Merged Processes — a New Condensed Representation of Petri Net Behaviour

Victor Khomenko[1], Alex Kondratyev[2], Maciej Koutny[1], and Walter Vogler[3]

[1] School of Computing Science, University of Newcastle, NE1 7RU, U.K.
[2] Cadence Berkeley Labs, Berkeley, CA 94704, USA
[3] Institut für Informatik, Universität Augsburg, D-86135 Germany

Abstract. Model checking based on Petri net unfoldings is an approach widely applied to cope with the state space explosion problem.
In this paper we propose a new condensed representation of a Petri net's behaviour called *merged processes*, which copes well not only with concurrency, but also with other sources of state space explosion, viz. sequences of choices and non-safeness. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for the latter can be re-used. Experimental results indicate that the proposed representation of a Petri net's behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking.
Keywords: Merged processes, Petri net unravelling, Petri net unfolding, state space explosion, model checking, formal verification.

1 Introduction

A reactive system is commonly described by a set of concurrent processes that interact with each other. Processes typically have descriptions which are short and manageable, and the complexity of the behaviour of the system as a whole comes from highly complicated interactions between them. One way of coping with this complexity problem is to use formal methods and, especially, computer aided verification tools implementing model checking (see, e.g., [1]) — a technique in which the verification of a system is carried out using a finite representation of its state space.

The main drawback of model checking is that it suffers from the *state space explosion* problem [16]. That is, even a relatively small system specification can (and often does) yield a very large state space. To cope with this, several techniques have been developed, which usually aim either at a compact representation of the full state space of the system, or at the generation of a reduced state space (that is still sufficient for a given verification task). Among them, a prominent technique is McMillan's (finite prefixes of) Petri net unfoldings (see, e.g., [5, 7, 11]). They rely on the partial order view of concurrent computation, and represent system states implicitly, using an acyclic *unfolding prefix*.

There are several common sources of state space explosion. One of them is concurrency, and the unfolding techniques were primarily designed for efficient

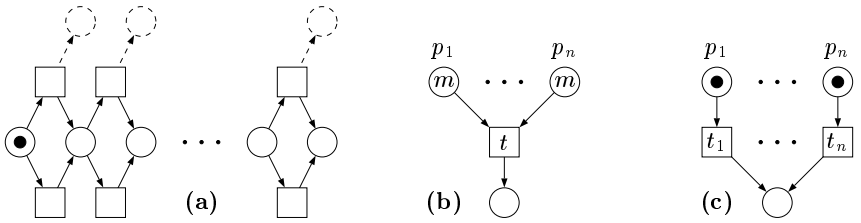

Fig. 1. Examples of Petri nets.

verification of highly concurrent systems. Indeed, complete prefixes are often exponentially smaller than the corresponding reachability graphs, because they represent concurrency directly rather than by multidimensional 'diamonds' as it is done in reachability graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the reachability graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will be isomorphic to the net itself. However, unfoldings do not cope well with some other important sources of state space explosion, in particular with sequences of choices and non-safeness. Below we consider examples illustrating this problem.

First, consider Figure 1(a) with the dashed part not taken into account. The cut-off condition proposed in [5] copes well with this Petri net (since the marking reached after either choice on each stage is the same — in fact, the Petri net has very few reachable markings), and the resulting prefix is linear in the size of the original Petri net. However, if the dashed part of the figure is taken into account, the smallest complete prefix is exponential in the size of the Petri net, since no event can be declared a cut-off (intuitively, each reachable marking of the Petri net 'remembers' its past). Thus Petri nets performing a sequence of choices leading to different markings may yield exponential prefixes.

Another problem arises when one tries to unfold non-safe Petri nets. Consider the Petri net in Figure 1(b). Its smallest complete unfolding prefix contains m^n instances of t, since the unfolding *distinguishes between different tokens on the same place.* One way to cope with non-safe nets is to convert them into safe ones and unfold the latter, as was proposed in [5]. However, such an approach destroys the concurrency and can lead to very large prefixes; e.g., this approach applied to the Petri net in Figure 1(c) would yield a prefix exponential in the size of the original Petri net, while the traditional unfolding technique would yield a prefix which is linear in its size [5].

The described problems with Petri net unfoldings should be viewed in the light of the fact that all the above examples have a very simple structure — viz. they are all acyclic, and thus many model checking techniques, in particular those based on the *marking equation* [7, 13, 15], could be applied *directly to the original Petri nets.* And so it may happen that a prefix exponential in the size of the Petri net is built for a relatively simple problem!

In this paper we propose a new condensed representation of a Petri net's behaviour called *merged processes,* which remedies the problems described above.

It copes well not only with concurrency, but also with other sources of state space explosion we mentioned, viz. sequence of choices and non-safeness. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for unfoldings can be re-used.

The main idea behind this representation is to fuse some equally labelled nodes in the complete prefix of the Petri net being verified, and use the resulting net as the basis for verification. For example, the unfolding of the Petri shown in Figure 1(a) (even with the dashed part taken into account) will collapse back to the original net after the fusion. In fact, this will happen in all the examples considered above. Of course, such a fusion can result in various problems, in particular cycles can appear and the marking equation alone is not sufficient for verification of such nets. The rest of this paper is devoted to formally defining this transformation and solving some of the arising problems. The experimental results indicate that the proposed representation of a Petri net's behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking.

All the proofs and further examples can be found in the technical report [8] (available on-line).

2 Basic notions

In this section we introduce the basic notions concerning Petri nets and their unfoldings (see also [5, 7, 9, 11, 13–15]).

