Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

F.W. Vaandrager

Some observations on redundancy in a context

Computer Science/Department of Software Technology Report CS-R8812 March

69F12, 69F31, 69C24

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 paper 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 $\bar{x}$ which is identical to x, except for the fact that $\bar{x}$ 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 simple workcell architecture is presented.

1985 Mathematical Subject Classification: 68Q10, 68Q60.
1980 Mathematical Subject Classification: 68B10.
1982 CR Categories: F.1.2, F.3.1, C.2.4.
Key Words & Phrases: process algebra, redundancy in a context, trace semantics, modularity, compositionality, computer integrated manufacturing.
Note: Partial support received from the European Communities under ESPRIT project no. 432, An Integrated Formal Approach to Industrial Software Development (METEOR).

§1. Introduction

The discussion of this paper takes place in the framework of ACP, the Algebra of Communicating Processes of Bergstra & Klop. This is not an introductory paper on process algebra. For a survey of the ACP formalism we refer the reader to [3, 5, 6].

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 diagram for the process *IMPL* with the expansion theorem, apply an abstraction operator to this diagram, and then simplify the resulting diagram to the diagram 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 also 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 need for proof methods with a more *modular/compositional* character.

Report CS-R8812
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

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 expression $\partial_H(X_1 \| X_2 \| X_3)$. A possible step in a modular verification could be that X_1 and X_2 are replaced by Y_1 and Y_2. 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 $\tau_I \circ \partial_H(.. \| X_3)$. 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 paper 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.

Alphabet Calculus. A simple calculus, introduced in [4], which is often applied in the setting of process algebra in order to enhance the modularity of verifications, is the alphabet calculus, used in conjunction with the conditional axioms. This calculus makes it possible to use information about the context in the correctness proof of a substitution. Axiom CA2, for instance, says that under certain circumstances actions of a component can be hidden because they will be hidden later on anyway.

$$\frac{\alpha(x) | (\alpha(y) \cap I) = \emptyset}{y = \tau_I(y) \text{ in the context } \tau_I(x \| ..)}$$

Sometimes all information about contexts which is needed in a compositional verification, can be coded in terms of alphabets of processes. However, there are many applications where it is not only important to know which actions can be performed by processes, but also in which order this can be done.

Example. We give a specification of a Dutch coffee machine similar to the one described in [15].

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

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

$$DU = (kof + 30c \cdot kof) \cdot talking \cdot DU$$

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.[1] Communication is given by: $\gamma(kof, \overline{kof}) = kof^*$, $\gamma(30c, \overline{30c}) = 30c^*$,

1. Dutch users do not occur in [15]. In the modelling as presented here, the thrift of the Dutch user is not taken into account:

$\gamma(choc,\overline{choc})=choc^*$. 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:

$$\overline{DU} = 30c \cdot kof \cdot talking \cdot \overline{DU}$$

because

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

Now notice that the correctness of the substitution of $\overline{DU}$ for DU cannot be concluded from information about alphabets alone.

Redundancy in a context. One can say that a process is influenced by the environment (context) in which it is placed, because this environment blocks certain actions at every moment. 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 $\partial_H(..\|y)$ which blocks a whenever the process is in s. In situations like this, the a-step from s is *redundant in the context* $\partial_H(..\|y)$. 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, the alphabet calculus is in often not strong enough. New proof rules are needed here. In this paper 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 [17]. The present paper can be viewed as a thorough revision of section 6 from that paper.

Trace-specifications. It is argued by many authors (see for instance [8]), 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 [12, 15, 16, 18], is to give information about the traces, ready pairs and failure pairs of these processes. This leads to the notation

$$x \text{ sat } S$$

which expresses that process x satisfies property S. When we use the notation in this paper, 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.

