Verification of Parallel Systems via Decomposition

Jan Friso Groote[1]

Department of Software Technology, CWI

P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

email: jfg@cwi.nl

Faron Moller

Laboratory for Foundations of Computer Science

James Clerk Maxwell Building, University of Edinburgh,

Edinburgh EH9 3JZ, Scotland

email: fm@lfcs.ed.ac.uk

Abstract

Recently, Milner and Moller have presented several decomposition results for processes. Inspired by these, we investigate decomposition techniques for the verification of parallel systems. In particular, we consider those of the form

$$\|_{i=1}^{n} p_i \; = \; \|_{j=1}^{m} q_j \tag{I}$$

where p_i and q_j are (finite) state systems. We provide a decomposition procedure for all p_i and q_j and give criteria that must be checked on the decomposed processes to see whether (I) does or does not hold. We analyse the complexity of our procedure and show that it is polynomial in n, m and the sizes of p_i and q_j if there is no communication. We also show that with communication the verification of (I) is co-NP hard, which makes it very unlikely that a polynomial complexity bound exists. But by applying our decomposition technique to Milner's cyclic scheduler we show that verification can become polynomial in space and time for practical examples, where standard techniques are exponential.

Note: The authors are supported by the European Communities under ESPRIT Basic Research Action 3006 (CONCUR).

[1]The first author's current affiliation is University of Utrecht, Department of Philosophy, P.O.Box 80126, 3508 TC Utrecht, email jfg@phil.ruu.nl. A full version of this paper has also appeared as technical report ECS-LFCS-92-193, Department of Computer Science, University of Edinburgh.

1 Introduction

Most common techniques for the automated verification of parallel systems are based on some kind of state-space exploration. Contemporary computer technology limits exploration to state spaces of about 10^7 states. However, state spaces of most parallel systems are substantially larger.

This problem is identified by many researchers, and various solutions have been proposed. For instance one may apply minimisation techniques when constructing state spaces [2], one may represent the state space using hash techniques [11], or one may restrict the state space using some additional information [7]. A more successful approach seems to be the smart encoding of state spaces, employing the regularity that is often present in the state spaces of parallel systems. In particular, the results based on binary decision diagrams (BDD's) seem more than promising [3]. An argument that one could raise against BDD's is that it is not directly based on notions inherent to processes, such as amount of communication, the structure of processes or the structure of communication, etc. This may obscure the true causes of the success of BDD's, and it may hinder further developments and a proper understanding of applicability.

Recently, some interesting decomposition results have emerged in process theory [16, 17]. Inspired by these results, we study whether decomposition techniques can be applied in order to obtain alternative means for the verification of parallel systems. Basically, the idea is as follows: Consider processes $p = \|_{i=1}^{n} p_i$ and $q = \|_{j=1}^{m} q_j$. We want to establish whether $p = q$ where '=' represents some reasonable process equivalence. In order to do so, we decompose each p_i into $p_{i1} \ldots p_{im}$ and each q_j into $q_{j1} \ldots q_{jn}$ according to some particular decomposition rules. Then we must verify whether $p_{ij} = q_{ji}$ for all i and j. The method is beneficial if the combination of performing the decompositions of the p_i's and q_j's along with checking each $p_{ij} = q_{ji}$ is considerably more efficient than checking $p = q$ directly. We show that this is indeed so in particular cases, but we show also that it is very unlikely to be true in general.

This paper first presents the decomposition scheme (after some preliminaries). Then we analyse what we have actually gained. It turns out that when there is no communication, verification via decomposition has a polynomial time and space complexity in the number and size of the processes p_i and q_j. In the case where communication is allowed, we provide a straightforward proof that verification is co-NP hard even in the case where the p_i's and q_j's are finite and determinate. More results of this kind can be found in [18]. Hence, polynomial verification is rather unlikely in this case.

In order to understand whether this intractability result rules out application of our techniques, we consider an example. This is Milner's scheduler [14], which is generally used as a benchmark for verification tools [6, 10, 12], due to its simple description, and its exponentially growing state spaces that it generates (in the number of 'cyclers' from which the scheduler is constructed). Verification via decomposition uses only polynomial time and linear space. The largest intermediate state space that is used in the verification has size $3k$ where k is the number of cyclers in the scheduler.

Our conclusions from the complexity analysis is that decomposition can indeed be a good technique for the verification of parallel systems. When there is little communication, i.e. in the case where the system has been adequately structured, the benefits of this technique may be especially high.

2 Preliminaries

In this paper we do not employ a particular process language. Rather, it turns out to be handy to work in a setting where processes are viewed as (possibly infinite) transition systems.

Definition 2.1. A *transition system* (*TS*) $p = (S_p, \alpha_p, \longrightarrow_p, s_p)$ is a four tuple, where

- S_p is a non-empty set of *states*;
- α_p is a set of *actions*;
- $\longrightarrow_p \subseteq S_p \times \alpha_p \times S_p$ is a *transition relation*; and
- $s_p \in S_p$ is the *initial state* of the transition system.

We use p, q, r to range over transition systems, and α to range over sets of actions. Elements (t, a, t') of a transition relation $\longrightarrow_p$ are often written as $t \xrightarrow{a}_p t'$. We also write $t \xrightarrow{a_1 \cdots a_n}_p t'$ for $t \xrightarrow{a_1}_p \cdots \xrightarrow{a_n}_p t'$. A function α gives the set of actions of a transition system, e.g. $\alpha\Big((S_p, \alpha_p, \longrightarrow_p, s_p)\Big) = \alpha_p$. The TS p is *finite-state* if S_p is finite, and it is *finite* if there is no infinite sequence $t_1 \xrightarrow{a_1}_p t_2 \xrightarrow{a_2}_p \cdots \xrightarrow{a_{i-1}}_p t_i \xrightarrow{a_i}_p t_{i+1} \cdots$.

Definition 2.2. A TS $p = (S, \alpha, \rightarrow, s)$ is called *determinate* with respect to some equivalence relation $\sim$ iff for all $t \in S$ and $a \in \alpha$: $t \xrightarrow{a} t_1$ and $t \xrightarrow{a} t_2$ implies $t_1 \sim t_2$. In general it will be clear which equivalence relation is meant, in which case we will simply say that p is determinate.

Definition 2.3. Let α be a set of actions. We have the following 'standard' transition systems.

- The *willing* process on α is the process that can always do an action from α:
$$W_\alpha \stackrel{\text{def}}{=} \Big(\{s\}, \alpha, \longrightarrow, s\Big) \qquad \text{where} \qquad \longrightarrow = \Big\{\langle s, a, s\rangle \mid a \in \alpha\Big\}.$$
- The *nil* process is not willing to do anything: $nil \stackrel{\text{def}}{=} W_\emptyset$.

Definition 2.4. Let $p = (S_p, \alpha_p, \longrightarrow_p, s_p)$ and $q = (S_q, \alpha_q, \longrightarrow_q, s_q)$ be TS's. We can define the following useful operations on TS's.