Petri nets. A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, F)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. A *marking* of N is a multiset M of places, i.e., $M : P \to \mathbb{N} \stackrel{\text{df}}{=} \{0, 1, 2, \ldots\}$. The standard rules about drawing nets are adopted in this paper, viz. places are represented as circles, transitions as boxes, the flow relation by arcs, and the marking is shown by placing tokens within circles. As usual, ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid (y, z) \in F\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid (z, y) \in F\}$ denote the *pre-* and *postset* of $z \in P \cup T$. In this paper, the presets of transitions are restricted to be non-empty, i.e., ${}^\bullet t \neq \varnothing$ for every $t \in T$. A *net system* (or *Petri net*) is a pair $\Sigma \stackrel{\text{df}}{=} (N, M_0)$ comprising a finite net N and an *initial* marking M_0. It is assumed that the reader is familiar with the standard notions of Petri net theory, such as the *enabledness* and *firing* of a transition, *reachability* of a marking, the *marking equation*, *safe Petri net* and *deadlock* (see, e.g., [15] for a brief introduction).

Branching processes. A *branching process* [5, 7] β of a Petri net Σ is a finite or infinite acyclic net which can be obtained through unfolding Σ, by successive firings of transition, under the following assumptions: (i) for each new firing a fresh transition (called an *event*) is generated; and (ii) for each newly produced token a fresh place (called a *condition*) is generated. There exists a unique (up to isomorphism) maximal (w.r.t. the prefix relation) branching process of Σ called

the *unfolding* of Σ. For example, the unfolding of the Petri net in Figure 2(a) is shown in part (b) of this figure (with the dashed lines ignored).

The unfolding is infinite whenever Σ has an infinite run; however, if Σ has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events) without loss of essential information. The sets of conditions, events, arcs and cut-off events of β will be denoted by B, E, G and E_{cut}, respectively, (note that $E_{cut} \subseteq E$), and the labelling function mapping the nodes of β to the corresponding nodes of Σ will be denoted by h.

Since β is acyclic, the transitive closure of its flow relation is a partial order $<$ on $B \cup E$, called the *causality relation*. (The reflexive order corresponding to $<$ will be denoted by $\leq$.) Intuitively, all the events which are smaller than an event $e \in E$ w.r.t. $<$ must precede e in any valid execution of β containing e. To make this precise, consider the implicit initial marking of β, obtained by putting a single token in each condition which does not have an incoming arc. Note that h is a *homomorphism*, i.e., it maps the conditions in the preset (postset resp.) of an event e bijectively to the preset (postset resp.) of $h(e)$ and, intuitively, it maps the (implicit) initial marking of β to the initial marking of Σ. Such as any homomorphism, h maps runs of β to runs of Σ. It is known that in acyclic nets like β, a marking is reachable if and only if the corresponding marking equation has a solution [15], and hence branching processes can be used for efficient model checking [6, 7, 10–13].

Two nodes $x, y \in B \cup E$ are in *conflict*, denoted $x \# y$, if there are distinct events $e, f \in E$ such that ${}^\bullet e \cap {}^\bullet f \neq \varnothing$ and $e \leq x$ and $f \leq y$. Intuitively, no valid execution of β can contain two events in conflict. Two nodes $x, y \in B \cup E$ are *concurrent*, denoted $x \; co \; y$, if neither $y \# y'$ nor $y \leq y'$ nor $y' \leq y$. Intuitively, two concurrent events can be enabled simultaneously, and executed in any order, or even concurrently. For example, in the branching process shown in Figure 2(b) the following relationships hold: $e_1 < e_5$, $e_3 \# e_4$ and $c_1 \; co \; c_4$.

Due to structural properties of branching processes (such as acyclicity), the reachable markings of Σ can be represented using *configurations* of β. A *configuration* is a finite set of events $C \subseteq E$ such that for all $e, f \in C$, $\neg(e \# f)$ and, for every $e \in C$, $f < e$ implies $f \in C$. For example, in the branching process shown in Figure 2(b) $\{e_1, e_3, e_5\}$ is a configuration whereas $\{e_1, e_2, e_3\}$ and $\{e_1, e_5\}$ are not (the former includes events in conflict, $e_1 \# e_2$, while the latter does not include e_3, a causal predecessor of e_5). Intuitively, a configuration is a partial-order execution, i.e., an execution where the order of firing of some of its events is not important.

After starting β from the implicit initial marking and executing all the events in C, one reaches the marking denoted by $Cut(C)$. $Mark(C)$ denotes the corresponding marking of Σ, reached by firing a transition sequence corresponding to the events in C. A branching process β is *marking-complete w.r.t. a set* $E_{cut} \subseteq E$ if for every reachable marking M of Σ there is a configuration C of β such that $C \cap E_{cut} = \varnothing$ and $Mark(C) = M$; moreover, β is *complete* if it is marking-complete and for each configuration C of β such that $C \cap E_{cut} = \varnothing$ and each

event $e \notin C$ of the unfolding such that $C \cup \{e\}$ is a configuration of the unfolding, e is in β (e may be in E_{cut}); this additional *preservation of firings* is sometimes used for deadlock detection. Complete branching processes are often called complete (unfolding) *prefixes*. One can build such a complete prefix ensuring that the number of non-cut-off events $|E \setminus E_{cut}|$ in it does not exceed the number of reachable markings of Σ [5, 7].

3 Merged processes

In this section we introduce the notion of a *merged process,* which is the main construction investigated in this paper.

Definition 1 (occurrence-depth). *Let β be a branching process of a Petri net Σ, and x be one of its nodes (condition or event). The* occurrence-depth *of x is defined as the maximum number of $h(x)$-labelled nodes on any directed path starting at a minimal (w.r.t. $<$) condition and terminating at x in the directed graph representing β.*

The above notion is well-defined since there is always at least one directed path starting at a minimal (w.r.t. $<$) condition and terminating at x, and the number of all such paths is finite. In Figure 2(b) the occurrence-depths of conditions are shown in brackets.