A desirable state of affairs is that for every process p, there exists a predicate S_p which expresses *all* properties of p. This observation stimulated many authors to study models where a process *is* the set of its traces, ready pairs, failure pairs, etc., and to build proof systems which are based upon these models. We however, prefer not to follow this approach. In recent years it has become abundantly clear that there are many notions of 'process'. The idea that a process, in general, *is* the set of its traces, ready pairs or failure pairs is just false, for instance because these notions of process do not capture features like real-time and fairness. We are interested in proof rules which express 'universal' truths about processes, and which are not tied to some particular model.

In this paper we use, in verifications, the laws of interleaved bisimulation semantics. However, we conjecture that the proof rule based on trace-specifications, as presented in this paper, also holds in partial order semantics (see [10]). Probably the correctness proof of the CIM-architecture which is presented in section 5, when reorganised a little bit, is also valid in partial order semantics. It is a topic for future research to substantiate these claims.

we can think of an environment where process DU performs 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].

§2. Trace sets

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 $\sigma * \sigma'$, 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: $\sigma \leqslant \rho$ iff, 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 *(an explicit model)*: Let $a, b \in A$, $V, W \in \mathbb{T}$, $\sigma, \sigma_1, \sigma_2 \in T$, $f : A_{\tau\delta} \rightarrow A_{\tau\delta}$ with $f(\tau) = \tau$ and $f(\delta) = \delta$, and let $n \in \mathbb{N}$. First we define some operators on trace sets.

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

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

$$\lambda \| a\sigma = a\sigma \| \lambda = a(\lambda \| \sigma)$$

$$\lambda \| \lambda = \{\lambda\}$$

$$\surd \| \sigma = \sigma \| \surd = \{\sigma\}$$

3. $\rho_f(V) ::= \{\rho_f(\sigma) \mid \sigma \in V\}$. The function ρ_f on traces is given by:

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

$$\rho_f(\lambda) = \lambda$$

$$\rho_f(\surd) = \surd$$

4. $\pi_n(V) ::= \{\sigma \in V \cap A^* \mid \sharp\sigma \leqslant n\} \cup \{\sigma\surd \in V \mid \sharp\sigma \leqslant n\}$[1]
5. $\alpha(V) ::= \{\alpha(\sigma) \mid \sigma \in V\}$. The function $\alpha : T \rightarrow Pow(A)$ is given by:

$$\alpha(a * \sigma) = \{a\} \cup \alpha(\sigma)$$

$$\alpha(\lambda) = \varnothing$$

$$\alpha(\surd) = \varnothing$$

Let $\Sigma(\mathrm{SACP}_\tau)$ ('Stripped' ACP_τ) be the signature obtained from the signature of ACP_τ by deleting the

1. The π_n-operators we define here, satisfy the same axioms as the ones defined in [9]: $\pi_n(\tau) = \tau$, $\pi_0(ax) = \delta$, $\pi_{n+1}(ax) = a \cdot \pi_n(x)$, etc.

operators $\|\!\lfloor$ and $|$. We give an *explicit* or *denotational* model[1] $\mathfrak{A}(TR_{den})$ for the constants and operators in the signature of SACP$_\tau$ + RN + PR + AB with domain $\mathbb{T}$[2]. The interpretation $[\![.]\!]_{tr}$ of (finite) terms in the trace domain $\mathbb{T}$ and the alphabet domain A is given by:

1. $[\![\delta]\!]_{tr} = \{\lambda\}$
2. $[\![\tau]\!]_{tr} = \{\lambda, \surd\}$
3. $[\![a]\!]_{tr} = \{\lambda, a, a\surd\}$
4. $[\![x+y]\!]_{tr} = [\![x]\!]_{tr} \cup [\![y]\!]_{tr}$
5. $[\![x \cdot y]\!]_{tr} = [\![x]\!]_{tr} \cdot [\![y]\!]_{tr}$
6. $[\![x \| y]\!]_{tr} = [\![x]\!]_{tr} \| [\![y]\!]_{tr}$
7. $[\![\rho_f(x)]\!]_{tr} = \rho_f([\![x]\!]_{tr})$
8. $[\![\pi_n(x)]\!]_{tr} = \pi_n([\![x]\!]_{tr})$
9. $[\![\alpha(x)]\!]_{tr} = \alpha([\![x]\!]_{tr})$

