Decompositional Minimisation of Monolithic Processes

Decompositional Minimisation of Monolithic Processes

Maurice Laveaux and Tim A.C. Willemse

Eindhoven University of Technology, Eindhoven, The Netherlands
{m.laveaux, t.a.c.willemse}@tue.nl

Abstract. Compositional minimisation can be an effective technique to reduce the state space explosion problem. This technique considers a parallel composition of several processes. In its simplest form, each sequential process is replaced by an *abstraction*, simpler than the corresponding process while still preserving the property that is checked. However, this technique cannot be applied in a setting where parallel composition is first translated to a non-deterministic sequential monolithic process. The advantage of this monolithic process is that it facilitates static analysis of global behaviour. Therefore, we present a technique that considers a monolithic process with data and decomposes it into two processes where each process defines behaviour for a subset of the parameters of the monolithic process. We prove that these processes preserve the properties of the monolithic process under a suitable synchronisation context. Moreover, we prove that state invariants can be used to improve its effectiveness. Finally, we apply the decomposition technique to several specifications.

1 Introduction

The mCRL2 language [10] is a process algebra that can be used to specify the behaviour of communicating processes with data. The corresponding mCRL2 toolset [4] translates the parallel composition and action synchronisation present in the process specification to an equivalent (non-deterministic sequential) monolithic process. The advantages of this translation are that the design of further static analysis techniques and the implementation of state space exploration can be greatly simplified. However, the static analysis techniques available at the moment are not always strong enough to mitigate the state space explosion problem for this monolithic process even though its state space can often be minimised modulo some equivalence relation after state space exploration.

In the literature there are several promising techniques to reduce this problem and one of them is *compositional minimisation*. The general idea is that the state space explosion often occurs due to all the possible interleaving of several processes in a parallel composition. To reduce the interleaving, in compositional minimisation the state space of each sequential process, referred to as a component, is replaced by an *abstraction*, simpler than the corresponding state space such that their composition preserves the property that is checked [18,17].

This naive approach is not always useful, because the size of the state spaces belonging to individual components summed together might exceed the size of the whole state space [7]. In particular, the state space can become infinitely large for components that rely on synchronisation 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 *semantic compositional minimisation* [8,5]. Furthermore, the order in which intermediate components are explored, minimised and subsequently composed heavily influences the size of the intermediate state spaces. There are heuristics for these problems that can be very effective in practice, as shown by the CADP [6] (Construction and Analysis of Distributed Processes) toolset.

Unfortunately, in our context where parallel composition is removed by a translation step the aforementioned compositional techniques, which rely on the user-defined parallel composition and the (sequential) processes, are not applicable. In this paper we define a decomposition technique of a monolithic process based on a partitioning of its data parameters that results in two components, which we refer to as a *cleave*, and show that it is correct. Furthermore, we show that *state invariants* can be used to retain more global information in both components to improve its effectiveness. Finally, we perform a case study to evaluate the decomposition technique and the advantage of state invariants in practice.

The advantages of decomposing the monolithic process are the same as for compositional minimisation; in that by minimising the state spaces of intermediate components the composed state space can be immediately smaller than the state space obtained by exploring the monolithic process. Indeed, the case studies on which we report support both observations. Furthermore, state space exploration relies on the evaluation of data expressions of the higher-level specification language and that can be costly, whereas, composition can be computed without evaluating expressions. An advantage of the decomposition technique over compositional minimisation is that constraints resulting from the synchronisation of these processes can be used when deriving the components. Another advantage is that the components resulting from the decomposition are not restricted to the user-defined processes present in the specification, which could yield a more optimal composition.

Related Work. Several different techniques are related to this type of decomposition. Most notably, the work on decomposing of petri nets into a set of automata [2] also aims to speed up state space exploration by means of decomposition. The work on functional decomposition [3] describes a technique to decompose a specification based on a partitioning of the action labels instead of a partitioning of the data parameters. In [12] it was shown how this type of decomposition can be achieved for mCRL2 processes. Furthermore, a decomposition technique was used in [9] to improve the efficiency of equivalence checking. However, that work considers processes that were already in a parallel composition and further decomposes them based on the actions that occur in each component.

Outline. In Section 2 the syntax and semantics of the considered process algebra are defined. The decomposition problem is defined in Section 3 and the cleave technique is presented. In Section 4 the cleave technique is improved with state invariants. In Section 5 a case study is presented to illustrate the effectiveness of the decomposition technique in practice. Finally, a conclusion and future work is presented in Section 6.

2 Preliminaries

We assume the existence of an abstract data theory that describes data sorts. Each sort D has an associated non-empty semantic domain denoted by $\mathbb{D}$. The existence of sorts B and N with their associated Boolean ($\mathbb{B}$) and natural number ($\mathbb{N}$) semantic domains respectively, with standard operators is assumed. Furthermore, we assume the existence of an infinite set of *sorted variables*. We use $e : D$ to indicate that e is an expression (or variable) of sort D. We use $\mathsf{FV}(e)$ to denote the set of free variables of an expression e. A variable that is not free is called *bound*. An expression e is *closed* iff $\mathsf{FV}(e) = \emptyset$.

We use an *interpretation* function, denoted by $[\![\ldots]\!]$, which maps syntactic objects to values within their corresponding semantic domain. We assume that for any closed expression e of a sort described by the data theory that $[\![e]\!]$ is already defined. We typically use boldface for semantic objects to differentiate them from syntax, *e.g.*, the semantic object associated with the expression $1 + 1$ is $\mathbf{2}$. For most operators we use the same symbol in both syntactic and semantic domains. However, we denote *data equivalence* by $e \approx f$, which is true iff $[\![e]\!] = [\![f]\!]$.

We use the following notation for vectors. Given a *vector* $\vec{d} = \langle d_0, \ldots, d_n \rangle$ of length $n+1$. Two vectors are equivalent, denoted by $\langle d_0, \ldots, d_n \rangle \approx \langle e_0, \ldots, e_n \rangle$, iff their elements are *pairwise equivalent*, *i.e.*, $d_i \approx e_i$ for all $0 \leq i \leq n$. Given a vector $\langle d_0, \ldots, d_n \rangle$ and a subset $I \subseteq \mathbb{N}$, we define the *projection*, denoted by $\langle d_0, \ldots, d_n \rangle_{|I}$, as the vector $\langle d_{i_0}, \ldots, d_{i_l} \rangle$ for the largest $l \in \mathbb{N}$ such that $i_0 < i_1 < \ldots < i_l \leq n$ and $i_k \in I$ for $0 \leq k \leq l$. We write $\vec{d} : \vec{D}$ for a vector of $n+1$ variables $d_0 : D_0, \ldots, d_n : D_n$ and denote the projection for a subset of indices $I \subseteq \mathbb{N}$ by $\vec{d}_{|I} : \vec{D}_{|I}$. Finally, we define $\mathsf{Vars}(\vec{d}) = \{d_0, \ldots, d_n\}$.

Given a set A we consider any total function $A \to \mathbb{N}$ to denote a *multi-set*. Let $m, m' : A \to \mathbb{N}$. *Inclusion*, denoted by $m \subseteq m'$, is defined as $m \subseteq m'$ iff for all elements $a \in A$ it holds that $m(a) \leq m'(a)$. Furthermore, we define the binary operator $m + m'$ as the pointwise addition: $\forall a \in A : (m + m')(a) = m(a) + m'(a)$. Similarly, we define the dual operator $m - m'$ such that $\forall a \in A : (m - m')(a) = \max(m(a) - m'(a), 0)$. Given an element $a \in A$ we refer to $m(a)$ as the *multiplicity* of a. We use the notation $\wr \ldots \wr$ for a multi-set where the multiplicity of each element is either written next to it or omitted when it is one, *e.g.*, $\wr a : 2, b \wr$ has elements a and b with multiplicity two and one respectively (and all other elements have multiplicity zero).

2.1 Labelled Transition Systems

Let Λ be the set of (sorted) action labels. We use D_a to indicate the sort of action label $a \in \Lambda$.

Definition 1. Multi-actions *are defined as follows:*

$$\alpha \ ::= \ \tau \ \mid \ a(e) \ \mid \ \alpha|\alpha$$

Where τ denotes the invisible *action and $a \in \Lambda$ is an action label with an expression e of sort D_a. Finally, $\alpha|\alpha$ denotes the simultaneous occurrence of two (multi-)actions.*

The set of all multi-sets over $\{a(\mathbf{e}) \mid a \in \Lambda, \mathbf{e} \in \mathbb{D}_a\}$ is denoted Ω. The semantics of a multi-action α, denoted by $[\![\alpha]\!]$, is an element of Ω and defined inductively as follows: $[\![\tau]\!] = \wr\wr$, $[\![a(e)]\!] = \wr a([\![e]\!])\wr$ and $[\![\alpha|\beta]\!] = [\![\alpha]\!] + [\![\beta]\!]$.

We assume the existence of a singleton sort $\bot$ and omit the parenthesis and expression of action labels of sort $\bot$ to provide a uniform treatment of actions with data and actions without relevant data parameters.

Definition 2. *A labelled transition system with multi-actions, abbreviated LTS, is a tuple $\mathcal{L} = (S, s_0, Act, \rightarrow)$ where S is a set of states; $s_0 \in S$ is an initial state; $Act \subseteq \Omega$ and $\rightarrow \subseteq S \times Act \times S$ is a labelled transition relation.*

We typically use ω to denote an element of Ω and we write $s \xrightarrow{\omega} t$ whenever $(s, \omega, t) \in \rightarrow$. We depict LTSs as edge-labelled directed graphs, where vertices represent states and the labelled edges between vertices represent the transitions. An incoming arrow with no starting state and no multi-action indicates the initial state. An example of an LTS that models the behaviour of a machine is depicted in Figure 1.