- For an action a the *a-prefix* of p is the TS
$$a{:}p \stackrel{\text{def}}{=} \Big(S_p \cup \{s\}, \alpha_p \cup \{a\}, \longrightarrow_p \cup \{\langle s, a, s_p\rangle\}, s\Big) \qquad \text{for } s \notin S_p.$$
- Assuming (without loss of generality) that $S_p \cap S_q = \emptyset$, the *sum* or *choice* of p and q is the TS
$$p + q \stackrel{\text{def}}{=} \Big(S_p \cup S_q \cup \{s_{p+q}\}, \alpha_p \cup \alpha_q, \longrightarrow_{p+q}, s_{p+q}\Big) \qquad \text{for } s_{p+q} \notin S_p \cup S_q,$$
where
$$\longrightarrow_{p+q} = \longrightarrow_p \;\cup\; \longrightarrow_q \;\cup \Big\{\langle s_{p+q}, a, s'\rangle \mid s_p \xrightarrow{a}_p s' \text{ or } s_q \xrightarrow{a} s'\Big\}.$$
- The *parallel composition* or *synchronisation merge* of p and q is the TS
$$p \parallel q \stackrel{\text{def}}{=} \Big(S_p \times S_q, \alpha_p \cup \alpha_q, \longrightarrow_{p\parallel q}, \langle s_p, s_q\rangle\Big)$$
where
$$\langle s_1, s_2\rangle \xrightarrow{a}_{p\parallel q} \langle s_1', s_2'\rangle \quad \text{iff} \quad \begin{cases} s_1 \xrightarrow{a}_p s_1' \text{ and } s_2 \xrightarrow{a}_q s_2', & \text{or} \\ s_1 \xrightarrow{a}_p s_1',\ s_2 = s_2' \text{ and } a \notin \alpha_q, & \text{or} \\ s_2 \xrightarrow{a}_q s_2',\ s_1 = s_1' \text{ and } a \notin \alpha_p. \end{cases}$$

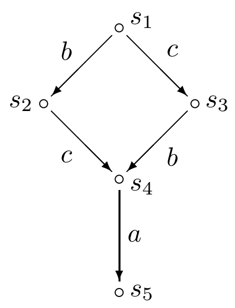


Figure 1: The process $p \; = \; b{:}a{:}nil \parallel c{:}a{:}nil$

The synchronisation merge thus forces common actions to synchronise. We write $\|_{i=1}^{n} p_i$ for $p_1 \parallel \ldots \parallel p_n$ and $\|_{i=1,i\neq k}^{n}$ for $p_1 \parallel \ldots \parallel p_{k-1} \parallel p_{k+1} \parallel \ldots \parallel p_n$. It is clear from the definition that the associativity of the composition operator is immaterial.

- Let α_1, α_2 be two sets of actions. The (α_1, α_2)-*projection* of p is the TS

  $$\restriction_{\alpha_2}^{\alpha_1}(p) \;\overset{\text{def}}{=}\; \Big(S_p, \alpha_2 \cap \alpha_p, \xrightarrow{a}_{\restriction_{\alpha_2}^{\alpha_1}(p)}, s_p\Big)$$

  where

  $$s \xrightarrow{a}_{\restriction_{\alpha_2}^{\alpha_1}(p)} s' \quad \text{iff} \quad \begin{cases} s \overset{b_1\cdots b_n a}{\longrightarrow}_p s' \text{ with } b_i \notin \alpha_2 \;\&\; a \in \alpha_1 \cap \alpha_2, & \text{or} \\ s \xrightarrow{a}_p s' \text{ for } a \in \alpha_2. \end{cases}$$

  The projection operator $\restriction$ is also used for traces: $(a_1 \cdots a_n) \restriction_\alpha$ is the trace $a_1 \cdots a_n$ from which the actions $a_i \notin \alpha$ are removed.

Remark 2.5. The projection operator $\restriction_{\alpha_2}^{\alpha_1}$ has, as far as we know, not appeared in the literature. In this article, it is solely introduced for the purpose of defining the decompositions. For an idea how this operator works, consider the process p, given by the diagram in figure 1. This represents a transition system with actions a, b and c, states s_1, s_2, s_3, s_4 and s_5, initial state s_1, and a transition relation as suggested by the arrows. Clearly p is the result of composing $p_1 = b{:}a{:}nil$ and $p_2 = c{:}a{:}nil$ in parallel. Using the projection operator $\restriction_{\alpha_2}^{\alpha_1}$ we can project p onto its parallel components, where α_1 contains those actions through which the components communicate and α_2 contains all the actions of that component. That is,

$$p_1 \;=\restriction_{\{a,b\}}^{\{a\}}(p) \qquad \text{and} \qquad p_2 \;=\restriction_{\{a,c\}}^{\{a\}}(p).$$

In the composition, the actions a and b appear in p_1, a and c appear in p_2, and a is the action through which p_1 and p_2 communicate. Note that when calculating p_1 and p_2, the possibility of extending actions backwards is essentially used. Also note that if we take $\alpha_1 = \emptyset$, then the projection operator $\restriction_{\alpha_2}^{\emptyset}(p)$ behaves as the encapsulation operator $\partial_{\alpha(p)\setminus\alpha_2}(p)$ from ACP [1] and the restriction operator $p\backslash\Big(\alpha(p) \setminus \alpha_2\Big)$ from CCS [15].

Remark 2.6. We now have three ways of specifying transition systems. We can describe them explicitly, we can write them down algebraically using the operators that have just been introduced, or we can draw a diagram such as in figure 1. In this paper, we also specify transition systems by simple recursive equations containing only choice, action prefix and a single variable. A construction that is sufficient for the examples in this paper is the following. Consider an equation

$$X = e(X) \tag{1}$$

where e consists of action prefixes and choices only. Define the self-loop TS

$$r = \Big(\{s\}, \{\star\}, \{(s, \star, s)\}, s\Big)$$

where $\star \notin \alpha\Big(e(nil)\Big)$. Construct the TS $e(r) = (S, \alpha, \longrightarrow, t)$. The TS defined by (1) is then the TS $p = (S, \alpha \setminus \{\star\}, \longrightarrow_p, t)$ where

$$\longrightarrow_p = \Big(\longrightarrow \cap \big(S \times \alpha\big(e(nil)\big) \times S\big)\Big) \cup \Big\{\langle t_1, a, t_2\rangle \mid t_1 \xrightarrow{\star} t_1 \text{ and } t \xrightarrow{a} t_2\Big\}.$$

For the examples in this paper, this definition coincides with the generally accepted interpretation of equations.

Remark 2.7. We can give operational characterisations of the above operators. We do not go into this any further except to list them as follows, and refer the interested yet uninitiated reader to e.g. [9] for understanding in interpreting these.

$$a{:}p \xrightarrow{a} p \qquad\qquad W_{\alpha\cup\{a\}} \xrightarrow{a} W_{\alpha\cup\{a\}}$$

$$\frac{p \xrightarrow{a} p'}{p + q \xrightarrow{a} p'} \qquad\qquad \frac{q \xrightarrow{a} q'}{p + q \xrightarrow{a} q'}$$

$$\frac{p \xrightarrow{a} p'}{p \parallel q \xrightarrow{a} p' \parallel q}\Big(a \notin \alpha(q)\Big) \qquad \frac{q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p \parallel q'}\Big(a \notin \alpha(p)\Big) \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{a} q'}{p \parallel q \xrightarrow{a} p' \parallel q'}$$

$$\frac{p \xrightarrow{a} p'}{\upharpoonright^{\alpha_1}_{\alpha_2}(p) \xrightarrow{a} \upharpoonright^{\alpha_1}_{\alpha_2}(p')}\Big(a \in \alpha_2\Big) \qquad \frac{p \xrightarrow{b} p' \quad \upharpoonright^{\alpha_1}_{\alpha_2}(p') \xrightarrow{a} p''}{\upharpoonright^{\alpha_1}_{\alpha_2}(p) \xrightarrow{a} p''}\Big(a \in \alpha_1, b \notin \alpha_2\Big)$$

3 Basic axioms

We will prove our results using axioms for $\parallel$, $\upharpoonright$ and W only. In this section we introduce these. The axioms hold in strong bisimulation semantics, and therefore in most other reasonable semantics as well.

Definition 3.1. Let $p = (S_p, \alpha_p, \longrightarrow_p, s_p)$ and $q = (S_q, \alpha_q, \longrightarrow_q, s_q)$ be TS's. We call a relation $\mathcal{R} \subseteq S_p \times S_q$ a (p, q)-*bisimulation relation* iff $t\mathcal{R}u$ implies

1. if $t \xrightarrow{a}_p t'$ then $u \xrightarrow{a}_q u'$ for some $u' \in S_q$ with $t'\mathcal{R}u'$; and
2. if $u \xrightarrow{a}_q u'$ then $t \xrightarrow{a}_p t'$ for some $t' \in S_p$ with $t'\mathcal{R}u'$.

