A Polynomial Time Flow for Implementing Free-Choice Petri-Nets

Pavlos M. Mattheakis, Christos P. Sotiriou
FORTH-ICS, Crete, Greece and
University of Crete, Greece.
emails: pmat@csd.uoc.gr, sotiriou@csd.uoc.gr

Peter A. Beerel
Department of Electrical Engineering,
University of Southern California, USA.
email: pabeerel@usc.edu

***Abstract*—FSM and PTnet control models are pertinent in both software and hardware applications as both specification and implementation models. The state-based, monolithic FSM model is directly implementable in software or hardware, but cannot model concurrency without state explosion. Interacting FSM models have so far lacked the formal rigor for expressing the synchronising interactions between different FSMs. The event-based, PTnet model is able to model both concurrency and choice within the same model, however lacks a polynomial time flow to implementation, as current methods of exposing the event state space require a potentially exponential number of states. In this work, we present a polynomial complexity flow for transforming a Free-Choice PTnet into a new formalism for Interacting FSMs, *i.e* Multiple, Synchronised FSMs (MSFSMs), a compact Interacting FSMs model, potentially implementable using any existing monolithic FSM implementation method. We believe that such a flow can in the long term bridge the event and state-based models. We present execution time and state space results of exercising our flow on 25 large PTnet specifications, describing asynchronous control circuits, and contrast our results to the popular Petrify tool for PTnet state space exploration and circuit implementation. Our results indicate a very significant reduction in both state space size and execution time.**

I. Introduction

The monolithic Finite State Machine (FSM) model is the protagonist of contemporary formal control models. The monolithic FSM model's simplicity, its seemingly vast expressive power, combined with its implementability, include some of the key reasons for its popularity in the fields of hardware and software design. In the software field, the FSM model is used by and within Computer Aided Software Engineering (CASE) tools to represent a software application's control properties or dependencies, where inputs typically represent stimuli and outputs actions. Several, well known techniques exist, in the literature, for implementing an FSM in software [1]. The straightforward approach for translating an FSM into programming language code is to to represent FSM states by state variables, and use two case statements for determining the output and next state values, *i.e.* one for current states and another for the inputs. In the hardware field, the FSM model represents the atomic unit for specifying and implementing sequential circuits. In the FSM circuit implementation literature, output signal timing, state changes and input sampling are usually synchronised to a clock. However, asynchronous FSM implementations have also been proposed, which typically impose assumptions on input arrival, output departure, and state changes. Similarly to software, a variety of algorithms exist for optimising and implementing an FSM both in the synchronous [2], [3], [4], as well as the asynchronous paradigm [5], [6].

Despite being ubiquitously used, and rich in implementation options, the monolithic FSM model is not scalable when a single FSM is used for representing, or analysing a concurrent system. The exponential state explosion manifested in such a case can render the use of the, otherwise very powerful, formalism of the monolithic FSM as prohibitive. Several software and hardware applications have been indeed affected by this shortcoming, including formal verification and circuit design, ultimately leading to research in higher-order control models. For example, model checking a concurrent specification comprised of multiple FSMs, requires the computation of the monolithic, composite (product) FSM [7], the size of which is exponential with respect to the total number of original states. Another example lies in the field of asynchronous circuit synthesis [8], whereby the state space of the circuit's input and output signals is explored, for the purpose of logic synthesis, as a monolithic FSM, the size of which is exponential with respect to the number of circuit signals. This motivation for higher-order control models has led to the exploration of models such as Interacting FSMs [9] and Place Transition nets (PTnets) [10].

Interacting FSMs or FSM networks are an evolution of the single FSM. They essentially represent a compact, non-exponential equivalent of a monolithic FSM, whereby multiple FSMs interact with each other, through designated output and input signals or transitions. As Interacting FSMs are an evolutionary model, they straightforwardly inherit the properties of monolithic FSMs, such as synchronous or asynchronous timing. However, interacting FSMs lack, so far, the formal rigor for expressing synchronisation, as well as the analysis capabilities of PTnets. PTnets represent an graph-based, event-oriented alternative control model, in contrast to the state-oriented FSM models. PTnets exhibit certain specific benefits, including the combination of concurrency and choice within the same model, as well as, more importantly, static analysis algorithms for verifying important control system properties, such as deadlock-freedom and boundedness. The PTnet model, being event-based, is typically considered asynchronous in nature. However, with the introduction of assumptions similar to those of synchronous FSMs, *i.e.* when input and output events are fired with respect to a clock signal, the PTnet model may also be used to specify or implement synchronous control systems [11]. The sole,yet significant, drawback of PTnets is that they do not currently possess a low-complexity path to implementation, similar to that of state-based models.

Hence, whether under synchronous or asynchronous timing, the monolithic FSM model possesses a low-complexity implementation path, whereas PTnets do not. Their intermediate counterpart, interacting FSMs, lack the formal expressiveness to express synchronisation, this is why they are used the least. A number of prior works [12], [8] and [11] have attempted to transform PTnets into a monolithic or multiple FSMs respectively, however they either suffer from state explosion or cannot guarantee the implementability of any live and bounded PTnet.

In this work, we introduce a polynomial complexity flow capable of transforming any Free-Choice PTnet into a set of Synchronised FSMs (MSFSMs), a compact interacting FSMs model which we introduce in this paper, capable of exposing state and expressing inter-FSM synchronisation, while being potentially implementable with any existing monolithic FSM implementation technique. We present state space and execution time results of our flow, on a set of 25 benchmarks, and comparison to results from PTnet implementation tool Petrify. Results indicate a significantly smaller state space footprint for our flow, due to its polynomial complexity, as well as a respective reduction in execution time, for exposing this state space.