Definition 2 (merged process). *Given a branching process β, the corresponding* merged process $\mu = \mathfrak{Merge}(\beta)$ *is a Petri net which is obtained in two steps, as follows:*
Step 1: *the places of μ, called* mp-conditions, *are obtained by fusing together all the conditions of β which have the same labels and occurrence-depths; each mp-condition inherits its label and arcs from the fused conditions, and its initial marking is the total number of minimal (w.r.t. $<$) conditions which were fused into it.*
Step 2: *the transitions of μ, called* mp-events, *are obtained by merging all the events which have the same labels, presets and postsets (after step 1 was performed); each mp-event inherits its label from the merged events (and has exactly the same connectivity as either of them), and it is declared cut-off iff all the events merged into it were cut-off events in β.*

Figure 2(b,c) illustrates this notion. In the sequel, $\hbar$ will denote the homomorphism mapping the nodes of β to the corresponding nodes of μ, and $\widehat{E}$, $\widehat{B}$, $\widehat{G}$, $\widehat{M_0}$, $\widehat{E_{cut}}$ and $\widehat{h}$ will denote the set of its mp-events, the set of its mp-conditions, its flow relation, its initial marking, the set of its cut-off events and the homomorphism mapping the nodes of μ to the corresponding nodes of Σ (note that $\widehat{h} \circ \hbar = h$). The merged process corresponding to the (full) unfolding of Σ will be called the *unravelling* of Σ. A few simple properties of merged processes are listed below:

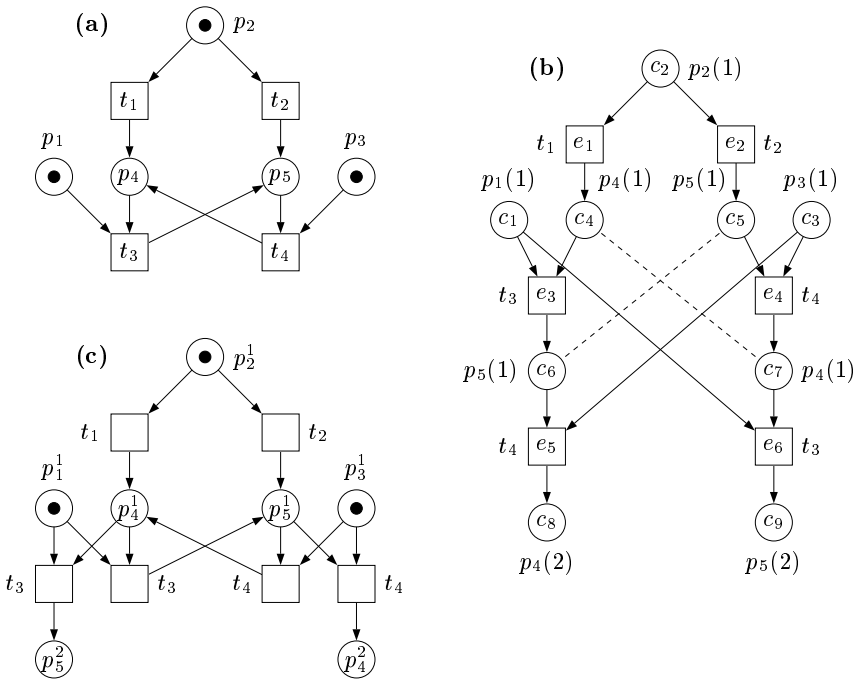

Fig. 2. A Petri net **(a)**, its unfolding with the occurrence-depths of conditions shown in brackets and the conditions to be fused connected by dashed lines **(b)**, and its unravelling **(c)**.

1. There is at most one mp-condition p^k resulting from the fusion of conditions labelled by place p of Σ occurring at depth $k \geq 1$.
2. Two distinct conditions in β having the same label and occurrence-depth are either concurrent or in conflict. Hence, if the original Petri net was safe then all the conditions in β which were fused into the same mp-condition p^k of μ were in conflict.
3. For two mp-conditions, p^k and p^{k+1}, there is a directed path from the former to latter. Moreover, if p^{k+1} is present and $k \geq 1$ then p^k is also present.
4. In general, μ is not acyclic (cycles can arise due to criss-cross fusions of conditions, as illustrated in Figure 2(b,c)). This, in turn, leads to complications for model checking, in particular the marking equation can have *spurious* solutions, i.e., solutions which do not correspond to any reachable marking. To simplify model checking, one could stop fusing conditions in Definition 2 when this leads to cycles, but this is not a satisfactory solution, since μ is not uniquely defined in such a case; moreover, this would lead to lower compression. So we chose to allow cycles, and strengthen the marking equation with additional constraints excluding spurious solutions (see Proposition 6).
5. There can be events consuming conditions in the postset of a cut-off mp-event.

6. There is a strong correspondence between the runs of Σ and those of its unravelling: σ is a run of Σ iff $\sigma = \widehat{h}(\widehat{\sigma})$ for some run $\widehat{\sigma}$ of the unravelling of Σ.

A multiset $\widehat{C}$ of mp-events is an *mp-configuration* of μ if $\widehat{C} = \hbar(C)$ for some configuration C of the unfolding of Σ (that we refer to the full unfolding rather than β here is a subtle point explained in [8]). If $\widehat{C}$ is an mp-configuration then the corresponding *mp-cut* $Cut(\widehat{C})$ is defined as the marking of μ reached by executing all the events of $\widehat{C}$ starting from the initial marking $\widehat{M_0}$. ($Cut(\widehat{C})$ can be efficiently computed using, e.g., the marking equation.) Moreover, $Mark(\widehat{C})$ is defined as $\widehat{h}(Cut(\widehat{C}))$. Note that if $\widehat{C} = \hbar(C)$ then $Mark(\widehat{C}) = Mark(C)$.