2.3. The Trace Operator (TO). The *trace operator* $tr{:}P \rightarrow \mathbb{T}$ 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 processes, $f{:}A_{\tau\delta} \rightarrow A_{\tau\delta}$ with $f(\tau)=\tau$ and $f(\delta)=\delta$, and $n \in \mathbb{N}$)

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

TABLE 1.

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$_\tau$ with recursion and auxiliary operators that can be mapped homomorphically to the trace algebra $\mathfrak{A}(TR_{den})$. This is no serious restriction because all 'interesting' process algebras are in this class. A similar approach is followed in [1].

Examples.

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

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

$$\begin{aligned} tr(ax) &= tr(a) \cdot tr(x) = \{\lambda, a, a\surd\} \cdot tr(x) = \{\lambda, a\} \cup a{*}tr(x) \overset{(1)}{=} \\ &= \{\lambda\} \cup \{a\} \cup a{*}(tr(x) \cup \{\lambda\}) = \{\lambda\} \cup a{*}tr(x) \end{aligned} \tag{2}$$

$$\begin{aligned} tr(<X|X=aX>) &= \bigcup_{n \geqslant 0} \pi_n(tr(X)) = \bigcup_{n \geqslant 0} tr(\pi_n(X)) = \bigcup_{n \geqslant 0} tr(a^n \cdot \delta) = \\ &= \{\lambda\} \cup \{\lambda, a\} \cup \{\lambda, a, aa\} \cup \cdots = \{\lambda, a, aa, \cdots\} \end{aligned} \tag{3}$$

1. The phrases *explicit* and *denotational* express that the operators and constants are defined directly on the domain and not indirectly, for instance by means of action relations on a domain of process expressions (see [9]).
2. The auxiliary operator $\|\!\lfloor$ cannot be added consistently to the model $\mathfrak{A}(TR_{den})$. For a discussion of this issue we refer to [11].

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. When we speak about trace-specifications, a special role will always be played by a variable σ of type trace. A trace-specification S describes the set of traces which, when assigned to free occurrences of σ in S, make the predicate true: $\{\sigma | S\}$. 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, $\sharp\sigma$, $\rho_f(\sigma)$, equality predicates for the integers, actions and traces, and quantification over integers and traces. This syntax is almost equivalent to the syntax proposed in [15], 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 $L = \{\sigma | S_L\}$. In section 4.5 it will be argued that we need such predicates. All predicates that we will use in this paper 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 paper 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 [12] 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. DEFINITION: Let $\sigma \in T$, $B \subseteq A$ and $a \in A$.

1. $\sigma \upharpoonright B$ gives the *projection* of trace σ onto the actions of B:

$$\sigma \upharpoonright B = \tau_{A-B}(\sigma)$$

2. $\sigma \downarrow a$ denotes the number of occurrences of a in σ:

$$\sigma \downarrow a = \begin{cases} \sharp(\sigma \upharpoonright \{a\}) - 1 & \text{if } \sigma = \sigma' \surd \\ \sharp(\sigma \upharpoonright \{a\}) & \text{otherwise} \end{cases}$$

3. Even though our trace-specification language contains no alphabet operator, we can talk about alphabets in predicates:

$$\alpha(\sigma) \subseteq B \Leftrightarrow \sigma \upharpoonright B = \sigma$$

2.4.2. Example. The Dutch coffee machine from example in section 1 satisfies:

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

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.

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

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 [13], such communication functions will be characterised as *trijective*. The assumption that communication is trijective will simplify the discussion of this paper.

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 $\overline{}: R\rightarrow S$ and $^\circ: R\rightarrow C$ such that for every $a,b,c\in A$:

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

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').

