Property-Preserving Generation of Tailored Benchmark Petri Nets

Bernhard Steffen*, Marc Jasper*, Jeroen Meijer† and Jaco van de Pol†
*Chair of Programming Systems, TU Dortmund University, Dortmund, Germany
Email: {steffen,marc.jasper}@cs.tu-dortmund.de
†Formal Methods and Tools, University of Twente, Enschede, The Netherlands
Email: {j.j.g.meijer,j.c.vandepol}@utwente.nl

***Abstract*—Bottleneck of the validation and evaluation of analysis and verification tools for distributed systems is the shortage of benchmark problems. Specifically designed benchmark problems are typically artificial, rare, and small, and it is difficult to guarantee challenging properties of realistic benchmarks. This paper shows how to systematically construct arbitrarily complex Petri Nets with guaranteed safety properties. Key to our construction is a top-down parallel decomposition based on lightweight assumption commitment specifications. We will illustrate how a specific strategy for design choices, which may well be automated, leads to benchmarks that grow exponentially with the number of its parallel components, and that are very difficult to verify. In particular, we will report numbers from a systematic sequence of concrete corresponding verification attempts using today's leading verification technology.**

I. Introduction

Today's software verification and analysis tools are increasingly complex and often comprise diverse technologies like SMT solving, data and process mining, statistical methods or even runtime analysis. This hybrid tool structure makes traditional verification of verification tools almost intractable and asks for alternative validation support. Bottleneck of experimental evaluation approaches, in particular for analysis and verification tools for distributed systems, is the shortage of adequate benchmark problems [22], [21] which are of challenging size and structure, and guaranteed to exhibit/violate interesting (temporal) properties. 'Realistic' benchmark systems come with the risk that it is unknown whether the considered property holds [21]. In such cases, the presumed solution is often chosen by some kind of majority vote which is, of course, no guarantee for correctness. On the other hand, manual benchmark design typically does not scale and therefore does not provide truly challenging verification scenarios. Work on the systematic construction of benchmark systems, like [18], [32], [11], is still very limited for distributed systems.

In this paper, we systematically apply the approach presented in [33] by sketching an incremental process to expand a given benchmark scenario $B(M, \Phi)$ that consists of a Model Transition System (MTS) [25] specification M for some concurrent implementation of controllable size[1] together with a set of properties Φ that is guaranteed to be correct for M[2]. This expansion results in a system with an arbitrary degree of parallelism, where all parallel components need to be considered for validation.

Key to our approach is the property-preserving parallel decomposition in a light-weight assumption commitment style. Property preservation is guaranteed on the basis of *Modal Contracts* (MCs) that permit a (weak) refinement into a component and its context while supporting the propagation of dependencies that are vital for the validity of considered properties. More technically, our development is based on the weak refinement [19] of convergent systems which preserves an interesting class of temporal properties [24], [4].

Given an initial benchmark scenario $B(M, \Phi)$, we systematically construct a corresponding parallel benchmark scenario via property preserving decomposition in two phases:

1) Phase I: Constructing a contract from $B(M, \Phi)$ in terms of an MC I by

- choosing a sub-alphabet Γ of M's alphabet Σ,
- marking must transitions with labels from Γ by coloring them green, and
- (randomly) adding red transitions labelled with Γ in a way that does not conflict with existing may transitions.

The intuition here is that the to be constructed context system M_c must guarantee to provide communication partners for green transitions, while it has to make sure that red transitions will never find a communication partner.

2) Phase II: Decomposing I into a system component M_s and a context component M_c both of which may be further decomposed during successive iterations. Important is here that both components are vital for the validity of Φ: Neither M_s nor M_c alone suffice to guarantee Φ, but their parallel composition does.

Based on this approach it is possible to device strategies guaranteeing that generated benchmarks grow exponentially with the number of its parallel components. We will illustrate the effectiveness of this approach by reporting numbers from a sequence of systematic, concrete corresponding verification attempts using today's leading verification technology.

[1]What we mean here is that M can be conveniently model checked wit state of the art technology.

[2]Our exposition focuses on the preservation of validity. It should be noted that our MTS-based approach also maintains the existence of counterexamples, which is something different for linear time temporal formulas.

Our approach harnesses the power of correctness by construction [23] where the essential dependencies are designed and therefore known during the iterative decomposition process. Revealing them afterwards during a-posteriori verification is a very different challenge, similar in flavor to the difference between proof checking and proof construction.

In contrast to classical assumption commitment [15] and approaches like the ones presented in [13], [14], the iterative decomposition based on MCs scales very well. However, admittedly, to achieve a different kind of goal because we do not require completeness and can therefore focus on a simplicity-oriented approach [26]. This scalability, which intuitively exists due to the difference between a posteriori verification and correctness by construction, can be regarded as the essence of our benchmark generation approach [18], [32], [11].

After introducing relevant preliminaries in Section II, Section III introduces our notion of Modal Contracts, the basis for our corresponding decomposition process, before the construction of an adequate context MTS M_c is described in Section IV. Subsequently, Section V illustrates our approach by developing benchmark scenarios of exploding size. Section VI shows how these benchmark scenarios can be transformed into Petri nets. Our experimental data is presented in Section VII, before Section VIII concludes this paper and presents some directions to future work.

II. Preliminaries

The Modal Contracts (MCs) proposed in this paper are an extension of modal transition systems (MTSs). This section introduces fundamental definitions that are important for understanding the remainder of this paper. We assume that the reader is familiar with regular languages and related automata.

