Some Observations on Redundancy in a Context

Frits W. Vaandrager
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

Let x be a process which can perform an action a when it is in state s. In this article we consider the situation where x is placed in a context which blocks a whenever x is in s. The option of doing a in state s is *redundant* in such a context and x can be replaced by a process x' which is identical to x, except for the fact that x' cannot do a when it is in s (irrespective of the context). A simple, compositional proof technique is presented, which uses information about the traces of processes to detect redundancies in a process specification. As an illustration of the technique, a modular verification of a workcell architecture is presented.

1. Introduction

We are interested in the verification of distributed systems by means of algebraic manipulations. In process algebra, verifications often consist of a proof that the behaviour of an implementation $IMPL$ equals the behaviour of a specification $SPEC$, after abstraction from internal activity: $\tau_I(IMPL)=SPEC$.

The simplest strategy to prove such a statement is to derive first the transition system (process graph) for the process $IMPL$ with the expansion theorem, apply an abstraction operator to this transition system, and then simplify the resulting system to the system for $SPEC$ using the laws of (for instance) bisimulation semantics. This 'global' strategy however, is often not very practical due to combinatorial state explosion: the number of states of $IMPL$ can be of the same order as the product of the number of states of its components. Another serious problem with this strategy is that it provides almost no 'insight' in the structure of the system being verified. It is impossible to use the approach for the design of distributed systems, i.e. the stepwise construction of an implementation starting from a specification. This makes that there is a strong need for proof methods with a more *modular/compositional* character.

Partial support received from the European Community under ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR).

1.1. Modularity and compositionality. For the purpose of verification, we are interested in proof principles which transform a system locally, so that for a correctness proof of a local transformation one does not have to deal with the complexity of the system as a whole. A *modular* verification transforms an expression $\tau_I(IMPL)$ gradually into $SPEC$ by a sequence of local transformation steps. Consider, as an example, the case where $IMPL$ represents the parallel composition of components X_1, X_2 and X_3, where the actions in a set H have to synchronise: $IMPL = \partial_H(X_1 \| X_2 \| X_3)$. A possible step in a modular verification could be that X_1 and X_2 are replaced by Y_1 and Y_2. In that case one has to prove that:

$$\tau_I \circ \partial_H(X_1 \| X_2 \| X_3) = \tau_I \circ \partial_H(Y_1 \| Y_2 \| X_3).$$

It is sufficient to prove that $X_1 \| X_2 = Y_1 \| Y_2$. However, this will not be possible in general. It can be the case that processes $X_1 \| X_2$ and $Y_1 \| Y_2$ are only equal in the context $\tau_I \circ \partial_H(\cdots \| X_3)$. And even if the processes are equal, then still it is often not a good strategy to prove this. If one shows that two processes are equal, then one shows that they are interchangeable in any context, not only in the context in which they actually occur. In order to bring about successful substitutions, it is therefore desirable (or even necessary) to incorporate information about the context in which components are placed in correctness proofs of substitutions. A proof technique which allows one to do this to a sufficiently large degree is called modular. It is also possible to use a modular proof system the other way around. In that case one starts with a specification, which is refined to an implementation by a sequence of transformation steps.

A proof rule is called *compositional* if it helps to prove properties of the system as a whole from properties of the individual components. Compositional proof rules are essential for modular verifications.

In this article we present a proof principle which can be used to enhance the modularity of verifications. We claim that the principle captures a simple intuition about the behaviour of concurrent systems, and moreover makes it possible to give short, modular proofs in quite a large number of situations.

1.2. Example. We give a specification of a Dutch coffee machine similar to the one described in [14]:

$$KM = \overline{30c} \cdot (\overline{kof} + \overline{choc}) \cdot zoem \cdot KM.$$

After inserting 30 cents, the user may select 'koffie' or 'chocolade'. Dutch coffee machines make a humming sound ('zoemen') when they produce a drink. The behaviour of a typical Dutch user of such a machine can be described by the recursive equation below:

$$DU = (kof + 30c \cdot kof) \cdot talk \cdot DU.$$

Dutch people are widely known for their thrift, and they will never spend 30 cents for a cup of coffee if they can get it for free*. Synchronisation of actions

* Dutch users do not occur in [14]. In the modelling as presented here, the thrift of the Dutch user is not really taken into account: we can think of an environment where process DU performs

is given by: $\gamma(kof, \overline{kof}) = kof^*$, $\gamma(30c, \overline{30c}) = 30c^*$ and $\gamma(choc, \overline{choc}) = choc^*$. Let $H = \{kof, \overline{kof}, choc, \overline{choc}, 30c, \overline{30c}\}$. Consider the system $\partial_H(DU \| KM)$. It will be clear that in this environment the thrift of the Dutch user makes no sense. This behaviour is *redundant in the given context.* More 'realistic' is the behaviour $\overline{DU} = 30c \cdot kof \cdot talk \cdot \overline{DU}$, because $\partial_H(DU \| KM) = \partial_H(\overline{DU} \| KM)$.

1.3. Redundancy in a context. The example above is an instance of a situation which occurs very often: a process x has, in principle, the possibility to perform an action a when it is in state s, but is placed in an environment $\partial_H(\cdots \| y)$ which blocks a whenever the process is in s. In situations like this, the a-step from s is *redundant in the context* $\partial_H(\cdots \| y)$. We want to have the possibility to replace x by a component $\bar{x}$, that is identical to x except for the fact that $\bar{x}$ cannot do action a when it is in state s (irrespective of the context). For a compositional proof of the correctness of this type of substitutions new proof rules are needed. In this article we will show that in most situations partial information about the *(finite, sequential) traces* of processes is sufficient to prove that a summand in a specification is redundant and can be omitted. The notion 'redundancy in a context' was introduced in [16]. The present article can be viewed as a thorough revision of Section 6 from that paper.

1.4. Trace-specifications. It is argued by many authors (see for instance [5]), that if one is interested in program development by stepwise refinement, one needs to have the possibility of mixing programming notation with specification parts. A natural way to specify aspects of concurrent processes, advocated by [9, 14, 15, 17], is to give information about the traces, ready pairs and failure pairs of these processes. This leads to the notation

$$x \textbf{ sat } S$$

which expresses that process x satisfies property S. When we use the notation in this article, S will always be a property of the traces of x. Without any problem we can also include other information in S but we don't need that here.

In recent years it has become abundantly clear that there are many notions of 'process'. For instance, the idea that a process, in general, *is* the set of its traces, ready pairs or failure pairs is just false, because these notions of process do not capture features like real-time and fairness. Therefore we are interested in proof rules which express 'universal' truths about processes, and which are not tied to some particular model.

The point which is new in this article is that we use statements of the form x **sat** S, i.e. information about the traces of processes, in proofs that processes are equal in a sense different from (and finer than) trace equivalence. Thus we combine the advantages of a linear trace semantics with the distinctive power of finer equivalences.

an action $30c$ even though it has the possibility to perform an action *kof* instead. Preference of a process for certain actions can be modelled by means of the 'priority operator' of [2].

1.5. Workcell architecture. As an illustration of our technique, we present in Section 5 of this article a specification and verification of a workcell architecture, i.e. a system consisting of a number of workcells which cooperate in order to manufacture a certain product. The verification is not only modular, but also short when compared with the non-modular verifications of the same system by Biemans and Blonk [4], and Mauw [13]. In the first steps of the verification we remove the redundant summands in the process specification of the workcell architecture. Often the information that some summand is redundant has some importance of its own. It allows one to replace one component by another which is simpler cq. cheaper. In our modular proof this information becomes available as a by-product.

1.6. Related work. This is not the first article which is concerned with modular verification in the setting of process algebra. Work in this area has also been done by Larsen and Milner [11, 12], and Koymans and Mulder [10]. We think that our approach has basically two advantages when compared with this work. The first advantage is that our approach is technically speaking much simpler. People have strong intuitions concerning the trace behaviour of concurrent systems. Our proof rule makes it possible to use these intuitions quite directly in verifications. The intuitions behind the techniques of [10-12] are more involved and a lot of technical machinery is needed to formalize them. Our approach is probably less general than the approaches of [10-12], but we think that for almost all practical applications it can be used just as well.

