Decompositional Minimization of Monolithic Processes
Abstract

Maurice Laveaux

August 7, 2019

1 Introduction

The mCRL2 language [7] is an expressive process algebra that can specify communication of data between processes executing concurrently. The corresponding mCRL2 toolset [1] translates many operators, such as parallel composition and action synchronization, to an equivalent (non-deterministic sequential) monolithic process. The advantages of this procedure are that the design of further analysis techniques and the implementation of state space exploration can be greatly simplified. Unfortunately, the available static analysis techniques are not powerful enough to mitigate the state space explosion of this monolithic process even though it could be reduced modulo some equivalence after construction.

In the literature there are several promising techniques to reduce the state space explosion problem and one of them is *compositional minimization*. In its simplest form, each sequential process (a component) is replaced by an *abstraction*, simpler than the corresponding process while still preserving the property that is checked [9, 10]. One challenge in this approach is that the size of the state spaces of individual components summed together might exceed the size of the whole state space [3]. In particular, the state space can become infinitely large for components that rely on synchronization to bound their behaviour. This can be avoided by specifying or generating *interface constraints* (also known as *environmental constraints* or *context constraints*) leading to a *refined compositional minimization* [5, 2]. Another challenge is that the order in which intermediate components are explored, minimized and subsequently composed heavily influences the size of the intermediate state spaces. However, this technique has been successfully [4] used by the CADP [3] (Construction and Analysis of Distributed Processes) tool set.

Unfortunately, the previous compositional techniques which rely on the user defined processes are not applicable in the setting where parallel composition was removed first. It might be possible to abandon the translation step that removes parallelism or to change it such that information about the user-defined processes is retained. However, we focus our attention on decomposing the resulting monolithic process into individual components based on its data parameters instead. These individual components can then be used for compositional minimization afterwards. Related work on this type of decomposition, referred to as (unique) *parallel decomposition*, was used in [6] to improve the efficiency of equivalence checking. However, this work considers processes that were already in a parallel composition and only further decomposes them based on the actions that occur in each component. In [8] it was also shown that mCRL2 processes can be split such that only one component contains a subset of (multi)-actions. Here, both components still contain the full behaviour of the original process and are as such not really useful for minimization.

The advantages of decomposing the monolithic process are the same as for compositional minimization; in that by minimizing intermediate components the composed state space might be smaller than its original state space. Furthermore, exploring the syntactic representation of a process relies on term rewriting to evaluate its conditions and expressions, and enumeration to explore summations. The complexity of these parts highly depend on the structure of the data specification and the term rewrite system. On the other hand, the composition of labelled transition systems has a complexity that depends on the number of transitions, the size of the transition labels and the number of synchronization rules. An additional advantage thus-far seems to be that by decoupling the decomposition from the user-defined processes the interface constraints can be obtained more easily, resulting in better bounds on the behaviour of the individual processes.

The research is still in its relatively early stages, but in the remainder of this abstract we identify the main research questions. Several initial results show promise in the feasibility of applying this decomposition technique using the mCRL2 process algebra by relying on some unique language features. However, the explored techniques are not yet strong enough to obtain useful decompositions for realistic process specifications.

2 Preliminaries

We assume the existence of an abstract data language over sorts which we denote by symbols D and E. In favour of conciseness we abstract away from defining the semantics explicitly. We assume the existence of booleans $\mathbb{B}$ consisting of true and false and the existence of an infinite set of sorted *variables*. Let Σ be a set of sorted action labels. First, we define the notion of a multi-action, which is a multi-set (or bag) of actions that occur simultaneously.

Definition 2.1. The set of *multi-actions* Act is inductively defined as:

$$\alpha, \beta \ ::= \ \tau \ \mid \ a(\vec{e}) \ \mid \ \alpha | \beta$$

Where $a \in \Sigma$ is an action label, $\vec{e}$ a tuple of values and τ the empty multi-action representing the *invisible* action.

Definition 2.2. A labelled transition system, abbreviated as LTS, is a four tuple $\mathcal{L} = (S, \mathit{init}, \mathit{Act}, \rightarrow)$ where S is a set of *states*; $\mathit{init} \in S$ is the *initial* state; Act is the set of multi-actions and $\rightarrow \ \subseteq S \times \mathit{Act} \times S$ is a labelled transition relation.

The mCRL2 language is an expressive process algebra that includes operators such as alternative (+), sequential (.), parallel (||) composition, summations (Σ) over sorted variables and many more. The monolithic process, which is a normal-form for the mCRL2 process algebra, is defined by a number of *condition-action-effect* statements, called *summands*, in an alternative composition where the values of its parameters determine the current state. Non-deterministically, for each statement where the condition evaluates to true the corresponding action can be performed and its parameters are updated by the effect.