Canonical merged processes Since $\mathfrak{Merge}$ is a deterministic transformation, one can easily define the *canonical* merged process as $\mathfrak{Merge}(\beta)$, where β is the canonical unfolding prefix [9]. This allows for an easy import of the results of [7, 9] related to the canonicity.

The size of a merged process One can see that in Definition 2 the fusion of conditions can only decrease the number of conditions without affecting the number of events or arcs; moreover, merging events can only decrease the number of events and arcs, without affecting the number of conditions. Hence, the following result holds:

Proposition 1 (size). *If β is finite then μ is finite and $|\widehat{B}| \leq |B|$, $|\widehat{E}| \leq |E|$ and $|\widehat{G}| \leq |G|$.*

This result allows to import all the upper bounds proved for unfolding prefixes [5, 7, 9]; in particular, since for every safe Petri net Σ one can build a marking-complete branching process with the number of events not exceeding the number of reachable markings of Σ, the corresponding merged process μ has the same upper bound on the number of its events. However, the upper bound given by Proposition 1 is rather pessimistic; in practice, merged processes turn out to be much more compact than the unfolding prefixes.

Tables 1 and 2 show the results of our experiments. The popular set of benchmarks collected by J.C. Corbett [2] has been attempted. The meaning of the columns is as follows (from left to right): the name of the problem; the number of places and transitions in the original Petri net; the number of conditions, events and cut-off events in the unfolding prefix; the time taken by deadlock checking based on unfoldings (discussed in the next section); the number of mp-conditions and mp-events in the corresponding merged process; the time taken by deadlock checking based on merged processes (discussed in the next section); and the ratios $|\widehat{E}|/|T|$ and $|E|/|\widehat{E}|$ giving measures of compactness of the merged process relative to the original Petri net and its unfolding prefix, respectively. The unfolding prefixes in our experiments were built using the algorithm described in [5, 7, 9], and the corresponding merged processes were obtained by

Problem	Net		Unfolding				Unravelling				
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	MC [s]	$\|\widehat{B}\|$	$\|\widehat{E}\|$	MC [s]	$\|\widehat{E}\|/\|T\|$	$\|E\|/\|\widehat{E}\|$
Q	163	194	16123	8417	1188	<1	248	256	<1	1.32	32.88
Speed	33	39	4929	2882	1219	<1	92	175	<1	4.49	16.47
Dac(6)	42	34	92	53	0	<1	42	35	<1	1.03	1.51
Dac(9)	63	52	167	95	0	<1	63	53	<1	1.02	1.79
Dac(12)	84	70	260	146	0	<1	84	71	<1	1.01	2.06
Dac(15)	105	88	371	206	0	<1	105	89	<1	1.01	2.31
Dp(6)	36	24	204	96	30	<1	60	37	<1	1.54	2.59
Dp(8)	48	32	368	176	56	<1	80	49	<1	1.53	3.59
Dp(10)	60	40	580	280	90	<1	100	61	<1	1.53	4.59
Dp(12)	72	48	840	408	132	<1	120	73	<1	1.52	5.59
Elev(1)	63	99	296	157	59	<1	73	89	<1	0.90	1.76
Elev(2)	146	299	1562	827	331	<1	150	241	<1	0.81	3.43
Elev(3)	327	783	7398	3895	1629	<1	304	588	<1	0.75	6.62
Elev(4)	736	1939	32354	16935	7337	<1	634	1387	<1	0.72	12.21
Hart(25)	127	77	179	102	1	<1	153	102	<1	1.32	1.00
Hart(50)	252	152	354	202	1	<1	303	202	<1	1.33	1.00
Hart(75)	377	227	529	302	1	<1	453	302	<1	1.33	1.00
Hart(100)	502	302	704	402	1	<1	603	402	<1	1.33	1.00
Key(2)	94	92	1310	653	199	<1	147	402	<1	4.37	1.62
Key(3)	129	133	13941	6968	2911	<1	201	1086	11	8.17	6.42
Key(4)	164	174	135914	67954	32049	<1	255	2054	69	11.80	33.08
Mmgt(1)	50	58	118	58	20	<1	61	58	<1	1.00	1.00
Mmgt(2)	86	114	1280	645	260	<1	111	282	<1	2.47	2.29
Mmgt(3)	122	172	11575	5841	2529	2	159	662	<1	3.85	8.82
Mmgt(4)	158	232	92940	46902	20957	10	207	1206	<1	5.20	38.89
Sent(25)	104	55	383	216	40	<1	120	81	<1	1.47	2.67
Sent(50)	179	80	458	241	40	<1	195	106	<1	1.33	2.27
Sent(75)	254	105	533	266	40	<1	270	131	<1	1.25	2.03
Sent(100)	329	130	608	291	40	<1	345	156	<1	1.20	1.87

Table 1. Experimental results for benchmarks with deadlocks.

application of the algorithm given by Definition 2. (The time taken by this algorithm is not included in the tables because it was negligible.) The algorithm for building merged processes directly from Petri nets is a matter of future research [8] (significant progress has already been made).

One can see that merged processes can be by orders of magnitude smaller than unfolding prefixes, and, in many cases, are just slightly greater than the original Petri nets. In fact, in some of the examples merged processes are *smaller than the original Petri nets* due to the elimination of dead transitions. However, merged processes are much more amenable to model checking than general safe Petri nets — e.g., most of 'interesting' behaviourial properties are known to be $\mathcal{PSPACE}$-complete for safe Petri nets [4], whereas in Section 4 we develop a non-deterministic polynomial-time algorithm for checking reachability-like properties of merged processes, i.e., many behaviourial properties of merged processes are in $\mathcal{NP}$. Since many such properties are known to be $\mathcal{NP}$-complete already for unfolding prefixes, the complexity class is not worsened if one uses merged processes rather than unfolding prefixes.