The second advantage of our technique is that it is independent of the particular process semantics which is used. This in contrast to the work of [10-12], which is tied heavily to bisimulation semantics. In the discussion below we employ the laws of interleaved bisimulation semantics. However, we could just as well work with the laws of failure equivalence, ready equivalence or trace equivalence. Working with bisimulation semantics only makes our results stronger. We conjecture that the proof rule based on trace-specifications, as presented in this article, also holds in partial order semantics (see [7]). Probably the correctness proof of the workcell architecture which is presented in Section 5, when reorganized a little bit, is also valid in partial order semantics. It is a topic for future research to substantiate these claims.

2. Traces and trace-specifications

A *trace* of a process is a finite sequence that gives a possible order in which atomic actions can be performed by that process. A trace can end with the symbol $\surd$ (pronounce 'tick'), to indicate that, after execution of the last atomic action, successful termination can occur. After some preliminary definitions we give, in Section 2.3, axioms that relate processes to trace sets.

2.1. Definition.

1. For any alphabet Σ, we use Σ^* to denote the set of finite sequences over alphabet Σ. We write λ for the empty sequence and a for the sequence consisting of the single symbol $a \in \Sigma$. By $\sigma\star\sigma'$, often abbreviated $\sigma\sigma'$, we

denote the concatenation of sequences σ and σ'.

2. Let σ be a sequence and V be a set of sequences. We use notation $\sigma{*}V$ (or σV) for the set $\{\sigma{*}\rho \mid \rho \in V\}$, and notation $V{*}\sigma$ (or $V\sigma$) for the set $\{\rho\sigma \mid \rho \in V\}$.
3. By $\sharp\sigma$ we denote the length of a sequence σ.
4. On sequences we define a partial ordering $\leqslant$ (the *prefix ordering*) by: $\sigma \leqslant \rho$ if and only if, for some sequence σ', $\sigma\sigma' = \rho$. A set of sequences V is *closed under prefixing* if, for all $\sigma \leqslant \rho$, $\rho \in V$ implies that $\sigma \in V$.
5. $A_{\surd} = A \cup \{\surd\}$ is the set of atomic actions together with the termination symbol. Elements from $(A_{\surd})^*$ are called *traces* or *histories*. τ acts as the identity over $(A_{\surd})^*$ and is therefore replaced by λ when occurring in traces.
6. $\mathbb{T}$ is the set of nonempty, countable subsets of $T = A^* \cup A^{*}{*}\surd$ which are closed under prefixing.

2.2. DEFINITION. Let $a,b \in A$, $V,W \in \mathbb{T}$, $\sigma,\sigma_1,\sigma_2 \in T$. We define the following ACP-operators on trace sets*:

1. Sequential composition.
   $V \cdot W ::= (V \cap A^*) \cup \{\sigma_1{*}\sigma_2 \mid \sigma_1 \surd \in V \text{ and } \sigma_2 \in W\}$.
2. Parallel composition. $V \| W ::= \{\sigma \mid \exists \sigma_1 \in V, \sigma_2 \in W : \sigma \in \sigma_1 \| \sigma_2\}$. The set $\sigma_1 \| \sigma_2$ of traces is defined inductively by:

$$a\sigma_1 \| b\sigma_2 = \begin{cases} a(\sigma_1 \| b\sigma_2) \cup b(a\sigma_1 \| \sigma_2) \cup \gamma(a,b)(\sigma_1 \| \sigma_2) & \text{if } \gamma(a,b) \in A \\ a(\sigma_1 \| b\sigma_2) \cup b(a\sigma_1 \| \sigma_2) & \text{otherwise} \end{cases}$$

$$\lambda \| a\sigma = a\sigma \| \lambda = a(\lambda \| \sigma), \quad \lambda \| \lambda = \{\lambda\}, \quad \surd \| \sigma = \sigma \| \surd = \{\sigma\}.$$

   Here γ: $A_\delta \times A_\delta \to A_\delta$ is a given function which describes the synchronisation between atomic actions. γ is commutative, associative and has δ as zero-element.
3. Encapsulation. Let $H \subseteq A$. $\partial_H(V) ::= V \cap (A_{\surd} - H)^*$.
4. Abstraction. Let $I \subseteq A$. $\tau_I(V) ::= \{\tau_I(\sigma) \mid \sigma \in V\}$. The function τ_I on traces is given by:

$$\tau_I(a{*}\sigma) = \begin{cases} \tau_I(\sigma) & \text{if } a \in I \\ a{*}\tau_I(\sigma) & \text{otherwise} \end{cases}$$

$$\tau_I(\lambda) = \lambda, \quad \tau_I(\surd) = \surd.$$

5. Renaming. Let $f: A_{\tau\delta} \to A_{\tau\delta}$ with $f(\tau) = \tau$ and $f(\delta) = \delta$. $\rho_f(V) ::= \{\rho_f(\sigma) \mid \sigma \in V\}$. The function ρ_f on traces is given by:

$$\rho_f(a{*}\sigma) = \begin{cases} f(a){*}\rho_f(\sigma) & \text{if } f(a) \neq \delta \\ \lambda & \text{otherwise} \end{cases}$$

*The auxiliary operator $\|\_$ cannot be defined on trace sets. For a discussion of this issue we refer to [8].

$$\rho_f(\lambda) = \lambda, \quad \rho_f(\surd) = \surd.$$

6. Projection. Let $n \in \mathbb{N}$.
   $\pi_n(V) ::= \{\sigma \in V \cap A^* \mid \sharp\sigma \leqslant n\} \cup \{\sigma\surd \in V \mid \sharp\sigma \leqslant n\}$†.
7. Alphabets. $\alpha(V) ::= \{\alpha(\sigma) \mid \sigma \in V\}$. The function $\alpha: T \to Pow(A)$ is given by:

$$\alpha(a \star \sigma) = \{a\} \cup \alpha(\sigma), \quad \alpha(\lambda) = \alpha(\surd) = \varnothing.$$

2.3. The Trace Operator (TO). Let P be the sort of processes. The *trace operator* $tr: P \to \mathbb{T}$ relates to every process the set of traces that can be executed by that process. The operator satisfies the axioms of Table 1. ($a \in A$, $x,y \in P$, $H,I \subseteq A$, $f: A_{\tau\delta} \to A_{\tau\delta}$ with $f(\tau)=\tau$ and $f(\delta)=\delta$, and $n \in \mathbb{N}$)

$tr(\delta) = \{\lambda\}$	TO1	$tr(\partial_H(x)) = \partial_H(tr(x))$	TO7
$tr(\tau) = \{\lambda, \surd\}$	TO2	$tr(\tau_I(x)) = \tau_I(tr(x))$	TO8
$tr(a) = \{\lambda, a, a\surd\}$	TO3	$tr(\rho_f(x)) = \rho_f(tr(x))$	TO9
$tr(x+y) = tr(x) \cup tr(y)$	TO4	$tr(\pi_n(x)) = \pi_n(tr(x))$	TO10
$tr(x \cdot y) = tr(x) \cdot tr(y)$	TO5	$\alpha(x) = \alpha(tr(x))$	TO11
$tr(x \| y) = tr(x) \| tr(y)$	TO6		

TABLE 1. Axioms for the trace operator

When calculating with trace sets we implicitly use ZF. This means that the considerations of this paper are not of a completely algebraic nature. We restrict our attention to the models of the theory ACP_τ with recursion and auxiliary operators that can be mapped homomorphically to the trace algebra. This is no serious restriction because all 'interesting' process algebras are in this class. A similar approach is followed in [1].