3.1.3. Note. In the rest of this paper we assume that communication is trijective.

3.2. Observability. We are interested in the behaviour of a process x when it is placed in a context $..\|y$. 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.

From the fact that a set B of actions is observable, we can conclude that $B\cap B|A=\varnothing$. 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* $..\|y$, 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 $..\|y$, we 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 we can tell which action from x participates in the communication. Observe that the fact that x is observable in context $..\|y$ does not imply that y is observable in context $..\|x$.

3.3. Localisation. The 'localisation' of actions from x in a context $..\|y$ 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 [17]). 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* $\nu(B): A_{\tau\delta} \rightarrow A_{\tau\delta}$ is the renaming function defined by:

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

Example. Consider the example from section 1. The communication function in this example is trijective. Furthermore $\alpha(DU) = \{kof, 30c, talking\}$ is observable. Process DU is observable in the context $..\|KM$. DU is however not observable in the context $..\|(DU\|KM)$. 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 $\partial_H(..\|KM)$. We derive:

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

$$= \rho_{\nu(\alpha(DU))}(30c^* \cdot hum \cdot kof^* \cdot talking \cdot \partial_H(DU\|KM)) =$$

$$= 30c \cdot \tau \cdot kof \cdot talking \cdot \rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM) =$$

$$= 30c \cdot kof \cdot talking \cdot \rho_{\nu(\alpha(DU))} \circ \partial_H(DU\|KM) \overset{RSP}{=} \overline{DU}$$

3.3.2. THEOREM: *Let p,q be closed terms with p observable in context $..\|q$. Then $ACP_\tau + RN + AB \vdash \tau \cdot \rho_{\nu(\alpha(p))}(p\|q) \cdot \delta = \tau \cdot p \cdot \delta$.*
PROOF: Easy. □

3.3.3. THEOREM: *Let x,y be processes, with x observable in context $..\|y$. Then $TO \vdash tr(\rho_{\nu(\alpha(x))}(x\|y)) \subseteq tr(x)$.*
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 $\gamma(a,b) \in A$

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

(Apply induction hypothesis) □

Notice that the $\subseteq$-sign in theorem 3.3.3 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 $tr(x)$.

3.3.4. THEOREM: *Let x,y be processes, with x observable in context ..$\|$y, and let $H \subseteq A$. Then TO $\vdash$ $tr(\rho_{\nu(\alpha(x))} \circ \partial_H(x \| y)) \subseteq tr(x)$.*
PROOF: Just like we did in the proof of theorem 3.3.3, we reformulate the statement. Let $V, W \in \mathbb{T}$ with $\alpha(V)$ observable, and $\alpha(W)$ disjoint from $\alpha(V)$ and $\alpha(V)|A$. 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 3.3.3 we conclude:

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

The following corollary of theorem 3.3.4 plays an important role in this paper 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).

3.3.5. 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

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

§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 paper 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 $X \in C$:

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

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 (over A). 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 the system of equations obtained from E by replacing summand aX_2 in the equation for X_1 by a 'fresh' atom t. Fresh means here that $t \notin \alpha(x)$. Let x,x',y be processes, with $x=<X_0 \mid E>$, $x'=<X_0 \mid E'>$ and x observable in context $..\|y$. Let $H \subseteq A$. The summand aX_2 of $<X_0 \mid E>$ *is redundant in the context* $\partial_H(..\|y)$ if:

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

Comment. One can say that the set $\{\sigma a \mid \sigma t \in tr(x')\}$ is the contribution of the summand aX_2 to $tr(x)$. Theorem 3.3.4 gives that $tr(\rho_{\nu(\alpha(x))} \circ \partial_H(x \| y))$ is also a subset of $tr(x)$. If the summand aX_2 is redundant, this means that all behaviours of x of the form 'go from state X_1 with an a-step to state X_2' are not possible if x is placed in the context $\partial_H(..\|y)$.