Since merged processes are inherently more compact than unfolding prefixes, it would be natural to seek sharper upper bounds than the trivial ones given by Proposition 1. In particular, it would be interesting to identify subclasses of Petri nets whose unfolding prefixes can be exponential in the size of the original Petri

Problem	Net		Unfolding				Unravelling				
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	MC [s]	$\|\widehat{B}\|$	$\|\widehat{E}\|$	MC [s]	$\|\widehat{E}\|/\|T\|$	$\|E\|/\|\widehat{E}\|$
ABP	43	95	337	167	56	<1	75	83	<1	0.87	2.01
BDS	53	59	12310	6330	3701	<1	145	359	<1	6.08	17.63
FTP	176	529	178085	89046	35197	16	304	875	<1	1.65	101.77
CYCLIC(3)	23	17	52	23	4	<1	39	21	<1	1.24	1.10
CYCLIC(6)	47	35	112	50	7	<1	84	45	<1	1.29	1.11
CYCLIC(9)	71	53	172	77	10	<1	129	69	<1	1.30	1.12
CYCLIC(12)	95	71	232	104	13	<1	174	93	<1	1.31	1.12
DME(2)	135	98	487	122	4	<1	309	98	<1	1.00	1.24
DME(3)	202	147	1210	321	9	<1	463	148	<1	1.01	2.17
DME(4)	269	196	2381	652	16	<1	617	197	<1	1.01	3.31
DME(5)	336	245	4096	1145	25	<1	771	246	<1	1.00	4.65
DME(6)	403	294	6451	1830	36	<1	925	295	<1	1.00	6.20
DME(7)	470	343	9542	2737	49	<1	1079	344	<1	1.00	7.96
DME(8)	537	392	13465	3896	64	<1	1233	393	<1	1.00	9.91
DME(9)	604	441	18316	5337	81	<1	1387	442	<1	1.00	12.07
DME(10)	671	490	24191	7090	100	2	1541	491	<1	1.00	14.44
DME(11)	738	539	31186	9185	121	2	1695	540	<1	1.00	17.01
DPD(4)	36	36	594	296	81	<1	81	78	<1	2.17	3.79
DPD(5)	45	45	1582	790	211	<1	102	100	<1	2.22	7.90
DPD(6)	54	54	3786	1892	499	<1	123	122	<1	2.26	15.51
DPD(7)	63	63	8630	4314	1129	<1	144	144	<1	2.29	29.96
DPFM(2)	7	5	12	5	2	<1	10	5	<1	1.00	1.00
DPFM(5)	27	41	67	31	20	<1	31	31	<1	0.76	1.00
DPFM(8)	87	321	426	209	162	<1	89	209	<1	0.65	1.00
DPFM(11)	1047	5633	2433	1211	1012	<1	313	1211	<1	0.21	1.00
DPH(4)	39	46	680	336	117	<1	87	108	<1	2.35	3.11
DPH(5)	48	67	2712	1351	547	<1	129	293	<1	4.37	4.61
DPH(6)	57	92	14590	7289	3407	<1	198	904	2313	9.83	8.06
DPH(7)	66	121	74558	37272	19207	1	277	2773	>10 hrs	22.92	13.44
FURN(1)	27	37	535	326	189	<1	70	98	<1	2.65	3.33
FURN(2)	40	65	4573	2767	1750	<1	121	432	<1	6.65	6.41
FURN(3)	53	99	30820	18563	12207	<1	180	1224	<1	12.36	15.17
GASNQ(2)	71	85	338	169	46	<1	87	103	<1	1.21	1.64
GASNQ(3)	143	223	2409	1205	401	<1	173	325	<1	1.46	3.71
GASNQ(4)	258	465	15928	7965	2876	6	308	748	21	1.61	10.65
GASNQ(5)	428	841	100527	50265	18751	321	505	1449	4455	1.72	34.69
GASQ(1)	28	21	43	21	4	<1	35	21	<1	1.00	1.00
GASQ(2)	78	97	346	173	54	<1	96	111	<1	1.14	1.56
GASQ(3)	284	475	2593	1297	490	<1	316	509	<1	1.07	2.55
GASQ(4)	1428	2705	19864	9933	4060	9	1540	3004	34	1.11	3.31
OVER(2)	33	32	83	41	10	<1	51	39	<1	1.22	1.05
OVER(3)	52	53	369	187	53	<1	89	97	<1	1.83	1.93
OVER(4)	71	74	1536	783	237	<1	138	217	<1	2.93	3.61
OVER(5)	90	95	7266	3697	1232	<1	186	375	<1	3.95	9.86
RING(3)	39	33	97	47	11	<1	58	40	<1	1.21	1.18
RING(5)	65	55	339	167	37	<1	110	97	<1	1.76	1.72
RING(7)	91	77	813	403	79	<1	160	146	<1	1.90	2.76
RING(9)	117	99	1599	795	137	<1	210	194	<1	1.96	4.10
RW(6)	33	85	806	397	327	<1	51	85	<1	1.00	4.67
RW(9)	48	181	9272	4627	4106	<1	75	181	<1	1.00	25.56
RW(12)	63	313	98378	49177	45069	<1	99	313	<1	1.00	157.12

Table 2. Experimental results for deadlock-free benchmarks.