2.3.1. Examples.

$$tr(x) = tr(\delta + x) = tr(\delta) \cup tr(x) = \{\lambda\} \cup tr(x). \tag{1}$$

So λ is member of the trace set of every process.

$$tr(ax) = tr(a) \cdot tr(x) = \{\lambda, a, a\surd\} \cdot tr(x) = \{\lambda, a\} \cup a \star tr(x) \overset{(1)}{=} \tag{2}$$
$$= \{\lambda\} \cup \{a\} \cup a \star (tr(x) \cup \{\lambda\}) = \{\lambda\} \cup a \star tr(x).$$

Let X be given by the recursive equation $X = aX$.

$$tr(X) = \bigcup_{n \geqslant 0} \pi_n(tr(X)) = \bigcup_{n \geqslant 0} tr(\pi_n(X)) = \bigcup_{n \geqslant 0} tr(a^n \cdot \delta) \tag{3}$$

† The π_n-operators we define here, satisfy the same axioms as the ones defined in [6]: $\pi_n(\tau)=\tau$, $\pi_0(ax)=\delta$, $\pi_{n+1}(ax)=a \cdot \pi_n(x)$, etc.

$$= \{\lambda\} \cup \{\lambda, a\} \cup \{\lambda, a, aa\} \cup \cdots = \{\lambda, a, aa, \ldots\}.$$

The first identity in derivation (3) follows from the structure of $\mathbb{T}$ and the definition of the π_n-operators on $\mathbb{T}$.

2.4. Trace-specifications. A *trace-specification* is a predicate. A trace-specification S describes the set of traces which, when assigned to free occurrences of a chosen variable σ of type trace in S, make the predicate true: $\{\sigma \mid S\}$. The syntax for trace-specifications we have in mind is a first-order language with integers, actions, traces, some simple functions like addition and multiplication, taking the i-th element of a trace, $\sharp\sigma$, $\rho_i(\sigma)$, equality predicates for the integers, actions and traces, and quantification over integers and traces. This syntax is almost equivalent to the syntax proposed in [14], except for the fact that we moreover have multiplication. This increases the expressiveness of our logic, and makes it for instance possible to define for each regular trace-language L a predicate S_L such that $L = \{\sigma \mid S_L\}$. In Section 4.5 it will be argued that such predicates are useful. All predicates that we will use in this article are definable in terms of the syntax which is described informally above.

A process x *satisfies* a trace-specification S for trace variable σ, notation

$$x \ \mathbf{sat}_\sigma \ S,$$

if

$$\forall \sigma \in tr(x) : S.$$

Because in nearly all cases we will use a fixed trace-variable σ, we often omit the subscript σ and write x **sat** S. In this article we regard x **sat** S merely as a notation. The proofs take place on the more elementary level of the *tr*-operator and trace sets. In [9] an elegant proof system is given which takes x **sat** S as a primitive notion. This system contains for instance rules like

$$\frac{x \ \mathbf{sat} \ S, \ x \ \mathbf{sat} \ S'}{x \ \mathbf{sat} \ S \wedge S'} \qquad \frac{x \ \mathbf{sat} \ S, \ S \Rightarrow S'}{x \ \mathbf{sat} \ S'}.$$

2.4.1. Notation. Let $\sigma \in T$, $B \subseteq A$ and $a \in A$.

1. $\sigma \restriction B$ gives the *projection* of trace σ onto the actions of B: $\sigma \restriction B = \tau_{A-B}(\sigma)$.
2. $\sigma \downarrow a$ denotes the number of occurrences of a in σ:
$$\sigma \downarrow a = \begin{cases} \sharp(\sigma \restriction \{a\}) - 1 & \text{if } \sigma = \sigma' \surd \\ \sharp(\sigma \restriction \{a\}) & \text{otherwise} \end{cases}$$
3. Even though our trace-specification language contains no alphabet operator, we can talk about alphabets in predicates: $\alpha(\sigma) \subseteq B \Leftrightarrow \sigma \restriction B = \sigma$.

2.4.2. Example. The coffee machine from Example 1.2 satisfies

$$KM \textbf{ sat } \alpha(\sigma) \subseteq \{\overline{kof}, \overline{choc}, \overline{30c}, zoem\} \wedge (\sigma \downarrow \overline{kof} \leqslant \sigma \downarrow \overline{30c}).$$

The number of cups of 'koffie' produced by the machine is always less or equal to the number of times 30 cents have been paid. The Dutch user however, takes care that never more than 30 cents are paid in advance:

$$DU \textbf{ sat } \alpha(\sigma) \subseteq \{kof, 30c, talk\} \wedge (\sigma \downarrow kof \geqslant (\sigma \downarrow 30c - 1)).$$

2.4.3. Remark. Sometimes we write a specification as $S(\sigma)$, to indicate that the specification will normally contain σ as a free variable. In that case we use the notation $S(te)$ to denote the predicate obtained from $S(\sigma)$ by substituting all free occurrences of σ by an expression *te* of sort trace, avoiding name clashes.

3. Observability and localisation

The parallel combinator $\|$ is in some sense related to the cartesian product construction. In the graph model of [3], the set of nodes of a graph $g\|h$ is defined as the set of ordered pairs of the nodes of g and h. Still the $\|$-operator lacks an important property of cartesian products, namely the existence of projection operators. It is not possible in general to define operators l and r such that $l(x\|y)=x$ and $r(x\|y)=y$. In this section we show that, if we impose a number of constraints on the communication function, and on x and y, it becomes possible to define an operator which, given the alphabet of x, can recover x almost completely from $x\|y$:

$$\tau \cdot \rho_{\nu(\alpha(x))}(x\|y) \cdot \delta = \tau \cdot x \cdot \delta.$$

The conditions on x and y make that x is *observable*, the operator $\rho_{\nu(\alpha(x))}$ *localises* x in $x\|y$.

3.1. Communication. For the specification of distributed systems, we mostly use the read/send communication scheme, or communications of type $\gamma(kof, \overline{kof}) = kof^*$. Following [10], such communication functions will be characterized as *trijective*. The assumption that communication is trijective will simplify the discussion of this article.

3.1.1. Definition. A communication function γ is *trijective* if three pairwise disjoint subsets $R,S,C \subseteq A$ can be given, and bijections $\bar{}: R \to S$ and $^\circ: R \to C$ such that for every $a,b,c \in A$:

$$\gamma(a,b)=c \Rightarrow (a \in R \wedge b = \bar{a} \wedge c = a^\circ) \vee (b \in R \wedge a = \bar{b} \wedge c = b^\circ).$$

In the rest of this article we assume that communication is trijective.

3.1.2. Remark. Observe that a trijective communication function γ satisfies the following three properties, and that each γ satisfying these properties is trijective ($a,b,c,d \in A$):
1. $\gamma(a,a) = \delta$,
2. if $\gamma(a,b) \neq \delta$ and $\gamma(a,c) \neq \delta$ then $b = c$ (γ is 'monogamous'),
3. if $\gamma(a,b) = \gamma(c,d) \neq \delta$ then $a = c$ or $a = d$ (γ is 'injective').

Observe further that a trijective γ satisfies $\gamma(\gamma(a,b),c) = \delta$ ('handshaking').

3.2. Observability. We are interested in the behaviour of a process x when it is placed in a context $\cdots \| y$. In order to keep things simple, we will always choose x and y in such a way that x is observable in context with y: every action of $x \| y$ is either an action from x, or an action from y, or a synchronisation between x and y. In the last case we moreover know which action from x participates in the synchronisation. Below we give a formal definition of this notion of observability.

3.2.1. DEFINITION. Let $B \subseteq A$ be a set of atomic actions. B is called *observable* if for each triple $a,b,c \in A$ with $\gamma(a,b) = c$ at most one element of $\{a,b,c\}$ is a member of B.

Let for $A_1, A_2 \subseteq A$: $A_1 | A_2 = \{\gamma(a_1,a_2) \in A \mid a_1 \in A_1, a_2 \in A_2\}$. From the fact that a set B of actions is observable, we can conclude that $B \cap B | A = \emptyset$. Because γ is injective, we know in addition that γ has an 'inverse' on $B | A$: for each $c \in B | A$, there is exactly one $b \in B$ such that an $a \in A$ exists with $\gamma(a,b) = c$. In this case we write $b = \gamma_B^{-1}(c)$.

3.2.2. DEFINITION. Let x,y be processes. Process x is called *observable in context* $\cdots \| y$, if $\alpha(x)$ is observable, and $\alpha(y)$ is disjoint from $\alpha(x)$ and $\alpha(x) | A$.

If a process x is observable in a context $\cdots \| y$, then one can tell for each action from $x \| y$ whether it is from x, from y, or from x and y together. In the last case one can also tell which action from x participates in the communication. Observe that the fact that x is observable in context $\cdots \| y$ does not imply that y is observable in context $\cdots \| x$.