Two states $t \in S_p$ and $u \in S_q$ are (p, q)-*bisimilar*, written $t \leftrightarrow_{p,q} u$, iff there is a (p, q)-bisimulation relation $\mathcal{R}$ relating t and u. We abbreviate $\leftrightarrow_{p,p}$ by $\leftrightarrow_p$. The two TS's p and q are bisimilar, written $p \leftrightarrow q$, if $\alpha(p) = \alpha(q)$ and $s_p \leftrightarrow_{p,q} s_q$.

Lemma 3.2 *(Congruence). $\leftrightarrow$ is a congruence with respect to action prefix, choice, parallel composition and (α_1, α_2)-projection.*

The axioms that we use are presented in table 1. We do not strive for completeness of the axiomatisation. Rather, the axioms need only be sufficiently complete to satisfy our goal.

Lemma 3.3 *(Soundness). The axioms in table 1 are sound with respect to $\leftrightarrow$.*

Example 3.4. The following examples show why the conditions in R_4, R_5 and R_6 of the last theorem are necessary. For the condition in R_4, observe that

$\Vert_1$	$p \parallel (q \parallel r) \;=\; (p \parallel q) \parallel r$	
$\Vert_2$	$p \parallel q \;=\; q \parallel p$	
R_1	$p \;= \upharpoonright^{\alpha}_{\alpha(p)} (p)$	
R_2	$\upharpoonright^{\alpha_1}_{\alpha_2} (p) \;= \upharpoonright^{\alpha_1}_{\alpha_2 \cap \alpha(p)} (p)$	
R_3	$\upharpoonright^{\alpha_1}_{\alpha_2} (p) \;= \upharpoonright^{\alpha_1 \cap \alpha_2}_{\alpha_2} (p)$	
R_4	$\upharpoonright^{\alpha_1}_{\alpha_2} (p) \;= \upharpoonright^{\alpha_1}_{\alpha_2} \left(\upharpoonright^{\alpha_1 \cup \alpha}_{\alpha_2 \cup \alpha} (p)\right)$	if $\alpha_2 \cap \alpha \;=\; \emptyset$
R_5	$\upharpoonright^{\alpha_1}_{\alpha_2} (p \parallel q) \;= \upharpoonright^{\alpha_1}_{\alpha_2} (p) \parallel \upharpoonright^{\alpha_1}_{\alpha_2} (q)$	if $\alpha_1 \subseteq \alpha(p) \cap \alpha(q) \subseteq \alpha_2$
R_6	$p \;=\; p \parallel \upharpoonright^{\alpha}_{\alpha} (p)$	if $\upharpoonright^{\alpha}_{\alpha} (p)$ is determinate
R_7	$\upharpoonright^{\alpha}_{\emptyset} (p) \;=\; nil$	
W_1	$p \parallel W_{\alpha(p)} \;=\; p$	
W_2	$W_{\alpha_1} \parallel W_{\alpha_2} \;=\; W_{\alpha_1 \cup \alpha_2}$	
W_3	$\upharpoonright^{\alpha_1}_{\alpha_2} (W_\alpha) \;=\; W_{\alpha_2 \cap \alpha}$	

Table 1: Basic axioms for operators

$$\upharpoonright^{\emptyset}_{\{b\}} (a{:}b{:}nil) \;\underline{\leftrightarrow}\; nil_b \qquad \text{whereas} \qquad \upharpoonright^{\emptyset}_{\{b\}} \left(\upharpoonright^{\{b\}}_{\{b\}} (a{:}b{:}nil)\right) \;\underline{\leftrightarrow}\; b{:}nil.$$

By nil_b, we mean the TS *nil* with alphabet $\{b\}$, which can be defined by $\upharpoonright^{\emptyset}_{\{b\}} (a{:}b{:}nil)$. For the first condition in R_5, observe that

$$\upharpoonright^{\{c\}}_{\{b,c\}} \Big((a{:}nil + b{:}nil) \parallel c{:}nil\Big) \;\underline{\leftrightarrow}\; b{:}nil \parallel c{:}nil \;+\; c{:}nil \qquad \text{whereas}$$
$$\upharpoonright^{\{c\}}_{\{b,c\}} (a{:}nil + b{:}nil) \;\parallel\; \upharpoonright^{\{c\}}_{\{b,c\}} (c{:}nil) \;\underline{\leftrightarrow}\; b{:}nil \parallel c{:}nil.$$

For the second condition in R_5, observe that

$$\upharpoonright^{\{a\}}_{\{a\}} \Big(b{:}a{:}nil \parallel (a{:}nil + b{:}nil)\Big) \;\underline{\leftrightarrow}\; nil_a \qquad \text{whereas}$$
$$\upharpoonright^{\{a\}}_{\{a\}} (b{:}a{:}nil) \;\parallel\; \upharpoonright^{\{a\}}_{\{a\}} (a{:}nil + b{:}nil) \;\underline{\leftrightarrow}\; a{:}a{:}nil.$$

Finally, for the condition in R_6, observe that for $p = a{:}b{:}a{:}nil + a{:}b{:}b{:}nil$,

$$p \;\parallel\; \upharpoonright^{\{a\}}_{\{a\}} (p) \;\underline{\leftrightarrow}\; p \;+\; a{:}b{:}nil.$$

4 Verification via decomposition

In this section we formulate our main result which explains how the verification of an equation $p = q$ with $p = \Vert_{i=1}^{n} p_i$ and $q = \Vert_{j=1}^{m} q_j$ can be performed via decomposition. In theorem 4.4 we describe the decomposition and we give some conditions that must be checked in order for the method to be applicable. In the theorem, we use p and q on both the left and right hand sides, so that nothing is apparently gained by applying the theorem. However in remark 4.6 we show how p and q can be eliminated from the right hand side.

We begin with some straightforward lemmata that are used in the proofs to follow.

Lemma 4.1. *If $\alpha \subseteq \alpha(p)$, then $p = p \parallel W_\alpha$. In particular, $p = p \parallel nil$.*

Proof. $p \overset{W_1}{=} p \parallel W_{\alpha(p)} \overset{W_2}{=} p \parallel W_{\alpha(p)} \parallel W_\alpha \overset{W_1}{=} p \parallel W_\alpha$. □

Lemma 4.2. *Assume that $p = p_1 \parallel p_2$. If $\alpha \subseteq \alpha(p)$ and $\restriction^\alpha_\alpha (p_2 \parallel W_\alpha)$ is determinate, then $p = p \parallel \restriction^\alpha_\alpha (p_2 \parallel W_\alpha)$.*

Proof.

$$\begin{aligned} p &\overset{lemma\ 4.1}{=} p_1 \parallel p_2 \parallel W_\alpha \\ &\overset{R_6}{=} p_1 \parallel p_2 \parallel W_\alpha \parallel \restriction^\alpha_\alpha \big(p_2 \parallel W_\alpha\big) \\ &\overset{lemma\ 4.1}{=} p \parallel \restriction^\alpha_\alpha (p_2 \parallel W_\alpha) \end{aligned}$$

□

Lemma 4.3. *If $\alpha \cap \beta = \emptyset$ and $\restriction^\alpha_\alpha (p)$ is determinate, then $\restriction^\alpha_{\alpha\cup\beta} (p) \parallel \restriction^\alpha_\alpha (p) = \restriction^\alpha_{\alpha\cup\beta} (p)$.*

Proof.