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

$$X_0 = a \cdot X_0 + X_1$$

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

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

$$X_0 = a \cdot X_0 + b \cdot c \cdot \delta$$

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

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

If the condition in definition 4.2 that C is isolated would be absent, then the summand $a \cdot X_0$ would (by definition) be redundant in context $\partial_H(..\|\bar{b})$. However, the summand cannot be omitted: outside the cluster it plays an essential role!

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

A redundant summand can be omitted

Below we formally present this principle as a theorem.

4.3. THEOREM: *Let $x = <X_0 | E>$ and $y = <Y_0 | F>$, with E and F guarded recursive specifications, such that x is observable in context $..\|y$. 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 $\bar{E}$ be the recursive specifications which are obtained from E by resp. replacing the summand aX_2 by a fresh atom t and omitting it. Let $x' \equiv <X_0 | E'>$ and $\bar{x} \equiv <X_0 | \bar{E}>$. Suppose that it is provable that the summand aX_2 is redundant: $ACP_\tau + RDP + RN + PR + TO \vdash$*

$$tr(\rho_{\nu(\alpha(x))} \circ \partial_H(x \| y)) \cap \{\sigma a \mid \sigma t \in tr(x')\} = \varnothing \qquad (1)$$

Then: $\mathrm{ACP}_\tau + \mathrm{RDP} + \mathrm{PR} + \mathrm{AIP}^- \vdash$

$$\partial_H(x \| y) = \partial_H(\bar{x} \| y).$$

PROOF: Omitted. □

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 proof techniques which allow us to prove that summands are redundant. The following strategy will work 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 $S(\sigma)$:

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

$$\rho_{\nu(\alpha(x))} \circ \partial_H(x \| y) \text{ sat } \neg S(\sigma)$$

If the cluster C is finite, then $\{\sigma a \mid \sigma t \in tr(x')\}$ 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 of section 1 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 ($DU = UD$):

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

In example 2.4.2 we already observed that:

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

Because of corollary 3.3.5 we also have:

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

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:

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

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

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

Define UD' by:

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

Of course we have

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

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

This example shows how one can give 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 [7, 14].

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 organised) 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 paper, 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.

$$\boxed{\begin{array}{ll} SPEC & = \sum_{n=0}^{N} r1(n)\cdot SPEC^{n}\cdot SPEC \\[1ex] SPEC^{0} & = s0(r) \\[1ex] SPEC^{n+1} & = s10(proc(p1))\cdot SPEC^{n} \end{array}}$$

Via port 1, the workcell accepts an order to produce n products of type $proc(p1)$ 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.

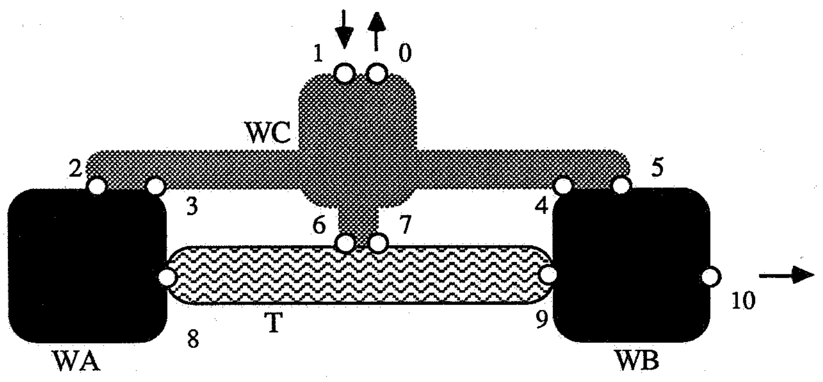

FIGURE 1.

