congruence if it preserves the operations of the algebra. Since our operations are defined on multisets, we have to lift congruences accordingly. We say $\wr x_1, \ldots, x_i \int \equiv \wr x'_1, \ldots, x'_i \int$ if $x_m \equiv x'_m$ for $1 \leq m \leq i$. Conversely, given two such equivalent multisets of size i, we require orderings $1, \ldots, i$ of the two multisets such that $x_m \equiv x'_m$ for $1 \leq m \leq i$.

Definition 7 *An equivalence relation $\equiv$ on a Σ-multialgebra is a* ***Σ-congruence*** *if for every operation $f^{j \longleftarrow i}$ in Σ, $k \geq i$, $C \in \binom{k}{i}$, if $t \equiv t'$ then $t;_C f \equiv t';_C f$. It is an* ***$n\Sigma$-congruence*** *if there is a distinguished congruence class e of the domain of sort n such that for any t, $e;_C t \equiv t$.*

The canonical map from an element of the domain of a Σ-multialgebra to its congruence class is a homomorphism.

From the definition of congruences, it is clear that they are closed under intersections. For two congruences of finite index, the smallest congruence which includes the union of both is a congruence of finite index.

The Myhill-Nerode congruence for an $n\Sigma$-language L is defined by: $r \approx_L s$ if for all right contexts t, we have $r;t \in L$ iff $s;t \in L$. It is the maximal congruence on $n\Sigma$-programs which saturates L.

We can now prove a Myhill-Nerode theorem.

Theorem 8 *The following classes of bounded width $n\Sigma$-languages are equivalent:*

(1) The set of $n\Sigma$-programs accepted by a finite $n\Sigma$-net system.

(2) The $n\Sigma$-language which is the inverse homomorphic image of a subset, not containing the zero, of a finite symmetric nilpotent $n\Sigma$-multialgebra.

(3) The $n\Sigma$-language saturated by a finite index $n\Sigma$-congruence.

(4) The $n\Sigma$-language whose Myhill-Nerode congruence has finite index.

(5) The $n\Sigma$-language accepted by a finite 1-safe deterministic $n\Sigma$-net system.

Proof. For the implication from (1) to (2), let $\mathcal{N}$ be a finite $n\Sigma$-net system accepting L, with set of initial markings I. We take the subset multialgebra for $\mathcal{N}$, which will also accept L. Since L is bounded width, we can collapse all the sorts beyond the bound into a single zero element. As $\mathcal{N}$ is finite, the set of possible markings is finite and the algebra is finite and nilpotent. The accessible elements of this algebra are the desired subset, not containing the zero, which L will map to.

For the implication from (2) to (3), we take the kernel of the homomorphism h saturating L.

(3) to (4) follows from maximality of $\approx_L$.

For the implication from (4) to (5), we need to define a finite 1-safe deterministic $n\Sigma$-net system $\mathcal{N}$ accepting L.

The places of this net are going to be congruence classes of elements of the domain.

Let the congruence class of the distinguished element e be $M_0 = \{p_1, \dots, p_n\}$. This will be the initial marking of the net. We say the empty program (of signature $n\Sigma$) in $\mathcal{N}$ uses the places $p_1, \dots, p_n$ for the congruence class $[\{v_1, \dots, v_n\}]$. (Formally, this means extending the syntax of programs by this one program.)

Inductively, suppose an $O := \Sigma(I)$ program r of arity $o \leftarrow n$ using the places $p_1, \dots, p_q$ has $O = \{u_1, \dots, u_i, \dots, u_o\}$. Consider now the $O' := \Sigma(I)$ program $s = (r; \{u'_1, \dots, u'_j\} := f\{u_{i+1}, \dots, u_o\})$ with $O' = \{u_1, \dots, u_i, u'_1, \dots, u'_j\}$. This defines a transition of the net labelled f. Its pre-set is the set of places corresponding to the multiset $[\{u_{i+1}, \dots, u_o\}]$ with the weight of a place the number of u's it corresponds to, and the post-set is the set of places corresponding to $[\{u'_1, \dots, u'_j\}]$ with the weight defined similarly. Clearly the transition respects Σ-labelling.

Suppose s has a prefix s_1 (that is, $s = s_1; s_2$) which it is congruent to—that is, s_1 is an $O' := \Sigma(I)$ program with $O' = \{v_{k_1}, \dots, v_{k_{i+j}}\}$ given by the congruence class of places $p_{k_1}, \dots, p_{k_{i+j}}$. Then we will also have $[\{u_1, \dots, u_i, u'_1, \dots, u'_j\}] = \wr p_{k_1}, \dots, p_{k_{i+j}} \int$. But if some of the v_{k_m} are new, the set of places used has to be extended beyond $p_1, \dots, p_q$.

