CORRECTNESS OF CONCURRENT PROCESSES

Ernst-Rüdiger Olderog

Centrum voor Wiskunde en Informatica, Amsterdam
Vakgroep Programmatuur, Universiteit van Amsterdam
Institut für Informatik und Praktische Informatik, Universität Kiel

ABSTRACT. A new notion of correctness for concurrent processes is introduced and investigated. It is a relationship P *sat* S between process terms P built up from operators of CCS [Mi 80], CSP [Ho 85] and COSY [LTS 79] and logical formulas S specifying sets of finite communication sequences as in [Zw 89]. The definition of P *sat* S is based on a Petri net semantics for process terms [Ol 89]. The main point is that P *sat* S requires a simple liveness property of the net denoted by P. This implies that P is divergence free and externally deterministic.

Process correctness P *sat* S determines a new semantic model for process terms and logical formulas. It is a modification $\mathfrak{R}^*$ of the readiness semantics [OH 86] which is fully abstract with respect to the relation P *sat* S. The model $\mathfrak{R}^*$ abstracts from the concurrent behaviour of process terms and certain aspects of their internal activity. In $\mathfrak{R}^*$ process correctness P *sat* S boils down to semantic equality: $\mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![S]\!]$. The modified readiness equivalence is closely related to failure equivalence [BHR 84] and strong testing equivalence [DH 84].

1. INTRODUCTION

A process is designed to serve the needs of one or more users. Internally it may exhibit a complicated, nondeterministic and concurrent behaviour. However, for the users only its externally visible reactions to communications are relevant. In particular, such reactions should occur within a finite amount of time. *Process correctness* links the internal process behaviour to the external communication behaviour.

Formally, it is a relationship between processes and specifications which states when a given process P *satisfies* or is *correct with respect to* a given specification S, abbreviated

P *sat* S.

Every notion of process correctness brings about some abstraction from the internal process behaviour according to the following principle:

> For a process the internal structure is irrelevant as long as
> it exhibits the specified communication behaviour.

The purpose of this paper is to present a simple new notion of process correctness and investigate its impact on abstraction.

To motivate this notion, we stipulate a rudimentary user interface of processes consisting of the following:

(1) a power switch for starting and halting the process (switch on or off),
(2) a stability light that indicates when the internal process activity has ceased, and
(3) communication buttons, one for each communication the process may engage in. A communication is possible only when the stability light is on and it is done by depressing the corresponding communciation button.

Processes may have more comfortable user interfaces, but we rely only on the above one.

To define correctness, we have to discuss what the communication behaviour of such a process is. Many answers are possible and meaningful. We aim at a simple, but widely applicable definition and therefore let it be a set of finite communication sequences that are possible between user and process. These sequences are known as *histories* or *traces* [Ho 78]. Since tracces are insensitive to intervening internal actions and concurrent process activities, this definition achieves abstraction from both internal activity and concurrency. Our viewpoint is here that internal activity and concurrency are only part of the process construction, not of the specified communication behaviour.

Of course, other viewpoints are possible. For example, in the work of Mazurkiewicz [Mz 77] even the word "trace" is used for something more elaborate, viz. the equivalence class of finite communication sequences modulo an independence relation on communications expressing concurrency. To avoid confusion, we prefer to call these equivalence classes "Mazurkiewicz-traces" and reserve the word "trace" for finite sequences.

As specification language for trace sets we use a *many-sorted first-order predicate logic*. Since its main sort is "trace", it is called *trace logic* and its formulas are called *trace formulas*. Informal use of trace logic appears in a number of papers (e.g. [CHo 81, MC 81, Os 83, Sn 85, Rm 87, WGS 87]). Precise syntax and semantics, however, is given only in [Zw 89]. We shall adopt Zwiers' proposal, but we need only a simplified version of it because we deal here only with atomic communications instead of messages sent along channels.