There are four components:	
WA:	Workcell A
WB:	Workcell B
T:	Transport service
WC:	Workcell Controller

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 $p1$. 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.

$$\boxed{\begin{aligned} WA &= \sum_{n=0}^{N} r2(n)\cdot XA^{n} \\ XA^{0} &= s3(r)\cdot WA \\ XA^{n+1} &= s8(p1)\cdot XA^{n} \end{aligned}}$$

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 $p1\in PROD$.

$$\boxed{\begin{aligned} WB &= \sum_{n=0}^{N} r4(n)\cdot XB^{n} \\ XB^{0} &= s5(r)\cdot WB \\ XB^{n+1} &= \sum_{p\in PROD} r9(p)\cdot s10(proc(p))\cdot XB^{n} \end{aligned}}$$

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 $p,q \in PROD$.

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

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

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 $s0(r)$.

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

$$XC^{0} = r5(r)\cdot s0(r)\cdot WC$$

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

5.1.7. $\mathbb{D} = \{n \mid 0 \leq n \leq N\} \cup \{r,tc,ar\} \cup PROD$ 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:

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

Important sets of actions are:

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

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

The implementation as a whole can now be described by:

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

5.1.8. Remark. The implementation description as presented above, is slightly different from the one in [14]. There we find for instance the following system of equations for workcell A.

$$\boxed{\begin{array}{ll} WA & = \sum_{n=0}^{N} r2(n)\cdot XA^n \cdot s3(r)\cdot WA \\ XA^0 & = t \\ XA^{n+1} & = s8(p1)\cdot XA^n \end{array}}$$

Here t is an internal action which is hidden at the system-level. Application of axiom CA2 makes it possible to make this abstraction local to WA. It then becomes obvious that our specification of WA is equivalent to the one in [14]. Besides the t-action, the only difference between both systems is that our system is written down in a more 'regular' way. This has the advantage that in verifications the expressions do not grow too much. Similar differences in specifications also occur with the other components of the system.

5.2. THEOREM *(correctness implementation):* ACP_τ + SC + RDP + PR + AIP^- + AB + 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 $p1$. This means that:

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

Consequently

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

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:

$$\boxed{\overline{WA} = r2(1)\cdot s8(p1)\cdot s3(r)\cdot \overline{WA}}$$

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 $p1$. This means that:

$$IMPL \text{ sat } \sigma{\downarrow}c8(p) = 0 \quad \text{for } p \neq p1 \qquad (1)$$

From the behaviour of component WC we conclude:

$$IMPL \textbf{ sat } \sigma{\downarrow}c6(tc) \leqslant \sigma{\downarrow}c3(r) \tag{2}$$

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

$$IMPL \textbf{ sat } \sigma{\downarrow}c3(r) \leqslant \sigma{\downarrow}c8(p1) \tag{3}$$

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 $p1$ that are handed to the transport service at port 8:

$$IMPL \textbf{ sat } \sigma{\downarrow}c6(tc) \leqslant \sigma{\downarrow}c8(p1) \tag{4}$$

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

$$IMPL \textbf{ sat } \sigma{\downarrow}c8(p1) \leqslant \sigma{\downarrow}c2(1) \tag{5}$$

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:

$$IMPL \textbf{ sat } \sigma{\downarrow}c2(1) \leqslant \sigma{\downarrow}c7(ar) + 1 \tag{6}$$

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

$$IMPL \textbf{ sat } \sigma{\downarrow}c8(p1) \leqslant \sigma{\downarrow}c7(ar) + 1 \tag{7}$$

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

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

where T is given by:

$$\boxed{T = r8(p1) \cdot r6(tc) \cdot s9(p1) \cdot s7(ar) \cdot T}$$

The transport service delivers at port 9 only products of type $p1$. Therefore all summands in the specification of WB which correspond to the acceptance of another product, are redundant.