In this way the infinite set of terms will map into a finite set of places, depending on the number of Myhill-Nerode congruence classes. Since the congruence is finite index, the net is finite.

Concurrency is separated out right from the initial marking $[\{v_1, \dots, v_n\}] = \{p_1, \dots, p_n\}$ (no repeated elements). The only thing that separates p_1 from p_2 is that the former is concurrent to $p_2, p_3, \dots, p_n$ and the latter is concurrent to $p_1, p_3, \dots, p_n$. The program $v' := g(v_1)$ will yield a g-transition from $[v_1]$ to $[v']$. $[v_1]$ maps to one of the places $p_1, \dots, p_n$ by the ordering chosen by the congruence relation. There is no nondeterminism.

In fact, the program $r; \{v'_1, \dots, v'_j\} := f\{u_{i+1}, \dots, u_n\}$ can be renamed to the program s above, so only one transition labelled f can be enabled at the multiset of places $[\{u_{i+1}, \dots, u_o\}]$. Hence the net is deterministic.

Finally, since the net system $\mathcal{N}$ accepts L of bounded width, there is a k-safe net system accepting L, for some k. Best and Wimmel[14] show how to produce a 1-safe net system accepting the same poset language. Both their "colouring" and "unfolding" constructions will preserve deterministic labelling. Their "balancedness" condition ensures that the result is a Σ-net. $\square$

As in the case of finite automata, the Myhill-Nerode theorem also provides a determinization. This is not that interesting a result because the usual "branching" behaviour for Petri nets is stricter than just a language of programs. In concurrency jargon, $a(b + c)$ is distinguished from $ab + ac$. We turn next to this notion of behaviour.

5 Event structures

A (labelled) **event structure** $ES = (E, \leq, \#, \ell)$ is a (labelled) poset $(E, \leq, \ell)$ with an irreflexive symmetric conflict relation $\#$ on E, which is "inherited," that is, $e_1 \# e_2 \leq e_3$ implies $e_1 \# e_3$. Two events e_1 and e_2 of ES are said to be concurrent if they are not related by $\leq$, $\geq$ or $\#$.

We will assume an event structure to be finitary (for each element e, its left-closure $\downarrow e = \{x \mid x \leq e\}$ of elements preceding it is finite). This implies that the conflict relation is generated from an immediate conflict relation $\#_\mu$, where $e_1 \#_\mu e_3$ if $e_1 \# e_3$ and there is no e_2 such that $e_1 \# e_2 \leq e_3$ (and hence by symmetry no e_2 such that $e_3 \# e_2 \leq e_1$).

An event structure is said to be deterministic if $e_1 \#_\mu e_2$ implies $\ell(e_1) \neq \ell(e_2)$. An event structure is conflict-free if its conflict relation is empty. Clearly a conflict-free event structure is deterministic.

A deterministic event structure is said to be a trace event structure if there is a symmetric "dependence" relation over the set of labels such that events related by the immediate successor relation or immediate conflict relation are dependent, and conversely, dependent events are not concurrent.

A **configuration** c of ES is a left-closed subset of E ($\downarrow c = c$) which is conflict-free (that is, a poset). The "remainder" of the event structure $ES \setminus c$ is said to be a **residue** of ES. By the finiteness of left-closure, a configuration is a finite poset, but residues can of course be infinite.

From a configuration c of a Σ-labelled event structure, we can define the hypergraph $(H, \{E_f | f \in \Sigma\})$, where H is the set of edges in the Hasse diagram of c and E_f takes ${}^\bullet e$ to $e^\bullet$ precisely when e is labelled f. Since the hypergraph is acyclic and unbranched, we can further define a program t. We write $c = forget(t)$. Since the labelling of the event structure need not form a signature, t might not be a Σ-program.

It is well known[15] that a (labelled) net $\mathcal{N}$ "unfolds" into a (labelled) event structure $Unf(\mathcal{N})$. We say a (labelled) event structure ES is generated by a (labelled) net $\mathcal{N}$ if ES is isomorphic to $Unf(\mathcal{N})$.

Now we lift some definitions from infinite trees to event structures. Two configurations c and d are right-invariant (we write $c \approx_R d$) if $ES \setminus c$ and $ES \setminus d$ are isomorphic as event structures. Clearly this is an equivalence relation. An infinite event structure ES is regular if the right-invariance has finite index.