As description language for processes we use terms built up from operators of CCS, CSP and COSY [Mi 80, Ho 85, LTS 79]. The operational behaviour of such process terms will be described by labelled transitions of Petri nets. Full details of this approach are given in [Ol 88/89, Ol 89]. With these preparations, we can define process correctness as a relationship

P *sat* S.

between process terms and trace formulas. The main point is how we use the trace formulas S. In most previous papers [CHo 81, MC 81, Os 83, ZRE 85, Zw 89] trace formulas express only *safety properties* or *partial correctness* (cf. [OL 82]). Then P *sat* S if every trace of P satisfies the formula S. This does not exclude the possibility that P diverges or deadlocks. As a consequence, there exists a single process term which satisfies every trace specification with the same alphabet. Such a process term is called a *miracle* after Dijkstra [Di 76].

This is unsatisfactory because we would like to use the notion of process correctness also for *process construction*, i.e. given a trace formula S construct a process term P with

P *sat* S. With miracles this task becomes trivial and meaningless. Therefore we shall be more demanding and use trace formulas to express also a simple type of *liveness property* implying *total correctness* (cf. [OL 82]). Essentially, P *sat* S requires the following:

* Safety: P may only engage in traces satisfying S.
* Liveness: P must engage in every trace satisfying S.

The notions of "may" and "must" are defined by looking at the Petri net transitions of P. The terminology of "may" and "must" originates from [DH 88] but the details are different here. The liveness condition is due to [OH 86] and related to the idea of Misra and Chandy to use so-called *quiescent* infinite trace specifications to express liveness in the setting of asynchronous communication (see [Jo 87]). It implies that every process P satisfying a trace formula S is divergence free and *externally deterministic*. That is: in every run of the process the user has exactly the same possibilities of communication, no matter which actions the process has pursued internally. This implies deadlock freedom of P.

Thus in our approach trace formulas can specify only a subset of processes. We are interested in this subset because, as demonstrated in [OI 88/89], it has many applications and yields simple compositional transformation rules for process construction and verification. We believe that in computing it is essential to identify subclasses of problems or programs where things work better than in the general case.

2. TRACE LOGIC

We start from an infinite set Comm of unstructured *communications* with typical elements a, b. By a *communication alphabet* or simply *alphabet* we mean a finite subset of Comm. We let letters A, B range over alphabets. Syntax and semantics of trace logic we adopt from Zwiers [Zw 89]. It is a many-sorted predicate logic with the following sorts:

trace	(finite communication sequences)
nat	(natural numbers)
comm	(communications)
log	(logical values)

Trace logic then consists of sorted expressions built up from sorted constants, variables and operator symbols. For notational convenience, trace formulas count here as expressions of sort *log*.

All communications appear as constants of sort *trace* and *comm*, and all natural numbers $k \geq 0$ appear as constants of sort *nat*. The set Var of variables is partitioned into a set Var:*trace* of variables t of sort *trace* and a set Var:*nat* of variables n of sort *nat*. Among the trace variables there is a *distinguished trace variable* called h; it will be used in the definiton of trace *specification*. For all communication alphabets A and all communications a, b there are unary operator symbols $\cdot\restriction A$ and $\cdot[b/a]$ of sort *trace* —> *trace*. Further on, there are binary operator symbols $\cdot.\cdot$ of sort *trace* × *trace* —> *trace* and $\cdot[\cdot]$ of sort *trace* × *nat* —> *comm*, and a unary operator symbol $|\cdot|$ of sort *trace* —> *nat*. The remaining symbols used in trace logic are all standard.

Definition. The syntax of trace logic is given by a set

$$\text{Exp} = \text{Exp:}\mathit{trace} \cup \text{Exp:}\mathit{nat} \cup \text{Exp:}\mathit{comm} \cup \text{Exp:}\mathit{log}$$

of *expressions* ranged over by xe. The constituents of Exp are defined as follows.

(1) The set Exp:*trace* of *trace expressions* consists of all expressions te of the form

$$\text{te::} = \varepsilon \mid \text{a} \mid \text{t} \mid \text{te}_1 \,.\, \text{te}_2 \mid \text{te}\restriction \text{A} \mid \text{te[b/a]}$$

where every trace variable t in te occurs within a subexpression of the form $\text{te}_0\restriction \text{A}$.