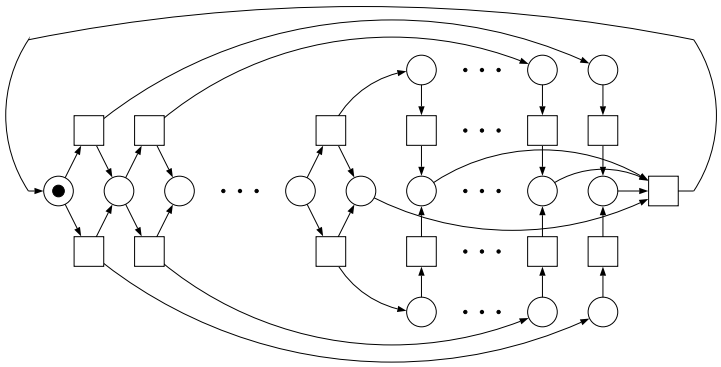

Fig. 3. An $LSFC^2$ Petri net whose unfolding prefix is exponential in its size.

net, but whose merged prefixes are guaranteed to be only polynomial. Below, we present two such results.

Proposition 2 (unravelling of an acyclic Petri net). *If Σ is an acyclic Petri net then its unravelling is isomorphic to the Petri net obtained from Σ by removing all its dead transitions and unreachable places.*

This result easily follows from the fact that no token in an acyclic Petri net can 'visit' a place more than once, and thus the occurrence-depth of every condition in the unfolding of Σ is 1. On the other hand, unfolding prefixes of even safe acyclic Petri nets can be exponential in the size of the original nets, e.g., this is the case for the acyclic Petri net in Figure 1(a) with the dashed part taken into account.

In the discussion below, $LSFC^k$ denotes the class of live and safe free-choice Petri nets [3] whose transitions' postsets have cardinality less than or equal to $k \in \mathbb{N} \cup \{\infty\}$; hence, $LSFC^\infty$ denotes the whole class of live and safe free-choice Petri nets. It turns out that if $k \neq \infty$ then the marking-complete merged processes for the nets in $LSFC^k$ are polynomial in the size of the original nets, even though their unfolding prefixes can be exponential; e.g., one can make the Petri net in Figure 1(a) (with the dashed part taken into account) live by adding a subnet 'gathering' tokens at the end of the execution and returning a token to the initial place, as shown in Figure 3. This net is in $LSFC^2$ and its complete prefix is exponential in its size.

Proposition 3 (merged processes of $LSFC^k$-nets [8]). *For any $k \in \mathbb{N}$, there exist marking-complete merged processes of $LSFC^k$-nets polynomial in the sizes of the original nets.*

This result is unlikely to be generalised to $LSFC^\infty$ [8]. However, one should note that the expressive power of $LSFC^k$ for $k \geq 2$ is comparable with that of $LSFC^\infty$, since every transition of an $LSFC^\infty$-net with postset of cardinality greater than k can be replaced by a tree of transitions with postsets of cardinality not exceeding k, and the resulting Petri net will be in $LSFC^k$.

Finiteness of a merged process In view of Proposition 1, μ is finite if β is. However, it is not obvious that the reverse holds, since, in general, infinitely many nodes of β can correspond to a single node of μ [8]. However, by the analog of König's lemma for branching processes [7, 9], if β is infinite then there exists an infinite path in β. Since the number of places in Σ is finite, some place $p \in P$ is repeated infinitely many times along this path, and so the occurrence-depth of its instances grows unboundedly in β. Thus there are infinitely many instances of p after fusion, and the following result holds:

Proposition 4. *μ is finite iff β is finite.*

Again, this result allows to import into the new framework all the finiteness results proved for unfolding prefixes [5, 7, 9].

Completeness of a merged process The marking-completeness of a merged process is defined similarly to the marking-completeness of a branching process. A merged process μ is *marking-complete w.r.t. a set* $\widehat{E_{cut}} \subseteq \widehat{E}$ if for every reachable marking M of Σ there exists an mp-configuration $\widehat{C}$ of μ such that $\widehat{C} \cap \widehat{E_{cut}} = \varnothing$ and $Mark(\widehat{C}) = M$.

Let C be a configuration of β and $\widehat{C} = \hbar(C)$ be the corresponding configuration in μ. One can easily show that if C contains no cut-off event then $\widehat{C}$ contains no cut-off mp-events, and that $Mark(C) = Mark(\widehat{C})$. Hence:

Proposition 5. *If β is marking-complete then μ is marking-complete.*

However, no such result holds for full completeness [8]; therefore, model checking algorithms developed for unfolding prefixes relying on the preservation of firings (e.g., some of the deadlock checking algorithms in [6, 7, 11–13]) cannot be easily transferred to merged processes. However, marking-completeness is sufficient for most purposes, as the transitions enabled by the final state of an mp-configuration can be easily found using the original Petri net. The model checking algorithm proposed in the next section does not make use of cut-off mp-events, and so they can be removed from the merged process before model checking.

4 Model checking based on merged processes

Model checking algorithms [6, 7, 10–13] working on complete prefixes of Petri net unfoldings are usually based on the following non-deterministic algorithm:

choose a set of events $C \subseteq E \setminus E_{cut}$
if C is a configuration violating the property (e.g., deadlock-freeness)
then accept /* C is a certificate convertible to a witness trace */
else reject

Various kinds of solvers have been employed to implement it, e.g., ones based on mixed-integer programming [13], stable models of logic programs [6], integer

programming [7] and Boolean satisfiability (SAT) [10]. More precisely, a system of constraints having for each non-cut-off event e of the prefix a variable conf_e is built (it might also contain other variables), and for every satisfying assignment A, the set of events $C \stackrel{\text{df}}{=} \{e \mid A(\mathsf{conf}_e) = 1\}$ is a configuration such that $\mathit{Mark}(C)$ violates the property being checked. This system of constraints usually has the form $\mathcal{CONF}\&\mathcal{VIOL}$. The role of the *configuration constraint,* $\mathcal{CONF}$, is to ensure that C is a configuration of the prefix (not just an arbitrary set of events), and the role of the *violation constraint,* $\mathcal{VIOL}$, is to capture the property violation condition for a configuration C, so that if a configuration C satisfying this constraint is found then the property (e.g., deadlock-freeness) does not hold, and any ordering of events in C consistent with the causal order on the events of the prefix is a violation trace.