$$\begin{aligned} \restriction^\alpha_{\alpha\cup\beta} (p) &\overset{R_2,R_3,R_6}{=} \restriction^{\alpha\cap\alpha(p)}_{(\alpha\cup\beta)\cap\alpha(p)} \Big(p \parallel \restriction^{\alpha\cap\alpha(p)}_{\alpha\cap\alpha(p)} (p)\Big) \\ &\overset{R_5}{=} \restriction^{\alpha\cap\alpha(p)}_{(\alpha\cup\beta)\cap\alpha(p)} (p) \parallel \restriction^{\alpha\cap\alpha(p)}_{(\alpha\cup\beta)\cap\alpha(p)} \Big(\restriction^{\alpha\cap\alpha(p)}_{\alpha\cap\alpha(p)} (p)\Big) \\ &\overset{R_1,R_2,R_3,R_4}{=} \restriction^\alpha_{\alpha\cup\beta} (p) \parallel \restriction^\alpha_\alpha (p) \end{aligned}$$

□

Theorem 4.4 *(Verification via decomposition)***.** *Let $p = \|_{i=1}^n p_i$ and $q = \|_{j=1}^m q_j$. Let α consist of the synchronous (communicating) actions of p and q. That is,*

$$\alpha \overset{\text{def}}{=} \bigcup_{1\le i<j\le n} \Big(\alpha(p_i) \cap \alpha(p_j)\Big) \ \cup\ \bigcup_{1\le i<j\le m} \Big(\alpha(q_i) \cap \alpha(q_j)\Big).$$

Assume that $\restriction^\alpha_\alpha (p_i \parallel W_\alpha)$ and $\restriction^\alpha_\alpha (q_j \parallel W_\alpha)$ are determinate for all $1 \le i \le n$ and $1 \le j \le m$. Then

$$p = q \qquad \textit{iff} \qquad \begin{cases} p_{ij} = q_{ji} & \text{for } 1 \le i \le n,\ 1 \le j \le m, \\ \restriction^\alpha_{\alpha\cup\alpha(p_i)} (p) = \displaystyle\mathop{\|}_{j=1}^{m} p_{ij} & \text{for } 1 \le i \le n, \text{ and} \\ \restriction^\alpha_{\alpha\cup\alpha(q_j)} (q) = \displaystyle\mathop{\|}_{i=1}^{n} q_{ji} & \text{for } 1 \le j \le m, \end{cases}$$

where

$$p_{ij} \overset{\text{def}}{=} \restriction^\alpha_{\alpha\,\cup\,\big(\alpha(p_i)\cap\alpha(q_j)\big)} (p) \qquad \text{and} \qquad q_{ji} \overset{\text{def}}{=} \restriction^\alpha_{\alpha\,\cup\,\big(\alpha(p_i)\cap\alpha(q_j)\big)} (q)$$

Proof.

($\Leftarrow$) For each $1 \leq i \leq n$ we can prove that:

$$p \overset{lemma\ 4.2}{=} p \parallel \mathop{\parallel}_{j=1, j\neq i}^{n} \upharpoonright_{\alpha}^{\alpha} (p_j \parallel W_\alpha).$$

By repeating this process for all i we get

$$\begin{aligned}
p &\overset{lemmas\ 4.2,4.1}{=} \Big(p \parallel W_\alpha\Big) \parallel \Big(\mathop{\parallel}_{i=1}^{n} \mathop{\parallel}_{j=1,j\neq i}^{n} \big(\upharpoonright_{\alpha}^{\alpha} (p_j \parallel W_\alpha)\big)\Big) \\
&\overset{\parallel_1,\parallel_2,W_2}{=} \mathop{\parallel}_{i=1}^{n} \Big(p_i \parallel W_\alpha\Big) \parallel \Big(\mathop{\parallel}_{i=1}^{n} \mathop{\parallel}_{j=1,j\neq i}^{n} \big(\upharpoonright_{\alpha}^{\alpha} (p_j \parallel W_\alpha)\big)\Big) \\
&\overset{\parallel_1,\parallel_2}{=} \mathop{\parallel}_{i=1}^{n} \Big(\Big(p_i \parallel W_\alpha\Big) \parallel \Big(\mathop{\parallel}_{j=1,j\neq i} \upharpoonright_{\alpha}^{\alpha} (p_j \parallel W_\alpha)\Big)\Big) \\
&\overset{R_1,R_2}{=} \mathop{\parallel}_{i=1}^{n} \Big(\upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (p_i \parallel W_\alpha) \parallel \Big(\mathop{\parallel}_{j=1,j\neq i}^{n} \upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (p_j \parallel W_\alpha)\Big)\Big) \\
&\overset{\parallel_1,\parallel_2}{=} \mathop{\parallel}_{i=1}^{n} \mathop{\parallel}_{j=1}^{n} \Big(\upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (p_j \parallel W_\alpha)\Big) \\
&\overset{R_5}{=} \mathop{\parallel}_{i=1}^{n} \Big(\upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} \Big(\mathop{\parallel}_{j=1}^{n} (p_j \parallel W_\alpha)\Big)\Big) \\
&\overset{lemma\ 4.1}{=} \mathop{\parallel}_{i=1}^{n} \Big(\upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (p)\Big) \\
&\overset{assumption}{=} \mathop{\parallel}_{i=1}^{n} \mathop{\parallel}_{j=1}^{m} p_{ij}
\end{aligned}$$

In the same way, we can deduce that $q = \parallel_{j=1}^{m} \parallel_{i=1}^{n} q_{ij}$. Hence from the assumption that $p_{ij} = q_{ji}$ for each $1 \leq i \leq n$ and $1 \leq j \leq m$, we can deduce that $p = q$.

($\Rightarrow$) First it is clear that $p = q$ immediately implies that $p_{ij} = q_{ji}$. So we now prove that $p = q$ implies the second condition of the theorem. For each $1 \leq i \leq n$ we can compute the following.

$$\begin{aligned}
\mathop{\parallel}_{j=1}^{m} p_{ij} &= \mathop{\parallel}_{j=1}^{m} \Big(\upharpoonright_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\big)}^{\alpha} (p)\Big) \\
&\overset{lemma\ 4.1}{=} \mathop{\parallel}_{j=1}^{m} \Big(\upharpoonright_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\big)}^{\alpha} (q \parallel W_\alpha)\Big) \\
&= \mathop{\parallel}_{j=1}^{m} \Big(\upharpoonright_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\big)}^{\alpha} \big(\mathop{\parallel}_{k=1}^{m} (q_k \parallel W_\alpha)\big)\Big) \\
&\overset{R_5}{=} \mathop{\parallel}_{j=1}^{m} \mathop{\parallel}_{k=1}^{m} \Big(\upharpoonright_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\big)}^{\alpha} (q_k \parallel W_\alpha)\Big) \\
&\overset{R_2}{=} \mathop{\parallel}_{j=1}^{m} \mathop{\parallel}_{k=1}^{m} \Big(\upharpoonright_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\cap\alpha(q_k)\big)}^{\alpha} (q_k \parallel W_\alpha)\Big) \\
&\overset{R_2}{=} \mathop{\parallel}_{j=1}^{m} \Big(\mathop{\parallel}_{k=1,k\neq j}^{m} \upharpoonright_{\alpha}^{\alpha} (q_k \parallel W_\alpha)\Big) \parallel \upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (q_j \parallel W_\alpha) \\
&\overset{lemma\ 4.3}{=} \mathop{\parallel}_{j=1}^{m} \Big(\upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (q_j \parallel W_\alpha)\Big) \\
&\overset{R_5}{=} \upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} \Big(\mathop{\parallel}_{j=1}^{m} (q_j \parallel W_\alpha)\Big) \\
&\overset{lemma\ 4.1}{=} \upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (q) \\
&= \upharpoonright_{\alpha\cup\alpha(p_i)}^{\alpha} (p)
\end{aligned}$$

Finally, the third condition can be deduced in the same way. $\square$

Remark 4.5. One may wonder whether it is enough simply to check $p_{ij} = q_{ji}$ in theorem 4.4. This would be a substantial optimisation. Unfortunately, this is not possible, as shown by the following example. Consider $p = (a{:}nil + b{:}nil) \parallel c{:}nil$ and $q = a{:}nil \parallel (b{:}nil + c{:}nil)$. One may try to verify that $p = q$ by applying theorem 4.4. In this case $\alpha = \emptyset$, so the determinacy constraints are easily satisfied. Calculating each p_{ij} and q_{ji} yields the following.

$$\begin{array}{lllll} p_{11} = q_{11} = a{:}nil & \qquad & p_{21} = q_{12} = b{:}nil \\ p_{12} = q_{21} = nil & & p_{22} = q_{22} = c{:}nil \end{array}$$

So clearly $p_{ij} = q_{ji}$ for all i and j, but $p \neq q$.