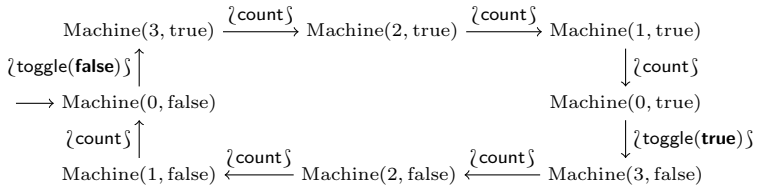


Fig. 1. Example LTS for the behaviour of a machine.

We recall the well-known strong bisimulation equivalence relation on LTSs [13].

Definition 3. *Let $\mathcal{L}_i = (S_i, s_i, Act_i, \rightarrow_i)$ for $i \in \{1, 2\}$ be two LTSs. A binary relation $R \subseteq S_1 \times S_2$ is a* strong bisimulation relation *iff for all $s\,R\,t$:*

- *if $s \xrightarrow{\omega}_1 s'$ then there is a state $t' \in S_2$ such that $t \xrightarrow{\omega}_2 t'$ and $s'\,R\,t'$, and*

– *if* $t \xrightarrow{\omega}_2 t'$ *then there is a state* $s' \in S_1$ *such that* $s \xrightarrow{\omega}_1 s'$ *and* $s' \mathrel{R} t'$.

Two states s *and* t *are* strongly bisimilar*, denoted by* $s \leftrightarroweq t$*, iff there is a strong bisimulation relation* R *such that* $s \mathrel{R} t$*. LTSs* $\mathcal{L}_1$ *and* $\mathcal{L}_2$ *are strongly bisimilar, denoted by* $\mathcal{L}_1 \leftrightarroweq \mathcal{L}_2$*, iff* $s_1 \leftrightarroweq s_2$.

2.2 Linear Process Equations

The (non-deterministic sequential) monolithic processes that we consider are defined by a number of *condition-action-effect* statements; which are called *summands*. Each summand symbolically represents a partial transition relation between the current and the next state for a multi-set of action labels. Let PN be a set of process *names*.

Definition 4. *A* linear process equation *(LPE) is an equation of the form:*

$$P(d : D) = \sum_{e_0 : E_0} c_0 \to \alpha_0 \,.\, P(g_0) + \;\; \ldots \;\; + \sum_{e_n : E_n} c_n \to \alpha_n \,.\, P(g_n)$$

Where $P \in PN$ *is the process* name *and* d *is the process parameter. Furthermore, for* $0 \leq i \leq n$ *it holds that:*

– E_i *is a sort over which* sum *variable* e_i *(where* $e_i \neq d$*) ranges,*
– c_i *is a boolean expression such that* $\mathsf{FV}(c_i) \subseteq \{d, e_i\}$ *defining the* enabling condition*, and*
– α_i *is a multi-action* τ *or* $a_i^1(f_i^1) | \ldots | a_i^{n_i}(f_i^{n_i})$ *such that* $a_i^k \in \Lambda$ *and* f_i^k *is an expression of sort* $D_{a_i^k}$ *such that* $\mathsf{FV}(f_i^k) \subseteq \{d, e_i\}$*, for* $1 \leq k \leq n_i$*, and*
– g_i *is an* update *expression such that* $\mathsf{FV}(g_i) \subseteq \{d, e_i\}$ *of sort* D.

The $+$ operator represents a non-deterministic choice among its operands, and the sum operator, denoted by $\sum$, expresses a non-deterministic choice for values of the sum variable. Note that the sum operator acts as a binder for the variable e_i. The sum operator is omitted whenever variable e_i does not occur freely within the condition, action and update expressions. We use $\mathop{+}_{i \in I}$ for a finite set of *indices* $I \subseteq \mathbb{N}$ as a shorthand for a number of summands.

We often consider LPEs where the parameter sort D represents a *vector* of a length $n + 1$; in that case we write $d_0 : D_0, \ldots, d_n : D_n$ to indicate that there are $n + 1$ parameters such that each d_i has sort D_i for $0 \leq i \leq n$. Similarly, we also generalise the action sorts and the sum operator in LPEs, where we permit ourselves to write $a(e_0, \ldots, e_k)$ and $\sum_{e_0 : E_0, \ldots, e_l : E_l}$, respectively.

The *operational semantics* of an LPE are defined by a mapping to an LTS. Given a substitution $\sigma = [x_0 \leftarrow \iota_0, \ldots, x_n \leftarrow \iota_n]$ and an expression e we use $\sigma(e)$ to denote the expression e where each occurrence of variable x_i is syntactically replaced by the *closed* expression ι_i, for $0 \leq i \leq n$. We assume the usual principle of substitutivity where for all variables x, expressions f and closed expressions g and h it holds that if $g \approx h$ then $[x \leftarrow g](f) \approx [x \leftarrow h](f)$. Let P be the set of symbols $P(\iota)$ such that $P(d : D) = \phi_P$, for any $P \in PN$, is an LPE and ι is a closed expression of sort D.

Definition 5. *Let* $P(d : D) = \sum_{i \in I} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P(g_i)$ *be an LPE and let* $\iota : D$ *be a closed expression. The semantics of* $P(\iota)$*, denoted by* $[\![P(\iota)]\!]$*, is an LTS* $(\mathsf{P}, P(\iota), \Omega, \to)$ *for which the following holds. For all values* $j \in I$*, closed expressions* $l : E_j$ *and closed expressions* $\iota' : D$ *such that* $\sigma = [d \leftarrow \iota', e_j \leftarrow l]$ *there is a transition* $P(\iota') \xrightarrow{[\![\sigma(\alpha_j)]\!]} P(\sigma(g_j))$ *iff* $[\![\sigma(c_j)]\!] = \mathbf{true}$*.*

We refer to the reachable part of the LTS that is the interpretation of an LPE with a given closed expression as the *state space* of that LPE. In the interpretation of an LPE a syntactic substitution is applied on the update expressions to define the reached state. This means that different closed syntactic expressions which correspond to the same semantic object, *e.g.*, $1 + 1$ and 2 for our assumed sort N, result in different states. However, such states are always strongly bisimilar as formalised below.

Lemma 1. *Given an LPE* $P(d : D) = \phi_P$ *and a closed expression* $\iota : D$*. For all closed expressions* $e, e' : D$ *such that* $[\![e \approx e']\!] = \mathbf{true}$ *we have* $[\![P(e)]\!] \leftrightarroweq [\![P(e')]\!]$*.*

Proof. We can show that the smallest relation R such that $P(e)\, R\, P(e')$ for all closed expressions $e, e' : D$ such that $e \approx e'$ is a strong bisimulation relation. This follows essentially from the principle of substitutivity and the definition of an LPE.

For any given state space we can therefore consider a *representative* state space where for each state a unique closed expression is chosen that is data equivalent. In an implementation it is very natural to generate this representative LTS directly, for example by always reducing expressions to the normal form if the data specification is based on a (terminating and confluent) term rewrite system. In examples we always consider the representative state space.

Example 1. Consider the LPE which models a machine that can be toggled with a delay of three counts. Whenever the counter reaches zero, *i.e.*, n is zero, it can be toggled again after which it counts down three times.

$$\begin{aligned} \mathrm{Machine}(n : N, s : B) = {} & (n > 0) \to \mathsf{count} \,.\, \mathrm{Machine}(n - 1, s) \\ & + (n \approx 0) \to \mathsf{toggle}(s) \,.\, \mathrm{Machine}(3, \neg s) \end{aligned}$$

Note that the sum operator has been omitted, because only parameter n occurs as a free variable in the expressions. A representative state space of the machine that is initially off, defined by $[\![\mathrm{Machine}(0, \mathsf{false})]\!]$, is shown in Figure 1. Initially, n is zero and therefore $[\![0 \approx 0]\!]$ is **true** and there is a transition labelled $[\![\mathsf{toggle}(\mathsf{true})]\!] = \wr \mathsf{toggle}(\mathbf{true}) \int$ to the state $\mathrm{Machine}(3, \mathsf{true})$. The other summand does not result in transitions for state $\mathrm{Machine}(0, \mathsf{false})$, because $[\![0 > 0]\!]$ is **false**. Similarly, there is an outgoing transition labelled $[\![\mathsf{count}]\!] = \wr \mathsf{count} \int$ for $\mathrm{Machine}(3, \mathsf{true})$, because $[\![3 > 0]\!]$ is **true**. Again, there is no other outgoing transition for state $\mathrm{Machine}(3, \mathsf{true})$ as $[\![3 \approx 0]\!]$ is **false**. The other transitions are derived in a similar way.

2.3 A Process Algebra of Communicating Linear Process Equations

We define a simple process algebra to express parallelism and interaction of LPEs. Let Comm be the set of *communication* expressions $a_0|\ldots|a_n \to c$ where $a_i, c \in \Lambda$ for $0 \leq i \leq n$ are action labels.

Definition 6. *The process algebra is defined as follows:*

$$S ::= \Gamma_C(S) \mid \nabla_A(S) \mid \tau_H(S) \mid S \parallel S \mid P(\iota)$$

Where $A \subseteq 2^{\Lambda \to \mathbb{N}}$ is a non-empty finite set of finite multi-sets of action labels, $H \subseteq \Lambda$ is a non-empty finite set of action labels and $C \subseteq \mathsf{Comm}$ is a finite set of communications. *Finally, we have $P(\iota) \in \mathsf{P}$.*

The set S contains all expressions of the process algebra. The operators describe *communication* (Γ_C), *action allowing* (∇_A), *action hiding* (τ_H) and *parallel composition* ($\parallel$). Finally, the elementary objects are the processes, defined as LPEs.

First, we introduce several auxiliary functions on Ω that are used to define the semantics of expressions in S.