It is natural to follow a similar approach for verification based on merged processes. However, one should bear in mind the following complications:

- An mp-configuration is generally a multiset (rather than a set) of mp-events. Though this is not a major problem, it does hamper verification employing Boolean solvers, as associating a single Boolean variable with each mp-event is no longer sufficient for representing an mp-configuration. But if the original Petri net is safe, the mp-configurations of its merged processes are sets.
- An easily testable characterisation of an mp-configuration is necessary (our 'indirect' definition of an mp-configuration as an $\hbar$-image of some configuration of the unfolding is not of much use for model checking). In what follows we develop such a characterisation for mp-configurations of merged processes of safe Petri nets. Some issues make it non-trivial to develop such a characterisation:

  Spurious solutions of the marking equation Many model checking algorithms working on unfolding prefixes [6, 7, 10, 13] are based on the marking equation (perhaps expressed not as integer linear constraints but in some other form, e.g., as a Boolean formula) and the fact that for acyclic Petri nets it cannot have spurious solutions [15]. Since merged processes are not generally acyclic, the marking equation *can* have spurious solutions. For example, the associated marking equations for the unravelling shown in Figure 2(c) has a spurious solution: if one 'borrows' a token in p_4^1 then the t_3- and t_4-labelled mp-events forming a cycle can be executed, returning the borrowed token to p_4^1 and leading to the spurious marking $\{p_2^1\}$.

  Spurious runs The correspondence between the runs and mp-configurations of μ is not very straightforward: some of its runs (e.g., the run comprised of the instance of t_1 followed by the left instance of t_3 in Figure 2(c)) do not form mp-configurations.

Below we solve these problems for merged processes of safe Petri nets.

The case of safe Petri nets

To capture the notion of an mp-configuration in the case when the original Petri net Σ is safe, we proceed as follows. Let C be a configuration of β, and $\widehat{C}$ be a

set of mp-events of μ. Below, $\mathcal{G}(C)$ and $\mathcal{G}(\widehat{C})$ will denote two graphs induced by the events of C together with their adjacent conditions and the minimal (w.r.t. $<$) conditions of β and by the mp-events of $\widehat{C}$ together with their adjacent mp-conditions and the initially marked mp-conditions of μ, respectively.

We say that $\widehat{C}$ satisfies: (a) $\mathcal{ME}$ if it is a solution of the marking equation for μ; (b) $\mathcal{ACYCLIC}$ if $\mathcal{G}(\widehat{C})$ is acyclic; and (c) $\mathcal{NG}$ (no-gap) if, for all $k > 1$ and all places p of Σ, the following holds: if p^k is a node in $\mathcal{G}(\widehat{C})$ then p^{k-1} is also a node in $\mathcal{G}(\widehat{C})$. Note that if $\widehat{C} = \hbar(C)$ then $\mathcal{G}(C)$ is isomorphic to $\mathcal{G}(\widehat{C})$ (including the labelling in terms of places and transitions). The next result gives a direct characterisation of mp-configurations and is crucial for model checking:

Proposition 6 (mp-configurations in the safe case [8]). *A set of mp-events $\widehat{C}$ is an mp-configuration iff $\mathcal{ME}\&\mathcal{ACYCLIC}\&\mathcal{NG}$ holds for $\widehat{C}$.*

Hence it is enough for model checking to take $\mathcal{CONF} \stackrel{\text{df}}{=} \mathcal{ME}\&\mathcal{ACYCLIC}\&\mathcal{NG}$ and apply an algorithm similar to that described in the beginning of this section for unfolding prefixes.

We implemented a deadlock checking algorithm based on merged processes using zChaff [14] as the underlying SAT solver. (Note that other reachability-like properties can also be implemented simply by adjusting the $\mathcal{VIOL}$ constraint.) All the experiments were conducted on a PC with a Pentium^{TM} IV/2.8GHz processor and 512M RAM.

The implementation of the $\mathcal{ME}$ and $\mathcal{VIOL}$ constraints as Boolean formulae is very similar to that for unfoldings and not discussed here. The $\mathcal{NG}$ constraint has been implemented as a conjunction of implications of the form $\mathsf{conf}_{p^k} \rightarrow \mathsf{conf}_{p^{k-1}}$, for all mp-conditions p^k such that $k > 1$. (Intuitively, $\mathsf{conf}_{p^k} = 1$ conveys that p^k is in $\mathcal{G}(\widehat{C})$; similarly, $\mathsf{conf}_{\widehat{e}} = 1$ conveys that $\widehat{e}$ is in $\mathcal{G}(\widehat{C})$, for each non-cut-off mp-event $\widehat{e}$ of μ.)

The implementation of $\mathcal{ACYCLIC}$ constraint is different from that in [8] (and so we report better results for deadlock checking). The problem can be re-formulated as follows: given a digraph $G = (V, E)$ (representing μ) with a boolean variable conf_v associated with each vertex $v \in V$, construct a boolean formula $\mathcal{ACYCLIC}$ (depending on the variables conf_* and, perhaps, other variables) such that, given an assignment to variables conf_*, the formula obtained from $\mathcal{ACYCLIC}$ by substituting the variables conf_* by their values is satisfiable iff the subgraph of G induced by the vertices whose corresponding variables were assigned to 1 is acyclic. (Note that $\mathcal{ME}$, $\mathcal{NG}$ and $\mathcal{VIOL}$ also contain the variables conf_*.)