Remark 4.6. The right hand side of theorem 4.4 can be calculated using the following observations.

$$\begin{array}{lcl} p_{ij} & = & \restriction^{\alpha}_{\alpha\cup\left(\alpha(p_i)\cap\alpha(q_j)\right)} \left(\mathop{\Big\|}_{k=1}^{n} p_k\right) \\ & \overset{lemma\ 4.1,\ R_5}{=} & \mathop{\Big\|}_{k=1}^{n} \left(\restriction^{\alpha}_{\alpha\cup\left(\alpha(p_i)\cap\alpha(q_j)\right)} (p_k \parallel W_\alpha)\right). \end{array}$$

We can calculate $\restriction^{\alpha}_{\alpha\cup\alpha(p_i)} (p)$ using the following:

$$\begin{array}{lcl} \restriction^{\alpha}_{\alpha\cup\alpha(p_i)} (p) & = & \restriction^{\alpha}_{\alpha\cup\alpha(p_i)} \left(\mathop{\Big\|}_{j=1}^{n} (p_j \parallel W_\alpha)\right) \\ & = & \mathop{\Big\|}_{j=1}^{n} \left(\restriction^{\alpha}_{\alpha\cup\alpha(p_i)} (p_j \parallel W_\alpha)\right). \end{array}$$

Of course this also applies to q_{ji} and $\restriction^{\alpha}_{\alpha\cup\alpha(q_j)} (q)$.

In section 6 we give an application of the above technique which takes advantage of the preceding remark. However we first analyse the verification problem to demonstrate the benefit of the technique.

5 On the complexity of verification by decomposition

In this section we consider the complexity of verification through decomposition. We do this in the setting of bisimulation equivalence, as the verification of trace based equivalences is generally intractable on finite state systems [13]. We show that in the case where there is no communication between the components, the verification is polynomial. In the case where there is communication between the components, we show that the verification is co-NP hard, and hence inherently intractable. The proof that we give is a simplified variant of those given in [18]. From these observations we draw the conclusion that verification via decomposition is especially worthwhile when there are relatively many asynchronous or non-communicating actions, and that its use is rather limited if almost every action is used for communication. But it is exactly the former case that leads to enormous state graphs, while in the latter case state graphs remain relatively small, and therefore, they can be more readily handled by existing means.

We start out by reformulating theorem 4.4, but now with the restriction that there are no communication actions among the component processes, which means that $\alpha = \emptyset$. For convenience, we write $\restriction_\beta$ for $\restriction^{\emptyset}_{\beta}$.

Corollary 5.1. *Let $p = \|_{i=1}^{n} p_i$ and $q = \|_{j=1}^{m} q_j$ with $\alpha(p_i) \cap \alpha(p_j) = \emptyset$ for all $1 \leq i < j \leq n$ and $\alpha(q_i) \cap \alpha(q_j) = \emptyset$ for all $1 \leq i < j \leq m$. Then*

Equality	Time complexity Space complexity
$p_{ij} = q_{ji} \qquad (1 \le i \le n,\ 1 \le j \le m)$	$O\Big(m\,n\,\big(\max_{i,j}(\mid \longrightarrow_{p_i} \mid + \mid \longrightarrow_{q_j} \mid)\big) \log\big(\max_{i,j}(\mid S_{p_i}\mid + \mid S_{q_j}\mid)\big)\Big)$ $O\Big(\max_{i,j}\big(\mid \longrightarrow_{p_i} \mid + \mid \longrightarrow_{q_j} \mid\big)\Big)$
$p_i = \Big\Vert_{j=1}^{m} p_{ij} \qquad (1 \le i \le n)$	$O\Big(m\,n\,\max_i \mid \longrightarrow_{p_i} \mid \log(\max_i \mid S_{p_i}\mid)\Big)$ $O\Big(\max_i \mid \longrightarrow_{p_i} \mid\Big)$
$q_i = \Big\Vert_{i=1}^{n} q_{ji} \qquad (1 \le j \le m)$	$O\Big(m\,n\,\max_j \mid \longrightarrow_{q_j} \mid \log(\max_j \mid S_{q_j}\mid)\Big)$ $O\Big(\max_j \mid \longrightarrow_{q_j} \mid\Big)$
$p = q$	$O\Big(m\,n\,\big(\max_{i,j}(\mid \longrightarrow_{p_i} \mid + \mid \longrightarrow_{q_j} \mid)\big) \log\big(\max_{i,j}(\mid S_{p_i}\mid + \mid S_{q_j}\mid)\big)\Big)$ $O\Big(\max_{i,j}(\mid \longrightarrow_{p_i} \mid + \mid \longrightarrow_{q_j} \mid)\Big)$

Table 2: Complexities of deciding bisimulation in non-communicating processes

$$p = q \qquad \textit{iff} \qquad \begin{cases} p_{ij} = q_{ji} & \textit{for } 1 \le i \le n,\ 1 \le j \le m, \\ p_i = \Big\Vert_{j=1}^{m} p_{ij} & \textit{for } 1 \le i \le n, \text{ and} \\ q_j = \Big\Vert_{i=1}^{n} q_{ji} & \textit{for } 1 \le j \le m, \end{cases}$$

where

$$p_{ij} \overset{\text{def}}{=} \upharpoonleft_{\alpha(q_j)} (p_i) \qquad \textit{and} \qquad q_{ji} \overset{\text{def}}{=} \upharpoonleft_{\alpha(p_i)} (q_j)$$

Proof. From R_1, R_2, R_7, lemma 4.1 and remark 4.6, we can show that $p_i = \upharpoonleft_{\alpha(p_i)} (p)$ and $q_j = \upharpoonleft_{\alpha(q_j)} (q)$, and from R_2, R_7, lemma 4.1 and remark 4.6, we can show that $\upharpoonleft_{\alpha(p_i)\cap\alpha(q_j)} (p) = \upharpoonleft_{\alpha(q_j)} (p_i)$ and $\upharpoonleft_{\alpha(p_i)\cap\alpha(q_j)} (q) = \upharpoonleft_{\alpha(p_i)} (q_j)$. The result then follows directly from theorem 4.4. □

In order to verify that $p = q$, we must check the three identities at the right hand side of the curly bracket in corollary 5.1. In table 2 we have put the complexities for each step and the complexity for the total calculation. Here, S_r and $\longrightarrow_r$ represent the sets of states and transitions, respectively, of TS r. We assume that the number of states of our TS's is smaller than the number of transitions, as it is reasonable to assume that all states are reachable. The complexities in table 2 are motivated as follows.

1. In order to calculate p_{ij}, we take p_i and remove all transitions labelled with actions not in $\alpha(q_j)$. Then we remove all unreachable states, along with their outgoing transitions. This takes $O(\mid \longrightarrow_{p_i} \mid)$ time and space. In the same way we construct q_{ji}. In order to calculate $p_{ij} = q_{ji}$, we apply a standard bisimulation algorithm [13], which takes $O\Big((\mid \longrightarrow_{p_i} \mid + \mid \longrightarrow_{q_j}$

$|)\log(|S_{p_i}|+|S_{q_j}|)\Big)$ time and $O(|\longrightarrow_{p_i}|+|\longrightarrow_{q_j}|)$ space. As this must be repeated for each $1\leq i\leq n$ and $1\leq j\leq m$, we obtain the complexities as given in table 2.1.