II. Control Models

A. Monolithic and Interacting FSMs

Since the 1970's there has been an extensive body of work in FSM-based design [2][13]. Early work focused on state minimisation, encoding and implementation of completely specified FSMs. In the mid 1990's, mature algorithms were developed for incompletely specified FSMs [3].

Definition 2.1 (FSM): An FSM, M, is a five-tuple, M = $(I, O, S, \delta, \lambda)$, where I is a finite, nonempty set of inputs, O is a finite, nonempty set of outputs, S is a finite nonempty set of states, $\delta : I \times S \rightarrow S$ is the next state function, and $\lambda : I \times S \rightarrow O$ (for a Mealy machine), or $\lambda : S \rightarrow O$ (for a Moore machine) is the output function.

All practical FSMs, possess an initial state S_0, assumed during system initialisation. If, the next state and output functions, δ and λ, are specified for all possible inputs, then the FSM is completely specified, otherwise the FSM is incompletely specified with next state and output Don't Cares (DCs) respectively. FSMs are typically visualised using State Graphs, Flow Tables or Cube Tables [4]. FSM synthesis flows today still operate at the grain of the monolithic FSM, thus complex control systems are synthesised and implemented one FSM at the time, with little notion, expression or exposure of the inter-FSM interaction.

Other key fundamental single FSM operations are composition and decomposition. Composition is a rather intuitive FSM operation [4], whereas decomposition is more complex, and may be achieved using state partition theory [14], [13], or by identifying repeated patterns within an FSM [15]. The drawbacks of the monolithic FSM model include its inability to model concurrency, and the fact that it is not scalable, *i.e.* composing multiple FSMs into a single one is generally intractable [16].

Interacting FSMs is a relatively informal model, which we include for completeness.

Definition 2.2 (Interacting FSMs): A network of n Interacting FSMs, M_1, M_2, ..., M_n, is a set of n FSMs, where each FSM may communicate, by exchanging inputs and outputs with other FSMs, without any restriction.

B. The FCPTnet Model

PTnets are divided into classes, depending on structural interactions between choice and concurrency. Below, we present the fundamental PTnet definitions.

Definition 2.3 (Net): A net N is a triple (S, T, F) where S, T are two finite disjoint sets, corresponding to the Places and Transitions of the net respectively, and F is a flow relation on $S \cup T$, such that $F \cap (S \times S) = F \cap (T \times T) = \emptyset$.

Given a net N, the set ${}^\bullet x = \{y | (y, x) \in F\}$ is the pre-set of node x and the set $x^\bullet = \{y | (x, y) \in F\}$ is its post-set. A triple $N' = (S', T', F')$ is a Net's $N = (S, T, F)$ subnet if S', T' are subsets of S, T respectively and F' consists of the subset of $(S' \times T') \cup (T' \times S')$ which is in F. A Net's marking assigns to every Place a non-negative integer. This number is the marking of the specified Place. The visualization of a Net's marking is a number of tokens in each Place, equal to Place's marking. A marking M enables a Transition t_i, if every Place in ${}^\bullet t$ is marked. If a Transition t is enabled in a marking M_j, then it fires reducing the marking of ${}^\bullet t$ by 1 and increasing the marking of $t^\bullet$ by 1. The set of all markings reachable from M is denoted as $[M >$.

Definition 2.4 (PTnet System, Reachable Markings): A Place, Transition Net system is a pair (N, M_0), where N is a connected net with at least one Place and one Transition, and M_0 is a marking of N, called the initial marking. A marking is called reachable in a system, if it is in $[M_0 >$.

The fundamental difference between a net and a PTnet system is that the former is merely a set of static relations, whereas the latter is a dynamic system able to model concurrency, conflict (choice) and complex interactions of the two. A Signal Transition Graph (STG) is a FCPTnet, whereby transitions of the net represent input or output signal transitions (+/-) to Boolean values (0/1).

Definition 2.5 (STG): An STG is a tuple $G = (P, T, F, M_O, In, Out, l)$ where (P, T, F, M_o) is a PTnet System, and In and Out are disjoint sets of input and output signals. For $Sig := In \cup Out$ being the set of all signals, $l : T \rightarrow Sig \times \{ +, - \}$ is the labeling function. $Sig \times \{ +, - \}$ is the set of signal edges or signal transitions.

We now review fundamental PTnet properties.

Definition 2.6 (Liveness and Deadlock Freedom): A system is live if, for every reachable marking M and every

Transition t, there exists a marking $M' \in [M >$ enabling t. If (N, M_0) is a live system, then M_0 is denoted as a live marking of N. A system is deadlock free, if every reachable marking enables at least one Transition, *i.e.* no dead marking is reachable from the initial marking.

Definition 2.7 (Place Bound, Bounded Systems): A system is bounded, if for every Place s, there exists a natural number b, such that $M(s) \leq b$, for every reachable marking M. The Place bound of s is $max\{M(s)|M \in [M_0\}$. A system is b-bounded if every Place is bounded by b.

Definition 2.8 (Well-formed nets): A net N is well-formed, if there exists a marking M_0 such that (N, M_0) is a live and bounded system.

An important predicate of a net, used by both analysis and covering algorithms, is the characterisation of a set of Places as a siphon.

Definition 2.9 (Siphons): A set R of Places of a net N is a siphon, iff $R \neq \emptyset \wedge {}^{\bullet}R \subseteq R^{\bullet}$.

Definition 2.10 (S-Component): A net N' is an S-Component of net N, generated by a nonempty set of nodes X, where for each Place s of X, ${}^{\bullet}s \cup s^{\bullet} \subseteq X$, $|{}^{\bullet}t| = 1 = |t^{\bullet}|$, for every Transition of N', N' is strongly connected and N' is a subnet of N.

Definition 2.11 (S-Cover): A set of S-Components, C, is an S-Cover, if every Place of a net belongs to an S-Component of C.