Definition 1 (Modal Transition Systems):
Let S be a set of states and Σ an alphabet of action symbols. $\boldsymbol{M = (S, s_0, \Sigma, \diamond, \square)}$ is called a **(rooted) modal transition system (MTS)** with root s_0 if the following condition holds:

$$\square \subseteq \diamond \subseteq (S \times \Sigma \times S)$$

Elements of $\diamond$ are called may transitions, those of $\square$ must transitions. We sometimes call the set $(\diamond \setminus \square)$ may-only transitions. Throughout this paper, the domain of all possible MTSs is referred to as $\mathcal{M}$.

We further define the operators $states(M) =_{def} S$, $alph(M) =_{def} \Sigma$, $may(M) =_{def} \diamond$, and $must(M) =_{def} \square$. For any $t = (p, \sigma, q) \in \diamond$, we call $sym(t) =_{def} \sigma$ the symbol or label of t. Operator $sym(\cdot)$ extends naturally to transition relations $T \subseteq (S \times \Sigma \times S)$ by $sym(T) =_{def} \{sym(t) \mid t \in T\}$.

An MTS can be seen as a generalisation of a traditional (rooted) labeled transition system (LTS), which allows the following definition:

Definition 2 (Labeled Transition Systems):
A **labeled transition system (LTS)** is an MTS $M = (S, s_0, \Sigma, \diamond, \square)$ with $\diamond = \square$.

The **language $\boldsymbol{\mathcal{L}(M)}$** of M is defined as the language of the related prefix-closed non-deterministic finite automaton (NFA) that results from marking all states in S as accepting.

We sometimes use an equivalent syntax $M = (S, s, \Sigma, \rightarrow)$ and denote transitions $(p, \sigma, q) \in \rightarrow$ as $p \xrightarrow{\sigma} q$ when modalities are no longer relevant.

Intuitively speaking, a may transition in an MTS stands for an underspecification and indicates a transition that may or may not be present in an actual implementation. A modal transition system therefore specifies a set of LTSs. These LTSs can be retrieved by refinement according to the following definition [25]:

Definition 3 (MTS Refinement):
Let $M_p = (S_p, s_0^p, \Sigma_p, \diamond_p, \square_p), M_q = (S_q, s_0^q, \Sigma_q, \diamond_q, \square_q) \in \mathcal{M}$ be two MTSs. A relation $\boldsymbol{\lesssim} \subseteq (S_p \times S_q)$ is called a **refinement** if the following hold for all $(p, q) \in \lesssim$:

1.) $\forall (p, \sigma, p') \in \diamond_p, \exists (q, \sigma, q') \in \diamond_q : (p', q') \in \lesssim$
2.) $\forall (q, \sigma, q') \in \square_q, \exists (p, \sigma, p') \in \square_p : (p', q') \in \lesssim$

We write $M_p \lesssim M_q$ if there exists a refinement $\lesssim$ with $(s_0^p, s_0^q) \in \lesssim$. In addition, we call M_p a strict refinement of M_q, denoted as $M_p \lesssim\!\!\!\!\!\!\diagup\; M_q$, if $M_p \lesssim M_q$ and $M_q \not\lesssim M_p$.

For the construction of adequate contexts, the maximal language defined by an MTS is important.

Definition 4 (Largest Language of an MTS):
Let $M = (S, s_0, \Sigma, \diamond, \square)$ be an MTS. We call the language

$$\mathcal{L}_\top(M) =_{def} \mathcal{L}((S, s_0, \Sigma, \diamond, \diamond))$$

the **largest language of $\boldsymbol{M}$**.

The parallel composition operator we consider in this paper for MTSs is reminiscent of CSP [17] with synchronization of components on their common alphabets.

Definition 5 (Parallel MTS Composition):
Let $M_p = (S_p, s_0^p, \Sigma_p, \diamond_p, \square_p), M_q = (S_q, s_0^q, \Sigma_q, \diamond_q, \square_q) \in \mathcal{M}$ be two MTSs, and let $T \in \{\diamond, \square\}$ identify the type of transition. The **parallel composition**

$$(M_p \mid\mid M_q) =_{def} (S_p \times S_q, (s_0^p, s_0^q), \Sigma_p \cup \Sigma_q, \diamond, \square)$$

is then defined as the least commutative and associative operation satisfying the following operational rules: [3]

$$\frac{p \xrightarrow{\sigma}_T p' \quad q \xrightarrow{\sigma}_T q'}{(p, q) \xrightarrow{\sigma}_T (p', q')} \qquad \frac{p \xrightarrow{\sigma}_T p' \quad \sigma \notin \Sigma_q}{(p, q) \xrightarrow{\sigma}_T (p', q)}$$

This allows us to define our notion of a Benchmark Scenario:

Definition 6 (Benchmark Scenarios):
Let $M = (M_1 \mid\mid ... \mid\mid M_n)$ be the parallel composition of n MTSs and Φ a set of properties. Then we call $\boldsymbol{B(M, \Phi)}$ a **benchmark scenario** if each property $\phi \in \Phi$ is either satisfied or violated by M.

It is straightforward to establish that $\mid\mid$ preserves refinement for both operands:

[3] This definition depends on the fact that each must transition is also a may transition.

Proposition 1 (Refinement Monotonicity):
Let $M, M', M'' \in \mathcal{M}$ be three arbitrary MTSs. Refining a component of a parallel composition also refines the composition:

$$(M \lesssim M') \implies ((M \parallel M'') \lesssim (M' \parallel M''))$$

Note that due to the commutativity of operator $\parallel$ this monotonicity holds for both components of a composition.