3.3. Localisation. The 'localisation' of actions from x in a context $\cdots \| y$ as described informally above, can be expressed formally by means of renaming operators. In the literature other definitions of the notions observability and localisation can be found (see [1] and [16]). In the choice of the definitions, there is a trade-off between the degree of generality (the capability of operators to localise actions) and the length of the definitions.

3.3.1. DEFINITION. Let $B \subseteq A$ be observable. The *localisation function* $\nu(B)$: $A_{\tau\delta} \to A_{\tau\delta}$ is the renaming function defined by:

$$\nu(B)(a) = \begin{cases} a & \text{if } a \in B \cup \{\tau,\delta\} \\ \gamma_B^{-1}(a) & \text{if } a \in B | A \\ \tau & \text{otherwise} \end{cases}$$

3.3.2. Example. The communication function in Example 1.2 is trijective. Furthermore $\alpha(DU) = \{kof, 30c, talk\}$ is observable. Process DU is observable in the context $\cdots \| KM$. DU is however not observable in the context $\cdots \|(DU\|KM)$. The expression

$$\rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM)$$

denotes the process corresponding to the behaviour of the Dutch user in a context $\partial_H(\cdots \| KM)$. We derive:

$$\begin{aligned}
\rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM) &= \\
&= \rho_{\nu(\alpha(DU))}(30c^* \cdot kof^* \cdot (talk \cdot zoem + zoem \cdot talk) \cdot \partial_H(DU\|KM)) \\
&= 30c \cdot kof \cdot (talk \cdot \tau + \tau \cdot talk) \cdot \rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM) \\
&= 30c \cdot kof \cdot talk \cdot \rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM)
\end{aligned}$$

Hence $\rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM)$ and $\overline{DU}$ satisfy the same guarded recursion equation. Application of the *Recursive Specification Principle (RSP)* now gives that both processes are equal.

3.3.3. Remark. It may seem that one needs the τ-law T2 ($\tau x = \tau x + x$) in the verification above. Surprisingly we can perform the verification using only the τ-law T1 ($x\tau = x$):

$$kof \cdot (talk \cdot \tau + \tau \cdot talk) = kof \cdot (\tau \| talk) = kof \cdot \tau \lfloor talk = kof \lfloor talk = kof \cdot talk.$$

In fact we claim that all the verifications in this article can be done using the τ-law T1 only. So we also do not need the law T3 ($a(\tau x + y) = a(\tau x + y) + ax$).

3.3.4. THEOREM. *Let p,q be closed terms with p observable in context $..\|q$. Then* $\mathrm{ACP}_\tau + \mathrm{RN} + \mathrm{AB} \vdash \tau \cdot \rho_{\nu(\alpha(p))}(p\|q) \cdot \delta = \tau \cdot p \cdot \delta$.

PROOF. Easy. □

3.3.5. THEOREM. *Let x,y be processes, with x observable in context $..\|y$. Then we can prove using the axioms TO that: $tr(\rho_{\nu(\alpha(x))}(x\|y)) \subseteq tr(x)$.*

PROOF. Using the axioms from Table 1, we rewrite the statement we have to prove into:

$$\rho_{\nu(\alpha(tr(x)))}(tr(x)\|tr(y)) \subseteq tr(x).$$

Because $tr(x)$, $tr(y) \in \mathbb{T}$, it is sufficient to prove that for every $V, W \in \mathbb{T}$ with $\alpha(V)$ observable and $\alpha(W)$ disjoint from $\alpha(V)$ and $\alpha(V) | A$:

$$\rho_{\nu(\alpha(V))}(V\|W) \subseteq V.$$

First we apply the definition of the merge-operator on trace sets:

$$\rho_{\nu(\alpha(V))}(V\|W) = \rho_{\nu(\alpha(V))}(\{\sigma \mid \exists v \in V,\ w \in W : \sigma \in v\|w\}).$$

The theorem is proved if we show for all $v \in V$ and $w \in W$ that:

$$\rho_{\nu(\alpha(V))}(v \| w) \subseteq V.$$

We prove a slightly stronger fact: Let $v = v_1 * v_2 \in V$ and let $w \in W$. Then:

$$v_1 * \rho_{\nu(\alpha(V))}(v_2 \| w) \subseteq V.$$

The proof goes by means of simultaneous induction on the structure of v_2 and w.

Case 1: $v_2 = \surd$

$v_1 * \rho_{\nu(\alpha(V))}(\surd \| w) = v_1 * \rho_{\nu(\alpha(V))}(\{w\}) = v_1 * \{\rho_{\nu(\alpha(V))}(w)\} \subseteq v_1 * \{\lambda, \surd\} \subseteq V$

Here we use that V is closed under prefixing.

Case 2: $w = \surd$

$v_1 * \rho_{\nu(\alpha(V))}(v_2 \| \surd) = v_1 * \rho_{\nu(\alpha(V))}(\{v_2\}) = v_1 * \{\rho_{\nu(\alpha(V))}(v_2)\} = v_1 * \{v_2\} = \{v\} \subseteq V$

Case 3.1: $v_2 = \lambda$ en $w \in A^*$

$v_1 * \rho_{\nu(\alpha(V))}(\lambda \| w) = v_1 * \rho_{\nu(\alpha(V))}(\{w\}) = v_1 * \{\rho_{\nu(\alpha(V))}(w)\} = v_1 * \{\lambda\} = \{v\} \subseteq V$

Case 3.2: $v_2 = \lambda$ en $w = w_1 \surd$

$v_1 * \rho_{\nu(\alpha(V))}(\lambda \| w_1 \surd) = v_1 * \rho_{\nu(\alpha(V))}(\{w_1\}) = v_1 * \{\rho_{\nu(\alpha(V))}(w_1)\} = v_1 * \{\lambda\} = \{v\} \subseteq V$

Case 4.1: $v_2 \in A^*$ en $w = \lambda$

$v_1 * \rho_{\nu(\alpha(V))}(v_2 \| \lambda) = v_1 * \rho_{\nu(\alpha(V))}(\{v_2\}) = v_1 * \{\rho_{\nu(\alpha(V))}(v_2)\} = v_1 * \{v_2\} = \{v\} \subseteq V$

Case 4.2: $v_2 = v_3 \surd$ en $w = \lambda$

$v_1 * \rho_{\nu(\alpha(V))}(v_3 \surd \| \lambda) = v_1 * \rho_{\nu(\alpha(V))}(\{v_3\}) = v_1 * \{\rho_{\nu(\alpha(V))}(v_3)\} = v_1 * \{v_3\} = \{v_1 * v_3\} \subseteq V$

(V is closed under prefixing.)

Case 5.1: $v_2 = av_3$, $w = bw_1$ en $\gamma(a,b) = \delta$

$$\begin{aligned} v_1 * \rho_{\nu(\alpha(V))}(av_3 \| bw_1) &= v_1 * \rho_{\nu(\alpha(V))}(a(v_3 \| bw_1) \cup b(av_3 \| w_1)) \\ &= v_1 * a * \rho_{\nu(\alpha(V))}(v_3 \| bw_1) \cup v_1 * \rho_{\nu(\alpha(V))}(av_3 \| w_1) \subseteq V \end{aligned}$$

(Apply induction hypothesis.)

Case 5.2: $v_2 = av_3$, $w = bw_1$ en $\gamma(a,b) \in A$

$v_1 * \rho_{\nu(\alpha(V))}(av_3 \| bw_1) = v_1 * \rho_{\nu(\alpha(V))}(a(v_3 \| bw_1) \cup b(av_3 \| w_1) \cup \gamma(a,b)(v_3 \| w_1))$

$$= v_1 * a * \rho_{\nu(\alpha(V))}(v_3 \| bw_1) \cup v_1 * \rho_{\nu(\alpha(V))}(av_3 \| w_1) \cup$$
$$\cup v_1 * a * \rho_{\nu(\alpha(V))}(v_3 \| w_1) \subseteq V$$

(Apply induction hypothesis.) □

Notice that the $\subseteq$-sign in Theorem 3.3.5 cannot be changed into an $=$-sign. If $tr(y)$ contains no traces ending on $\surd$, then $tr(\rho_{\nu(\alpha(x))}(x \| y)$ will also contain no such traces, even if they are in $tr(x)$.