Definition 2.12 (Free-Choice PTnet System, Net [17]): A net $N = (S,T,F)$ is free-choice, if $(s,t) \in F$ implies ${}^{\bullet}t \times s^{\bullet} \subseteq F$ for every Place s and Transition t. A system (N, M_0) is free-choice if its underlying net is free-choice.

FCPTnets represent the most ubiquitous form of PTnets, due to the following two Theorems.

Lemma 2.1 (S-Coverability [17]): Well-formed, connected free-choice nets are covered by S-Components.

S-Coverability has been shown to be achievable in polynomial time for Free-choice [18] and Extended Free-choice (EFC) [19] Nets. A practical algorithm for the minimisation of the obtained S-Cover is presented in [20]. The inverse of the previous theorem, *i.e.* that an FCPTnet which is covered by S-Components is well-formed, also holds.

Lemma 2.2 (Well-formedness of FCPTnet[21]): For a live and bounded FCPTNet, N, it holds that:

- N is strongly connected,
- each Place belongs to a minimal siphon R,
- each N's subnet (R, $R^{\bullet} \cup^{\bullet} R$, $(R \times R^{\bullet}) \cup ({}^{\bullet}R \times R)$) is an S-Component,
- each S-Component is initially marked.

Higher-order PTnet classes, *e.g.* Asymmetric-Choice (ACPTnets) and General (GPTnets) PTnets, exhibit confusion, *i.e.* complex interactions between concurrency and choice. In practice, algorithms which, for FCPTnets exhibit polynomial complexity, such as S-Covering, have been shown to be intractable for the ACPTnets, GPTnets classes.

III. The Multiple Synchronised FSMs Model

The multiple Synchronised FSM model is a compact representation of a set of Interacting FSMs, which explicitly models inter-FSM synchronisation, using a set of two basic synchronisation primitives, Wait States and Transition Barriers.

A. Definitions

Definition 3.1 (MSFSM Set): An MSFSM set, MS, is a five-tuple (I,O,S,Δ,Λ), where I is a finite, nonempty set of global inputs, O is a finite, nonempty set of global outputs, S is a finite nonempty set of N FSMs, with state sets S_i and corresponding local output sets, $\lambda_i : I \times S_i \rightarrow O_i$ (if S_i is a Mealy machine), or $\lambda_i : S_i \rightarrow O_i$ (if S_i is a Moore machine), Δ is a set of next state functions, one per FSM i, where $\Delta_i : I \times O_1 \times O_2 \times \ldots \times S_i \times \ldots \times O_N \rightarrow S_i$, and $\Lambda : I \times O_1 \times O_2 \times \ldots \times O_N \rightarrow O$ is the global output generation function.

The MSFSM set is a set of shared input support, interacting FSMs, where each FSM changes state, or produces its local outputs (in the usual Mealy or Moore fashion), based on both the global inputs, I, and the state-dependent outputs of other FSMs of the set. Local FSM outputs can then be combined, combinationally, to generate global outputs. The simplest case of a local output, O_i, is when it is directly dependent upon a local state, *i.e.* $O_i = S_i$, thus each FSM's state change or local output generation may directly depend to the state of other FSMs. The initial state of the MSFSM set corresponds to the set of initial states of its member machines.

An MSFSM set may be represented as a set of flow tables (or state transition graphs), one per FSM in the set, and a set of combinational logic equations. Each flow table, i, represents the next state function, Δ_i, and the local outputs, O_i of FSM i, whereas the set of combinational logic equations generate the global outputs, *i.e.* global output function Λ. An MSFSM set corresponds to a potentially implementable specification, as each FSM of the set is indeed implementable, using existing monolithic FSM implementation techniques.

B. MSFSM Synchronisation Primitives

The fact that Δ_i functions of FSMs, implicitly include the current state of other FSMs in the set, allows for the inter-FSM communication, and more importantly, synchronisation to be exposed. We define two synchronisation primitives, Wait States and Transition Barriers, which stem from the Δ_i functions interaction, and prove that these are sufficient when only conjunctive (AND) synchronisation is allowed between Synchronised FSMs.

Definition 3.2 (MSFSM Wait State): In an MSFSM set, MS, a Wait State W is a state of a machine M, which belongs to MS, where the next state function for state W, $\Delta_i(W)$, depends on a combinational function f of the global inputs I, and on a product of local outputs of a set j of the FSMs of MS, *i.e.* is of the form: $\Delta_i(W) = f(I) \cdot \prod_{j \in N} O_j$

The next state logic of a Wait state W of M, will be activated, not only when the corresponding inputs function, $f(I)$ is activated, but also when a set of local FSM outputs, O_j, are activated as well. These, in turn, depend on local states of their corresponding FSMs, *i.e.* a conjunction of outputs. Thus, transitively, a Wait state W awaits for a set

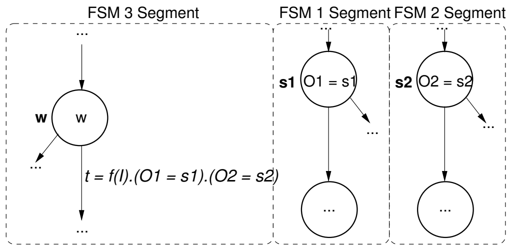

Fig. 1: Wait State Example

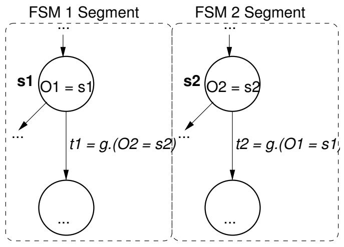

Fig. 2: Transition Barrier Example

of FSM states to be reached and potentially a set of global inputs as well. This form of synchronisation is unidirectional and represents conjunctive (AND) causality. A Wait State synchronisation may be represented by a tuple (W, t, $s1$, $s2$, ..., sm), where W, t and $s1$ to sm correspond to state W, the relevant transition function, $f(I)$, and the relevant FSM states of MS generating O_j respectively. Figure 1 illustrates a simple wait state dependency, where transition t of a state w of FSM 3, is activated by states $s1$ and $s2$, of FSMs 1 and 2 respectively.