The following notion of conjunction for MTSs is very close to that of parallel compostion:

Definition 7 (MTS Conjunction):
Let $M_p = (S_p, s_0^p, \Sigma, \diamond_p, \Box_p), M_q = (S_q, s_0^q, \Sigma, \diamond_q, \Box_q) \in \mathcal{M}$ be two MTSs, and let $T \in \{\diamond, \Box\}$ identify the type of transition. The **conjunction**

$$(M_p \wedge M_q) =_{def} (S_p \times S_q, (s_0^p, s_0^q), \Sigma, \diamond, \Box)$$

of M_p and M_q is then defined as a commutative and associative operation satisfying the following operational rules: [4]

$$\frac{p \xrightarrow{\sigma}_\Box p' \quad q \xrightarrow{\sigma}_\diamond q'}{(p,q) \xrightarrow{\sigma}_\Box (p',q')} \qquad \frac{p \xrightarrow{\sigma}_\diamond p' \quad q \xrightarrow{\sigma}_\diamond q'}{(p,q) \xrightarrow{\sigma}_\diamond (p',q')}$$

$$\frac{p \xrightarrow{\sigma}_\Box p' \quad q \not\xrightarrow{\sigma}_\diamond}{(p,q) \xrightarrow{\sigma} error}$$

Whenever an error occurs, the conjunction of M_p and M_q is undefined.

The MTS conjunction of Def. 7 guarantees that a refining MTS refines both components:

Proposition 2 (Conjunction of Refinement Constraints):
Let $M, M_p, M_q \in \mathcal{M}$ be three MTSs. If $(M_p \wedge M_q)$ is defined, then the following holds:

$$(M \lesssim (M_p \wedge M_q)) \iff (M \lesssim M_p \wedge M \lesssim M_q)$$

Alphabet abstraction and, in particular hiding, is an important means to improve the scalability of verification methods. We apply these methods here for generating challenging benchmarks:

Definition 8 (Label Hiding):
Let $M = (S, s_0, \Sigma, \diamond, \Box) \in \mathcal{M}$ be an MTS. Let $\Gamma \subseteq \Sigma$ be a sub-alphabet. The **Γ-hiding**

$$hid_\Gamma(M) =_{def} (S, s_0, ((\Sigma \setminus \Gamma) \cup \{\tau\}), hid(\diamond), hid(\Box))$$

of M relabels all transitions t of M such that $sym(t) \in \Gamma$ with the special symbol $\boldsymbol{\tau}$ and therefore features the following transition relations for all $T \in \{\diamond, \Box\}$:

$$hid(T) = \{(p, \tau, q) \mid \exists \gamma \in \Gamma : (p, \gamma, q) \in T\} \cup \{(p, \sigma, q) \in T \mid \sigma \in (\Sigma \setminus \Gamma)\}$$

In order to prepare the (standard) definition of weak MTS refinement, we define the usual observational relation of a transition relation:

[4]This definition again depends on the fact that each must transition is also a may transition.

Definition 9 (Observational Relation):
Let $(\Sigma \cup \{\tau\})$ be an alphabet and let $T \subseteq (S \times (\Sigma \cup \{\tau\}) \times S)$ be a transition relation between states in S. Let $p, p', q, q' \in S$. We define the **observational relation $obs(T)$** of T as follows.

Let $p \xrightarrow{\sigma} p'$ denote a feasible transition $(p, \sigma, p') \in T$ and $p \xRightarrow{\sigma} p'$ a feasible transition $(p, \sigma, p') \in obs(T)$. The transition relation $obs(T)$ results from an exhaustive application of the following three rules for all $\sigma \in \Sigma$:

$$p \xRightarrow{\epsilon} p \qquad \frac{p \xrightarrow{\tau} p' \quad p' \xRightarrow{\epsilon} q}{p \xRightarrow{\epsilon} q}$$

$$\frac{p \xRightarrow{\epsilon} p' \quad p' \xrightarrow{\sigma} q' \quad q' \xRightarrow{\epsilon} q}{p \xRightarrow{\sigma} q}$$

The observational MTS is now simply defined by replacing the original transition relations by their observable counterparts:

Definition 10 (Observational MTS):
Let $M = (S, s_0, \Sigma, \diamond, \Box) \in \mathcal{M}$ be an MTS. The observational MTS $\omega(M)$ of M is based on the observational expansion of its transition relations (Def. 9):

$$\omega(M) =_{def} (S, s_0, ((\Sigma \setminus \{\tau\}) \cup \{\epsilon\}), obs(\diamond), obs(\Box))$$

This is sufficient to introduce weak MTS refinement [19]:

Definition 11 (Weak MTS Refinement):
Let $M, M' \in MTS$ be two MTSs. **Weak refinement** $\lessapprox$ is defind as follows:

$$(M \lessapprox M') \iff (\omega(M) \lesssim \omega(M'))$$

Weak refinement is insensitive to *divergence*, i.e. the possibility that the system engages in an infinite τ sequence, and therefore does not preserve liveness properties. In order to partially repair this drawback we will reduce our attention to convergent systems:

Definition 12 (Convergent MTS):
An MTS is called **c**onvergent, if every allowed τ sequence is finite.

In the following we show how to systematically construct benchmark scenarios of challenging nature.

III. Modal Contracts

This section establishes our notion of a Modal Contract, which is designed to support the property-preserving decomposition of its argument MTS into two components that need both to be considered for verification. This requires:

Definition 13 (Label Projection):
Let T be a transition relation with $sym(T) = \Sigma$ and $\Gamma \subseteq \Sigma$ be a subset of Σ. We call the transition relation

$$\alpha_\Gamma(T) =_{def} \{(p, \gamma, q) \in T \mid \gamma \in \Gamma\}$$

the **(label) projection** of T onto Γ.