(2) The set Exp:*nat* of *natural number* consists of the following expressions ne:

$$\text{ne::} = \text{k} \mid \text{n} \mid \text{ne}_1 + \text{ne}_2 \mid \text{ne}_1 * \text{ne}_2 \mid |\text{te}|$$

(3) The set Exp:*comm* of *communication expressions* consists of the following expressions ce:

$$\text{ce::} = \text{a} \mid \text{te[ne]}$$

(4) The set Exp: *log* of *trace formulas* or *logical expressions* consists of the following expressions le:

$$\begin{aligned}\text{le::} = {} & \mathit{true} \mid \text{te}_1 \le \text{te}_2 \mid \text{ne}_1 \le \text{ne}_2 \mid \text{ce}_1 = \text{ce}_2 \\ & \mid \neg\ \text{le} \mid \text{le}_1 \wedge \text{le}_2 \mid \exists \text{t.}\ \text{le} \mid \exists \text{n.}\ \text{le}\end{aligned}$$

□

Let xe{te/t} denote the result of *substituting* the trace expression te for every free occurrence of the trace variable t in xe. Furthermore, let xe{b/a} denote the result of literally replacing every occurrence of the communication a in xe by b.

The *standard semantics* or *interpretation of trace logic* is introduced along the lines of Tarski's semantic definition for predicate logic. It is a mapping

$$\mathfrak{I} : \text{Exp} \longrightarrow (\text{Env}_{\mathfrak{I}} \longrightarrow \text{DOM}_{\mathfrak{I}})$$

assigning a value to every expression with the help of so-called *environments*. These are mappings

$$\rho \in \text{Env}_{\mathfrak{I}} = \text{Var} \longrightarrow \text{DOM}_{\mathfrak{I}}$$

assigning values to the free variables in expressions. The semantic domain of $\mathfrak{I}$ is

$$\text{DOM}_{\mathfrak{I}} = \text{Comm}^* \cup \mathbb{N}_0 \cup \text{Comm} \cup \{\bot\} \cup \{\text{true, false}\},$$

and the environments ρ respect sorts, i.e. trace variables t get values in Comm^* and natural number variables n get values in $\mathbb{N}_0$.

Definition. With the above conventions the standard semantics $\mathfrak{I}$ of trace logic is defined as follows.

(1) *Semantics of trace expressions* yielding values in Comm^*:

$\mathfrak{I}[\![\varepsilon]\!](\rho) = \varepsilon$, the empty trace

$\mathfrak{I}[\![\text{a}]\!](\rho) = \text{a}$

$\mathfrak{I}[\![\text{t}]\!](\rho) = \rho(\text{t})$

$\mathfrak{I}[\![\text{te}_1 \,.\, \text{te}_2\ ]\!](\rho) = \mathfrak{I}[\![\text{te}_1\ ]\!](\rho) \,._{\mathfrak{I}}\, \mathfrak{I}[\![\text{te}_2\ ]\!](\rho)$, the *concatenation* of the traces

$\mathfrak{I}[\![\text{te}\restriction \text{A}\ ]\!](\rho) = \mathfrak{I}[\![\text{te}]\!](\rho) \restriction_{\mathfrak{I}} \text{A}$, the *projection* onto A, i.e. with all communications outside A removed

$\mathfrak{I}[\![\text{te[b/a]}]\!](\rho) = \mathfrak{I}[\![\text{te}]\!](\rho)\ \{\text{b/a}\}$, i.e. every occurrence of a is *renamed* into b. Brackets [...] denote an unevaluated renaming operator and brackets {...} its evaluation.

(2) *Semantics of natural number expressions* yielding values in $\mathbb{N}_0$:

$\mathfrak{I}[\![k]\!](\rho) = k$ for $k \in \mathbb{N}_0$

$\mathfrak{I}[\![n]\!](\rho) = \rho(n)$

$\mathfrak{I}[\![|te|]\!](\rho) = |\mathfrak{I}[\![te]\!](\rho)|_{\mathfrak{I}}$, the *length* of the trace

Expressions $ne_1 + ne_2$ and $ne_1 * ne_2$ are interpreted as addition and multiplication.