Definition 2.3. A *linear process equation*, abbreviated LPE, is an mCRL2 process of the following form:

$$P(d : D) = \sum_{i \in I} \sum_{e_i : E_i} c_i(d, e_i) \rightarrow \alpha_i(d, e_i) \, . \, P(g_i(d, e_i))$$

Where $I \subseteq \mathbb{N}$ is a finite index set and $\sum_{i \in I}$ a meta level construct to define a number of *summands*. For $i \in I$ symbol E_i is a data sort over which variables e_i range. A boolean expression $c_i(d, e_i)$ represents the *enabling condition* and $\alpha_i(d, e_i)$ is a multi-action of action labels over expressions containing variables d and e_i. Finally, $g_i(d, e_i)$ is an *update* expression of sort D.

The operational semantics of a linear process equation are defined by an LTS.

Definition 2.4. Let $P(d : D)$ be a linear process equation and $u \in D$ a value. The corresponding LTS denoted by $[\![P(u)]\!]$ is $(D, u, \mathit{Act}, \rightarrow)$ where there is a transition $v \xrightarrow{\alpha_i(v,t)} g_i(v, t)$ for $v \in D$, $t \in E_i$ and $i \in I$ iff $c_i(v, t)$ evaluates to true.

Two processes are considered to be equivalent whenever they are *strongly bisimilar*.

Definition 2.5. Let $\mathcal{L} = (S, \mathit{init}, \mathit{Act}, \rightarrow)$ be a labelled transition system. A symmetric binary relation $R \subseteq S \times S$ is called a *strong bisimulation relation* if and only if for all states $s, t \in S$ such that $s \, R \, r$ and all actions $a \in \mathit{Act}$ that if there is a transition $s \xrightarrow{a} s'$, then there is a state $t' \in S$ such that transition $t \xrightarrow{a} t'$ exists and $s' \, R \, t'$.

Two states s and t are *strongly bisimilar*, denoted by $s \leftrightarrows t$, if and only if there is a strong bisimulation relation R such that $s \, R \, t$. Two LTSs $\mathcal{L}_1$ and $\mathcal{L}_2$ are *strongly bisimilar*, denoted by $\mathcal{L}_1 \leftrightarrows \mathcal{L}_2$, if and only if their initial states are strongly bisimilar in both LTS.

3 Decomposition

The parameter of an LPE can be generalised to tuples or a vector of parameters. We are interested in finding a decomposition procedure that constructs n component processes of which each contains a subset (or a projection) of the parameters from the original monolithic process such that under some context strong bisimilarity is preserved. This immediately leads us to the first research question on determining when a decomposition is valid.

Question 1. Let $P(\vec{d} : \vec{D})$ be an LPE. What are the necessary conditions such that n component processes ranging $P_0(\vec{d_0} : D_0), \ldots, P_n(\vec{d_n} : D_n)$ and some context C satisfy that for all $\vec{u} \in D$ the LTS $[\![C(P_0(\vec{u}_0) \parallel \ldots \parallel P_n(\vec{u}_n))]\!]$ is strongly bisimilar to $[\![P(\vec{u})]\!]$ for suitable projections of $\vec{u}$ onto $\vec{u}_0$ to $\vec{u}_n$.

At first, we restrict ourselves to determining a decomposition that results in exactly two components, which we refer to a *cleave*. For this setting we have already defined several cleaves where $\vec{d_i}$ and $\vec{u}_i$ are disjoint projections of the original parameters and corresponding sorts to validate the feasibility of this approach. We illustrate an example for one of the proposed cleaves, which takes dependencies between the components into account. Note that we only determined a correct decomposition, but not yet a procedure to determine such a decomposition and many decisions need to be made by the user.

Example 3.1. Consider a stopwatch process that can count down for n steps emitting COUNT actions and a machine that has an action TOGGLE to switch between an 'on' and 'off' state indicated by parameter s. The machine has to wait for ten COUNT actions before it can switch state again. The following monolithic process corresponds to this description:

$$\begin{aligned} P(n : \mathbb{N}, s : \mathbb{B}) = {} & (n > 0) \rightarrow \textsc{count} \,.\, P(n-1, s) \\ & + (n = 0) \rightarrow \textsc{toggle} \,.\, P(10, \neg s) \end{aligned}$$

The cleave of this process on parameters n and s results in two processes $P_n(n : \mathbb{N})$ and $P_s(s : \mathbb{B})$ with the same summands without the updates on removed parameters. The second summand of process P_s needs the value of n to decide whether it can perform the TOGGLE action. We introduce two sync_1^n and sync_1^s actions that synchronize the value of n. Here, the superscript n and s indicate the component and the subscript the index of the summand. Now, components P_n and P_s become:

$$\begin{aligned} P_n(n : \mathbb{N}) &= (n > 0) \rightarrow \textsc{count} \,.\, P_n(n-1) + (n = 0) \rightarrow \mathsf{sync}_1^n(n) \,.\, P_n(10) \\ P_s(s : \mathbb{B}) &= \sum_{n \in \mathbb{N}} (n = 0) \rightarrow \textsc{toggle}|\mathsf{sync}_1^s(n) \,.\, P_s(\neg s) \end{aligned}$$

Note that the TOGGLE action is only present in P_s. Here, we see that the constraint $(n = 0)$ is now present in both components to bound the possible values of n. Furthermore, by identifying that the first summand can be performed independently in P_n it can be removed from P_s. The context C is now defined such that whenever the parallel composition of P_s and P_n performs $\textsc{toggle}|\mathsf{sync}_1^n(0)|\mathsf{sync}_1^s(0)$ atomically the synchronization actions are renamed to τ and from the axiom $\alpha|\tau = \alpha$ follows that only TOGGLE is visible. The other interleavings are disallowed by the context and the resulting composition is strongly bisimilar to P.