Our notion of modal contract is now defined as follows:

Definition 14 (Modal Contract):
Syntax: Let $M = (S, s_0, \Sigma, \diamond, \Box)$ be an MTS and $\Gamma \subseteq \Sigma$. A

Modal Contract (MC) of M with communication alphabet $\Gamma(I) =_{def} \Gamma$ is a tuple

$$I = (S, s_0, \Sigma, \diamond, \Box, G, R)$$

where

- $G =_{def} \alpha_\Gamma(\Box)$, and
- R is a set of transitions over the alphabet Γ that do not exist in $\diamond$ and such that they are not in conflict with G, meaning there do not exist two paths of may transitions in M with the same label sequence such that one ends with a transition in G and the other with one in R.[5]

Moreover $G(I) =_{def} G$ and $R(I) =_{def} R$, and we color transitions of $G(I)$ green and transitions of $R(I)$ red.

*Definition 15 (**Meaning** of an MC):*
Let $I = (S, s_0, \Sigma, \diamond, \Box, G, R)$ be an MC, and let r be a new sink state $r \notin S$. Define

$$R' =_{def} \{(p, \sigma, r) \mid \exists q \in S : (p, \sigma, q) \in R\}$$

be a redirection of transitions in R to the new sink, and

$$R^* =_{def} R' \cup \{(r, \sigma, r) \mid \sigma \in \Sigma\}$$

the extension of R with arbitrary subsequent behavior. Then I defines a so called **system MTS**

$$M_s(I) =_{def} ((S \uplus \{r\}), s_0, \Sigma, (\diamond \cup R^*), \Box)$$

and a set of corresponding **context MTSs**

$$\mathcal{M}_C(I) =_{def} \{M_c(I) \mid (M_s(I) \parallel M_c(I)) \lesssim M\}$$

An MTS $M_c(I) \in \mathcal{M}_C(I)$ is called a **correct context** of I.

Intuitively speaking, an MC specifies an assume-guarantee contract [15], [2] based on an MTS M such that the parallel composition of the system MTS and a corresponding context component is guaranteed to refine M.

IV. Context Generation

Given an MC I, we now first define a specific MTS $M_c^*(I)$, called the green/red context of I, such that $M_c^*(I)$ is correct with regards to I (cf. Theorem 1). Our goal is to specify an $M_c^*(I)$ with minimal constraints in order to have many possible choices in the actual implementation of this context. Following [33], we first define the green-only context $M_c^g(I)$ and subsequently refine it with the separately defined red-only context $M_c^r(I)$ for I. It is easy to see that the conjunction of the green-only context and the red-only context satisfies the requirements of I (cf. Theorem 1). Subsequently, we generalize our notion of context via alphabet extension, and we show that this generalization can be realized via parallel composition with an independently defined MTS for alphabet extension.

Definition 16 (Language Projection):
Let Σ, Γ be two alphabets with $\Gamma \subseteq \Sigma$. For any word $w = (\sigma_1, ..., \sigma_n) \in \Sigma^*$, the **projection $\alpha_\Gamma(w)$** of w onto Γ results from skipping symbols $\sigma_i \notin \Gamma$. This projection extends naturally to languages.

Using this projection, we can now define the green-only context:[6]

Definition 17 (Green-Only Contexts):
Let $M \in \mathcal{M}$ be an MTS and let I (Def. 14) be an MC of M and F_d be the minimal DFA that describes the prefix-closed language $\alpha_{\Gamma(I)}(\mathcal{L}_\top(M))$.

We define the **green-only context $M_c^g(I)$** as the MTS that is the result of the following transformation based on F_d:

1.) Consider all incoming and outgoing transitions of the unique non-accepting sink state as may-only transitions.
2.) Consider all other transitions as must transitions.
3.) Disregard the property of accepting/non-accepting. states.

The red-only context $M_c^r(I)$ is defined as follows:

Definition 18 (Red-Only Contexts):
Let I (Def. 14) be an MC, and $\mathcal{L}_R$ be the language of words for which a path in I exists that contains a red transition $t \in R$. Let F_d be the minimal DFA that describes the prefix-closed language $(\Gamma(I)^* \setminus \alpha_{\Gamma(I)}(\mathcal{L}_R))$ (see also Def. 16).

We define the **red-only context MTS $M_c^r(I)$** as the MTS that results from the following transformations of F_d:

1.) Remove all incoming and outgoing transitions of the unique non-accepting sink state together with this sink state itself.
2.) Consider all remaining transitions as may-only transitions.
3.) Disregard the property of accepting/non-accepting states.

Green/red contexts are now simply defined via MTS conjunction:

Definition 19 (Green/Red Contexts):
Let I be an MC with red-only context $M_c^r(I)$ (Def. 18) and green-only context $M_c^g(I)$ (Def. 17). Then the corresponding **green/red context $M_c^*(I)$** is defined as follows:

$$M_c^*(I) =_{def} (M_c^r(I) \wedge M_c^g(I))$$

As green and red transitions are guaranteed to be non-conflicting (see Def. 14), the following theorem follows straightforwardly [33]:

Theorem 1 (Correctness of Green/Red Context):
Let $M \in \mathcal{M}$ be an MTS and I be an MC of M (Def. 14) with its green/red context $M_c^*(I)$ according to Def. 19. Then $M_c^*(I)$ is well-defined and the following holds:

$$(M_s(I) \parallel M_c^*(I)) \lesssim M$$

Weak refinement of convergent systems preserves an interesting class of temporal properties [19], [24], [4]. We will

[5] Such a conflict can easily be detected via the determinization of the may automaton for I.

[6] In order to simplify this exposition, the definition of the green-only context is kept simple here. It can be generalized based on the notion of weak refinement and a definition of MTS determinization.

therefore restrict our attention to properties of this class, be they branching time, linear time, safety or liveness properties.

Definition 20 ($\boldsymbol{\Sigma_E}$ Context Extensions):
Let $M \in \mathcal{M}$ be an MTS and let I be an MC of M. Let Σ_E be a new alphabet, i.e. $(\Sigma_E \cap alph(M)) = \emptyset$.