Another synchronisation primitive can be defined by considering the special case of two, or more, mutually dependent Wait States between two or more FSMs of MS. For the two state case, a Transition Barrier (named after barrier synchronisation) corresponds to two Wait States mutually dependent upon each other, and is generalisable to any number of Wait States.

Definition 3.3 (MSFSM Transition Barrier): In an MSFSM set, MS, a Transition Barrier T, is a set of Wait State transitions of different FSMs of MS, with identical combinational function $f(I)$, and an equivalent output product in the respective Δ_i's, *i.e.* each transition of the synchronisation barrier T and corresponding wait state local output product, $\prod_{j \in N} O_j$, includes all other wait states of T.

Thus, the transitions of a Barrier may only be activated simultaneously, when all of the relevant Wait States, of the respective FSMs are reached. As each Wait State represents conjunctive (AND) causality, the barrier also represents the same.

A Transition Barrier may be represented by an unordered transition set, *i.e.* $\{t1, t2, \ldots, tm\}$, where $t1$ to tm correspond to the set of Wait State transitions of the barrier. Figure 2 illustrates the simplest form of a Transition Barrier, $\{t1, t2\}$, whereby transitions $t1$ and $t2$, which belong to FSMs 1 and 2 respectively, possess the same combinational function g and are mutually dependent, as $t1$ is activated by $O2$, which in turn is set by $s2$ and $t2$ by $O1$, which is set by $s1$. Thus, one awaits for the other, and they may only be activated simultaneously.

IV. Free Choice PTnet to MSFSM Set Transformation

In this section, we introduce a polynomial complexity flow for transforming a FCPTnet into an MSFSM set. This flow bridges the PTnet and monolithic FSM models, tackles the state explosion problem associated with existing FCPTnet implementation approaches, as well as guarantees existence for any FCPTnet implementation.

Free-choice PTnets have been shown to be decomposable not to the selfsame model, but to a set of S-Components (or T-components). As illustrated in Section II-B, an FCPTnet is decomposable into an S-Cover, where each S-Component is an FSM-like graph, *c.f.* Definitions 2.10, 2.11, Theorems 2.1 and 2.2. In fact, S-Coverability is achievable *in polynomial time* [19], and it has been shown that a non-exponential, practical algorithm may be used to derive a minimal S-Cover [20]. Hence, this path represents a very viable and practical path for the PTnet to MSFSM transformation. Our contribution in the transformation step lies in the conversion of S-Covers to "proper" FSMs, and MSFSMs, whereby (i) input transition relevant PTnet Places are eliminated, as they don't represent state in the FSM sense, and (ii) FSMs include all the necessary interaction signals for the purposes of synchronisation, and are thus behaviourally equivalent, to the original PTnet. The latter is performed through extraction of the aforementioned synchronisation primitives.

The polynomial time FCPTnet to MSFSM set transformation is comprised of the following five polynomial complexity steps, and respective complexities.

1) *FCPTnet S-Covering [17], [19], [20]*: $O(PT + P^2)$
2) *S-Component to Non-Interactive FSM mapping*: $O(P^2T^2)$
3) *FSM Collapsing*: $O(P^2T^4)$
4) *Synchronisation Primitive Extraction*: $O(P^3T^2)$
5) *Inter-MSFSM Synchronisation Integration*: $O(P^2T)$

In the following sections, these transformation steps are presented in detail, with the aid of a FCPTnet specification example borrowed from the field of asynchronous circuit design, $adfast$, an A/D converter controller [22]. In the relevant diagrams, solid black and white transitions are used represent output and input events respectively.

A. FCPTnet S-Covering

Extracting all possible S-Components of an FCPTnet is known to be exponential in complexity. However, prior work has shown that extracting a single S-Cover is achievable in $O(PT)$ time [19]. Further on, an extra step of $O(P^2)$ complexity can be used to extract a minimal (not the minimum) S-Cover, as shown in [20]. Hence, prior work has indeed established that covering an FCPTnet with a minimal S-Cover is achievable in polynomial time. It should be noted that the total number of places included in the S-Cover will typically be larger than that of the original FCPTnet. This is because the S-Covering process distributes and replicates transitions with multiple input or output places to different S-Components. Hence, some of the input and output places of such transitions are thus instantiated multiple times in the derived S-Components.

Provided that the FCPTnet is well-formed, *i.e.* live and bounded, *c.f.* 2.2, each S-Component includes an initially marked place. This initially marked place corresponds to the initial state of the subsequently derived FSM.

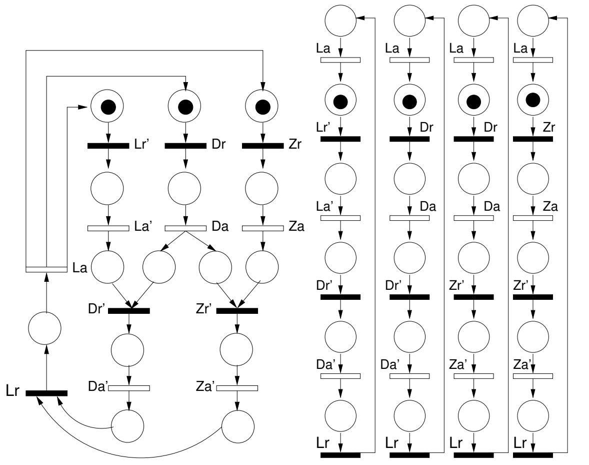

Fig. 3: $adfast$ - FCPTnet S-Covering

Figure 3 illustrates the S-Covering process, with the LHS and RHS of Figure 3 illustrating the original FCPTnet specification, and a corresponding, minimal S-Cover of $adfast$ respectively (black transitions correspond to output, white to input signals). It is evident that while the original net contains a total of 15 Places and 12 Transitions, the S-Cover contains 24 Places and 24 Transitions, due to Place (state) and Transition replication (*e.g.* transition Lr).