3.3.6. THEOREM. *Let x,y be processes, with x observable in context ..‖y, and let $H \subseteq A$. Then we can prove using the axioms TO that: $tr(\rho_{\nu(\alpha(x))} \circ \partial_H(x \| y)) \subseteq tr(x)$.*

PROOF. Just like we did in the proof of Theorem 3.3.5, we reformulate the statement. Let $V, W \in \mathbb{T}$ with $\alpha(V)$ observable, and $\alpha(W)$ disjoint from $\alpha(V)$ and $\alpha(V) \mid A$. We have to prove:

$$\rho_{\nu(\alpha(V))} \circ \partial_H(V \| W) \subseteq V.$$

For $X, Y \in \mathbb{T}$ we have that $\partial_H(X) \subseteq X$ and $X \subseteq Y \Rightarrow \rho_f(X) \subseteq \rho_f(Y)$. Hence

$$\rho_{\nu(\alpha(V))} \circ \partial_H(V \| W) \subseteq \rho_{\nu(\alpha(V))}(V \| W).$$

From the proof of Theorem 3.3.5 we conclude:

$$\rho_{\nu(\alpha(V))}(V \| W) \subseteq V. \quad \square$$

The following corollary of Theorem 3.3.6 plays an important role in this article because it allows us to derive a property of a system as a whole from a property of a component (this is the essence of compositionality).

3.3.7. COROLLARY. *Let x,y be processes, with x observable in context ..‖y, let $H \subseteq A$ and suppose $f = \nu(\alpha(x))$. If x* **sat** *$S(\sigma)$, then:*

$$\rho_f \circ \partial_H(x \| y) \textbf{ sat } S(\sigma)$$

and consequently

$$\partial_H(x \| y) \textbf{ sat } S(\rho_f(\sigma)).$$

3.4. REMARK. The formal definitions of the notions 'observable' and 'localisation' in this section are quite complex. The definitions are much simpler if one works with the synchronisation-merge $\|_A$ of Olderog and Hoare [15] instead of the parallel combinator $\|$ of ACP. In fact the whole discussion of this article can be simplified considerably if one uses the $\|_A$-combinator. The main reason for this is that the combinator corresponds quite directly with logical conjunction of trace-specifications (see [14]).

Still, one cannot say that $\|_A$ is a better operator than $\|$ in general. The synchronisation format of the $\|$-operator is very flexible and often allows for elegant specifications. An unpleasant property of the $\|_A$-operator is that it is

not associative (in general $(x\|_B y)\|_C z \neq x\|_B(y\|_C z)$). We think that the operators $\|$ and $\|_A$ are both very useful and that therefore notions like 'observable', 'localisation' and 'redundancy in context' should be worked out for both.

4. Redundancy in a context

We want to prove, in a compositional way, that in a given context a summand in a specification can be omitted. We will restrict ourselves in this article to the case where the summand occurs in a 'linear' equation.

4.1. Definition. Let $E = \{X = t_X \mid X \in V_E\}$ be a recursive specification. A set $C \subseteq V_E$ of variables is called a *cluster* if for each $X \in C$, t_X is of the form:

$$\sum_{k=1}^{m} a_k \cdot X_k + \sum_{l=1}^{n} Y_l$$

for actions $a_k \in A_\tau$, variables $X_k \in C$ and $Y_l \in V_E - C$. Cluster C is called *isolated* if variables from C do not occur in the terms for the variables from $V_E - C$.

4.2. Definition. Let $E = \{X = t_X \mid X \in V_E\}$ be a recursive specification and let C be an isolated cluster in E. Let $X_0, X_1, X_2 \in C$, $a \in A_\tau$ and let aX_2 be a summand of t_{X_1}. Let E' be obtained from E by replacing summand aX_2 in t_{X_1} by a 'fresh' atom t. Write $p \leftrightarrows \langle X_0 \mid E \rangle$ and $p' \leftrightarrows \langle X_0 \mid E' \rangle$. Let y be a process with p observable in context $..\|y$. Let $H \subseteq A$. The summand aX_2 of p *is redundant in the context* $\partial_H(..\|y)$ if:

$$tr(\rho_{\nu(\alpha(p))} \circ \partial_H(p\|y)) \cap \{\sigma a \mid \sigma t \in tr(p')\} = \varnothing.$$

4.2.1. Comment. One can say that the set $\{\sigma a \mid \sigma t \in tr(p')\}$ is the contribution of summand aX_2 to $tr(p)$. Theorem 3.3.6 gives that $tr(\rho_{\nu(\alpha(p))} \circ \partial_H(p\|y))$ is also a subset of $tr(p)$. If summand aX_2 is redundant, this means that all behaviours of p of the form 'go from state X_1 with an a-step to state X_2' are not possible if p is placed in the context $\partial_H(..\|y)$.

We give an example which shows why we require in Definition 4.2 that cluster C is isolated. Assume a trijective communication function γ with $\gamma(a,\bar{a}) = a^*$ and $\gamma(b,\bar{b}) = b^*$. Assume further that $H = \{a,\bar{a},b,\bar{b}\}$ en $I = \{a^*,b^*\}$. Consider the following recursive specification E:

$$X_0 = aX_0 + X_1$$

$$X_1 = b \cdot \tau_I \circ \partial_H(X_0 \| \bar{a} \cdot c)$$

In this system X_0 forms a cluster which is not isolated. We derive:

$$X_0 = aX_0 + b \cdot c \cdot \delta.$$

From this equation it is easy to see that X_0 is observable in context $..\|\bar{b}$. We have:

$$\rho_{\nu(\alpha(X_0))} \circ \partial_H(X_0 \| \overline{b}) = b \cdot c \cdot \delta.$$

If the condition in Definition 4.2 that C is isolated would be absent, then the summand aX_0 would (by definition) be redundant in context $\partial_H(.. \| \overline{b})$. However, the summand cannot be omitted: outside the cluster it plays an essential role!

We can now formulate the central proof principle of this article:

A redundant summand can be omitted.

Below we formally present this principle as a theorem.

4.3. THEOREM. *Let $p \equiv \langle X_0 \mid E \rangle$ and $q \equiv \langle Y_0 \mid F \rangle$, with E and F guarded recursive specifications, and p observable in context $.. \| q$. Let $H \subseteq A$. Let C be an isolated cluster in E with $X_0, X_1, X_2 \in C$, $a \in A_\tau$ and aX_2 a summand of t_{X_1}. Let E' and $\overline{E}$ be obtained from E by resp. replacing aX_2 by a fresh atom t, and omitting it. Let $p' \equiv \langle X_0 \mid E' \rangle$ and $\overline{p} \equiv \langle X_0 \mid \overline{E} \rangle$. Suppose that* ACP_τ + RDP + RN + PR + TO *proves that summand aX_2 is redundant. Then:* ACP_τ + RDP + RN + PR + AIP^- $\vdash \partial_H(p \| q) = \partial_H(\overline{p} \| q)$.

PROOF. Omitted. The proof uses a bisimulation model generated by Plotkin like action rules. It is proved that the (infinitary) axiom system ACP_τ + RDP + RN + PR + AIP^- is sound and complete for processes represented by a guarded specification. Consequently it is enough to prove that $\partial_H(p \| q)$ and $\partial_H(\overline{p} \| q)$ are bisimilar. The proof that the obvious candidate for a bisimulation between these processes indeed *is* a bisimulation uses the fact that every trace of actions in the transition system of an expression p is also a (provable) element of $tr(p)$. □

4.4. Remark. A summand which can be omitted is in general not redundant. In every context the second summand of the equation

$$X = aX + aX$$

can be omitted, even if it is not redundant. At present we have no idea how a 'reversed version' of Theorem 4.3 would look like.

4.5. Proving redundancies. Now we know that a redundant summand can be omitted, it becomes of course interesting to look for techniques which allow us to prove that summands are redundant. The following strategy works in most cases.

Let E, C, X_0, etc., be as given in Definition 4.2. In order to prove that the summand is redundant, it is enough to show that for some predicate $S(\sigma)$:

$$p' \textbf{ sat } \forall \sigma' : \sigma = \sigma' t \Rightarrow S(\sigma' a) \quad \text{and}$$

$$\rho_{\nu(\alpha(p))} \circ \partial_H(p \| y) \textbf{ sat } \neg S(\sigma).$$

If the cluster C is finite, then $\{\sigma a \mid \sigma t \in tr(p')\}$ is a regular language and can be denoted by a predicate in the trace-specification language of Section 2.4. Consequently we can in such cases always express that a summand is redundant.