Definition 7. *Given $\omega \in \Omega$ we define γ_C, where $C \subseteq \mathsf{Comm}$, as follows:*

$$\gamma_\emptyset(\omega) = \omega$$
$$\gamma_C(\omega) = \gamma_{C \setminus C_1}(\gamma_{C_1}(\omega)) \text{ for } C_1 \subset C$$
$$\gamma_{\{a_0|\ldots|a_n \to c\}}(\omega) = \begin{cases} \wr c(\mathbf{d}) \int + \gamma_{\{a_0|\ldots|a_n \to c\}}(\omega - \wr a_0(\mathbf{d}), \ldots, a_n(\mathbf{d}) \int) \\ \quad \text{if } \wr a_0(\mathbf{d}), \ldots, a_n(\mathbf{d}) \int \subseteq \omega \\ \omega \quad \text{otherwise} \end{cases}$$

For this function to be well-defined we require that the labels in the left-hand sides of the communications should not *overlap*. Furthermore, the action label on the right-hand side must not occur in any *other* left-hand side. For example $\gamma_{\{a|b \to c\}}(a|d|b) = c|d$ according to the definition, but $\gamma_{\{a|b \to c, a|d \to c\}}(a|d|b)$ and $\gamma_{\{a|b \to c, c \to d\}}(a|d|b)$ are not allowed.

Definition 8. *Given $\omega \in \Omega$ we define $\theta_H(\omega)$, where $H \subseteq \Lambda$ is a set of action labels, such that $\theta_H(\omega) = \omega'$ where:*

$$\omega'(a(\mathbf{d})) = \begin{cases} 0 & \text{if } a \in H \\ \omega(a(\mathbf{d})) & \text{otherwise} \end{cases}$$

Given a multi-action α we define $\underline{\alpha}$ to obtain the multi-set of action labels, *e.g.*, $\underline{a(3)|b(5)} = \wr a, b \int$. Formally, $\underline{a(e)} = \wr a \int$, $\underline{\tau} = \wr \int$ and $\underline{\alpha|\beta} = \underline{\alpha} + \underline{\beta}$. We define $\underline{\omega}$ for $\omega \in \Omega$ in a similar way.

Using these functions we can define the semantics, using *structured operational semantics*, as an LTS.

Definition 9. *The operational semantics of an expression Q of S, denoted $[\![Q]\!]$, are defined by the corresponding LTS $(\mathsf{S}, Q, \Omega, \rightarrow)$ with its transition relation defined by the rules below and the transition relation given in Definition 5 for each expression in P. For any $\omega \in \Omega$ and P, P', Q, Q' expressions of S:*

$$\textsc{Com}\ \frac{P \xrightarrow{\omega} P' \qquad C \subseteq \mathsf{Comm}}{\Gamma_C(P) \xrightarrow{\gamma_C(\omega)} \Gamma_C(P')} \qquad \textsc{Allow}\ \frac{P \xrightarrow{\omega} P' \qquad A \subseteq 2^{\Lambda \to \mathbb{N}} \qquad \underline{\omega} \in A}{\nabla_A(P) \xrightarrow{\omega} \nabla_A(P')}$$

$$\textsc{Hide}\ \frac{P \xrightarrow{\omega} P' \qquad H \subseteq \Lambda}{\tau_H(P) \xrightarrow{\theta_H(\omega)} \tau_H(P')} \qquad \textsc{Par}\ \frac{P \xrightarrow{\omega} P' \quad Q \xrightarrow{\omega'} Q'}{P \parallel Q \xrightarrow{\omega + \omega'} P' \parallel Q'}$$

$$\textsc{ParR}\ \frac{Q \xrightarrow{\omega} Q'}{P \parallel Q \xrightarrow{\omega} P \parallel Q'} \qquad \textsc{ParL}\ \frac{P \xrightarrow{\omega} P'}{P \parallel Q \xrightarrow{\omega} P' \parallel Q}$$

3 Decomposition

We are interested in decomposing an LPE into n LPEs, where the latter are referred to as *components*, such that each component contains a subset of the original parameters. This decomposition is considered *valid* iff the original state space is strongly bisimilar to the state space of these components under a suitable context.

Definition 10. *Let $P(\vec{d} : \vec{D}) = \phi$ be an LPE and $\vec{\iota} : \vec{D}$ a closed expression. The LPEs $P_0(\vec{d}_{|I_0} : \vec{D}_{|I_0}) = \phi_0$ to $P_n(\vec{d}_{|I_n} : \vec{D}_{|I_n}) = \phi_n$, for indices $I_0, \ldots, I_n \subseteq \mathbb{N}$, are a valid* decomposition *of P and $\vec{\iota}$ under a context C iff:*

$$[\![P(\vec{\iota})]\!] \leftrightarrows [\![\mathsf{C}[P_0(\vec{\iota}_{|I_0}) \parallel \ldots \parallel P_n(\vec{\iota}_{|I_n})]]\!]$$

Where $\mathsf{C}[P_0(\vec{\iota}_{|I_0}) \parallel \ldots \parallel P_n(\vec{\iota}_{|I_n})]$ is an expression in S. We refer to the expression $\mathsf{C}[P_0(\vec{\iota}_{|I_0}) \parallel \ldots \parallel P_n(\vec{\iota}_{|I_n})]$ as the composition.

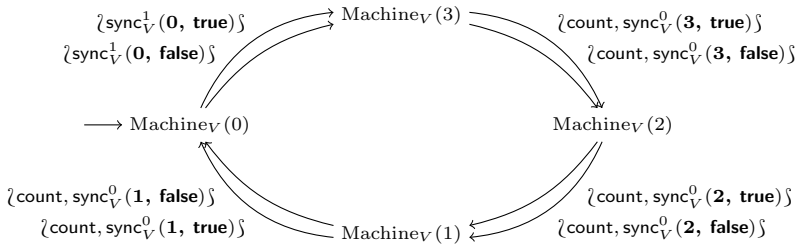
In our case the context C is an expression with operators from S that is used to define the synchronisation between the individual components, see for example the composition expression in Definition 12. The primary benefit of a valid decomposition is that a state space that is equivalent to the original state space can be obtained as follows. First, the state space of each component is derived separately. Then the result of the composition expression can be derived from the component state spaces based on the rules of the operational semantics. Furthermore, the component state spaces can be minimised modulo an equivalence relation that is a congruence with respect to the operators of S before deriving the results of the composition expression. Strong bisimilarity is known to be a congruence with respect to our operators. This is referred to as *compositional minimisation*.

For the remainder of this paper we consider a decomposition technique that results in exactly two components. First, we present an example to show that a valid decomposition can be achieved using the presented process algebra and which shows that not every decomposition is necessarily useful for state space exploration. This decomposition exploits the fact that τ represents the empty multi-action, which means that $\alpha|\tau = \alpha$ holds for all multi-actions α.

Example 2. Consider the LPE of Example 1 again. For the decomposition we introduce the two components shown below.

$$\begin{aligned}
\mathrm{Machine}_V(n:N) &= \sum_{s:B}(n>0) \to \mathsf{count}|\mathsf{sync}^0_V(n,s)\,.\,\mathrm{Machine}_V(n-1)\\
&\quad + \sum_{s:B}(n \approx 0) \to \tau|\mathsf{sync}^1_V(n,s)\,.\,\mathrm{Machine}_V(3)\\
\mathrm{Machine}_W(s:B) &= \sum_{n:N}(n>0) \to \tau|\mathsf{sync}^0_W(n,s)\,.\,\mathrm{Machine}_W(s)\\
&\quad + \sum_{n:N}(n \approx 0) \to \mathsf{toggle}(s)|\mathsf{sync}^1_W(n,s)\,.\,\mathrm{Machine}_W(\neg s)
\end{aligned}$$

Each component describes part of the behaviour where the parameter value, either n or s, is known. However, the value of the other parameter is unknown. To cater for this, we add it as a sum variable. Consider the state space of $\mathrm{Machine}_V(0)$ shown below. It describes the behaviour of M where the value of n is known, but at each state it allows both values of s.



The synchronisation actions make the values of the parameters, which are chosen non-deterministically for the unknown parameters, visible in the behaviour. This can be used to achieve a valid decomposition by enforcing the synchronisation of these actions as follows:

$$\nabla_{\{⟅\mathsf{toggle}⟆,⟅\mathsf{count}⟆\}}(\tau_{\{\mathsf{sync}^0,\mathsf{sync}^1\}}(\\ \Gamma_{\{\mathsf{sync}^0_V|\mathsf{sync}^0_W\to\mathsf{sync}^0,\mathsf{sync}^1_V|\mathsf{sync}^1_W\to\mathsf{sync}^1\}}(\mathrm{Machine}_V(0) \parallel \mathrm{Machine}_W(\mathsf{false}))))$$

Unfortunately the state space of $\mathrm{Machine}_W(\mathsf{false})$ is infinitely branching from its initial state and it has no finite state space that is strongly bisimilar to it. Therefore, the previously described state space construction cannot be applied to this decomposition. However, it can be verified that this is a valid decomposition.

For the purpose of state space exploration the effectiveness of a valid decomposition is determined by the size, *i.e.*, the sum of the number of states and transitions, of the state space corresponding to each component. As a minimum requirement the size of each component state space should be smaller than the size of the original state space. Furthermore, it would be considered useful whenever the composition of the minimised components is smaller than the original state space.

3.1 Separation Tuples

Example 2 hinted at the construction that we use to achieve a valid decomposition. However, as we have seen this decomposition could not yet be used for the compositional state space construction. Furthermore, we need to consider the restrictions that the resulting components should satisfy in order to be a valid decomposition in the general case.

To obtain a useful decomposition it can be beneficial to reduce the number of parameters that occur in the synchronisation actions, because these actions become visible as transitions in the state spaces of the individual components. Furthermore, in some cases we can actually remove the synchronisation for summands completely. For instance, in the first summand of Machine in Example 1 we can observe that the value of parameter s remains unchanged and the condition is only an expression containing parameter n. Therefore, we could allow component Machine_V to result in a transition labelled with count without a corresponding summand in Machine_W that synchronises the values of s and n unnecessarily. We refer to these kind of summands as *independent* summands.

An independent summand can result in transitions without synchronising with the other component in the composition expression. However, this might introduce an issue when the action labels in all action expressions are *not* disjoint. For example, consider an LPE that contains two summands with a as action expression that are independent in different components and another summand with $a|a$ as action expression. Then $a|a$ would be allowed in the composition expression to ensure that these transitions can occur, but then the independent summands could also result in a simultaneous transition due to rule Par, which was not a possibility in the original LPE. To prevent this issue we introduce a *tag* action label and only allow independent actions with a single tag in the composition.