B. S-Component to Non-Interactive FSM mapping

A one-to-one and onto mapping (bijection relation) exists between each S-Component of the FCPTnet and a Mealy FSM, which can be implemented as a sequence of three steps.

In the first step, each S-Component transition is mapped to a corresponding FSM transition, and accordingly each S-Component place is mapped to a corresponding FSM state. In the second step, the FSM's input and output sets, I and O, are extracted from the corresponding S-Component's T set, depending on whether T is labeled as an input or output.

Finally, in the third and last step, each FSM's δ and λ functions are constructed, based on the corresponding S-Component's flow relation F. The next state function, δ, at this point assumes its monolithic FSM form, *i.e.* $\delta : I \times S \rightarrow S$, as no explicit inter-FSM interaction is yet expressed. In the final step of the flow, Section IV-E, δ is promoted to Δ, *i.e.* includes inter-FSM synchronisations.

Hence, for each couple of an S-Component's flow relation pairs, *i.e.* (p, t), (t', p'), whereby $t = t'$, state transition (s, t_f, s') is extracted, for s, s', the respective states which correspond to the places of F. Such a state transition is appropriately added to the respective FSM's δ or λ functions, depending on whether t_f is an input or output. The corresponding complexities of the three steps are $O(T+P)$, $O(T)$ and $O(P^2T^2)$ respectively. Figure 5 illustrates the mapping of the first S-Component of $adfast$ to an FSM.

The set of Non-Interactive FSMs, which stem from this second step of the transformation flow, are implicitly synchronised, as per the original FCPTnet semantics. This implicit synchronisation is implied between states and transitions of the Non-Interactive FSMs which stem from, and map to the same original place or transition of the covered FCPTnet. Exposing this implicit synchronisation is necessary, so as to produce an Interactive FSMs model. The fourth step of the transformation flow, extracts the synchronisation primitives for this particular purpose.

C. FSM Collapsing

Each element (s, i, s') of an FSM's δ set is comprised of two states s and s' and an input transition i. Up to this point, the PTNet to FSM transformation assumed that each input corresponds to a single signal. However, FSM semantics dictate that each transition corresponds to a Boolean Function of input signals. For example, if $f_i = in_i.in_j$ then transition (s, f_i, s') is activated when both in_i and in_j are activated, independently of their activation order. Effectively, in_i and in_j are concurrent in f_i. This type of concurrency may also be exploited for the outputs in the elements of the λ set.

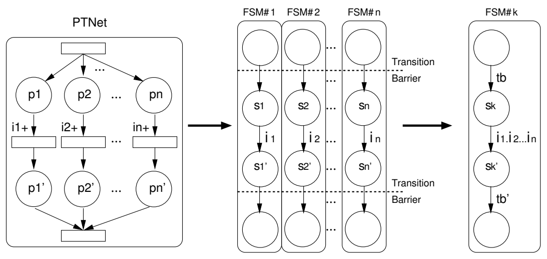

Fig. 4: FSM Collapsing - Example

Thus, in the case where multiple FSMs exhibit concurrency only between inputs or outputs, these may be collapsed into a single FSM. An example is shown in Figure 4, where n concurrent input events are distributed to n synchronised FSMs which only exhibit concurrency at the aforementioned input events.

The complexity for collapsing is O(P^2T^4), as each FSM's transitions are compared to the transitions of all other FSMs, so as to ascertain whether the latter, as well as their predecessor and successor transitions respectively stem from the same corresponding PTnet transitions (the number of FSMs is in worst case $|P|$ and the number of transitions in each FSM is in worst case $|T|$).

D. Synchronisation Primitive Extraction

This fourth step of the transformation flow identifies the complete set of implicit synchronisation primitives, *i.e.* Wait States and Transition Barriers (*c.f* Section III-B), as expressed by the S-Cover and S-Components of the FCPTnet. The latter are extracted by analysing the S-Component's flow relations.

This is achieved by transforming each flow-relation into the form $F' : S \times T \rightarrow S$. Common transitions in T of different F''s, *i.e.* shared between the S-Components, produce Wait States, as they implicitly describe a state synchronisation dependence for entering a state of the S-Component. Similarly, common States in S, *i.e.* originating from the same PTnet place, produce Transition Barriers, as they implicitly describe states of multiple S-Components which must be simultaneously entered or left.

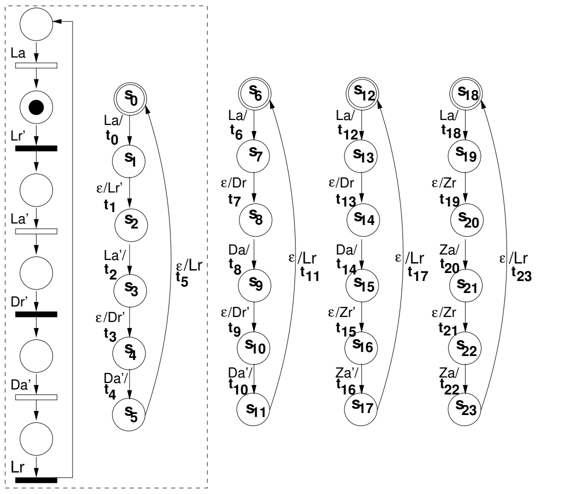

Fig. 5: $adfast$ - S-Component to FSM Conversion

Figure 5 illustrates the Non-Interacting FSMs of $adfast$. Synchronisation primitive analysis of the original FCPTnet derives 8 Transition Barriers, two of which, *i.e.* $\{t_0,t_6,t_{12},t_{18}\}$, $\{t_5,t_{11},t_{17},t_{23}\}$ synchronise all four FSMs, whereas the remaining six synchronise two FSMs each, *i.e.* $\{t_7,t_{13}\}$, $\{t_8,t_{14}\}$ synchronise FSM_1, FSM_2, $\{t_3,t_9\}$, $\{t_4,t_{10}\}$ synchronise FSM_0, FSM_1 and $\{t_{15},t_{21}\}$, $\{t_{16},t_{22}\}$ synchronise FSM_2, FSM_3.