(3) *Semantics of communication expressions* yielding values in Comm $\cup$ {$\perp$}:

$\mathfrak{I}[\![a]\!](\rho) = a$

$\mathfrak{I}[\![te[ne]]\!](\rho) = \mathfrak{I}[\![te]\!](\rho)[\mathfrak{I}[\![ne]\!](\rho)]_{\mathfrak{I}}$, the *selection* of the $\mathfrak{I}[\![ne]\!](\rho)$-th element of the trace $\mathfrak{I}[\![ne]\!](\rho)$ if it exists and $\perp$ otherwise

(4) *Semantics of trace formulas* yielding values in {true, false}:

$\mathfrak{I}[\![true]\!](\rho)$ = true

$\mathfrak{I}[\![te_1 \leq te_2]\!](\rho) = (\ \mathfrak{I}[\![te_1]\!](\rho) \leq_{\mathfrak{I}} \mathfrak{I}[\![te_2]\!](\rho))$, the *prefix* relation on Comm*

$\mathfrak{I}[\![ne_1 \leq ne_2]\!](\rho) = (\ \mathfrak{I}[\![ne_1]\!](\rho) \leq_{\mathfrak{I}} \mathfrak{I}[\![ne_2]\!](\rho))$, the standard ordering relation on $\mathbb{N}_0$

$\mathfrak{I}[\![ce_1 = ce_2]\!](\rho) = (\ \mathfrak{I}[\![ce_1]\!](\rho) =_{\mathfrak{I}} \mathfrak{I}[\![ce_2]\!](\rho))$, the *strong, non-strict equality* on $DOM_{\mathfrak{I}}$. Thus a value $\perp$, which is possible for a communication expression, does not propagate to the logical level.

Formulas $\neg$ le, $le_1 \wedge le_2$, $\exists$t.le, $\exists$n. le are interpreted as negation, conjunction and existential quantification over Comm and $\mathbb{N}_0$, respectively.

(5) A trace formula le is called *valid*, abbreviated $\models$le, if $\mathfrak{I}[\![le]\!](\rho)$ = true for all environments ρ. □

How to use trace logic for the specification of trace sets? The answer is that we use a certain subset of trace formulas.

Definition. The set Spec of *trace specifications* ranged over by S, T, U consists of all trace formulas where at most the distinguished variables h of sort *trace* is free. □

Thus the logical value $\mathfrak{I}[\![S]\!](\rho)$ of a trace specification S depends only on the trace value $\rho(h)$. We say that a trace $\mathfrak{h} \in$ Comm* *satisfies* S and write $\mathfrak{h} \models S$ if $\mathfrak{I}[\![S]\!](\rho)$ = true for $\rho(h) = \mathfrak{h}$. Note the following relationship between satisfaction and validity:

$$\mathfrak{h} \models S \quad \text{iff} \quad \models S\ \{\ \mathfrak{h}/h\ \}$$

A trace specification S specifies the set of all traces satisfying S. In fact, wether or not a trace satisfies a trace specification S depends only on the trace value within the *projection alphabet* $\alpha(S)$. This is the smallest set of communications such that h is accessed only via trace projections within $\alpha(S)$. The definition is not straightforward because expressions allow an arbitrary nesting of projection and renaming operators. Consider for example

$$S = (\langle lk.h\rangle \restriction \{dn\} \leq (\langle lk.h\rangle\ dn/lk]) \restriction \{lk,\ up\}.$$

Should the communication lk appear in $\alpha(S)$ or not? To solve this question, we follow [Zw 89] and first convert every expression into a cetain normal form where all trace projections $\cdot \restriction A$ are adjacent to the trace variables.