An event e is said to be enabled at a configuration c if $e \notin c$ and $c \cup \{e\}$ is a configuration. An event structure is said to have bounded enabling if there is a $b \in \mathbf{N}$ such that at any configuration of the event structure, the number of events enabled is bounded by b. Thiagarajan conjectured[1] that a well-known result linking regular infinite trees to tree automata generalizes to event structures with bounded enabling.

Conjecture 1 *(Thiagarajan's conjecture) An infinite regular event structure with bounded enabling is generated by a finite 1-safe net system.*

Thiagarajan proved the conjecture for trace event structures[2]. More recently, the conjecture was proved for the conflict-free case[3].

We use our Myhill-Nerode result to give a proof of the deterministic case of Thiagarajan's conjecture. That the trace case follows from this is easy to see. As a corollary of our theorem, we also prove the conflict-free case.

Hence we generalize both cases. The proof does not extend to a general event structure because the Myhill-Nerode theorem collapses nondeterminism to determinism.

Theorem 9 *(Deterministic case) An infinite regular deterministic event structure with bounded enabling is generated by a finite 1-safe net system.*

Proof. Let $ES = (E, \leq, \#, \ell)$ be the given deterministic event structure with $\ell : E \to \mathit{Dir}$ ("directions"), and R be the finite set of right-invariance classes of configurations of ES.

We refine this labelling to a new labelling $\lambda : E \to \Sigma$, where $\Sigma = \mathit{Dir} \times R$. To each element e of E we let $\lambda(e)$ have, in addition to $\ell(e)$, the right-invariance class of $\downarrow\!e$ in R. Notice that since we use isomorphism to define right-invariance, a residue is different from two copies of itself (separated by conflict or concurrency). Thus, in concurrency jargon, the event structures for ab, $a(b+b)$, $a(b+b+b)$, $a(b||b)$ and $a(b||b+b)$ will have different sets of residues.

We still have to determine the arities for Σ. Let c be a configuration. Consider any $e \in c$. Let j be the maximum size of a configuration in $ES \setminus c$ which has elements of $e^\bullet$. This is well defined since ES has bounded enabling. The post-arity of $\lambda(e)$ is defined to be j.

Consider now a configuration $c \cup \{e\}$ where the post-arity of all $\leq$-maximal events in c has been determined. Let i be the sum of the post-arities of $\leq$-maximal elements in c which are in ${}^\bullet e$. The pre-arity of $\lambda(e)$ is defined to be i. Determine its post-arity as above. Because of the isomorphism defining the right-invariance, the arities of equivalent elements will be the same.

All that remains now is assigning a pre-arity to the $\leq$-minimal events e in case no other element is labelled with $\lambda(e)$. In this case, we set the pre-arity of e to be 1.

Hence $\Sigma = \mathit{Dir} \times R$ with these arities is a signature and ES can be seen as an event structure labelled by a signature.

Let n be the maximal sum of pre-arities of such an "initial" configuration (which only has $\leq$-minimal events). Inductively, each configuration is represented by a $n\Sigma$-program. We lift $\approx_R$ to $n\Sigma$-programs. For programs x and y such that $c = forget(x)$ and $d = forget(y)$, we let $x \approx_R y$ if $c \approx_R d$. We map programs which do not represent configurations of ES to new zero elements of the appropriate "type" (pair of arities), which are congruence classes by themselves. (This is a finite set since Σ is finite.) $\approx_R$ is an equivalence relation on $n\Sigma$-programs of finite index; it remains to show that it is a congruence.

Suppose $u_k \approx_R u'_k$, i.e., $ES \setminus forget(u_k)$ and $ES \setminus forget(u'_k)$ are isomorphic, for $1 \leq k \leq i$, and $\{v_1, \ldots, v_j\} := f\{u_1, \ldots, u_i\}$ for f in Σ. This corresponds to a configuration with a new maximal element of arity $j \leftarrow i$ labelled f. Either such an element does not exist in the event structure and both programs map to zero, or by right-invariance, $ES \setminus f\{u_1, \ldots, u_i\}$ is isomorphic to $ES \setminus f\{u'_1, \ldots, u'_i\}$. In fact, there is some ordering of variables $v'_1, \ldots, v'_j$ such that $\{v'_1, \ldots, v'_j\} := f\{u'_1, \ldots, u'_i\}$, and $ES \setminus v_k$ is isomorphic to $ES \setminus v'_k$ for $1 \leq k \leq j$. Hence the right-invariance is a congruence.

Since the event structure has bounded enabling, at any configuration at most a bounded number of events can occur in parallel. Hence the size of its antichains is bounded and the corresponding set of $n\Sigma$ programs has bounded width.

Now applying Theorem 8, there is a finite 1-safe deterministic $n\Sigma$-net system $\mathcal{N}$ accepting the language L of configurations of ES.