1) An MTS M_E is called **$\boldsymbol{\Sigma_E}$ context extension** of $\Gamma(I)$ if it results from the following three-step construction.
   - Choose an arbitrary MTS over Σ_E with the following two properties:
     - M restricted to its must transitions is deadlock free
     - Each state of M is reachable via must transitions
   - Select a set S of transitions with the property that every infinite trace in M_E visits a state in S infinitely often.[7]
   - Replace each transition of S by a set of must transitions, one for each symbol in $\Gamma(I)$.
2) Let $M_c^*(I)$ be the green/red context of I and let M_E be a Σ_E context extension of $\Gamma(I)$. An MTS

$$M_c^*(I, M_E) =_{def} (M_c^*(I) \parallel M_E)$$

   is called a **$\boldsymbol{\Sigma_E}$-extended context** of I.

This allows us to formulate the main Theorem of [33].

Theorem 2 (Correctness of $\boldsymbol{\Sigma_E}$-Extended Context):
Let $M \in \mathcal{M}$ be an MTS, let I be an MC of M (Def. 14), and let M_E be a Σ_E context extension of $\Gamma(I)$. Then we have:

1.) $hid_{\Sigma_E}(M_s(I) \parallel M_c^*(I, M_E)) \lessapprox M$
2.) $hid_{\Sigma_E}(M_s(I) \parallel M_c^*(I, M_E))$ is convergent.

As argued in [33], it is quite easy to generate benchmark systems with arbitrary degree of parallelism where no component can be abstracted away. The most important feature of a framework for property-preserving benchmark generation, however, is that it can produce systems of arbitrary state size. This can easily be achieved via an informed use of green and red context construction and their successive alphabet extension. We will sketch such a strategy in the next section in a fashion that its generalization to the generation of more sophisticated benchmark scenarios should become clear.

V. Decomposition Pattern by Example

In order to illustrate a simple yet effective strategy towards automatically generating benchmark scenarios whose state spaces are guaranteed to grow exponentially with their degree of parallelism, we consider a very simple example scenario starting with Milner's four-state cycler (Figure 1):
Our decomposition strategy proceeds now in two dimensions:

The first dimension is characterized by replication in a fashion reminiscent of Milner's round robin scheduler. This can be achieved by defining a Modal Contract I_i with $\Gamma(I_i) =_{def} \{c_i\}$. We then choose $\Sigma_{E_i} =_{def} \{b_{i+1}, c_{i+1}, d_{i+1}\}$

[7]This definition is similar to the notion of cut points in Floyd's inductive assertion method.

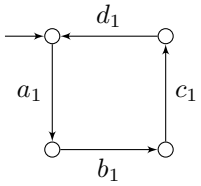


Fig. 1: Milner's four-state cycler

as the extension alphabet in order to generate the Σ_{E_i} context extension M_{E_i} illustrated in Figure 2. Note that in this specific scenario, both M_{E_i} and the resulting extended context $M_c^*(I_i, M_{E_i})$ are identical.

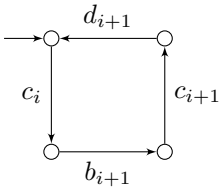


Fig. 2: Extended context and $(i+1)$'th component

This decomposition can be iterated to generate a potentially infinite chain. The elements of this chain are in a second dimension further decomposed into pairs of components based on a Modal Contract I_i' with communication alphabet $\Gamma(I_i') =_{def} \{b_i, d_i\}$ as depicted in Figure 3.[8]

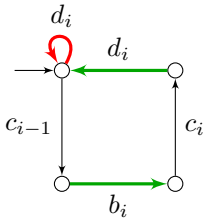


Fig. 3: Contract to generate a pair of components[9]

This contract I_i' yields the green/red context illustrated in Figure 4. Dashed arrows indicate may-only transitions. We do not extend this context with a new alphabet. Instead, we simply define a must transition for every existing may transition in both system and context in order to retrieve LTS implementations. The final pair of components resulting from a contract I_i' is shown within a single rectangular box of Figure 5. Figure 5 itself illustrates the resulting parallel system when generating a chain of n component pairs.
It is easy to see that the state space of this chain easily doubles whenever a further pair is added. Note that any temporal property that is preserved by weak refinement of divergent MTSs can be chosen for a corresponding benchmark scenario. An example can be found in Section VII-B.

We consider this construction as a kind of minimal skeleton for

[8]In case of $i = 2$, the transition labeled c_{i-1} would instead be labeled a_1.
[9]The d_i self loop is red and horizontal arrows labeled d_i and b_i are green.

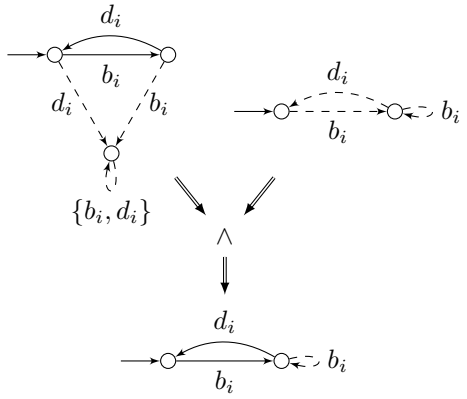


Fig. 4: Context of contract I_i'.