Since each cycle is contained in some strongly connected component of G, one can partition G into its strongly connected components, generate such a constraint for each of them separately and form $\mathcal{ACYCLIC}$ as their conjunction. For each strongly connected component $G_k = (V_k, E_k)$ of $G = (V, E)$, the vertices are sorted to heuristically minimise the number of *feedback vertices*, i.e., vertices $v \in V_k$ for which there exists $w \in V_k$ such that $(w, v) \in E_k$ and $w > v$ (since the vertices of G_k are ordered, we identify each vertex $v \in V_k$ with its position in this order). Then for each such a feedback vertex $v \in V_k$ the following formula

is generated (reach_* are auxiliary variables created separately for each such v):

$$(\mathsf{conf}_v \rightarrow \mathsf{reach}_v) \wedge \bigwedge_{\substack{(x,y)\in E_k \\ x \geq v \wedge y > v}} \big((\mathsf{reach}_x \wedge \mathsf{conf}_y) \rightarrow \mathsf{reach}_y\big) \wedge \bigwedge_{\substack{(w,v)\in E_k \\ w > v}} \neg \mathsf{reach}_w \ .$$

The idea behind this formula is to perform a reachability analysis in G_k starting from v and ignoring all the vertices which precede v in the chosen order or are not selected. Note that if the values of the variables conf_* are fixed then this formula is unsatisfiable iff at least one of the sources of the feedback arcs ending at v is reachable from v (and hence there is a cycle); moreover, the unsatisfiability can be proven by unit resolution alone, i.e., one can setup the solver not to branch on the variables reach_*.

The experimental results in Tables 1 and 2 show that the developed model checking algorithm is quite practical and it even outperformed the one working on unfolding prefixes on some of the benchmarks. On the other hand, its performance deteriorated on the DPH and GASNQ series. We reckon that this is due to our still inefficient implementation of the $\mathcal{ACYCLIC}$ constraint, and that this can be significantly improved (major improvements over the results reported in [8] have already been achieved due to a different implementation of $\mathcal{ACYCLIC}$).

The point we are making with these results is: merged processes are a more compact behaviour representation than unfolding prefixes, but still allow model checking of reachability-like properties in at least comparable time. Since space considerations are of utmost importance in model checking, we regard this as very promising — although, to make merged processes practical, we still have to develop an *unravelling algorithm* that builds them directly from Petri nets instead of deriving them from unfolding prefixes (significant progress has already been made).

5 Conclusions and future work

We proposed the notion of a merged process — a new condensed representation of a Petri net's behaviour allowing one to contain state space explosion arising not only from concurrency, but also from a sequence of choices and from non-safeness of the Petri net. Experimental results show that merged processes can be smaller by orders of magnitude than the corresponding unfolding prefixes, and are in many cases not much bigger than the original Petri nets. Many results developed for Petri net unfoldings (related to canonicity, finiteness, completeness and size) have been transferred to the new framework. Moreover, we proved sharper upper bounds for some of the net subclasses and directly characterised the mp-configurations of merged processes of safe Petri nets, which allowed us to develop a model checking algorithm.

We now identify possible directions for future study (see also the discussion in [8]): (i) direct characterisation of merged processes (cf. the characterisation of branching processes by occurrence nets); (ii) direct characterisation of (general)

mp-configurations (for non-safe Petri nets this is still an open problem); (iii) more efficient model checking; and (iv) direct unravelling algorithm.

Acknowledgements The authors would like to thank Keijo Heljanko for a helpful discussion about expressing $\mathcal{ACYCLIC}$ and Javier Esparza for sharing his expertise on $LSFC$ nets. This research was supported by the EC IST grant 511599 (RODIN).

References

1. E. M. Clarke, O. Grumberg and D. Peled: *Model Checking*. MIT Press (1999).
2. J. C. Corbett: Evaluating Deadlock Detection Methods for Concurrent Software. *IEEE Transactions on Software Engineering* 22 (1996) 161–180.
3. J. Desel and J. Esparza: *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press (1995).
4. J. Esparza: Decidability and Complexity of Petri Net Problems — an Introduction. In: *Lectures on Petri Nets I: Basic Models*, LNCS 1491 (1998) 374–428.
5. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20 (2002) 285–310.
6. K. Heljanko: Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets. *Fundamenta Informatica* 37 (1999) 247–268.
7. V. Khomenko: *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD Thesis, School of Computing Science, University of Newcastle upon Tyne (2003).
8. V. Khomenko, A. Kondratyev, M. Koutny and V. Vogler: Merged Processes — a New Condensed Representation of Petri Net Behaviour. Technical Report CS-TR-884, School of Computing Science, University of Newcastle (2005). URL: `http://homepages.cs.ncl.ac.uk/victor.khomenko/home.formal/papers/CS-TR-884.pdf`
9. V. Khomenko, M. Koutny and V. Vogler: Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica* 40 (2003) 95–118.
10. V. Khomenko, M. Koutny and A. Yakovlev: Detecting State Coding Conflicts in STG Unfoldings Using SAT. *Fundamenta Informatica* 62 (2004) 221–241.
11. K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *CAV'1992*, LNCS 663 (1992) 164–174.
12. K. L. McMillan: *Symbolic Model Checking: an Approach to the State Explosion Problem*. PhD thesis, CMU-CS-92-131 (1992).
13. S. Melzer and S. Römer: Deadlock Checking Using Net Unfoldings. Proc. of *Computer Aided Verification (CAV'97)*, LNCS 1254 (1997) 352–363.
14. S. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik: CHAFF: Engineering an Efficient SAT Solver. Proc. of *DAC'2001*, ASME Tech. Publ. (2001) 530–535.
15. T. Murata: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77 (1989) 541–580.
16. A. Valmari: The State Explosion Problem. In: *Lectures on Petri Nets I: Basic Models*, LNCS 1491 (1998) 429–528.