4.6. Example. We return to Example 1.2 and show how the statement

$$\partial_H(DU\|KM) = \partial_H(\overline{DU}\|KM)$$

can be proved with the notions presented in this section. KM is observable in context $DU\|..$, and DU is observable in context $..\|KM$. The specification of DU contains no isolated clusters, but using RSP we can give an equivalent specification where the set of variables as a whole forms an isolated cluster $(DU = UD)$.

$$\begin{aligned} UD &= 30c \cdot UD_1 + kof \cdot UD_2 \\ UD_1 &= kof \cdot UD_2 \\ UD_2 &= talk \cdot UD \end{aligned}$$

TABLE 2. Specification of DU

In Example 2.4.2 we already observed that:

$$KM \textbf{ sat } \sigma\downarrow\overline{kof} \leqslant \sigma\downarrow\overline{30c}.$$

Because of Corollary 3.3.7 we also have:

$$\rho_{\nu(\alpha(KM))} \circ \partial_H(UD\|KM) \textbf{ sat } \sigma\downarrow\overline{kof} \leqslant \sigma\downarrow\overline{30c}.$$

The alphabet of process $\partial_H(UD\|KM)$ contains no actions $\overline{kof}$ or $\overline{30c}$, because these actions are in H. This implies that occurrences of these actions in traces from $tr(\rho_{\nu(\alpha(KM))} \circ \partial_H(UD\|KM))$ 'originated' (by renaming) from actions kof^* and $30c^*$. Hence:

$$\partial_H(UD\|KM) \textbf{ sat } \sigma\downarrow kof^* \leqslant \sigma\downarrow 30c^*.$$

But since the alphabet of $\partial_H(UD\|KM)$ contains no actions kof and $30c$, this implies:

$$\rho_{\nu(\alpha(UD))} \circ \partial_H(UD\|KM) \textbf{ sat } \sigma\downarrow kof \leqslant \sigma\downarrow 30c.$$

Define UD' by:

$$\begin{aligned} UD' &= 30c \cdot UD'_1 + t \\ UD'_1 &= kof \cdot UD'_2 \\ UD'_2 &= talk \cdot UD' \end{aligned}$$

Of course we have

$$UD' \textbf{ sat } \forall\sigma' : \sigma = \sigma' t \Rightarrow (\sigma' kof)\downarrow kof > (\sigma' kof)\downarrow 30c.$$

This shows that the second summand in the equation from UD is redundant. □

In the example above, we gave a long proof of a trivial fact. The nice thing about the proof is however that it is compositional and only uses general properties of the separate components. This makes that the technique can be used also in less trivial situations where the number of states of the components is large.

In the sequel we will speak about redundant summands of equations which are not part of a cluster. What we mean in such a case is that the corresponding system of equations can be transformed into another system, that a certain summand in the new system is redundant, and that the system which results from omitting this summand is equivalent to the system obtained by omitting the summand in the original system that was called 'redundant'.

5. A WORKCELL ARCHITECTURE

In this section we present a modular verification of a small system which is described in [4, 13].

One can speak about *Computer Integrated Manufacturing (CIM)* if computers play a role in all phases of an industrial production process. In the CIM-philosophy one views a plant as a (possibly hierarchically organized) set of concurrently operating *workcells*. Each workcell is responsible for a well-defined part of the production process, for instance the filling and closing of bottles of milk.

In principle it is possible to specify the behaviour of individual workcells in process algebra. A composite workcell, or even a plant, can then be described as the parallel composition of a number of more elementary workcells. Proof techniques from process algebra can be applied to show that a composite workcell has the desired external behaviour.

In general, not all capabilities of a workcell which is part of a CIM-architecture will be used. A robot which can perform a multitude of tasks, can be part of an architecture where its only task is to fasten a bolt. Other possibilities of the robot will be used only when the architecture is changed. A large part of the behaviours of workcells will be redundant in the context of the CIM-architecture of which they are part. Therefore it can be expected that the notions which are presented in the previous sections of this article, will be useful in the verification of such systems.

5.1. Specification

5.1.1. The external behaviour. We want to construct a composite workcell which satisfies the following specification.

$$SPEC = \sum_{n=0}^{N} r\,1(n)\cdot SPEC^n \cdot SPEC$$

$$SPEC^0 = s\,0(r) \qquad SPEC^{n+1} = s\,10(proc(p\,1))\cdot SPEC^n$$

TABLE 3. Specification of a composite workcell

Via port 1, the workcell accepts an order to produce n products of type $proc(p\,1)$ and to deliver these products at port 10. Here $0 \leqslant n \leqslant N$ for a given upperbound $N > 0$. After execution of the order, the workcell gives a signal r at port 0, and returns to its initial state (r = ready).

5.1.2. Architecture. The architecture of the system that has to implement this specification is depicted in Figure 1.

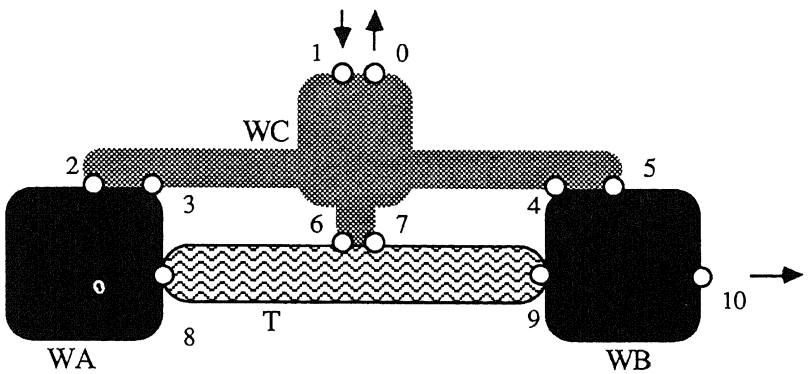

FIGURE 1

There are four components: Workcell A (WA), Workcell B (WB), the Transport service T, and the Workcell Controller WC.

5.1.3. Workcell A. By means of a signal n at port 2, Workcell A receives the order to produce n products of type $p\,1$. The cell performs the job and delivers the products to the Transport service T at port 8. Thereafter a message r is sent at port 3, to indicate that a next order can be given.

$$WA = \sum_{n=0}^{N} r\,2(n)\cdot XA^n$$

$$XA^0 = s\,3(r)\cdot WA \qquad XA^{n+1} = s\,8(p\,1)\cdot XA^n$$

TABLE 4. Specification of Workcell A

5.1.4. Workcell B. By means of a signal n at port 4, Workcell B receives the order to process n products. B receives products from a set $PROD$ at port 9. An incoming product p is processed and the result $proc(p) \in PROD$ is delivered at port 10 ($proc$ = processed). Thereafter a message r is sent at port 5 and the workcell returns to its initial state. We assume that $p1 \in PROD$.

$$WB = \sum_{n=0}^{N} r4(n) \cdot XB^n$$

$$XB^0 = s5(r) \cdot WB \qquad XB^{n+1} = \sum_{p \in PROD} r9(p) \cdot s10(proc(p)) \cdot XB^n$$

TABLE 5. Specification of Workcell B

5.1.5. Transport service T transports products in $PROD$ and behaves like a FIFO-queue. Products are accepted by T at port 8. Transport commands tc are given to T at port 6. The number of products accepted by the transport service should not exceed the number of transport commands which have been received by more than one. Each time a product leaves T at port 9, a signal $s7(ar)$ is given (ar = arrival). Variables in the specification below are indexed by the contents of the transport service: $\sigma \in PROD^*$ and $p, q \in PROD$.

$$T^{\lambda} = r6(tc) \cdot \Big(\sum_{p \in PROD} r8(p) \cdot T^p \Big) + \sum_{p \in PROD} r8(p) \cdot r6(tc) \cdot T^p$$

$$T^{\sigma q} = r6(tc) \cdot \Big(\sum_{p \in PROD} r8(p) \cdot T^{p\sigma q} \Big) + \sum_{p \in PROD} r8(p) \cdot r6(tc) \cdot T^{p\sigma q} + s9(q) \cdot s7(ar) \cdot T^{\sigma}$$