2. We obtain the second complexity measures via the following observation:

   Lemma 5.2. *Let $r_0=(S_{r_0},\alpha_{r_0},\longrightarrow_{r_0},s_{r_0})$ and $r_1=(S_{r_1},\alpha_{r_1},\longrightarrow_{r_1},s_{r_1})$ with $\alpha_{r_0}\cap\alpha_{r_1}=\emptyset$. For all $u,u'\in S_{r_0}$ and $v,v'\in S_{r_1}$:*

   $$u\leftrightarrows_{r_0}u' \text{ and } v\leftrightarrows_{r_1}v' \qquad \text{iff} \qquad \langle u,v\rangle\leftrightarrows_{r_0\|r_1}\langle u',v'\rangle.$$

   Proof. Straightforward. □

   Reading this lemma from right to left, it says that if $r_0\parallel r_1$ is not minimised with respect to bisimulation, i.e. it contains different states that are bisimilar, then this is due to the fact that either r_0 or r_1 was not minimal with respect to bisimulation. Reversing this reasoning says that if we ensure that r_0 and r_1 are minimal, then $r_0\parallel r_1$ will also be minimal.

   We use this observation as follows in constructing $\|_{j=1}^{m}p_{ij}$. First construct p_{i1} as indicated above. This takes $O(|\longrightarrow_{p_i}|)$ time and space. Minimise p_{i1} with respect to bisimulation, obtaining $\hat{p}_{i1}$. Using the ordinary bisimulation algorithms, this takes $O\Big(|\longrightarrow_{p_i}|\log(|S_{p_i}|)\Big)$ time and $O(|\longrightarrow_{p_i}|)$ space. Now construct p_{i2} and its minimised variant $\hat{p}_{i2}$ likewise. Then calculate $\hat{p}_{i1}\parallel\hat{p}_{i2}$, but stop if the number of states of the result exceed those of p_i. As $\hat{p}_{i1}$ and $\hat{p}_{i2}$ are minimal w.r.t. bisimulation, $\hat{p}_{i1}\parallel\hat{p}_{i2}$ is minimal. Hence if the number of states of $\hat{p}_{i1}\parallel\hat{p}_{i2}$ exceed the number of states of p_i, then p_i cannot be bisimilar to $\|_{j=1}^{m}p_{ij}$. The complexity of calculating $\hat{p}_{i1}\parallel\hat{p}_{i2}$ is therefore $O(|\longrightarrow_{p_i}|)$. We thus calculate $\|_{j=1}^{m}p_{ij}$ by stepwise adding $p_{i3},p_{i4},\ldots,p_{im}$ in the same way. This takes $O\Big(m|\longrightarrow_{p_i}|\log(|S_{p_i}|)\Big)$ time and $O(|\longrightarrow_{p_i}|)$ space. The verification of $p_i=\|_{j=1}^{m}p_{ij}$ can then be done without increasing the time and space complexities. The steps above must be repeated for each $1\leq i\leq n$. So we obtain the figures in table 2.

3. The analysis in this case is the same as in case 2, using q instead of p.

4. Combining the above gives these complexities for calculating $p\leftrightarrows q$.

The procedure sketched above is rather wasteful, e.g. p_{ij} and q_{ji} are calculated rather often. We have not investigated optimisations, as we expect that they will not improve the time and space complexities. However, the example in section 6 gives the impression that by using the regularity of processes p_i and q_j, substantial improvements can be expected.

In the case where there is communication between the processes, then the verification of $\|_{i=1}^{n}p_i=\|_{j=1}^{m}q_j$ becomes co-NP hard for each process equivalence between trace and bisimulation equivalence. We give a straightforward proof of this fact, actually showing that in the case that p_i and q_j are all finite and determinate, this verification is co-NP complete. In [18] it is shown that this verification becomes P-space hard if p_i and q_j are finite state. It also gives an EXPSPACE completeness result in case abstraction of actions is allowed.

The proof technique in this section is a straightforward reduction from 3SAT [4]: Let $x_1,\ldots,x_k$ be variables and $l_{ij}\in\{x_1,\ldots,x_k,\neg x_1,\ldots,\neg x_k\}$. The question whether

$$\bigwedge_{i=1}^{n}(l_{i1}\vee l_{i2}\vee l_{i3})$$

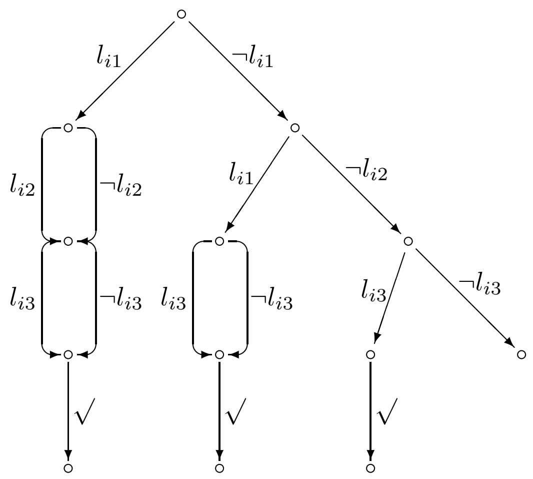
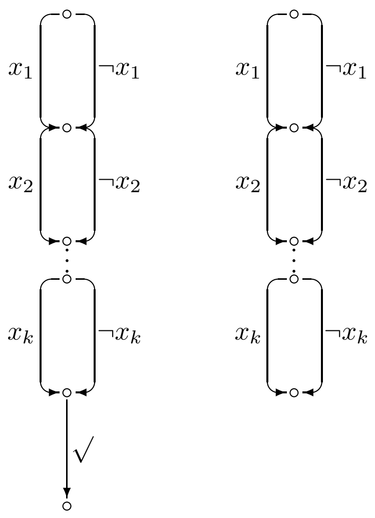


Figure 2: The processes p_i, p and p'

is satisfiable is well-known to be NP-complete. There is a straightforward polynomial way of reducing an instance of 3SAT to an instance of 3SAT such that $k_{i1} < k_{i2} < k_{i3}$ where l_{ij} refers to a variable $x_{k_{ij}}$[1]. So 3SAT with this restriction is still NP-complete.

Lemma 5.3. *Determining whether* $\|_{i=1}^{n} p_i = \|_{j=1}^{m} q_j$ *holds is co-NP complete for finite determinate* p_i *and* q_j.

Proof. First we show co-NP hardness by reducing from 3SAT with the ordering restriction to the question whether $(\|_{i=1}^{n} p_i) \parallel p = p'$, for finite determinate p_i, p and p', does not hold. Consider the following instance of 3SAT with restriction over variables $x_1, \ldots, x_k$:

$$\bigwedge_{i=1}^{n} (l_{i1} \vee l_{i2} \vee l_{i3}). \tag{2}$$

The processes p_i, p and p' are constructed as in figure 2. Process p_i has actions l_{i1}, l_{i2}, l_{i3}, $\neg l_{i1}$, $\neg l_{i2}$, $\neg l_{i3}$ and $\surd$. Here $\neg l_{ij}$ stands for $\neg x$ if $l_{ij} \equiv x$ and for x if $l_{ij} \equiv \neg x$. A step l_{ij} corresponds to considering a valuation σ that assigns true to l_{ij}, and a step $\neg l_{ij}$ corresponds to considering a valuation σ that assigns false to l_{ij}. Clearly, p_i can perform a $\surd$ step iff $\sigma(l_{i1} \vee l_{i2} \vee l_{i3})$ is true.

The process p is used to guarantee that in $(\|_{i=1}^{n} p_i) \parallel p$, first a step corresponding to x_1 must be performed, then one corresponding to x_2 etc. It has actions $x_1, \ldots, x_k$, $\neg x_1, \ldots, \neg x_k$ and $\surd$. The process p' is equal to p with the only difference being that it has no $\surd$ step at the end.