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

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

$$\boxed{\begin{array}{ll} \overline{WB} & = \sum_{n=0}^{N} r4(n) \cdot \overline{XB}^{n} \\ \overline{XB}^{0} & = s5(r) \cdot \overline{WB} \\ \overline{XB}^{n+1} & = r9(p1) \cdot s10(proc(p1)) \cdot \overline{XB}^{n} \end{array}}$$

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 gives:

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

Let W be given by:

$$\boxed{\begin{array}{ll} W & = \sum_{n=0}^{N} r1(n)\cdot s4(n)\cdot W^n \\ W^0 & = r5(r)\cdot s0(r)\cdot W \\ W^{n+1} & = \tau\cdot s9(p\,1)\cdot W^n \end{array}}$$

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.

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

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

$$\begin{aligned}
&\tau_{I'}\circ\partial_{H'}(XC^{n+1}\|\overline{WA}\|T) = \\
&= \tau_{I'}(c2(1)\cdot\partial_{H'}(s3(r)\cdot s6(tc)\cdot r7(ar)\cdot XC^n\|s8(p\,1)\cdot s3(r)\cdot\overline{WA}\|T)) = \\
&= \tau\cdot\tau_{I'}(c8(p\,1)\cdot\partial_{H'}(s3(r)\cdot s6(tc)\cdot r7(ar)\cdot XC^n\|s3(r)\cdot\overline{WA}\|r6(tc)\cdot s9(p\,1)\cdot s7(ar)\cdot T))= \\
&= \tau\cdot\tau\cdot\tau_{I'}(c3(r)\cdot\partial_{H'}(s6(tc)\cdot r7(ar)\cdot XC^n\|\overline{WA}\|r6(tc)\cdot s9(p\,1)\cdot s7(ar)\cdot T)) = \\
&= \tau\cdot\tau_{I'}(c6(tc)\cdot\partial_{H'}(r7(ar)\cdot XC^n\|\overline{WA}\|s9(p\,1)\cdot s7(ar)\cdot T)) = \\
&= \tau\cdot\tau_{I'}(s9(p\,1)\cdot\partial_{H'}(r7(ar)\cdot XC^n\|\overline{WA}\|s7(ar)\cdot T)) = \\
&= \tau\cdot s9(p\,1)\cdot\tau_{I'}(c7(ar)\cdot\partial_{H'}(XC^n\|\overline{WA}\|T)) = \\
&= \tau\cdot s9(p\,1)\cdot\tau_{I'}\circ\partial_{H'}(XC^n\|\overline{WA}\|T)
\end{aligned}$$

We have now derived:

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

Let V be given by:

$$\boxed{\begin{array}{ll} V & = \sum_{n=0}^{N} r1(n)\cdot V^n \\ V^0 & = \tau\cdot s0(r)\cdot V \\ V^{n+1} & = \tau\cdot s10(proc(p\,1))\cdot V^n \end{array}}$$

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

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

$$\tau_I \circ \partial_H(W^0 \| \overline{XB}^0) = \tau_I(c5(r) \cdot \partial_H(s0(r) \cdot W \| \overline{WB})) =$$
$$= \tau \cdot s0(r) \cdot \tau_I \circ \partial_H(W \| \overline{WB})$$
$$\tau_I \circ \partial_H(W^{n+1} \| \overline{XB}^{n+1}) = \tau \cdot \tau_I(c9(p\,1) \cdot \partial_H(W^n \| s10(proc(p\,1)) \cdot \overline{XB}^n)) =$$
$$= \tau \cdot s10(proc(p\,1)) \cdot \tau_I \circ \partial_H(W^n \| \overline{XB}^n)$$

(the last equality is not completely trivial). 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*.

$$SPEC = \sum_{n=0}^{N} r1(n) \cdot (\tau \cdot SPEC^n \cdot SPEC)$$
$$\tau \cdot SPEC^0 \cdot SPEC = \tau \cdot s0(r) \cdot SPEC$$
$$\tau \cdot SPEC^{n+1} \cdot SPEC = \tau \cdot s10(proc(p\,1)) \cdot (\tau \cdot SPEC^n \cdot SPEC)$$

Hence:

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

□

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 paper.