TABLE 6. Specification of Transport service

5.1.6. Workcell Controller WC is the boss of components WA, T and WB. From his superiors (via port 1), WC can get the order to take care of the manufacturing of n products $proc(p1)$. In order to execute this order, WC sends a stream of commands to his subordinates, receiving progress reports from these subordinates in between. When the controller thinks that the task has been completed, he generates a signal $s0(r)$.

$$WC = \sum_{n=0}^{N} r1(n) \cdot s4(n) \cdot XC^n$$

$$XC^0 = r5(r) \cdot s0(r) \cdot WC \qquad XC^{n+1} = s2(1) \cdot r3(r) \cdot s6(tc) \cdot r7(ar) \cdot XC^n$$

TABLE 7. Specification of Workcell Controller

5.1.7. $\mathbb{D} = \{n \mid 0 \leqslant n \leqslant N\} \cup \{r,tc,ar\} \cup PROD$ is the set of objects which can be communicated in the system, and $\mathbb{P} = \{0,1,...,10\}$ is the set of port-names used. Communication takes place following the read/send-scheme:

$$\gamma(rp(d),sp(d)) = cp(d) \quad \text{for } p \in \mathbb{P},\ d \in \mathbb{D}$$

and γ yields δ in all other cases. Important sets of actions are:

$$H = \{rp(d),sp(d) \mid 2 \leqslant p \leqslant 9 \text{ and } d \in \mathbb{D}\} \quad \text{and}$$

$$I = \{cp(d) \mid 2 \leqslant p \leqslant 9 \text{ and } d \in \mathbb{D}\}.$$

The implementation as a whole can now be described by:

$$\boxed{IMPL = \partial_H(WC \| WA \| T^{\lambda} \| WB)}$$

5.2. THEOREM (CORRECTNESS IMPLEMENTATION).

$$\mathrm{ACP}_{\tau} + \mathrm{SC} + \mathrm{RDP} + \mathrm{PR} + \mathrm{AIP}^{-} + \mathrm{AB} + \mathrm{CA} \vdash \tau_I(IMPL) = SPEC.$$

PROOF. In seven steps we transform $\tau_I(IMPL)$ to $SPEC$. Before we start with the 'real' calculations, we show in the first three steps that in the specifications of components WA, T and WB, a large number of summands can be omitted. Notice that communication is trijective and that each component of $IMPL$ is observable in context with the other components.

First we use that the only command which is given by the controller to Workcell A is a request to produce a single product p 1. This means that:

$$IMPL \textbf{ sat } \sigma\downarrow c2(n) = 0 \quad \text{for } n \neq 1.$$

Consequently

$$\rho_{\nu(\alpha(WA))}(IMPL) \textbf{ sat } \sigma\downarrow r2(n) = 0 \quad \text{for } n \neq 1.$$

Using the approach of Section 4.5, together with Theorem 4.3, we obtain that all the summands in the specification of WA which correspond to the acceptance of a command different from $r2(1)$ are redundant. We have

$$IMPL = \partial_H(WC \| \overline{WA} \| T^{\lambda} \| WB),$$

where $\overline{WA}$ is given by:

$$\boxed{\overline{WA} = r2(1) \cdot s8(p\,1) \cdot s3(r) \cdot \overline{WA}}$$

Hence:

$$\tau_I(IMPL) = \tau_I \circ \partial_H(WC \| \overline{WA} \| T^{\lambda} \| WB). \qquad \text{(step 1)}$$

Also component T^{λ} is clearly a candidate for simplification. With some simple

trace-theoretic arguments we show that nearly all summands in the specification of T^λ are redundant.

The only product which is delivered by $\overline{WA}$ at port 8 is $p\,1$. This means that:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,8(p) = 0 \quad \text{for } p \neq p\,1. \tag{1}$$

From the behaviour of component WC we conclude:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,6(tc) \leqslant \sigma{\downarrow}c\,3(r). \tag{2}$$

Further we deduce from the behaviour of $\overline{WA}$:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,3(r) \leqslant \sigma{\downarrow}c\,8(p\,1). \tag{3}$$

From (2) and (3) together we conclude that the number of transport commands at port 6 is less or equal to the number of products $p\,1$ that are handed to the transport service at port 8:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,6(tc) \leqslant \sigma{\downarrow}c\,8(p\,1). \tag{4}$$

From the specification of $\overline{WA}$ we learn that A does not deliver products without being asked for:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,8(p\,1) \leqslant \sigma{\downarrow}c\,2(1). \tag{5}$$

Further it follows from the specification of WC that the number of commands given to A by the controller, never exceeds the number of *ar*-signals with more than one:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,2(1) \leqslant \sigma{\downarrow}c\,7(ar) + 1. \tag{6}$$

From (5) and (6) together we conclude:

$$IMPL \textbf{ sat } \sigma{\downarrow}c\,8(p\,1) \leqslant \sigma{\downarrow}c\,7(ar) + 1. \tag{7}$$

From formulas (1), (4) and (7) it follows that nearly all summands in the specification of T^λ are redundant.

$$\tau_I{\circ}\partial_H(WC \| \overline{WA} \| T^\lambda \| WB) = \tau_I{\circ}\partial_H(WC \| \overline{WA} \| T \| WB) \qquad \text{(step 2)}$$

where T is given by:

$$\boxed{T = r\,8(p\,1){\cdot}r\,6(tc){\cdot}s\,9(p\,1){\cdot}s\,7(ar){\cdot}T}$$

The transport service delivers at port 9 only products of type $p\,1$. Therefore all summands in the specification of WB which correspond to the acceptance of another product, are redundant.

$$\tau_I{\circ}\partial_H(WC \| \overline{WA} \| T \| WB) = \tau_I{\circ}\partial_H(WC \| \overline{WA} \| T \| \overline{WB}) \qquad \text{(step 3)}$$

where $\overline{WB}$ is given by:

$$\overline{WB} = \sum_{n=0}^{N} r4(n)\cdot\overline{XB}^{n}$$

$$\overline{XB}^{0} = s5(r)\cdot\overline{WB} \qquad \overline{XB}^{n+1} = r9(p\,1)\cdot s\,10(proc(p\,1))\cdot\overline{XB}^{n}$$

We will now 'zoom in' on components WC, $\overline{WA}$ and T. Define:

$$H' = \{rp(d), sp(d) \mid p \in \{2,3,6,7,8\} \text{ and } d \in \mathbb{D}\} \quad \text{and}$$

$$I' = \{cp(d) \mid p \in \{2,3,6,7,8\} \text{ and } d \in \mathbb{D}\}.$$

Application of the conditional axioms CA gives:

$$\tau_I \circ \partial_H(WC \,\|\, \overline{WA} \,\|\, T \,\|\, \overline{WB}) = \tau_I \circ \partial_H(\tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T) \,\|\, \overline{WB}). \quad \text{(step 4)}$$

Let W be given by:

$$W = \sum_{n=0}^{N} r1(n)\cdot s4(n)\cdot W^{n}$$

$$W^{0} = r5(r)\cdot s0(r)\cdot W \qquad W^{n+1} = \tau\cdot s9(p\,1)\cdot W^{n}$$

We prove that $W = \tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T)$, by showing that process $\tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T)$ satisfies the defining equations of W.