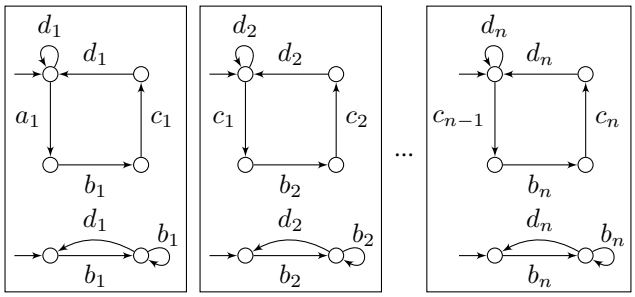


Fig. 5: Parallel components of exemplary benchmark scenario

benchmark generation. Of course for RERS, our components will be much bigger, and they will exploit the potential given by the many may transitions much more, but deep inside we will make sure that there are at least some dependencies of the kind described above in order to guarantee the exponential growth of the benchmarks with increasing degree of parallelism. Detecting these vital dependencies in the benchmarks is certainly one of the major challenges when solving the corresponding benchmark scenario.

VI. Generating Petri Nets

In this section we show how the generated benchmark processes can be transformed to Petri Nets. Naturally, we restrict ourselves to 1-safe Petri Nets. Subsequently, we validate if the verification problems are indeed hard for current model checking tools.

A. Transformation to Petri-Nets

A Petri Net is a tuple $(P, T, \rightarrow)$ of places P, transitions T, connected by arcs $\rightarrow \subseteq P \times T \cup T \times P$. Let $src(t) =_{def} \{p \in P \mid p \rightarrow t\}$ be the places preceding t, and similarly let $trg(t) =_{def} \{p \in P \mid t \rightarrow p\}$ be the places succeding t. A marking is a function $\mu : P \rightarrow \mathbb{N}$; $\mu(p)$ indicates the number of tokens in place p. Tokens can be moved around by firing transitions. A transition t is enabled in marking μ if all $p \in src(t)$ have $\mu(p) > 0$. The effect of firing t is to first decrease $M(p)$ for all places $p \in src(t)$ by one, and then to increase $M(p)$ for all places $p \in trg(t)$ by one. The LTS induced by a Petri Net and an initial marking is the set $\mathcal{R}$ of (reachable) markings, connected by transition firings. A Petri Net is 1-safe, if $\mu(p)$ never exceeds 1.

Given a generated benchmark process $M = M_1 \,||\, \cdots \,||\, M_n$, consisting of the parallel composition of LTSs $M_i = (S_i, s_i, \Sigma_i, \rightarrow_i)$, our goal is to transform it into a Petri Net, whose underlying LTS is equivalent to the LTS of P. For a single component M, the transition would be straightforward: Every state in the LTS corresponds to a place of the Petri Net, and every LTS transition $p \xrightarrow{a} q$ becomes a Petri Net transition t_a with arcs $p \rightarrow t_a$ and $t_a \rightarrow q$.

For a parallel composition we must take care of the synchronisation between actions. This is complicated by multiple occurrences of the same action within a single process: in such cases an a-step can potentially synchronize with any matching a-step in each component. So, given a label a, we will introduce a new Petri Net transition for each combination of LTS transitions from different components. Note that components in which a doesn't occur should not block the synchronisation.

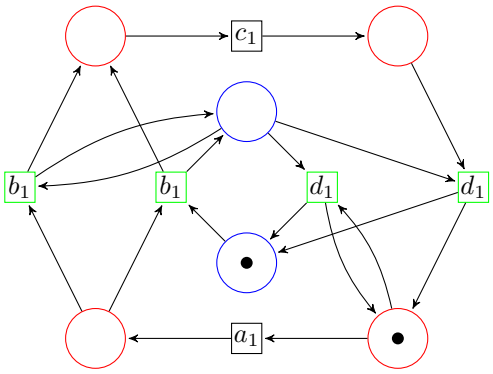
More formally, given action a, let index set $I(a) =_{def} \{i \mid a \in \Sigma_i\}$ describe the processes that synchronize on a. Let $E(a, i) =_{def} \{(p, a, q) \mid p \xrightarrow{a}_i q\}$ be the set of a-transitions in M_i. Then we define the Petri Net $(P, T, \rightarrow)$ as follows:

- The set of places $P =_{def} \bigcup_{i \in 1 \ldots n} S_i$ is the union of the states of all components.
- The set of transitions $T =_{def} \bigcup_{a \in \Sigma} \prod_{i \in I(a)} E(a, i)$ is the Cartesian product of all transitions (p, a, q) in those processes with a in their alphabet.
- For every (p, a, q) that participated in Petri Net transition t, we set the arcs $p \rightarrow t \rightarrow q$.
- The initial marking M has $M(s_i) = 1$ for the initial states s_i of M_i, and $M(s) = 0$, otherwise.

By construction, $(P, T, \rightarrow)$ is 1-safe, and the LTS of $(P, T, \rightarrow)$ is isomorphic to the LTS of M.

B. Running Example

We demonstrate the Petri Net generated from two processes in Section V, more specifically from the two components illustrated in the rectangle on the left-hand side of Figure 5. Red places (circles) originate from the respective process in the first row of Figure 5 and blue places originate from the respective process in the second row. The transitions a_1 and c_1 are internal to the first component, whereas the labels b_1 and d_1 are synchronized.



C. Implementation.

We implemented the translation in Python. It takes a description of M in dot-format, and generates a Petri Net in PNML[10] format. This format is also used in the MCC competition [21]. The PNML format allows extra annotations that can be used by model checkers. In particular, we expose the structure of our Petri Nets in NUPN format (Nested-Unit Petri Nets) [10] by declaring all places derived from a single component as a unit.