First, we present several restrictions on the structure of the component LPEs. In the following definition there is a set of indices K to partition the summands between dependent and *independent* summands. Furthermore, there is a set of indices J to indicate summands that are present in this component. Finally, we allow the condition, action and synchronisation expressions to be chosen freely, which can be used to further reduce the amount of parameter synchronisation. We use indexed sets for these expressions such that the index corresponds to the index of the summand, where for each element we indicate the index by a subscript.

Definition 11. *Let* $P(\vec{d} : \vec{D}) = +_{i\in I} \sum_{e_i:E_i} c_i \to \alpha_i \,.\, P(\vec{g}_i)$ *be an LPE. Let* $(P, U, K, J, c^U, \alpha^U, h^U)$ *be a* separation *tuple such that* $U \subseteq \mathbb{N}$ *is a set of parameter indices and* $K \subseteq J \subseteq I$ *are two sets of summand indices. Furthermore,* c^U, α^U *and* h^U *are indexed sets of condition, action and update expressions respectively such that for all* $i \in (J \setminus K)$ *it holds that* $\mathit{FV}(c_i^U) \cup \mathit{FV}(\alpha_i^U) \cup \mathit{FV}(\vec{h_i^U}) \subseteq \mathsf{Vars}(\vec{d}) \cup \{e_i\}$*. Finally, for all* $i \in K$ *it holds that* $\mathit{FV}(c_i) \cup \mathit{FV}(\alpha_i) \cup \mathit{FV}(\vec{g_{i|U}}) \subseteq \mathsf{Vars}(\vec{d_{|U}}) \cup \{e_i\}$*.*

The separation tuple induces an LPE, where $U^c = \mathbb{N} \setminus U$*, as follows:*

$$\begin{aligned} P_U(\vec{d_{|U}} : \vec{D}_{|U}) = \mathop{+}_{i\in(J\setminus K)} \sum_{e_i:E_i, \vec{d}_{|U^c}:\vec{D}_{|U^c}} & \\ c_i^U \to \alpha_i^U |\mathit{sync}_U^i(\vec{h_i^U}) \,.\, P_U(\vec{g_{i|U}}) & \\ + \mathop{+}_{i\in K} \sum_{e_i:E_i} c_i \to \alpha_i|\mathit{tag} \,.\, P_U(\vec{g_{i|U}}) & \end{aligned}$$

We assume that action labels sync_V^i *and* sync_W^i*, for any* $i \in I$*, and label* tag *does not occur in* α_j*, for any* $j \in I$*, to ensure that these action labels are fresh.*

The composition expression for these components is a generalisation of the composition expression presented in Example 2 where tag action labels are hidden at the highest level.

Definition 12. *Let* $P(\vec{d} : \vec{D}) = +_{i\in I} \sum_{e_i:E_i} c_i \to \alpha_i \,.\, P(\vec{g}_i)$ *be an LPE and* $(P, V, K_V, J_V, c^V, \alpha^V, h^V)$ *and* $(P, W, K_W, J_W, c^W, \alpha^W, h^W)$ *be separation tuples. Let* $P_V(\vec{d_{|V}} : \vec{D}_{|V}) = \phi_V$ *and* $P_W(\vec{d_{|W}} : \vec{D}_{|W}) = \phi_W$ *be the induced LPEs according to Definition 11. Let* $\iota : \vec{D}$ *be a closed expression. Then the composition expression is defined as:*

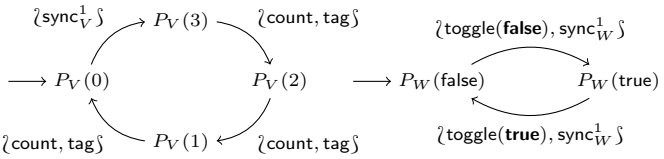
$$\begin{aligned} &\tau_{\{\mathsf{tag}\}}(\nabla_{\{\underline{\alpha_i}|i\in I\}\cup\{\underline{\alpha_i|\mathsf{tag}}|i\in K_V\}|i\in K_W\}\}}(\\ &\tau_{\{\mathsf{sync}^i|i\in I\}}(\Gamma_{\{\mathsf{sync}_V^i|\mathsf{sync}_W^i\to\mathsf{sync}^i|i\in I\}}(P_V(\vec{\iota_{|V}}) \parallel P_W(\vec{\iota_{|W}}))))) \end{aligned}$$

We revisit Example 1 to illustrate a valid decomposition that fits the given structure and unlike Example 2 allows for reduced synchronisation.

Example 3. Consider the LPE presented in Example 1 again. We obtain components P_V and P_W for the separation tuples $(P, V, \{0\}, \{0, 1\}, \{(n > 0)_0, (n \approx 0)_1\}, \{(\mathsf{count}|\mathsf{tag})_0, (\tau)_1\}, \{\langle\rangle_1\})$ and $(P, W, \emptyset, \{1\}, \{\mathsf{true}_1\}, \{\mathsf{toggle}(s)_1\}, \{\langle\rangle_1\})$ respectively.

$$\begin{aligned} P_V(n : N) &= (n > 0) \to \mathsf{count}|\mathsf{tag} \,.\, P_V(n - 1) \\ &\quad + (n \approx 0) \to \mathsf{sync}_V^1 \,.\, P_V(3) \\ P_W(s : B) &= \mathsf{true} \to \mathsf{toggle}(s)|\mathsf{sync}_W^1 \,.\, P_W(\neg s) \end{aligned}$$

The state spaces of components $P_V(0)$ and $P_W(\mathsf{false})$ are shown below.



We obtain the following composition according to Definition 12:

$$\tau_{\{\mathsf{tag}\}}(\nabla_{\{\wr\mathsf{toggle}\wr,\wr\mathsf{count}\wr,\wr\mathsf{count,tag}\wr\}}(\tau_{\{\mathsf{sync}^0,\mathsf{sync}^1\}}(\Gamma_{\{\mathsf{sync}^0_V|\mathsf{sync}^0_W\to\mathsf{sync}^0,\mathsf{sync}^1_V|\mathsf{sync}^1_W\to\mathsf{sync}^1\}}(P_V(0) \parallel P_W(\mathsf{false}))))))$$

One can verify that the state space of this expression is strongly bisimilar to the state space of Machine$(0, \mathsf{false})$ shown in Example 1. As shown above the state space of $P_V(0)$ has four states and transitions, and the state space of $P_W(\mathsf{false})$ has two states and transitions, which are both smaller than the original state space. Their composition is exactly the same size as the original state space, where no further minimisation can be achieved.

3.2 Cleave Correctness Criteria

Not every decomposition which satisfies Definition 12 yields a valid decomposition. For example, replacing the condition expression in Example 3 of the summand in P_W by false would not result in a valid decomposition. Therefore, we need to consider restrictions that should be imposed on the components such that the result of the composition expression defined in Definition 12 is *always* a valid decomposition, which means that the composition should be strongly bisimilar to the original state space for the given initial values. We essentially employ restrictions to preserve a relation between each original state and the two states of the components where the parameters have the same value.

Let us consider any decomposition according to Definition 12. We provide an intuition for each of the requirements that is stated in Definition 13. First of all, we need to ensure that every summand of the LPE either occurs as an independent summand in one of the components or it occurs in both components. This is stated by requirement SYN. Furthermore, this requirement ensures that the condition, action and update expressions for summands with indices in $(J_V \cap J_W)$ are defined in requirements ORG and COM.

In Definition 12 we see that summands with indices in K have condition, action and update expressions that only have the parameters in $\vec{d}_{|V}$ and the summand variable as free variables. Furthermore, to consider a summand as *independent* we require that the update expressions in $\vec{d}_{|W}$ do not modify the parameters. This is essential as otherwise these updates would be omitted, which would almost certainly not yield a valid decomposition. Therefore, we introduce requirement IND to ensure that elements of K_V and K_W are independent summands.

For the summands with an index in $(J_V \cap J_W)$ we need to consider when the synchronisation should occur and when it is not allowed. Whether it is allowed

depends on the outgoing transitions that occur due to the corresponding original summand, *i.e.*, the summand in the original LPE with the same index. If there is an outgoing transition due to such a summand then both components must result in a transition where the synchronisation action can communicate. This means that their conditions must be true, the synchronisation vectors ($\vec{h}^V$ and $\vec{h}^W$) must have equal values and the resulting action label must be equal to the original action label These are exactly the requirements stated by ORG.

Similarly, whenever both components have outgoing transitions that could synchronise then there must be a corresponding outgoing transition in the original state space. Here, the complication is that the values for the sum variables, which are both the original sum variable and the values of the other parameters, can be chosen non-deterministically, and therefore requirement COM ensures that for any choice of these variables there is a corresponding outgoing transition in the original state.

Definition 13. *Let* $P(\vec{d} : \vec{D}) = +_{i \in I} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P(\vec{g}_i)$ *be an LPE and* $(P, V, K_V, J_V, c^V, \alpha^V, h^V)$ *and* $(P, W, K_W, J_W, c^W, \alpha^W, h^W)$ *be separation tuples as defined in Definition 11. The two separation tuples are a* cleave *of* P *iff the following requirements hold.*

SYN $J_V = I \setminus K_W$ *and* $J_W = I \setminus K_V$.

IND *For all* $r \in K_V$ *it holds that* $\vec{g_r}_{|W} = \vec{d}_{|W}$, *and for all* $r \in K_W$ *it holds that* $\vec{g_r}_{|V} = \vec{d}_{|V}$.

ORG *For all* $r \in (J_V \cap J_W)$, *closed expressions* $\vec{\iota} : \vec{D}$ *and* $l : E_r$ *such that* $\sigma = [\vec{d} \leftarrow \vec{\iota}, e_r \leftarrow l]$ *it holds that if* $[\![\sigma(c_r)]\!]$ *then:*
- $[\![\sigma(c_r^V \wedge c_r^W)]\!]$, *and*
- $[\![\sigma(\vec{h_r^V} \approx \vec{h_r^W})]\!]$, *and*
- $[\![\sigma(\alpha_r^V | \alpha_r^W \approx \alpha_r)]\!]$.