We have the following fact, from which our co-NP hardness result follows immediately.

$$\bigwedge_{i=1}^{n} (l_{i1} \vee l_{i2} \vee l_{i3}) \text{ is satisfiable} \qquad \text{iff} \qquad (\mathop{\Big\|}_{i=1}^{n} p_i) \parallel p = p' \text{ does not hold.}$$

[1]First remove all clauses $l_{i1} \vee l_{i2} \vee l_{i3}$ that contain a variable occurring both with and without negation. Next remove double occurrences of variables in the clauses. Finally, introduce two new variables x_{k+1} and x_{k+2} and add these to incomplete clauses.

Here '=' represents any equivalence between trace and bisimulation equivalence [8]. We now prove this fact:

($\Rightarrow$) Let σ be a valuation satisfying (2). Then $(\|_{i=1}^{n} p_i) \parallel p$ can perform the trace $a_1 \cdots a_k \surd$ where

$$a_i = \begin{cases} x_i & \text{if } \sigma(x_i) = true, \\ \neg x_i & \text{if } \sigma(x_i) = false. \end{cases}$$

Clearly, such a trace cannot be performed by p'. So, $(\|_{i=1}^{n} p_i) \parallel p$ and p' are not trace equivalent.

($\Leftarrow$) If $(\|_{i=1}^{n} p_i) \parallel p$ can perform a trace $a_1 \cdots a_k \surd$, then the assignment σ defined as:

$$\sigma(x_i) = \begin{cases} \mathit{true} & \text{if } a_i = x_i, \\ \mathit{false} & \text{if } a_i = \neg x_i. \end{cases}$$

is clearly a satisfying truth assignment for (2). Thus if (2) is not satisfiable, then $(\|_{i=1}^{n} p_i) \parallel p$ cannot perform traces ending in $\surd$. So exactly the traces $a_1 \cdots a_k$ with $a_i \equiv x_i$ or $a_i \equiv \neg x_i$ can be performed by both $(\|_{i=1}^{n} p_i) \parallel p$ and p', and hence they are trace equivalent. As all processes are determinate, $(\|_{i=1}^{n} p_i) \parallel p$ and p' are also bisimulation equivalent [5].

For completeness it is sufficient to guess a trace $a_1 \cdots a_k \surd$ and to check whether for each $1 \leq i \leq n$, $a_{k_{i1}}\, a_{k_i 2}\, a_{k_{i3}} \surd$ is a trace of p_i, where l_{ij} refers to a variable $x_{k_{ij}}$. This can clearly be done in polynomial time. As $a_1 \cdots a_k \surd$ is always a trace of p, it must also be a trace of $(\|_{i=1}^{n} p_i) \parallel p$, while it cannot be a trace of p'. □

It is not difficult to extend the proof above to include only two-way communication (see [18]) or to use only two actions. However this is outside the setting of this paper, and it complicates matters slightly.

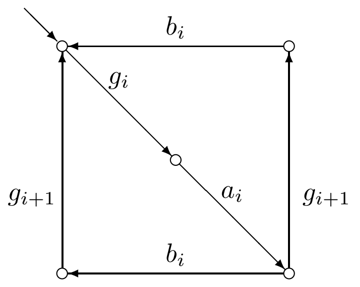
6 An application

In this section, we apply the decomposition theorem to Milner's scheduler [14], which is constructed out of simple components, called cyclers. The scheduler is often used as a benchmark for programmes which calculate process equivalences [6, 10, 12], because its state space grows exponentially with the number of cyclers. Using our decomposition technique, we can avoid this exponential blowup.

The scheduler schedules k processes in cyclic succession, so that the first process is reactivated after the kth process has been activated. However, a process must never be reactivated before it has terminated. It is constructed of k cyclers $C_0, ..., C_{k-1}$, as depicted in figure 3, where cycler C_i is dedicated to process i. The left part of the figure shows the transition system for cycler C_i, while the right part depicts the architecture of the scheduler. The dotted lines indicate where the cyclers synchronise. Cycler C_i first synchronises on a signal g_i which indicates that it may start. It then activates process i via an action a_i. Next, it waits for termination of process i, indicated by b_i, and in parallel, using g_{i+1}, activates the next cycler. Here, the indices are taken mod k, so that $g_k = g_0$. It then returns to its initial state. The cycler C_i is described by:

$$\begin{array}{lcll} C_0 & = & a_0{:}(b_0{:}g_1{:}g_0{:}C_0 \;+\; g_1{:}b_0{:}g_0{:}C_0), & \\ C_i & = & g_i{:}a_i{:}(b_i{:}g_{i+1}{:}C_i \;+\; g_{i+1}{:}b_i{:}C_i) & \text{for} \quad 1 \leq i < k. \end{array}$$

The first cycler is assumed to have been initiated. The complete scheduler for k processes is thus described by:



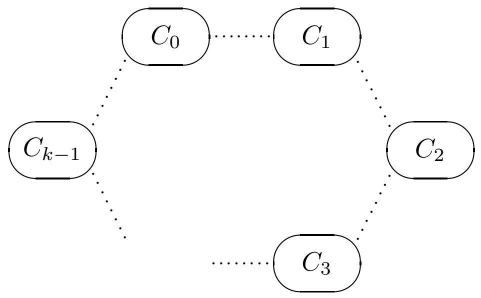


Figure 3: A cycler and a scheduler

$$Sched_k = C_0 \parallel C_1 \parallel \cdots \parallel C_{k-1}.$$

A correctness criterion for the scheduler has been formulated in [14]. The a_i and b_i actions must happen alternately, and the a_i actions must happen cyclically. For the purposes of this example, we are also interested in the precise relationship between the synchronisation actions g_i and the actions b_j. Therefore we prove the scheduler $Sched_k$ equal to the specification $Correct_k$ from which the behaviour of the scheduler can easily be understood. The process $Correct_k$ is defined by

$$Correct_k = D_0 \parallel D_1 \parallel \cdots \parallel D_{k-1} \parallel BB_k,$$

where

$$\begin{aligned} BB_k &= a_0{:}g_1{:}a_1{:}\cdots{:}g_{k-1}{:}a_{k-1}{:}g_0{:}BB_k, \\ D_0 &= a_0{:}b_0{:}g_0{:}D_0, \\ D_i &= g_i{:}a_i{:}b_i{:}D_i \qquad \text{for} \quad 1 \le i < k. \end{aligned}$$

The letters BB in BB_k stand for 'backbone'. It is easy to see that $Correct_k$ satisfies the correctness criteria as given by Milner. This can be shown formally by applying hiding, but as this is rather standard, we do not prove that here. For an idea of the proof, see the verification of the scheduler in [14].

We wish to apply theorem 4.4 to verify that $Sched_k = Correct_k$. We thus let $p_i = C_i$ for $0 \le i < k$, and define $q_j = D_j$ for $0 \le j < k$ and $q_k = BB_k$.