VII. Validation of Benchmark Hardness

In order to check how hard the generated benchmarks are, we analyse the feasibility of state space generation and model checking. As input, we take an increasing chain of n components, of the running example in Section V. We transformed the example to a Petri Net following Section VI. Note that the generated Petri Net could blow up exponentially in principle, but due to the regular communication structure this does not happen. The size of the generated Petri Nets grows in the number of red and blue components as follows:

- Number of red components = $|R|$
- Number of blue components = $|B|$
- Number of transitions = $|T| = 1 + 5 \cdot \frac{|R \cup B|}{2}$
- Number of places = $|P| = 2 \cdot |R| + 4 \cdot |B|$
- Number of states (reachable markings) = $|\mathcal{R}| = 4^{|R \cup B|}$

The (estimated) number of states indicates that explicit-state model checking will not get very far. We will now experiment with the symbolic model checker in LTSmin [20], which scored highest in the LTL category of the last MCC competion [21]. We performed two experiments: reachability analysis and model checking. All experiments were run on an AMD Opteron 4386 machine and 64GB memory. We used LTSmin version 3.0, with a timeout of 30 minutes.

A. Symbolic Reachability Analysis

In this experiment, we just compute the Decision Diagram consisting of all reachable states of the system. Symbolic Reachability can exploit many tricks. In particular, we used variable reordering and transition firing strategies. In LTSmin, we experimented with reordering strategies FORCE [1] and Sloan [27]. The Sloan strategy (based on Sloan's matrix bandwidth reduction algorithm) clearly wins. As transition firing we used a combination of chaining and saturation as reachability strategy. As underlying decision diagrams, we took the multicore Multiway Decision Diagrams from Sylvan [7].

Table I shows the size of the chain (number of red and blue components), the size of the state space, computed by the reachability tools, and the peak size and final size, indicating the maximal intermediate and final number of MDD nodes required to store the set of visited states symbolically.

[10]http://www.pnml.org/

B. Model Checking

Finally, we checked the following ACTL (action-based CTL) property: $\mathrm{AG\,AF}\, a_1$, which means that action a_1 occurs infinitely often along each path. Note that this formula is equivalent to the LTL property $\mathrm{A\,GF}\, a_1$.[11]

In LTSmin, we interpret this CTL formula in μ-calculus, using a direct translation of CTL operators to fixpoints. Then we use the straightforward algorithm [8] to evaluate μ-calculus formulas as sets of states, taking advantage of the fact that CTL formulas do not introduce any true μ/ν-alternations. This set-based algorithm can be directly converted to MDDs, as in [3]. It is possible to further optimize this algorithm specifically for CTL. The μ-calculus algorithm based on equation systems, provided by [6], is linear in the size of the CTL formula and the explicit LTS. For symbolic state spaces any model checking procedure for μ-calculus requires exponential time [29]. However, one could implement smart iteration heuristics, like saturation [35]. Our implementation evaluates fixed points in a strict breadth-first manner. As a consequence, the scalability of model checking CTL formulas is much lower than for reachability properties.

The last column in Table I indicates the number of MDD nodes required in model checking the ACTL property mentioned above. Note that the resources required by model checking grow much faster than the requirements for state space generation. We get a timeout when trying to model check with size 128, this is shown is the bottom right entry.

structure			**State space**		**CTL**
$\|R\|$	$\|B\|$	**states**	**peak**	**final**	**peak**
1	1	4	14	14	14
2	2	16	30	28	39
4	4	256	92	59	145
8	8	$6.55 \cdot 10^4$	301	117	1107
16	16	$4.29 \cdot 10^9$	430	224	10277
32	32	$1.84 \cdot 10^{19}$	4005	467	99813
64	64	$3.40 \cdot 10^{38}$	1816	896	875030
128	128	$1.15 \cdot 10^{77}$	3674	1792	TO

TABLE I: Number of states (left) and MDD nodes (right) with growing chain

VIII. Conclusion

In this paper, we have proposed a systematic approach to generate highly parallel benchmark systems, in particular in terms of complex Petri Nets, that are guaranteed to satisfy a set of given temporal properties. Key to our approach is the iterative property-preserving parallel decomposition in a light-weight assumption commitment style on the basis of Modal Contracts. We have illustrated how a specific strategy for design choices, which may well be automated, leads to benchmarks that grow exponentially with the number of its parallel components, and that are very difficult to verify. In particular, we have reported numbers from a sequence of systematic, concrete corresponding verification attempts using today's leading verification technology.

[11]Note that this is accidental: $\mathrm{AF\,AG}\, a_1$ is not equivalent to $\mathrm{A\,FG}\, a_1$.

Currently, generated context components are deterministic concerning the initial alphabet by construction, a property that may be exploited by potential verifiers. We are therefore planning to eliminate this determinisms via a notion of context-dependent semantic determinism, which we consider very hard to distinguish from full non-determinism. In this context, we will also investigate which influence dedicated properties have on the tools performance, e.g. when they come close to characteristic formulae [30], [31].

Benchmark problems generated with our method can be guaranteed to explode in size. It has to be seen whether our notion of hardness is stable in the context of advanced reduction/verification techniques such as the compositional minimization presented in [13], [14], (lazy) CEGAR [5], [16], and partial order reduction [34], [28], [12], [9]. The planned investigation of the corresponding interplay between the introduction and reduction of difficulty is envisioned to boost the progress of system verification.

Another line of future research is to include data and arithmetic in the modeling language, for example in the fashion proposed for sequential benchmarks in [32]. In particular when allowing shared memory between the components this imposes new challenges both for the generation process and, even more so, for the solution of resulting benchmark problems.

We plan to make our generation tool available open source in order to invite users to enhance the generation potential and to contribute to a library of characteristic benchmarks. Ideally, this will help to establish an accepted quality standard.