The complexity for extracting the synchronisation primitives is $O(P^3T^2)$, as each place, transition pair of the original PTNet is analysed with respect to the states and transitions sets of the $|P|$, in the worst case, S-Components.

E. Inter-MSFSM Synchronisation Integration

With the synchronisation primitives identified, this step completes the transformation process by promoting the monolithic δ_i functions of each FSM to Δ_i, so as to explicitly include the inter-MSFSM synchronisations dictated by the synchronisation primitives, and it also implements the global output generation function, Λ.

The finest grain of synchronisation, according to Definition 3.2, is the Wait State. A Transition Barrier, Definition 3.3 is indeed a set of interlocked Wait States. Thus, with respect to integrating inter-MSFSM synchronisation, it is both necessary and sufficient to integrate Wait State synchronisation and express each Δ_i as $\Delta_i : I \times O_1 \times O_2 \times \ldots \times S_i \times \ldots \times O_N \rightarrow S_i$ functions, *i.e.* determine the dependent intra-FSM outputs for a state change.

We now illustrate specifically how the synchronisation is integrated. Each Wait State (W, t, s_1, s_2, ..., s_m), which may indeed be a member of a Transition Barrier, dictates that states (s_1, s_2, ..., s_m) form corresponding state-generated outputs, ($o(s_1)$, $o(s_2)$, ..., $o(s_m)$), which, along with transition t form the transition function for entering State W. With respect to the state-generated outputs, $o(s_i)$ will evaluate to logic 1, if the FSM is currently in this corresponding state, or logic 0 otherwise. Thus, the transition function into W will be of the form $W_{enter} = (t \,.\, o(s_1).\, o(s_2) \,.\, \ldots \,.\, o(s_m))$, *i.e.* the Boolean conjunction of t and the state-generated outputs. In this way, each Wait State contributes to the generation of local state-generated outputs, and subsequently the formation of the Δ_i next state functions.

The global output generation function, Λ, where $\Lambda : I \times O_1 \times O_2 \times \ldots \times O_N \rightarrow O$, is expressed by forming the consensus of the corresponding local outputs. Hence, output transitions which exist in multiple FSM's λ_i functions must be combined, again through Boolean conjunction, so as to render the consensus of the local outputs.

The complexity of this step is equal to the worst case total number of wait states *i.e.* O(P^2T), as each place-transition pair in the original PTNet may have been cloned to all $|P|$ FSMs.

V. Transformation Completeness and Equivalence

In this section, we formally prove that the presented transformation flow is complete, *i.e.* any FCPTnet is transformable to an equivalent MSFSM set. Prior to the proof itself, we formally define the notions of global state for the MSFSM set, and input, output trace equivalence.

Definition 5.1 (FCPTnet, MSFSM Global State): The global state of a safe FCPTnet System (N, M_o), is its current Marking, M_C, *i.e.* the set of currently Marked Places.

The global current state of an MSFSM set M, composed of n FSMs, is the n-tuple, (CS_1, CS_2, ..., CS_n), formed by the current states of the n MSFSMs.

The presented transformation indeed forms a bijection between the global states of the FCPTnet and MSFSM set, as we prove in the following Lemma.

Lemma 5.1 (Global State Change Bijection): A bijection relation exists between the Global State Change of the FCPTnet and the Global State Change of the MSFSM set.

Proof: A global state change of the FCPTnet represents a change in the marking of the net. The latter is based on token movement, in accordance to the firing of input and output transitions. According to Lemmas 2.1 and 2.2, S-Cover's global state change is indeed a bijection of the global state of the FCPTnet, as any token movement in the latter has a direct correspondence to token movement in the former. Now, the MSFSM set is generated directly from the S-Cover, and as described in Sections IV-B-IV-E, both a unique and bi-directional structural pairing exists between places of the S-Covers and states of the MSFSM set, and a unique and bi-directional correspondence, *i.e.* a matching by name, exists between input and output transitions of the S-Covers and inputs and outputs of the MSFSM set. Further on, the next state transition functions, Δ_i, of the MSFSMs are formed precisely to correspond to, and implement the allowed token movement of the S-Cover. Thus, any input transition, t, simultaneous between S-Covers corresponds to a Transition Barrier, and in turn corresponds to Wait State Boolean transition functions $W_{enter} = (t \,.\, o(s_1).\, o(s_2) \,.\, \ldots \,.\, o(s_m))$. In addition, any output transition of the S-Cover, corresponds directly to a local output generation in the MSFSM set. From this bi-directional pairing between places, states, inputs and outputs and the formation of the transition functions to implement the allowed token movement of the S-Covers, it follows that any global state change of the S-Covers, through an allowed token movement, corresponds to an exactly matched, 1-1 and onto global state change in the MSFSM set, as dictated by the input and output transitions. As the opposite also holds, based on the above, the global state change relation is a bijection. ■

Now, proving input, output trace equivalence between an FCPTnet and its corresponding MSFSM set, requires defining input and output trace equivalence with respect the the global states of the two models.