REFERENCES

1 J.C.M. BAETEN & J.A. BERGSTRA (1987): *Global Renaming Operators in Concrete Process Algebra (revised version).* Report P8709, Programming Research Group, University of Amsterdam, to appear in I&C. This article is a revision of CWI Report CS-R8521.

2 J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1986): *Syntax and Defining Equations for an Interrupt Mechanism in Process Algebra.* Fund. Inf. IX(2), pp. 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), pp. 129-176.

4 J.C.M. BAETEN, J.A. BERGSTRA & J.W. KLOP (1987): *Conditional Axioms and α/β Calculus in Process Algebra.* In: Formal Description of Programming Concepts - III, Proceedings of the third IFIP WG 2.2 working conference, Ebberup 1986 (M. Wirsing, ed.), North-Holland, Amsterdam, pp. 53-75.

5 J.A. BERGSTRA & J.W. KLOP (1985): *Algebra of communicating processes with abstraction.* Theoretical Computer Science 37(1), pp. 77-121.

6 J.A. BERGSTRA & J.W. KLOP (1986): *Process Algebra: Specification and Verification in Bisimulation Semantics.* In: Mathematics and Computer Science II, CWI Monograph 4 (M. Hazewinkel, J.K. Lenstra & L.G.L.T. Meertens, eds.), North-Holland, Amsterdam, pp. 61-94.

7 F. BIEMANS & P. BLONK (1986): *On the formal specification and verification of CIM architectures*

using LOTOS. Computers in Industry 7(6), pp. 491-504.

8 E.W. DIJKSTRA (1976): *A Discipline of Programming,* Prentice-Hall, Englewood Cliff.

9 R.J. VAN GLABBEEK (1987): *Bounded Nondeterminism and the Approximation Induction Principle in Process Algebra.* In: Proceedings STACS 87 (F.J. Brandenburg, G. Vidal-Naquet & M. Wirsing, eds.), LNCS 247, Springer-Verlag, pp. 336-347.

10 R.J. VAN GLABBEEK & F.W. VAANDRAGER (1987): *Petri net models for algebraic theories of concurrency (extended abstract).* In: Proceedings PARLE conference, Eindhoven, Vol. II (Parallel Languages) (J.W. de Bakker, A.J. Nijman & P.C. Treleaven, eds.), LNCS 259, Springer-Verlag, pp. 224-242, full version to appear as CWI Report.

11 R.J. VAN GLABBEEK & F.W. VAANDRAGER (1988): *Curious Queues - With Applications of Module Algebra in Process Algebra.* Report CS-R88.., Centrum voor Wiskunde en Informatica, Amsterdam, to appear.

12 C.A.R. HOARE (1985): *Communicating Sequential Processes,* Prentice-Hall International.

13 C.P.J. KOYMANS & J.C. MULDER (1986): *A Modular Approach to Protocol Verification using Process Algebra.* Logic Group Preprint Series Nr. 6, CIF, State University of Utrecht.

14 S. MAUW (1987): *Process Algebra as a Tool for the Specification and Verification of CIM-Architectures.* Report P8708, Programming Research Group, University of Amsterdam.

15 E.-R. OLDEROG (1986): *Process Theory: Semantics, Specification and Verification.* In: Current Trends in Concurrency (J.W. de Bakker, W.-P. de Roever & G. Rozenberg, eds.), LNCS 224, Springer-Verlag, pp. 442-509.

16 E.-R. OLDEROG & C.A.R. HOARE (1986): *Specification-Oriented Semantics for Communicating Processes.* Acta Informatica 23, pp. 9-66.

17 F.W. VAANDRAGER (1986): *Verification of Two Communication Protocols by Means of Process Algebra.* Report CS-R8608, Centrum voor Wiskunde en Informatica, Amsterdam.

18 J. ZWIERS (1988): *Compositionality, Concurrency and Partial Correctness: Proof theories for networks of processes, and their connection.* Ph.D Thesis, Technical University Eindhoven.