COM *For all* $r \in (J_V \cap J_W)$, *closed expressions* $\vec{\iota}, \vec{\iota'}, \vec{\iota''} : \vec{D}$ *and* $l, l' : E_r$ *such that* $\sigma = [\vec{d}_{|V} \leftarrow \vec{\iota}_{|V}, \vec{d}_{|W \setminus V} \leftarrow \vec{\iota'}_{|W \setminus V}, e_r \leftarrow l]$ *and* $\sigma' = [\vec{d}_{|W} \leftarrow \vec{\iota}_{|W}, \vec{d}_{|V \setminus W} \leftarrow \vec{\iota''}_{|V \setminus W}, e_r \leftarrow l']$ *the following holds. If both* $[\![\sigma(c_r^V) \wedge \sigma'(c_r^W)]\!]$ *and* $[\![\sigma(\vec{h_r^V}) \approx \sigma'(\vec{h_r^W})]\!]$ *hold then there is a closed expression* $l'' : E_r$ *such that for the substitution* $\rho = [\vec{d} \leftarrow \vec{\iota}, e_r \leftarrow l'']$ *it holds that:*
- $[\![\rho(c_r)]\!]$, *and*
- $[\![\sigma(\alpha_r^V) | \sigma'(\alpha_r^W)]\!] = [\![\rho(\alpha_r)]\!]$, *and*
- $[\![\sigma(\vec{g_r}_{|V})]\!] = [\![\rho(\vec{g_r}_{|V})]\!]$, *and*
- $[\![\sigma'(\vec{g_r}_{|W})]\!] = [\![\rho(\vec{g_r}_{|W})]\!]$.

Note that requirements ORG and COM of Definition 13 are stated on the semantics of the condition, action and update expressions. In practice, we would need to effectively approximate these correctness requirements using static analysis. For example the requirements on the condition expression can be approximated by using the syntactic conjunctions that are present in it. However, how precise and efficient this static analysis can be is left as future work. The correctness of the cleave is established by the following theorem.

Theorem 1. *The composition expression defined in Definition 12 where the two separation tuples are a cleave according to Definition 13 is a valid decomposition as defined in Definition 10.*

Proof. Let R be the smallest relation such that for any closed expressions $\vec{\iota}' : \vec{D}$ the following holds.

$$P(\vec{\iota}') \;\; R \;\; \tau_{\{\mathsf{tag}\}}(\nabla_{\{\underline{\alpha_i}|i\in I\}\cup\{\underline{\alpha_i}|\mathsf{tag}|i\in K_V\}|i\in K_W\}}(\tau_{\{\mathsf{sync}^i|i\in I\}} (\Gamma_{\{\mathsf{sync}^i_V|\mathsf{sync}^i_W\to\mathsf{sync}^i|i\in I\}}(P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W})))))$$

We prove that R is a strong bisimulation *up to bisimilarity* $\leftrightarroweq$. The essential observation is that due to parameter synchronisation the original state vectors can always be traced to the two states (of P_V and P_W) that carry the same values, and vice versa. The requirements ensure that (the combination of) expressions evaluate to the same values as the corresponding original values. The full proof can be found in Appendix A.

Observe that the decompositions obtained in Example 2 and Example 3 are cleaves. However, we can also observe that the decomposition in Example 2 is not a particularly useful cleave for the purpose of compositional minimisation. In this case, as shown by Example 3 we could avoid the infinite branching of $\text{Machine}_W(\mathsf{false})$ by reducing the amount of synchronisation, but this might not always be possible. Therefore, we present an alternative technique to restrict the behaviour of the resulting components.

4 State Invariants

One way to restrict the behaviour of the components is to strengthen the condition expressions of each summand to avoid certain outgoing transitions. We show that so-called *state invariants* [1] can be used for this purpose. These state invariants are typically formulated by the user based on intuition of the model behaviour.

Definition 14. *Given an LPE* $P(d : D) = +_{i\in I} \sum_{e_i:E_i} c_i \to \alpha_i \,.\, P(g_i)$. *A boolean expression* ψ *such that* $\mathsf{FV}(\psi) \subseteq \{d\}$ *is called a* state invariant *iff the following holds: for all* $i \in I$ *and closed expressions* $\iota : D$ *and* $l : E_i$ *such that* $[\![[d \leftarrow \iota, e_i \leftarrow l](c_i \wedge \psi)]\!]$ *holds then* $[\![[d \leftarrow [d \leftarrow \iota, e_i \leftarrow l](g_i)](\psi)]\!]$ *holds as well.*

The essential property of a state invariant is that whenever it holds for the initial state it is guaranteed to hold for all reachable states in the state space. This follows relatively straightforward from its definition. Next, we define a *restricted* LPE where (some of) the condition expressions are strengthened with a boolean expression.

Definition 15. *Given an LPE* $P(d : D) = \sum_{i \in I} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P(g_i)$*, a boolean expression* ψ *such that* $\mathsf{FV}(\psi) \subseteq \{d\}$ *and a set of indices* $J \subseteq I$*. We define the restricted LPE, denoted by* $P^{\psi,J}$*, as follows:*

$$P^{\psi,J}(d : D) = \sum_{i \in J} \sum_{e_i : E_i} c_i \wedge \psi \to \alpha_i \,.\, P^{\psi,J}(g_i) + \sum_{i \in (I \setminus J)} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P^{\psi,J}(g_i)$$

Note that if the boolean expression ψ in Definition 15 is a state invariant for the given LPE then for all closed expressions $\vec{\imath} : \vec{D}$ such that $[\![[\vec{d} \leftarrow \vec{\imath}](\psi)]\!]$ holds, it holds that $[\![P(\vec{\imath})]\!] \leftrightarrows [\![P^{\psi,J}(\vec{\imath})]\!]$, for any $J \subseteq I$. Therefore, we can use a state invariant of an LPE to strengthen all of its condition expressions.

Moreover, a state invariant of the original LPE can *also* be used to restrict the behaviour of the components obtained from a cleave, as formalised in the following theorem. Note that the set of indices is used to only strengthen the condition expressions of summands that introduce synchronisation, because the condition expressions of independent summands cannot contain the other parameters as free variables. Furthermore, the restriction can be applied to independent summands before the decomposition.

Theorem 2. *Let* $P(\vec{d} : \vec{D}) = \sum_{i \in I} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P(\vec{g}_i)$ *be an LPE and* $(V, K_V, J_V, c^V, \alpha^V, h^V)$ *and* $(W, K_W, J_W, c^W, \alpha^W, h^W)$ *be separations tuples as defined in Definition 11. Let* ψ *be a state invariant of* P*. Given closed expressions* $\vec{\imath} : \vec{D}$ *such that* $[\![[\vec{d} \leftarrow \vec{\imath}](\psi)]\!]$ *holds the following expression, where* $C = J_V \cap J_W$*, is a valid decomposition:*

$$\tau_{\{\mathsf{tag}\}}(\nabla_{\{\underline{\alpha_i} | i \in I\} \cup \{\underline{\alpha_i | \mathsf{tag}} | i \in K_V\} | i \in K_W\}}(\tau_{\{\mathsf{sync}^i | i \in I\}}(\Gamma_{\{\mathsf{sync}^i_V | \mathsf{sync}^i_W \to \mathsf{sync}^i | i \in I\}}(P_V^{\psi,C}(\vec{\imath}_{|V}) \parallel P_W^{\psi,C}(\vec{\imath}_{|W})))))$$

Proof. This proof is similar to the proof of Theorem 1 with the strong bisimilation relation only defined for states where the invariant holds. The full proof can be found in Appendix B.

Observe that the predicate $n \leq 3$ is a state invariant of the LPE Machine in Example 1. Therefore, we can consider the process $\mathrm{Machine}_W^{\psi,I}$ in Example 2 for the composition expression, which is finite. This would yield two finite components, but the state space of $\mathrm{Machine}_W^{\psi,I}$ is larger than that of P_W in Example 3.

Finally, we remark that the restricted state space contains deadlock states whenever the invariant does not hold. These deadlocks can be avoided by applying the invariant to the update expression of each parameter instead of the parameter itself without affecting the correctness.

5 Case Study

We have implemented an automated translation that, given a partitioning of parameters defined by the user, uses a simple static analysis to obtain components

that are guaranteed to satisfy the requirements of a cleave. Each experiment is a specification written in the high-level language mCRL2 [10], a process algebra generalising the one of Section 2.3. To apply the decomposition technique we derive an LPE that is behaviourally equivalent using a normalisation procedure that is not discussed in more detail, but which is implemented in the mCRL2 toolset [4].

We compare size of the original state space to the sizes of the components. Furthermore, we have minimised the state spaces modulo strong bisimulation. Finally, we compute the state space of the composition expression applied to these minimised components to determine the effectiveness of the decomposition when compared to the original minimised state space. We do not present run time and memory usage for these benchmarks. However, the cost of computing the cleave was in the range of several milliseconds.

5.1 Alternating Bit Protocol

The alternating bit protocol (ABP) is a communication protocol that uses a single control bit, which is sent along the message, to implement a reliable communication channel over two unreliable channels [10]. The specification contains four processes for the sender, receiver and two unreliable communication channels.

First, we choose the partitioning of the parameters such that one component (ABP_V) contains the parameters of the sender and one communication channel, and the other component (ABP_W) contains the parameters of the receiver and the other communication channel. We observe that both components are larger than the original state space, and can also not be minimised further, illustrating that traditional compositional minimisation is, in this case, not particularly useful.