We use determinism to argue that ES is isomorphic to $Unf(\mathcal{N})$. Since ES is deterministically labelled by Dir, it is deterministically labelled by Σ, and each extension of a configuration is represented by a distinct program in L. The way the maximal events of the different extensions are related to each other can also be determined from L. Hence ES is determined upto isomorphism. The restriction $\mathcal{N}'$ of $\mathcal{N}$, where the R-component of the labelling is forgotten, generates ES. □

We can also use our result to give an alternate proof of the conflict-free case of Thiagarajan's conjecture, which was recently shown[3].

Corollary 1 *(Conflict-free case) An infinite regular conflict-free event structure with bounded enabling is generated by a finite 1-safe forward unbranched net system.*

Proof. Applying Theorem 9, we get a finite unlabelled 1-safe net system generating ES.

Suppose the net obtained is forward branched. That is, there is a place p with at least two transitions t_1, t_2 in $p^\bullet$. (That is, $t_1 \#_\mu t_2$.) Then there is a program accepted by the net in which the transition t_i occurs once but t_j does not occur (for $i \neq j \in \{1, 2\}$), but there is no program accepted by the net in which t_1, t_2 both occur once.

Corresponding to each of these programs, there is a configuration of ES in which the event corresponding to transition t_i occurs once as a maximal event but that corresponding to t_j does not occur (for $i \neq j \in \{1, 2\}$). But then there is a configuration of ES in which both these events are maximal and the program for this configuration too must be accepted by the net, a contradiction.

Hence the net is forward unbranched. □

Remarks

The earlier proofs of Thiagarajan and Nielsen[2, 3] take up several pages of difficult combinatorial argument, and have an explicit treatment of concurrency in the form of a trace labelling. The arguments in our two main theorems separate out into congruences and their labellings. The combinatorics and concurrency is left to the proof of Best and Wimmel[14] which we use. It would be interesting to find a proof of (4) to (5) in Theorem 8 which is independent of this.

Our approach should be of interest to the process algebra community. In particular, one can attempt to tackle the general conjecture by working with a syntax of terms with an explicit sum operation.

References

[1] P.S. Thiagarajan. Regular event structures and finite Petri nets: a conjecture, in *Formal and natural computing – essays dedicated to Grzegorz Rozenberg* (W. Brauer, H. Ehrig, J. Karhumäki and A. Salomaa, eds.), LNCS 2300 (2002) 244–256.

[2] P.S. Thiagarajan. Regular trace event structures, BRICS Research Abstracts (1996), http://www.brics.dk/RS/96/32/BRICS-RS-96-32.ps.gz.

[3] M. Nielsen and P.S. Thiagarajan. Regular event structures and finite Petri nets: the conflict-free case, *Proc. ICATPN*, Adelaide (J. Esparza and C. Lakos, eds.), LNCS 2360 (2002) 335–351.

[4] C.-A. Petri. Fundamentals of a theory of asynchronous information flow, *Proc. IFIP*, Munich (C.M. Popplewell, ed.), North-Holland (1962) 386–390.

[5] J. Meseguer and U. Montanari. Petri nets are monoids, *Inform. Comput.* 88 (1990) 105–155.

[6] J. Meseguer, U. Montanari and V. Sassone. Representation theorems for Petri nets, in *Foundations of computer science — festschrift for W. Brauer* (C. Freksa, M. Jantzen and R. Valk, eds.) LNCS 1337 (1997) 239–249.

[7] J.C.M. Baeten and T. Basten. Partial order process algebra (and its relation to Petri nets), in *Handbook of process algebra* (J.A. Bergstra, A. Ponse and S.A. Smolka, eds.), Elsevier (2001) 769–872.

[8] G. Ştefănescu. *Network algebra*, Springer (2000).

[9] J.R. Büchi. *Finite automata, their algebras and grammars: Towards a theory of formal expressions* (D. Siefkes, ed.), Springer (1989).

[10] K. Lodaya and P. Weil. Rationality in algebras with a series operation, *Inform. Comput.* 171,2 (2001) 269–293.

[11] U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets, *Inform. Comput.* 57 (1983) 125–147.

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

[13] W. Thomas. Uniform and nonuniform recognizability, *TCS* 292 (2003) 283–298.

[14] E. Best and H. Wimmel. Reducing k-safe Petri nets to pomset-equivalent 1-safe Petri nets, *Proc. ATPN*, Aarhus (M. Nielsen and D. Simpson, eds.), LNCS 1825 (2000) 146–165.

[15] M. Nielsen, G.D. Plotkin and G. Winskel. Petri nets, event structures and domains I, *TCS* 13 (1981) 85–108.