First note that $\alpha = \{a_i, g_i \mid 0 \le i < k\}$. A small calculation tells us that $\lceil^{\alpha}_{\alpha} (p_i \parallel W_\alpha)$ is bisimilar to $E_i \parallel W_\alpha$, where E_i is defined by

$$\begin{aligned} E_0 &= a_0{:}g_1{:}g_0{:}E_0, \\ E_i &= g_i{:}a_i{:}g_{i+1}{:}E_i \qquad \text{for} \quad 1 \le i < k, \end{aligned}$$

and that $\lceil^{\alpha}_{\alpha} (q_j \parallel W_\alpha)$ is bisimilar to $F_j \parallel W_\alpha$, where F_j is defined by

$$\begin{aligned} F_0 &= a_0{:}g_0{:}F_0, \\ F_j &= g_j{:}a_j{:}F_j \qquad \text{for} \quad 1 \le i < k, \\ F_k &= BB_k. \end{aligned}$$

Obviously these are all determinate, so theorem 4.4 is applicable. We use remark 4.6 to calculate p_{ij}, q_{ji}, $\lceil^{\alpha}_{\alpha \cup \alpha(p_i)} (p)$ and $\lceil^{\alpha}_{\alpha \cup \alpha(q_j)} (q)$. For $i \neq j$, we find that

$$p_{ij} = \mathop{\Big\|}_{l=0}^{k-1} \lceil^{\alpha}_{\alpha \cup \left(\alpha(p_i) \cap \alpha(q_j)\right)} (p_l \parallel W_\alpha)$$

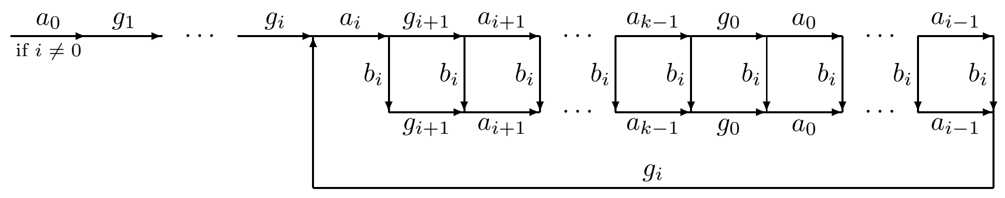
$$\begin{aligned}
&= \mathop{\Big\|}_{l=0}^{k-1} \upharpoonright^{\alpha}_{\alpha} (p_l \parallel W_\alpha) \\
&= \mathop{\Big\|}_{l=0}^{k-1} E_l \ \parallel\ W_\alpha \\
&= BB_k,
\end{aligned}$$

and

$$\begin{aligned}
q_{ji} &= \mathop{\Big\|}_{l=0}^{k} \upharpoonright^{\alpha}_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_j)\big)} (q_l \parallel W_\alpha) \\
&= \mathop{\Big\|}_{l=0}^{k} \upharpoonright^{\alpha}_{\alpha} (q_l \parallel W_\alpha) \\
&= \mathop{\Big\|}_{l=0}^{k} F_l \ \parallel\ W_\alpha \\
&= BB_k.
\end{aligned}$$

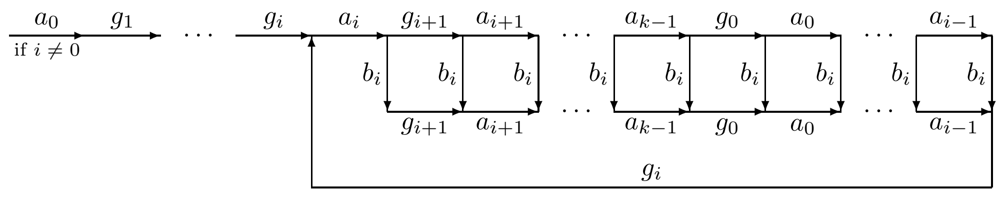
For $i = j$, we find that

$$\begin{aligned}
p_{ii} &= \mathop{\Big\|}_{l=0}^{k-1} \upharpoonright^{\alpha}_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_i)\big)} (p_l \parallel W_\alpha) \\
&= \mathop{\Big\|}_{l=0}^{k-1} \upharpoonright^{\alpha}_{\alpha\cup\{b_i\}} (p_l \parallel W_\alpha) \qquad (3) \\
&= \mathop{\Big\|}_{l=0,l\neq i}^{k-1} E_l \ \parallel\ C_i \ \parallel\ W_\alpha \\
&=
\end{aligned}$$



and

$$\begin{aligned}
q_{ii} &= \mathop{\Big\|}_{l=0}^{k-1} \upharpoonright^{\alpha}_{\alpha\cup\big(\alpha(p_i)\cap\alpha(q_i)\big)} (q_l \parallel W_\alpha) \\
&= \mathop{\Big\|}_{l=0}^{k-1} \upharpoonright^{\alpha}_{\alpha\cup\{b_i\}} (q_l \parallel W_\alpha) \qquad (4) \\
&= \mathop{\Big\|}_{l=0}^{k} F_l \ \parallel\ D_i \ \parallel\ W_\alpha \\
&=
\end{aligned}$$



The initial sequences of actions $a_0\, g_1 \cdots g_i$ in the two diagrams above are only present if $i \neq 0$. Obviously, p_{ij} and q_{ji} are thus equivalent. Note that the number of states of each intermediate term is always smaller than $3k$, i.e. linear in k.

Now note that $p_{ii} \parallel BB_k = p_{ii}$ and hence $\|_{j=0}^{k} p_{ij} = p_{ii}$. Similarly, $\|_{i=0}^{k-1} q_{ji} = q_{jj}$. Hence

$$\begin{aligned}
\lceil^{\alpha}_{\alpha\cup\alpha(p_i)}(p) &\overset{remark\ 4.6}{=} \mathop{\Big\|}_{j=0}^{k-1} \Big(\lceil^{\alpha}_{\alpha\cup\alpha(p_i)}(p_j \parallel W_\alpha)\Big) \\
&= \mathop{\Big\|}_{j=0}^{k-1} \Big(\lceil_{\alpha\cup\{b_i\}}(p_j \parallel W_\alpha)\Big) \\
&\overset{(3)}{=} \mathop{\Big\|}_{j=0}^{k} p_{ij}.
\end{aligned}$$

Equally, from remark 4.6 and (4) we have that $\lceil^{\alpha}_{\alpha\cup\alpha(q_j)}(q) = \|_{i=0}^{k-1} q_{ji}$ So according to theorem 4.4, it follows that $p = q$.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990.

[2] A. Bouajjani, J.-C. Fernandez and N. Halbwachs. Minimal model generation. Preliminary draft. 1991.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking 10^{20} states and beyond. In *Proceedings* 5th *Annual Symposium on Logic in Computer Science,* Philadelphia, USA, pages 428–439, 1990.

[4] S.A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the* 3rd *Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio*, pages 151–158, 1971.

[5] J. Engelfriet. Determinacy $\rightarrow$ (observation equivalence = trace equivalence). *Theoretical Computer Science*, 36(1):21–25, 1985.

[6] J.-C. Fernandez. An implementation of an efficient algorithm for bisimulation equivalence. *Science of Computer Programming*, 13:219–236, 1989/1990.

[7] J.-C. Fernandez and L. Mounier. “On the fly” verification of behavioural equivalences and preorders. In K.G. Larsen, editors, *Proceedings CAV’91,* Aalborg, pages 238–250. 1991.

[8] R.J. van Glabbeek. The linear time – branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings CONCUR’90,* Amsterdam, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer-Verlag, 1990.

[9] J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence (extended abstract). In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proceedings* 16th *ICALP,* Stresa, volume 372 of *Lecture Notes in Computer Science*, pages 423–438. Springer-Verlag, 1989. Full version to appear in *Information and Computation.*

[10] J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M.S. Paterson, editor, *Proceedings* 17th *ICALP,* Warwick, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer-Verlag, 1990.

[11] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall International, 1991.

[12] H. Qin. Efficient verification of determinate processes. In J.C.M. Baeten and J.F. Groote, editors, *Proceedings CONCUR'91,* Amsterdam, volume 527 of *Lecture Notes in Computer Science*, pages 471–494. Springer-Verlag, 1991.

[13] P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86:43–68, 1990.

[14] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[15] R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.

[16] R. Milner and F. Moller. Unique decomposition of processes. *Bulletin of the European Association for Theoretical Computer Science*, 41:226–232, 1990.

[17] F. Moller. *Axioms for concurrency.* PhD thesis, Report CST-59-89, Department of Computer Science, University of Edinburgh, 1989.

[18] A. Rabinovich. Checking equivalences between concurrent systems of finite agents (draft of extended abstract). Preprint, 1991.