Model	original		minimised	
	#states	#trans	#states	#trans
ABP	182	230	48	58
ABP_V	204	512	204	512
ABP_W	64	196	60	192
ABP_V^ψ	104	180	53	180
ABP_W^ψ	58	110	21	108
$\mathsf{ABP}_V^\psi \parallel \mathsf{ABP}_W^\psi$	172	220	48	58
ABP'_V	5	35	5	35
ABP'_W	78	118	28	42
$\mathsf{ABP}'_V \parallel \mathsf{ABP}'_W$	76	90	48	58

Table 1. Metrics for the alternating bit protocol.

Further analysis showed that the behaviour of each process heavily depends on the state of the other processes, which results in large components as this

information is lost. We can encode this global information as a state invariant based on the *control flow* parameters. The second cleave is obtained by obtaining two restricted components (ABP_V^ψ and ABP_W^ψ) using this invariant. This yields a useful decomposition.

Finally, we have obtained a cleave into components ABP'_V and ABP'_W where the partitioning is not based on the original processes. This yields a very effective cleave as shown in Table 1.

5.2 Practical Examples

In these experiments we consider several more practical specifications. We compare the results of the monolithic exploration and the exploration based on the decomposition in Table 2. The parameter partitioning for each case is our best effort to obtain the optimal decomposition

Table 2. State space metrics for various practical specifications.

Model	exploration		minimised	
	#states	#trans	#states	#trans
Register	914 048	1 885 824	1 740	3 572
$\mathsf{Register}_V$	464	10 672	464	10 672
$\mathsf{Register}_W$	97 280	273 408	5 760	16 832
$\mathsf{Register}_V \parallel \mathsf{Register}_W$	76 416	157 952	1 740	3 572
Chatbox	65 536	2 621 440	16	144
$\mathsf{Chatbox}_V$	128	4 352	128	3 456
$\mathsf{Chatbox}_W$	512	37 888	8	440
$\mathsf{Chatbox}_V \parallel \mathsf{Chatbox}_W$	1 024	22 528	16	144
WMS	155 034 776	2 492 918 760	44 526 316	698 524 456
WMS_V	212 992	5 144 576	212 992	2 801 664
WMS_W	1 903 715	121 945 196	414 540	26 429 911
$\mathsf{WMS}_V \parallel \mathsf{WMS}_W$	64 635 040	1 031 080 812	44 526 316	698 524 456

The Chatbox specification describes a chat room where four users that can join, leave and send messages [16]. This specification is interesting because it is described as a monolithic process, which means that compositional minimisation is not applicable. However, the decomposition technique can be used quite successfully.

The Register specification describes a wait-free handshake register which is presented in [11]. Finally, we consider the workload management system (WMS) specification described in [15]. For the latter two experiments we found that partitioning the parameters into a set of data parameters and so-called control flow parameters yielded the best results. Unfortunately, for these practical models we have not managed to improve the results by using an invariant.

We also consider the execution time and maximum amount of memory required to obtain the original state space using exploration and the state space obtained using the presented decomposition technique, for which the results can be found in Table 3. Here, we consider the maximum amount of memory used, because several tools are run sequentially and only the highest amount used at one time determines the amount of memory required to explore the state space. Finally, we should note that these times are for the state space obtained under "exploration" without considering the final minimisation step.

Table 3. Execution times and maximum memory usage measurements for various specifications.

Model	monolithic		decomposition	
	time	memory	time	memory
Chatbox	4.4s	21.3MB	0.2s	14.9MB
Register	6.9s	96.9MB	1.3s	23.5MB
WMS	2.4h	14.5GB	0.7h	11.5GB

6 Conclusion

We have presented a decomposition technique, referred to as cleave, that can be applied to any monolithic process with the structure of an LPE and have shown that the result is always a valid decomposition. Furthermore, we have shown that state invariants can be used to improve the effectiveness of the decomposition. For practical application we must consider improvements to the heuristics used to obtain the components and especially heuristics to choose the parameter partitioning automatically. Furthermore, it can also be interesting to consider cleaving into more than two components and even applying it recursively to the resulting components. Finally, the cleave is currently not well-suited for applying the typically more useful abstraction based on (divergence-preserving) branching bisimulation minimisation [19]. The reason for this is that τ-actions might be extended with synchronisation actions and tags. As a result they become visible, effectively reducing branching bisimilarity to strong bisimilarity.

References

1. Marc Bezem and Jan Friso Groote. Invariants in process algebra with data. In Bengt Jonsson and Joachim Parrow, editors, *CONCUR*, volume 836 of *LNCS*, pages 401–416. Springer, 1994.
2. Pierre Bouvier, Hubert Garavel, and Hernán Ponce de León. Automatic decomposition of petri nets into automata networks - A synthetic account. In Ryszard Janicki, Natalia Sidorova, and Thomas Chatain, editors, *Application and Theory of Petri Nets and Concurrency - 41st International Conference, PETRI NETS 2020, Paris, France, June 24-25, 2020, Proceedings*, volume 12152 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2020.

3. Ed Brinksma, Rom Langerak, and Peter Broekroelofs. Functionality decomposition by compositional correctness preserving transformation. In Costas Courcoubetis, editor, *CAV*, volume 697 of *LNCS*, pages 371–384. Springer, 1993.
4. Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mcrl2 toolset for analysing concurrent systems - improvements in expressivity and usability. In Tomás Vojnar and Lijun Zhang, editors, *TACAS*, volume 11428 of *LNCS*, pages 21–39. Springer, 2019.
5. Shing-Chi Cheung and Jeff Kramer. Context constraints for compositional reachability analysis. *ACM Trans. Softw. Eng. Methodol.*, 5(4):334–377, 1996.
6. Hubert Garavel, Frédéric Lang, Radu Mateescu, and Wendelin Serwe. CADP 2011: a toolbox for the construction and analysis of distributed processes. *STTT*, 15(2):89–107, 2013.
7. Hubert Garavel, Frédéric Lang, and Laurent Mounier. Compositional verification in action. In Falk Howar and Jiri Barnat, editors, *FMICS*, volume 11119 of *LNCS*, pages 189–210. Springer, 2018.
8. Susanne Graf, Bernhard Steffen, and Gerald Lüttgen. Compositional minimisation of finite state systems using interface specifications. *Formal Asp. Comput.*, 8(5):607–616, 1996.
9. Jan Friso Groote and Faron Moller. Verification of parallel systems via decomposition. In Rance Cleaveland, editor, *CONCUR*, volume 630 of *LNCS*, pages 62–76. Springer, 1992.
10. Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. MIT Press, 2014.
11. Wim H. Hesselink. Invariants for the construction of a handshake register. *Inf. Process. Lett.*, 68(4):173–177, 1998.
12. Sung-Shik T. Q. Jongmans, Dave Clarke, and José Proença. A procedure for splitting data-aware processes and its application to coordination. *Sci. Comput. Program.*, 115-116:47–78, 2016.
13. Robin Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25:267–310, 1983.
14. Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
15. Daniela Remenska, Tim A. C. Willemse, Kees Verstoep, Wan J. Fokkink, Jeff Templon, and Henri E. Bal. Using model checking to analyze the system behavior of the LHC production grid. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13-16, 2012*, pages 335–343. IEEE Computer Society, 2012.
16. Judi Romijn and Jan Springintveld. Exploiting symmetry in protocol testing. In Stanislaw Budkowski, Ana R. Cavalli, and Elie Najm, editors, *FORTE XI / PSTV XVIII*, volume 135 of *IFIP Conference Proceedings*, pages 337–352. Kluwer, 1998.
17. Kuo-Chung Tai and Pramod V. Koppol. Hierarchy-based incremental analysis of communication protocols. In *ICNP*, pages 318–325. IEEE Computer Society, 1993.
18. Kuo-Chung Tai and Pramod V. Koppol. An incremental approach to reachability analysis of distributed programs. In Jack C. Wileden, editor, *IWSSD*, pages 141–151. IEEE Computer Society, 1993.
19. R. J. van Glabbeek, B. Luttik, and N. Trčka. Branching bisimilarity with explicit divergence. *Fundam. Inform.*, 93(4):371–392, 2009.

A Proof of Theorem 1

The following definition and proposition is due to [14]. These are in the context of two LTSs $\mathcal{L}_1 = (S_1, s_1, Act_1, \rightarrow_1)$ and $\mathcal{L}_2 = (S_2, s_2, Act_2, \rightarrow_2)$. We introduce for a binary relation $R \subseteq S_1 \times S_2$ the following notation $\leftrightarroweq R \leftrightarroweq$ to denote the *relational composition* such that $\leftrightarroweq R \leftrightarroweq \; = \{(s,t) \in S_1 \times S_2 \mid \exists s' \in S_1, t' \in S_2 : s \leftrightarroweq s' \wedge s' \mathrel{R} t' \wedge t' \leftrightarroweq t\}$.

Definition 16. *A binary relation $R \subseteq S_1 \times S_2$ is a* strong bisimulation up to $\leftrightarroweq$ *iff for all $s \mathrel{R} t$ it holds that:*
- *if $s \xrightarrow{\omega} s'$ then there is a state $t' \in S_2$ such that $t \xrightarrow{\omega} t'$ and $s' \leftrightarroweq R \leftrightarroweq t'$.*
- *if $t \xrightarrow{\omega} t'$ then there is a state $s' \in S_1$ such that $t \xrightarrow{\omega} t'$ and $t' \leftrightarroweq R \leftrightarroweq s'$.*

Proposition 1. *If R is a strong bisimulation up to $\leftrightarroweq$ then $R \subseteq \leftrightarroweq$*

This result establishes that if R is a strong bisimulation up to $\leftrightarroweq$ then for any pair $(s,t) \in R$ we can conclude that $s \leftrightarroweq t$.

We introduce two auxiliary lemmas to relate the transition induced by some expression $P \in \mathsf{S}$ to the transitions induced by applying the allow, hide and communication operators, in the same order as the composition expression defined in Definition 12, to P.