Definition 5.2 (k-Distinguishable Global States): The global states of an FCPTnet System, (N, M_o), M_c, and the MSFSM set, M, (CS_1, CS_2, ..., CS_n), are distinguishable if and only if there exists at least one finite, allowed input sequence of length k, which, when applied to both N and M, with N and M residing in their corresponding initial global states, it causes different output sequences.

It thus follows that global states which are not k-distinguishable are k-equivalent. Thus, equivalence may be defined as follows.

Definition 5.3 (FCPTnet, MSFSM Equivalence): An FCPTnet System, (N, M_o) is equivalent to an MSFSM set, M, if and only if, for every *allowed input sequence*, the same output sequence will be produced from their corresponding initial global states of the FCPTnet N and MSFSM set M.

The allowed input sequences are the valid input transition sequences described in the FCPTnet specification. Now, equivalence between the original FCPTnet and resultant MSFSM set must be proved with respect to Definition 5.3.

Theorem 5.1 (Transformation Equivalence, Completeness): For any well-formed, STG-labeled FCPTnet System, (N, M_o), an equivalent MSFSM set, M, is derived by the presented flow.

Proof: (Induction) Assume the FCPTnet System (N, M_o), and the MSFSM set, M, *are not k-distinguishable*, according to Definition 5.2, and are thus k-equivalent, *i.e.* for all input sequences of length k (or less), output sequences match. We must subsequently prove that they are not (k + 1)-distinguishable and thus (k + 1)-equivalent.

According to our induction hypothesis, the global states of the FCPTnet System (N, M_o), and the MSFSM set, are not k-distinguishable, *i.e.* after k input transitions, the global states of the two are indeed equivalent, as they produce the same output sequences. Thus, starting from equivalent global states, we consider the arrival of an additional, allowed input; the next observable output will determine (k + 1)-distinguishability. Now, if an output is not generated for the (k + 1) input, it follows straightforwardly that (k + 1)-distinguishability does not hold. Thus, we consider the case that an observable output is indeed generated. According to Lemma 5.1, the global state change of the FCPTnet and that of the MSFSM set is a bijection, thus the global state change of the FCPTnet, which will produce the next observable output, will possess a corresponding global state change for the MSFSM set, and both will be reachable from the two equivalent global states of the k-input sequence. It thus follows that the (k + 1) input must also lead to equivalent global states for the FCPTnet and the MSFSM set, hence the (k + 1) allowed input sequence is not (k + 1)-distinguishable. ■

VI. Results

We now present experimental results of the FCPTnet to Interactive Synchronised FSM polynomial complexity flow presented, on a set of 25 PTnet benchmarks. The benchmarks used stem from the asynchronous circuit implementation field and represent asynchronous control circuit specifications, whereby PTnet events represent the assertion or deassertion of the relevant circuit signals. We contrast the state space and execution time of our flow to that of tool Petrify [23], which, in order to implement a PTnet specification, it expresses the latter's state space as a monolithic FSM, *i.e.* the event State Graph (SG).

Table I illustrates state space and execution time results of the flow presented in this paper, as well as the corresponding results of the Petrify tool. The first column of Table I classifies the PTnet of the relevant benchmark, between Marked Graph (MG), State Machine (SM), FC (Free-choice), or State-Machine Decomposable (SMD). The latter class refers to a General PTnet, coverable by S-Components.

Petrify results clearly indicate several cases of state explosion occurring while the tool explores the PTnet specification's state space, particularly for the more concurrent benchmarks of the set. As an example, `master_read2`, which consists of 52 places, requires a SG with 8×10^7 states for state-based, SG analysis! In comparison, the total number of states produced by our flow is 62, and correspond to a total of 10 Interacting, Synchronised FSMs. With respect to execution time, for a fair comparison, this includes solely the required time for state-space generation, without any additional time for SG processing required to generate the implementation. The execution time results illustrate significant runtime improvement over Petrify, with several orders of magnitude difference. We should emphasise that the resultant set of Interacting FSMs are not optimised in any way, for instance by running a state minimisation algorithm per FSM.

VII. Conclusions and Future Work

The contribution of this paper is a polynomial complexity flow for transforming an event-driven FCPTnet into a state-based Interacting FSMs model. This flow tackles the deficiencies of the PTnet and monolithic FSMs models, *i.e.* state explosion and efficient representation of concurrency, by acting as bridge between the these two models. The key to the