Based on the assumption that computing the transition relation is expensive the decomposition should be performed in such a way that the size (the number of transitions) of P_V and P_W is minimized. In the previous example the state space of $P_n(10)$ has ten COUNT transitions and one $\mathsf{sync}_1^n(0)$ transition from zero to ten. The state space of P_s has two states true and false with two transitions labelled $\textsc{toggle}|\mathsf{sync}_1^s(0)$ between them. The state space of P has 20 COUNT actions and two TOGGLE actions in a cycle and so in this particular case the decomposition is an improvement over exploring P directly. In particular, the state space of P_s could even be reduced to a single state with one self-loop as the states true and false are strongly bisimilar.

Unfortunately, for the cleave definitions that we have explored so-far there are examples where the optimal split cannot be achieved. For an optimal cleave it should hold that the sum of the size of P_V, denoted by $|P_V|$, and $|P_W|$ are minimal. In particular, the sizes of P_V and P_W should not exceed the size of the original state space in order to be an improvement over exploring the monolithic process itself as shown in the previous example. The next research question is necessary in order to show that a given cleave definition is both correct and can achieve the best possible decomposition.

Question 2. Can we find a cleave definition for which its restrictions on P_V and P_W are weak enough such that the optimal parallel decomposition fits this definition?

Here, it is also interesting to consider whether a cleave (only two components) is enough to preserve the optimal split. Defining this last research question exact is already difficult as the transformation to components, which should be LPEs, must be on a syntactic level to avoid certain unwanted, but optimal, examples where the individual components are first explored and then translated back to LPEs.

As the monolithic process is the result of applying various transformations to the user-provided specification it is particularly important that deriving the components can be automated as much as possible. Therefore, we are also interesting in defining an algorithm to decide, within a given cleave definition, the structure of P_V and P_W for a concrete instance where the initial state is known. Achieving the optimal result here might not be tractable (or in general even decidable) as the monolithic processes can be large. Preferably, the heuristics should either find a suitable split that is an improvement over exploring the monolithic process directly or it should not result in a decomposition at all.

Question 3. Can we define an exact algorithm or an algorithm based on heuristics to automatically construct the components for a given cleave definition?

The last question actually makes it also interesting to study cleave definitions that are too restrictive to handle the optimal split as for these definitions it might be easier to automate the construction of the components. It is also interesting to consider techniques that are not solely based on static analysis, but actually partially explore a component to determine (or improve heuristics to find) the structure of the individual components. Finally, these methods should also be applied to practice models and benchmarks to actually evaluate the proposed methods on realistic models.

References

[1] O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, W. Wesselink, A. Wijs, and T. A. C. Willemse. The mcrl2 toolset for analysing concurrent systems - improvements in expressivity and usability. In T. Vojnar and L. Zhang, editors, *TACAS*, volume 11428 of *LNCS*, pages 21–39. Springer, 2019.

[2] S. Cheung and J. Kramer. Context constraints for compositional reachability analysis. *ACM Trans. Softw. Eng. Methodol.*, 5(4):334–377, 1996.

[3] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *STTT*, 15(2):89–107, 2013.

[4] H. Garavel, F. Lang, and L. Mounier. Compositional verification in action. In F. Howar and J. Barnat, editors, *FMICS 2018*, volume 11119 of *LNCS*, pages 189–210. Springer, 2018.

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

[6] J. F. Groote and F. Moller. Verification of parallel systems via decomposition. In R. Cleaveland, editor, *CONCUR '92*, volume 630 of *LNCS*, pages 62–76. Springer, 1992.

[7] J. F. Groote and M. R. Mousavi. *Modeling and Analysis of Communicating Systems.* MIT Press, 2014.

[8] S. T. Q. Jongmans, D. Clarke, and J. Proença. A procedure for splitting data-aware processes and its application to coordination. *Sci. Comput. Program.*, 115-116:47–78, 2016.

[9] K. Tai and P. V. Koppol. Hierarchy-based incremental analysis of communication protocols. In *ICNP 1993*, pages 318–325. IEEE Computer Society, 1993.

[10] K. Tai and P. V. Koppol. An incremental approach to reachability analysis of distributed programs. In J. C. Wileden, editor, *IWSSD 1993*, pages 141–151. IEEE Computer Society, 1993.