Lemma 2. *Given expressions $P, Q \in \mathsf{S}$, a set of multi-sets of action labels $A \subseteq 2^{\Lambda \to \mathbb{N}}$, sets of events $H', H \subseteq \Lambda$, a set of communications $C \subseteq \mathsf{Comm}$. If $P \xrightarrow{\omega'} Q$ and $\underline{\theta_{H'}(\theta_H(\gamma_C(\omega')))} \in A$ then:*

$$\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P)))) \xrightarrow{\theta_{H'}(\theta_H(\gamma_C(\omega')))} \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(Q))))$$

Proof. We can derive the following:

$$\textsc{Hide}\;\frac{\textsc{Hide}\;\dfrac{\textsc{Allow}\;\dfrac{\textsc{Com}\;\dfrac{P \xrightarrow{\omega'} Q \qquad C \subseteq \mathsf{Comm}}{\Gamma_C(P) \xrightarrow{\gamma_C(\omega')} \Gamma_C(Q)} \qquad H \subseteq \Lambda}{\tau_H(\Gamma_C(P)) \xrightarrow{\theta_H(\gamma_C(\omega'))} \tau_H(\Gamma_C(Q))} \qquad A \subseteq 2^{\Lambda \to \mathbb{N}} \qquad \underline{\theta_H(\gamma_C(\omega'))} \in A}{\nabla_A(\tau_H(\Gamma_C(P))) \xrightarrow{\theta_H(\gamma_C(\omega'))} \nabla_A(\tau_H(\Gamma_C(Q)))} \qquad H' \subseteq \Lambda}{\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P)))) \xrightarrow{\theta_{H'}(\theta_H(\gamma_C(\omega')))} \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(Q))))}$$

Lemma 3. *Given expressions $P, Q \in \mathsf{S}$, a set of multi-sets of action labels $A \subseteq 2^{\Lambda \to \mathbb{N}}$, sets of events $H', H \subseteq \Lambda$, a set of communications $C \subseteq \mathsf{Comm}$ if:*

$$\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P)))) \xrightarrow{\omega} Q'$$

then there are $Q \in \mathsf{S}$ and $\omega' \in \Omega$ such that $Q' = \nabla_A(\tau_H(\Gamma_C(Q)))$, $\omega = \theta_{H'}(\theta_H(\gamma_C(\omega')))$, $P \xrightarrow{\omega'} Q$ and $\underline{\omega} \in A$.

Proof. Assume that $\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P)))) \xrightarrow{\omega} Q'$. From the structure of the premise conclusion we know that only rule HIDE is applicable, which can only be applied for some $Q'' \in \mathsf{S}$ such that:

$$\textsc{Hide}\ \frac{\nabla_A(\tau_H(\Gamma_C(P))) \xrightarrow{\theta_{H'}(\omega)} Q'' \qquad H' \subseteq \Lambda}{\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P)))) \xrightarrow{\theta_{H'}(\omega)} \tau_{H'}(Q'')} \tag{1}$$

Similarly, we derive the applicability of the ALLOW and COM rules such that we essentially can obtain (the only possible) derivation shown in the proof of Lemma 2.

Theorem 1. *The composition expression defined in Definition 12 where the two separation tuples are a cleave according to Definition 13 is a valid decomposition as defined in Definition 10.*

Proof. Pick an arbitrary closed expression $\vec{\iota''} : \vec{D}$. Let $(S_1, s_1, Act_1, \rightarrow_1) = [\![P(\vec{\iota''})]\!]$ and $(S_2, s_2, Act_2, \rightarrow_2) = [\![\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota''}_{|V}) \parallel P_W(\vec{\iota''}_{|W})))))]\!]$. Let $A = \{\underline{\alpha_i} \mid i \in I\} \cup \{\underline{\alpha_i|\mathsf{tag}} \mid i \in J_V\}\}$, $H' = \{\mathsf{tag}\}$, $H = \{\mathsf{sync}^i \mid i \in I\}$ and $C = \{\mathsf{sync}^i_V|\mathsf{sync}^i_W \rightarrow \mathsf{sync}^i \mid i \in I\}$.

Let R be the smallest relation R such that $P(\vec{\iota'})\, R\, \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota'}_{|V}) \parallel P_W(\vec{\iota'}_{|W})))))$, for any closed expression $\vec{\iota'} : \vec{D}$. We show that R is a strong bisimulation relation up to $\leftrightarrows$. Pick any arbitrary closed expression $\vec{\iota} : \vec{D}$ and suppose we have the following: $P(\vec{\iota})\ R\ \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W})))))$.

- Case $P(\vec{\iota}) \xrightarrow{\omega}_1 Q'$. There is an index $r \in I$ and a closed expression $l : E_r$ such that for $\sigma = [\vec{d} \leftarrow \vec{\iota}, e_r \leftarrow l]$ it holds that $[\![\sigma(c_r)]\!]$, $\omega = [\![\sigma(\alpha_r)]\!]$ and $Q' = P(\sigma(\vec{g_r}))$. There are three cases to consider based on the index r.
  - Case $r \in I \setminus (K_V \cup K_W)$. From SYN this means that $r \in (J_V \cap J_W)$. We derive the transitions using requirement ORG. First, from $[\![\sigma(c^V_r \wedge c^W_r)]\!]$ it follows that:

$$P_V(\vec{\iota}_{|V}) \xrightarrow{[\![\sigma(\alpha^V_r|\mathsf{sync}^V_r(\vec{h}^V_r))]\!]}_2 P_V(\sigma(\vec{g}_{r|V}))$$
$$\text{and } P_W(\vec{\iota}_{|W}) \xrightarrow{[\![\sigma(\alpha^W_r|\mathsf{sync}^W_r(\vec{h}^W_r))]\!]}_2 P_W(\sigma(\vec{g}_{r|W}))$$

  Furthermore, $[\![\sigma(\alpha_r)]\!] = [\![\sigma(\alpha^V_r|\alpha^W_r)]\!]$ and by rule PAR there is:

$$P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{[\![\sigma(\alpha_r)|\mathsf{sync}_V(\sigma(\vec{h}^V_r)))|\mathsf{sync}_W(\sigma(\vec{h}^W_r)))]\!]}_2 P_V(\sigma(\vec{g}_{r|V})) \parallel P_W(\sigma(\vec{g}_{r|W}))$$

  From $[\![\sigma(h^{\vec{V}}_r \approx h^{\vec{W}}_r)]\!]$ it follows that:

$$\theta_{H'}(\theta_H(\gamma_C([\![\sigma(\alpha_r)|\mathsf{sync}_V(\sigma(\vec{h}^V_r)))|\mathsf{sync}_W(\sigma(\vec{h}^W_r))]\!]))) = [\![\sigma(\alpha_r)]\!]$$

  From $\underline{[\![\sigma(\alpha_r)]\!]} \in A$ we know by Lemma 2 that:

$$\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W})))))) \xrightarrow{[\![\sigma(\alpha_r)]\!]}_2 \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g}_{r|V})) \parallel P_W(\sigma(\vec{g}_{r|W})))))))$$

Finally, $P(\sigma(\vec{g_r}))\ R\ \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g_{r|V}})) \parallel P_W(\sigma(\vec{g_{r|W}}))))))$.

- Case $r \in K_V$. We derive $P_V(\vec{\iota}_{|V}) \xrightarrow{[\![\sigma(\alpha_r)|\mathsf{tag}]\!]}_2 P_V(\sigma(\vec{g_{r|V}}))$, because $[\![\sigma(c_r)]\!]$ holds. There is a transition $P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{[\![\sigma(\alpha_r)|\mathsf{tag}]\!]}_2 P_V(\sigma(\vec{g_{r|V}})) \parallel P_W(\vec{\iota}_{|W})$ by From rule ParL. Furthermore, by definition $\theta_{H'}(\theta_H(\gamma_C(\sigma(\alpha_r)|\mathsf{tag}))) = \sigma(\alpha_r)$ and $\underline{\sigma(\alpha_r)} \in A$. From Lemma 2 we conclude that:

$$\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}))))) \xrightarrow{\sigma(\alpha_r)}_2 \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g_{r|V}})) \parallel P_W(\vec{\iota}_{|W}))))))$$

  By IND it holds that $\vec{g_{r|W}} = \vec{\iota}_{|W}$. Finally, by definition $P(\sigma(\vec{g_r}))\ R\ \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g_{r|V}})) \parallel P_W(\sigma(\vec{g_{r|W}}))))))$.
- Case $r \in K_W$. Follows from the same observations as $r \in K_V$.

– Case $\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}))))) \xrightarrow{\omega}_2 Q'$. By Lemma 3 there is an expression $Q \in \mathsf{S}$ such that $Q' = \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P))))(Q)$, and $\omega' \in \Omega$ such that $\omega = \theta_{H'}(\theta_H(\gamma_C(\omega')))$, $P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{\omega'}_2 Q$ and $\underline{\omega} \in A$. There are three cases where a parallel composition results in a transition. Suppose that $P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{\omega'}_2 Q$ is due to:

- Case $P_V(\vec{\iota}_{|V}) \xrightarrow{\omega_V}_2 P'_V$ and $P_W(\vec{\iota}_{|W}) \xrightarrow{\omega_W}_2 P'_W$ and rule Par. Then there is a transition $P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{\omega_V+\omega_W}_2 P'_V \parallel P'_W$ such that $\omega_V + \omega_W = \omega'$.
  From $\underline{\theta_{H'}(\theta_H(\gamma_C(\omega_V + \omega_W)))} \in A$ we know that $\gamma_C(\omega_V + \omega_W) = \omega + \wr\mathsf{sync}_r\wr$, for some index $r \in I$, because only a single tag is allowed with original action labels. Therefore, it also follows that $r \in I \setminus (K_V \cup K_W)$. Observe that variables (and the related closed expressions) in $\vec{d}_{|V}$ and $\vec{d}_{|(W\setminus V)}$ are disjoint. There are closed expressions $l, l' : E_r$, $\vec{m} : \vec{D}_{|(W\setminus V)}$, and $\vec{\iota'}, \vec{\iota''} : \vec{D}$ with the substitutions

$$\sigma = [\vec{d}_{|V} \leftarrow \vec{\iota}_{|V}, \vec{d}_{|(W\setminus V)} \leftarrow \vec{\iota'}_{|V\setminus W}, e_r \leftarrow l]$$
$$\text{and } \sigma' = [\vec{d}_{|W} \leftarrow \vec{\iota}_{|W}, \vec{d}_{|(V\setminus W)} \leftarrow \vec{\iota''}_{|W\setminus V}, e_r \leftarrow l']$$

  such that $[\![\sigma(c_r^V)]\!]$ holds, ω_V is equal to $[\![\sigma(\alpha_r^V)|\mathsf{sync}_r^V(\vec{h_r^V}))]\!]$ and $P'_V = P_V([\![\sigma(\vec{g_{r|V}})]\!])$. And $[\![\sigma'(c_r^W)]\!]$ holds, $\omega_W = [\![\sigma'(\alpha_r^W)|\mathsf{sync}_r^W(\vec{h_r^W})]\!])$ and $P'_W = P_W([\![\sigma'(\vec{g_{r|W}})]\!])$ and finally $[\![\sigma(\vec{h_r^V}) \approx \sigma'(\vec{h_r^W})]\!]$ holds.
  From the requirement COM it follows that there is a closed expression $l'' : E_r$ such that for $\rho = [\vec{d} \leftarrow \vec{\iota}, e_r \leftarrow l'']$ that $[\![\rho(c_r)]\!]$ holds and $[\![\sigma(\alpha_r^V)|\sigma'(\alpha_r^W)]\!] = [\![\rho(\alpha_r)]\!]$. We conclude that $P(\vec{\iota}) \xrightarrow{\omega}_1 P(\rho(\vec{g_r}))$. Furthermore, from $[\![\sigma(\vec{g_{r|V}})]\!] = [\![\rho(\vec{g_{r|V}})]\!]$, and $[\![\sigma'(\vec{g_{r|W}})]\!] = [\![\rho(\vec{g_{r|W}})]\!]$ and Lemma 1 it follows that:

$$P_V(\sigma(\vec{g_{r|V}})) \leftrightarrows P_V(\rho(\vec{g_{r|V}}))$$
$$\text{and } P_W(\sigma'(\vec{g_{r|W}})) \leftrightarrows P_W(\rho(\vec{g_{r|W}}))$$

By the congruence of strong bisimilarity with respect to S we obtain that:

$$\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g_r}_{|V})) \parallel P_W(\sigma'(\vec{g_r}_{|W})))))) \leftrightarroweq \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\rho(\vec{g_r}_{|V})) \parallel P_W(\rho(\vec{g_r}_{|W})))))$$

Finally, $P(\rho(\vec{g_r}))\, R\, \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\rho(\vec{g_r}_{|V})) \parallel P_W(\rho(\vec{g_r}_{|W})))))$.

- Case $P_V(\vec{\iota}_{|V}) \xrightarrow{\omega'}_2 P'_V$ and rule ParL such that $P_V(\vec{\iota}_{|V}) \parallel P_W(\vec{\iota}_{|W}) \xrightarrow{\omega'}_2 P'_V \parallel P_W(\vec{\iota}_{|W})$. Pick an arbitrary index $r \in J_V$. If $r \in J_V \setminus K_V$ then the action expression contains an action labelled sync_r^V, which means that $\underline{\theta_{H'}(\theta_H(\gamma_C(\omega')))} \notin A$. Contradiction.
As such $r \in K_V$ and the requirement $\mathsf{FV}(c_r) \cup \mathsf{FV}(\alpha_r) \cup \mathsf{FV}(\vec{g_r}_{|U}) \subseteq \mathsf{Vars}(\vec{d}_{|U}) \cup \{e_r\}$ holds. Therefore, there is a closed expression $l : E_r$ such that for $\sigma = [\vec{d}_{|V} \leftarrow \vec{\iota}_{|V}, e_r \leftarrow l]$ such that $[\![\sigma(c_r)]\!]$, $\omega' = [\![\sigma(\alpha_r)|\mathsf{tag}]\!]$ and $P'_V = P_V(\sigma(\vec{g_r}_{|V}))$.
We know that for $\sigma' = [\vec{d} \leftarrow \vec{\iota}, e_r \leftarrow l]$ that (syntactically) $\sigma(c_r) = \sigma'(c_r)$ and $\sigma(\alpha_r) = \sigma'(\alpha_r)$. Therefore, $[\![\sigma'(c_r)]\!]$ holds and $\omega' = [\![\sigma'(\alpha_r)]\!]$. Furthermore, from $\vec{g_r}_{|W} = \vec{\iota}_{|W}$ (IND) we conclude that $P(\vec{\iota}) \xrightarrow{\omega'}_1 P(\sigma'(\vec{g_r}))$. Finally, we conclude $P(\sigma(\vec{g_r}) R \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\sigma(\vec{g_r}_{|V})) \parallel P_W(\sigma(\vec{g_r}_{|W})))))))$.
- Case $P_W(\vec{\iota}_{|W}) \xrightarrow{\omega'}_2 P'_W$ and rule ParR, along the same lines as above.

Using Proposition 1 we conclude that $[\![P(\vec{\iota''}_{|V})]\!] \leftrightarrow [\![\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V(\vec{\iota''}_{|V}) \parallel P_W(\vec{\iota''}_{|W})))))]\!]$

B Proof of Theorem 2

Theorem 2. *Let $P(\vec{d} : \vec{D}) = +_{i \in I} \sum_{e_i : E_i} c_i \to \alpha_i \,.\, P(\vec{g}_i)$ be an LPE and $(V, K_V, J_V, c^V, \alpha^V, h^V)$ and $(W, K_W, J_W, c^W, \alpha^W, h^W)$ be separations tuples as defined in Definition 11. Let ψ be a state invariant of P. Given closed expressions $\vec{\iota} : \vec{D}$ such that $[\![[\vec{d} \leftarrow \vec{\iota}](\psi)]\!]$ holds the following expression, where $C = J_V \cap J_W$, is a valid decomposition:*

$$\tau_{\{\mathsf{tag}\}}(\nabla_{\{\underline{\alpha_i}|i \in I\} \cup \{\underline{\alpha_i|\mathsf{tag}}|i \in K_V\}|i \in K_W\}}(\tau_{\{\mathsf{sync}^i|i \in I\}}(\Gamma_{\{\mathsf{sync}^i_V|\mathsf{sync}^i_W \to \mathsf{sync}^i|i \in I\}}(P_V^{\psi,C}(\vec{\iota}_{|V}) \parallel P_W^{\psi,C}(\vec{\iota}_{|W})))))$$

Proof. Let ψ be a state invariant of P and let $\vec{\iota''} : \vec{D}$ be a closed expression such that $[\![[\vec{d} \leftarrow \vec{\iota''}](\psi)]\!]$ holds. Let $(S_1, s_1, Act_1, \to_1) = [\![P(\vec{\iota''})]\!]$ and $(S_2, s_2, Act_2, \to_2) = [\![\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V^{\psi,C}(\vec{\iota''}_{|V}) \parallel P_W^{\psi,C}(\vec{\iota''}_{|W})))))]\!]$.

Let R be the smallest relation such that $P(\vec{\iota'})\, R\, \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V^{\psi,C}(\vec{\iota'}_{|V}) \parallel P_W^{\psi,C}(\vec{\iota'}_{|W})))))$, for any closed expression $\vec{\iota'} : \vec{D}$ such that $[\![[\vec{d} \leftarrow \vec{\iota'}](\psi)]\!]$ holds. We show that R is a strong bisimulation relation up to $\leftrightarrow$. The rest of the proof follows the same structure as the proof presented in Appendix A.

Consider the case $P(\vec{\iota}) \xrightarrow{\omega}_1 Q'$. First of all, we know that $[\![[\vec{d} \leftarrow \vec{\iota}](\psi)]\!]$ holds by definition of R. In the case $r \in I \setminus (K_V \cup K_W)$, which means that $r \in (J_V \cap J_W)$, we can therefore conclude that $[\![\sigma(c_r^V \wedge c_r^W \wedge \psi)]\!]$ holds. Furthermore, by definition of a state invariant we know that $[\![[\vec{d} \leftarrow \sigma(\vec{g_r})](\psi)]\!]$ and thus $P(\sigma(\vec{g_r}))\, R\, \tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V^{\psi,C}(\sigma(\vec{g_{r|V}})) \parallel P_W^{\psi,C}(\sigma(\vec{g_{r|W}}))))))$. The other case deal with unrestricted summands and as such only the observation that $[\![[\vec{d} \leftarrow \sigma(\vec{g_r})](\psi)]\!]$ holds needs to be added.

Consider the case $\tau_{H'}(\nabla_A(\tau_H(\Gamma_C(P_V^{\psi,C}(\vec{\iota}_{|V}) \parallel P_W^{\psi,C}(\vec{\iota}_{|W})))))) \xrightarrow{\omega}_2 Q'$. We can easily see that the restricted condition imply the original condition as well. In the first case $P_V^{\psi,C}(\vec{\iota}_{|V}) \xrightarrow{\omega_V}_2 P'_V$ and $P_W^{\psi,C}(\vec{\iota}_{|W}) \xrightarrow{\omega_W}_2 P'_W$ and rule Par we observe that if $[\![\sigma(c_r^V \wedge \psi)]\!]$ holds then $[\![\sigma(c_r^V)]\!]$ holds as well, and similarly if $[\![\sigma'(c_r^W \wedge \psi)]\!]$ holds that $[\![\sigma'(c_r^W)]\!]$ holds. The remainder of the proof stays exactly the same. In the case $P_V^{\psi,C}(\vec{\iota}_{|V}) \xrightarrow{\omega'}_2 P'_V$ and rule ParL such that $P_V^{\psi,C}(\vec{\iota}_{|V}) \parallel P_W^{\psi,C}(\vec{\iota}_{|W}) \xrightarrow{\omega'}_2 P'_V \parallel P_W^{\psi,C}(\vec{\iota}_{|W})$ we observe that $r \in (J_V \setminus K_V)$ and as such the condition expression is not restricted.