References

[1] Aloul, F.A., Markov, I.L., Sakallah, K.A.: FORCE: a fast and easy-to-implement variable-ordering heuristic. In: Proc. of the 13th ACM Great Lakes Symposium on VLSI. pp. 116–119 (2003)

[2] Benveniste, A., Caillaud, B.: Synchronous interfaces and assume/guarantee contracts. This Festschrift (2017)

[3] Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10ˆ20 states and beyond. Information and Computation 98(2), 142–170 (1992)

[4] Butler, M.: Incremental design of distributed systems with event-b. Engineering Methods and Tools for Software Safety and Security 22, 131 (2009)

[5] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: International Conference on Computer Aided Verification. pp. 154–169. Springer (2000)

[6] Cleaveland, R., Klein, M., Steffen, B.: Faster model checking for the modal mu-calculus. In: Computer Aided Verification, CAV'92. pp. 410–422. LNCS 663 (1992)

[7] van Dijk, T., van de Pol, J.: Sylvan: Multi-core decision diagrams. In: TACAS 2015, Proceedings. pp. 677–691. LNCS 9035 (2015)

[8] Emerson, E.A., Lei, C.: Efficient Model Checking in Fragments of the Propositional Mu-Calculus (Extended Abstract). In: LICS. pp. 267–278 (1986)

[9] Flanagan, C., Godefroid, P.: Dynamic partial-order reduction for model checking software. In: ACM Sigplan Notices. vol. 40(1), pp. 110–121. ACM (2005)

[10] Garavel, H.: Nested-Unit Petri Nets: A Structural Means to Increase Efficiency and Scalability of Verification on Elementary Nets. In: PETRI NETS. pp. 179–199 (2015)

[11] Geske, M., Jasper, M., Steffen, B., Howar, F., Schordan, M., van de Pol, J.: RERS 2016: Parallel and sequential benchmarks with focus on LTL verification. In: International Symposium on Leveraging Applications of Formal Methods. pp. 787–803. Springer (2016)

[12] Godefroid, P., Van Leeuwen, J., Hartmanis, J., Goos, G., Wolper, P.: Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem, vol. 1032. Springer Heidelberg (1996)

[13] Graf, S., Steffen, B.: Compositional minimization of finite state processes. In: Computer-Aided Verification. vol. 90, pp. 57–73 (1990)

[14] Graf, S., Steffen, B., Lüttgen, G.: Compositional minimisation of finite state systems using interface specifications. Formal Aspects of Computing 8(5), 607–616 (1996)

[15] Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems (TOPLAS) 16(3), 843–871 (1994)

[16] Henzinger, T.A., Jhala, R., Majumdar, R., Sutre, G.: Lazy abstraction. ACM SIGPLAN Notices 37(1), 58–70 (2002)

[17] Hoare, C.A.R.: Communicating sequential processes. In: The origin of concurrent programming, pp. 413–443. Springer (1978)

[18] Howar, F., Isberner, M., Merten, M., Steffen, B., Beyer, D.: The RERS grey-box challenge 2012: analysis of event-condition-action systems. In: International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. pp. 608–614. Springer (2012)

[19] Hüttel, H., Larsen, K.G.: The use of static constructs in a model process logic. In: International Symposium on Logical Foundations of Computer Science. pp. 163–180. Springer (1989)

[20] Kant, G., et al.: LTSmin: High-Performance Language-Independent Model Checking. In: TACAS. pp. 692–707. LNCS 9035 (2015)

[21] Kordon, F., et al.: Complete Results for the 2016 Edition of the Model Checking Contest. http://mcc.lip6.fr/2016/results.php (June 2016)

[22] Kordon, F., Linard, A., Buchs, D., Colange, M., Evangelista, S., Lampka, K., Lohmann, N., Paviot-Adet, E., Thierry-Mieg, Y., Wimmel, H.: Report on the model checking contest at petri nets 2011. In: Transactions on Petri Nets and Other Models of Concurrency VI, pp. 169–196. Springer (2012)

[23] Kourie, D.G., Watson, B.W.: The Correctness-by-Construction Approach to Programming. Springer Science & Business Media (2012)

[24] Larsen, K.G., Nyman, U., Wasowski, A.: Modal i/o automata for interface and product line theories. In: European Symposium on Programming. pp. 64–79. Springer (2007)

[25] Larsen, K.G.: Modal specifications. In: International Conference on Computer Aided Verification. pp. 232–246. Springer (1989)

[26] Margaria, T., Steffen, B.: Simplicity as a driver for agile innovation. Computer 43(6), 90–92 (2010)

[27] Meijer, J., van de Pol, J.: Bandwidth and Wavefront Reduction for Static Variable Ordering in Symbolic Reachability Analysis. In: NFM, Proceedings. pp. 255–271. LNCS 9690 (2016)

[28] Peled, D.: All from one, one for all: on model checking using representatives. In: International Conference on Computer Aided Verification. pp. 409–423. Springer (1993)

[29] Rabinovich, A.M.: Symbolic model checking for μ-calculus requires exponential time. Theoretical Computer Science 243(1-2), 467–475 (2000)

[30] Steffen, B.: Characteristic formulae. In: International Colloquium on Automata, Languages, and Programming. pp. 723–732. Springer (1989)

[31] Steffen, B., Ingólfsdóttir, A.: Characteristic formulas for processes with divergence. Information and Computation 110(1), 149–163 (1994)

[32] Steffen, B., Isberner, M., Naujokat, S., Margaria, T., Geske, M.: Property-driven benchmark generation: synthesizing programs of realistic structure. International Journal on Software Tools for Technology Transfer 16(5), 465–479 (2014)

[33] Steffen, B., Jasper, M.: Property-preserving parallel decomposition. In: LNCS (to appear). Springer (2017)

[34] Valmari, A.: Stubborn sets for reduced state space generation. In: International Conference on Application and Theory of Petri Nets. pp. 491–515. Springer (1989)

[35] Zhao, Y., Ciardo, G.: Symbolic CTL model checking of asynchronous systems using constrained saturation. In: ATVA'09. pp. 368–381. LNCS 5799 (2009)