$$\tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T) = \sum_{n=0}^{N} r1(n)\cdot s4(n)\cdot \tau_{I'} \circ \partial_{H'}(XC^{n} \,\|\, \overline{WA} \,\|\, T)$$

$$\tau_{I'} \circ \partial_{H'}(XC^{0} \,\|\, \overline{WA} \,\|\, T) = r5(r)\cdot s0(r)\cdot \tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T)$$

$$\tau_{I'} \circ \partial_{H'}(XC^{n+1} \,\|\, \overline{WA} \,\|\, T) =$$

$$= \tau_{I'}(c2(1)\cdot \partial_{H'}(s3(r)\cdot s6(tc)\cdot r7(ar)\cdot XC^{n} \,\|\, s8(p\,1)\cdot s3(r)\cdot \overline{WA} \,\|\, T))$$

$$= \tau\cdot\tau_{I'}(c8(p\,1)\cdot \partial_{H'}(s3(r)\cdot s6(tc)\cdot r7(ar)\cdot XC^{n} \,\|\, s3(r)\cdot \overline{WA} \,\|\, r6(tc)\cdot s9(p\,1)\cdot s7(ar)\cdot T))$$

$$= \tau\cdot\tau\cdot\tau_{I'}(c3(r)\cdot \partial_{H'}(s6(tc)\cdot r7(ar)\cdot XC^{n} \,\|\, \overline{WA} \,\|\, r6(tc)\cdot s9(p\,1)\cdot s7(ar)\cdot T))$$

$$= \tau\cdot\tau_{I'}(c6(tc)\cdot \partial_{H'}(r7(ar)\cdot XC^{n} \,\|\, \overline{WA} \,\|\, s9(p\,1)\cdot s7(ar)\cdot T))$$

$$= \tau\cdot\tau_{I'}(s9(p\,1)\cdot \partial_{H'}(r7(ar)\cdot XC^{n} \,\|\, \overline{WA} \,\|\, s7(ar)\cdot T))$$

$$= \tau\cdot s9(p\,1)\cdot\tau_{I'}(c7(ar)\cdot \partial_{H'}(XC^{n} \,\|\, \overline{WA} \,\|\, T))$$

$$= \tau\cdot s9(p\,1)\cdot \tau_{I'} \circ \partial_{H'}(XC^{n} \,\|\, \overline{WA} \,\|\, T)$$

We have now derived:

$$\tau_I \circ \partial_H(\tau_{I'} \circ \partial_{H'}(WC \,\|\, \overline{WA} \,\|\, T) \,\|\, \overline{WB}) = \tau_I \circ \partial_H(W \,\|\, \overline{WB}). \quad \text{(step 5)}$$

Let V be given by:

$$\boxed{\begin{aligned} V &= \sum_{n=0}^{N} r1(n)\cdot V^n \\ V^0 &= \tau\cdot s0(r)\cdot V \qquad V^{n+1} = \tau\cdot s10(proc(p1))\cdot V^n \end{aligned}}$$

We show that $\tau_I\circ\partial_H(W\|\overline{WB})$ satisfies the defining equations of V.

$$\begin{aligned} \tau_I\circ\partial_H(W\|\overline{WB}) &= \sum_{n=0}^{N} r1(n)\cdot\tau_I\circ\partial_H(s4(n)\cdot W^n\|(\sum_{m=0}^{N} r4(m)\cdot\overline{XB}^m)) \\ &= \sum_{n=0}^{N} r1(n)\cdot\tau_I(c4(n)\cdot\partial_H(W^n\|\overline{XB}^n)) \\ &= \sum_{n=0}^{N} r1(n)\cdot\tau\cdot\tau_I\circ\partial_H(W^n\|\overline{XB}^n) \end{aligned}$$

$$\tau\cdot\tau_I\circ\partial_H(W^0\|\overline{XB}^0) = \tau\cdot\tau_I(c5(r)\cdot\partial_H(s0(r)\cdot W\|\overline{WB}))\ \tau\cdot s0(r)\cdot\tau_I\circ\partial_H(W\|\overline{WB})$$

$$\begin{aligned} \tau\cdot\tau_I\circ\partial_H(W^{n+1}\|\overline{XB}^{n+1}) &= \tau\cdot\tau_I(c9(p1)\cdot\partial_H(W^n\|s10(proc(p1))\cdot\overline{XB}^n)) \\ &= \tau\cdot s10(proc(p1))\cdot\tau_I\circ\partial_H(W^n\|\overline{XB}^n) \end{aligned}$$

(here we use that $\tau(\tau x\|y)=\tau\tau x\between y=\tau x\between y=\tau(x\|y)$). From the above derivation it follows that:

$$\tau_I\circ\partial_H(W\|\overline{WB}) = V. \qquad \text{(step 6)}$$

We show that $SPEC$ satisfies the defining equations of V.

$$\begin{aligned} SPEC &= \sum_{n=0}^{N} r1(n)\cdot(\tau\cdot SPEC^n\cdot SPEC) \\ \tau\cdot SPEC^0\cdot SPEC &= \tau\cdot s0(r)\cdot SPEC \\ \tau\cdot SPEC^{n+1}\cdot SPEC &= \tau\cdot s10(proc(p1))\cdot(\tau\cdot SPEC^n\cdot SPEC) \end{aligned}$$

Hence:

$$V = SPEC. \quad \square \qquad \text{(step 7)}$$

This example shows that a combination of trace-theoretic arguments and the use of alphabet calculus makes it possible to verify simple systems in a compositional and modular way.

ACKNOWLEDGEMENTS
I would like to thank the participants of the PAM seminar, especially Jos Baeten, Jan Bergstra and Hans Mulder, for their comments on earlier versions of this article.

REFERENCES
1. J.C.M. BAETEN, J.A. BERGSTRA (1988). Global renaming operators in concrete process algebra. *Information and Computation 78(3),* 205-245.
2. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1986). Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae IX(2),* 127-168.
3. J.C.M. BAETEN, J.A. BERGSTRA, J.W. KLOP (1987). On the consistency of Koomen's Fair Abstraction Rule. *Theoretical Computer Science, 51(1/2),* 129-176.
4. F. BIEMANS, P. BLONK (1986). On the formal specification and verification of CIM architectures using LOTOS. *Computers in Industry 7(6),* 491-504.
5. E.W. DIJKSTRA (1976). *A Discipline of Programming,* Prentice-Hall, Englewood Cliff.
6. R.J. VAN GLABBEEK (1987). Bounded nondeterminism and the approximation induction principle in process algebra. F.J. BRANDENBURG, G. VIDAL-NAQUET, M. WIRSING (eds.). *Proceedings STACS 87,* LNCS 247, Springer-Verlag, 336-347.
7. R.J. VAN GLABBEEK, F.W. VAANDRAGER (1987). Petri net models for algebraic theories of concurrency. J.W. DE BAKKER, A.J. NIJMAN, P.C. TRELEAVEN (eds.). *Proceedings PARLE Conference, Eindhoven, Vol. II (Parallel Languages),* LNCS 259, Springer-Verlag, 224-242.
8. R.J. VAN GLABBEEK, F.W. VAANDRAGER (1988). *Modular Specifications in Process Algebra - With Curious Queues,* CWI Report CS-R8821, Centre for Mathematics and Computer Science, Amsterdam. Extended abstract to appear in *Proceedings of the METEOR Workshop on Algebraic Methods: Theory, Tools and Applications,* LNCS, Springer-Verlag.
9. C.A.R. HOARE (1985). *Communicating Sequential Processes,* Prentice-Hall.
10. C.P.J. KOYMANS, J.C. MULDER (1989). *A Modular Approach to Protocol Verification using Process Algebra.* This volume.
11. K.G. LARSEN (1986). *Context-dependent Bisimulation between Processes,* Ph. D. Thesis, Department of Computer Science, University of Edinburgh.
12. K.G. LARSEN, R. MILNER (1987). A complete protocol verification using relativized bisimulation. TH. OTTMANN (ed.). *Proceedings 14th ICALP, Karlsruhe,* LNCS 267, Springer-Verlag, 126-135.
13. S. MAUW (1989). *Process Algebra as a Tool for the Specification and Verification of CIM-architectures.* This volume.
14. E.-R. OLDEROG (1986). Process theory: semantics, specification and verification. J.W. DE BAKKER, W.-P. DE ROEVER, G. ROZENBERG (eds.). *Current Trends in Concurrency,* LNCS 224, Springer-Verlag, 442-509.
15. E.-R. OLDEROG, C.A.R. HOARE (1986). Specification-oriented semantics

for communicating processes. *Acta Informatica 23,* 9-66.

16. F.W. VAANDRAGER (1986). *Verification of Two Communication Protocols by Means of Process Algebra,* CWI Report CS-R8608, Centre for Mathematics and Computer Science, Amsterdam.
17. J. ZWIERS (1988). *Compositionality, Concurrency and Partial Correctness: Proof Theories for Networks of Processes, and their Connection,* Ph.D. Thesis, Technical University Eindhoven.