Benchmark	Original PTnet			Petrify [23]		MSFSM			
	Type	Num. of Places	Num. of Trans.	Num. of States	Exec. Time(s)	Num. of FSMs	Total Num. of States	Total Num. of Trans.	Exec. Time(s)
`art_jordi_10_9`	MG	216	198	9.3×10^{17}	39	19	216	216	0.31
`two_pipes_weak3`	MG	23	14	160	0.03	7	28	28	<0.01
`two_pipes_weak6`	MG	47	26	$10\ \times10^{3}$	0.33	13	52	52	0.01
`two_pipes_weak9`	MG	71	38	6.5×10^{5}	1.54	19	76	76	0.02
`two_pipes_weak12`	MG	95	50	4.2×10^{7}	11.22	25	100	100	0.03
`two_pipes_arb3`	SMD	38	24	$1\ \times10^{3}$	0.17	11	61	61	0.01
`two_pipes_arb6`	SMD	62	36	6.9×10^{4}	1.19	23	115	115	0.02
`two_pipes_arb9`	SMD	86	48	4.4×10^{6}	19.2	33	158	158	0.04
`two_pipes_arb12`	SMD	95	50	2.8×10^{8}	550.4	25	100	100	0.07
`three_pipes_weak3`	MG	34	20	1.6×10^{3}	0.13	11	44	44	<0.01
`three_pipes_weak6`	MG	70	38	8.5×10^{5}	1.31	20	80	80	0.02
`three_pipes_weak9`	MG	106	56	4.3×10^{8}	38.4	29	116	116	0.04
`three_pipes_weak12`	MG	142	74	2.2×10^{11}	356.94	38	152	152	0.06
`three_pipes_arb3`	SMD	56	36	1.4×10^{4}	0.56	16	121	126	0.01
`three_pipes_arb6`	SMD	70	38	7.3×10^{6}	9.3	34	192	197	0.02
`three_pipes_arb9`	SMD	106	56	3.7×10^{9}	200.5	51	280	287	0.07
`three_pipes_arb12`	SMD	142	74	1.9×10^{12}	1045.7	69	335	340	0.11
`dup-4-pull.sl.3`	FC	121	112	155	0.62	17	1360	1496	0.88
`count2`	FC	19	16	27	0.01	5	41	46	<0.01
`dup-4-ph`	FC	133	123	169	0.95	20	1814	2014	0.93
`hybridf`	MG	26	16	80	0.03	8	48	48	<0.01
`master_read2`	MG	74	52	8×10^{7}	1.12	20	102	102	<0.01
`master_read`	MG	38	26	1882	0.17	10	62	76	<0.01
`mmu`	MG	20	16	174	0.24	5	31	31	<0.01
`tangram`	MG	98	92	426	0.48	6	348	348	0.06

TABLE I: MSFSM Generation for FCPTnets for Concurrent Benchmarks

flow is the definition of a new formalism for Interacting FSMs, which exposes the inter-FSM synchronising interactions, using a set of synchronisation primitives, *i.e.* Wait States and Transition Barriers. The new flow can be used for algorithms which need to explore the state space of a PTnet specification, *e.g.* PTnet implementation. Experimental results, on a set of 25 PTnet benchmarks from the field of asynchronous circuit design, demonstrate a significant reduction in both state space and execution time for our approach, when compared to the corresponding results of the Petrify tool.

ACKNOWLEDGEMENT

This work has been partly funded by FORTH-ICS, Greece.

REFERENCES

[1] F. Wagner, R. Schmuki, T. Wagner, and P. Woltenholme, *Modeling Software with Finite State Machines: A Practical Approach*. Auerbach Publications, 2006.

[2] J. Hopcroft, "An nlogn Algorithm for Minimizing States in a Finite Automaton," Stanford University, Computer Science Department, Tech. Rep. STAN-CS-71-190, Jan. 1971.

[3] J.-K. Rho *et al.*, "Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines," *IEEE Transactions on Computer-Aided Design*, vol. 13, no. 2, pp. 167–177, 1994.

[4] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996, iSBN 0-7923-9746-0.

[5] S. Nowick, "Automatic synthesis of burst-mode asynchronous controllers," Ph.D. dissertation, Stanford University, March 1995, (revised tech. report, Stanford Computer Systems Lab. CSL-TR-95-686, Dec. 1995).

[6] L. A. Hollaar, "Direct implementation of asynchronous control units," vol. C-31, no. 12, pp. 1133–1141, Dec. 1982.

[7] R. K. Brayton *et al.*, "VIS: A System for Verification and Synthesis," in *Proc. International Workshop on Computer Aided Verification*, 1996, pp. 428–432.

[8] J. Cortadella *et al.*, *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002.

[9] T. Kam, *Synthesis of Finite State Machines: Functional Optimization*. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[10] T. Murata, "Petri Nets: Properties, analysis and applications," *Proceedings of the IEEE*, pp. 541–580, Apr. 1989.

[11] K. Biliński, E. L. Dagless, J. M. Saul, and M. Adamski, "Parallel controller synthesis from a petri net specification," in *Proc. European Conference on Design Automation (EDAC)*, 1994.

[12] L. Lavagno, C. W. Moon, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Solving the state assignment problem for signal transition graphs," in *Proc. ACM/IEEE Design Automation Conference*, June 1992.

[13] Z. Kohavi, *Switching and Finite Automata Theory*. McGraw-Hill, June 1978.

[14] J. Hartmanis and R. Stearns, *Algebraic Structure Theory of Sequential Machines*. Prentice-Hall, 1966.

[15] S. Devadas and A. Newton, "Decomposition and factorization of sequential finite state machines," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 11, pp. 1206 –1217, nov 1989.

[16] J. Lind-Nielsen *et al.*, "Verification of Large State/Event Systems using Compositionality and Dependency Analysis," in *TACAS'98 Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, 1998.

[17] J. Desel and J. Esparza, *Free Choice Petri Nets*. Cambridge University Press, 1995, iSBN-10 0-521-01945-1.

[18] P. Kemper, "Linear time algorithm to find a minimal deadlock in a strongly connected free-choice net," in *Application and Theory of Petri Nets*, 1993, pp. 319–338.

[19] Peter Kemper, "O(PT) - Algorithm to Compute a Cover of S-components in EFC-nets," Informatik IV, University of Dortmund, Tech. Rep., 1994.

[20] D.-I. Lee, S. Kodama, and S. Kumagai, "Decomposition Algorithms for Live and Safe Free Choice Nets," *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 78, 1995.

[21] P. Kemper and F. Bause, "An efficient polynomial-time algorithm to decide liveness and boundedness of free-choice nets," in *Application and Theory of Petri Nets*. Springer, 1992, pp. 263–278.

[22] J. Carmona, J. Cortadella, and E. Pastor, "A Structural Encoding Technique for the Synthesis of Asynchronous Circuits," *Fundam. Inf.*, vol. 50, pp. 135–154, February 2002.

[23] J. Cortadella *et al.*, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE Trans. Inf. and Syst.*, vol. E80-D, no. 3, pp. 315–325, Mar. 1997.