Definition. A *trace expression* te is called *normal* if it can be generated by the following syntax rules:

$$te ::= \varepsilon \mid a \mid t \restriction A \mid te_1 \,.\, te_2 \mid te[b/a].$$

An arbitrary *expression* xe is *normal* if every maximal trace expression te in xe is normal. Maximal means that te is not contained in a larger trace expression in xe. □

Every other expression xe can be converted into a unique normal expression, called its *normal form* and denoted by xe_{norm}. This convertion is done by applying algebraic laws which move all projections $\cdot \restriction A$ in the trace expressions of xe down to the trace variables.

Definition. For normal trace expressions te the *projection alphabet* or simply *alphabet* $\alpha(te)$ is defined inductively as follows:

$$\alpha(\varepsilon) = \alpha(a) = \Phi$$
$$\alpha(h \restriction A) = A$$
$$\alpha(t \restriction A) = \Phi \text{ if } t \neq h$$
$$\alpha(te_1 \,.\, te_2) = \alpha(te_1) \cup \alpha(te_2)$$
$$\alpha(te[b/a]) = \alpha(te)$$

For arbitrary trace expressions te the alphabet is given by $\alpha(te) = \alpha(te_{norm})$. For arbitrary expressions (in particular trace specifications) xe the alphabet is

$$\alpha(xe) = \bigcup \alpha(te)$$

where the union is taken over all maximal trace expressions te in xe which contain an occurrence of h that is free in xe. If such a trace expression does not exist, the alphabet $\alpha(xe)$ is empty. □

Example. We determine the projection alphabet $\alpha(S)$ of the expression

$$S = (1k.h)\restriction\{dn\} \leq ((1k.h)[dn/1k])\restriction\{1k, up\} \,.$$

Maximal trace expressions of S are te1 = $(1k.h)\restriction\{dn\}$ and te2 = $((1k.h)[dn/1k])\restriction\{1k, up\}$. Their normal forms are

$$te1_{norm} = \varepsilon.h\restriction\{dn\} \quad \text{and} \quad te2_{norm} = \varepsilon.h\restriction\{up\}.$$

Thus we obtain $\alpha(S) = \alpha(te1) \cup \alpha(te2) = \alpha(te1_{norm}) \cup \alpha(te2_{norm}) = \{dn, up\}$. □

Projection Lemma. Let S be a trace specification. Then

$$\eta \vDash S \quad \text{iff} \quad \eta \restriction \alpha(S) \vDash S$$

for all traces $\eta \in Comm^*$. □

Since trace logic includes the standard interpretation of Peano arithmetic, viz. the model $(\mathbb{N}_0, 0, 1, +_{\mathfrak{Z}}, *_{\mathfrak{Z}}, =_{\mathfrak{Z}})$, trace specifications are very expressive. The following theorem is essentially stated in [Zw 89].

Expressiveness Theorem. Let $\mathfrak{T} \subseteq A^*$ be a recursively enumerable set of traces over the alphabet A. Then there exists a trace specification TRACE($\mathfrak{T}$) with projection alphabet α(TRACE($\mathfrak{T}$)) = A such that

$$\mathfrak{h} \in \mathfrak{T} \quad \text{iff} \quad \mathfrak{h} \models \text{TRACE}(\mathfrak{T})$$

for all traces $\mathfrak{h} \in A^*$. The same is true for sets $\mathfrak{T} \subseteq A^*$ whose complement in A^* is recursively enumerable. □

For practical specification, such a general expressiveness result is not very helpful. Then a concise and clear notation is important. We use the following:

* Natural number expressions *counting* the number of communications in a trace:

$$a \# te =_{df} | te\restriction \{a\} |$$

* Communication expressions *selecting* specific elements of a trace: e.g.

$$\mathit{last}\ te =_{df} te[|te|]$$

* Extended syntax for logical expressions: e.g. for $k \geq 3$

$$ne_1 \leq \ldots \leq ne_k =_{df} \bigwedge_{j=1}^{k-1} ne_j \leq ne_{j+1}$$

* *Regular expressions* denoting sets of traces.

3. PROCESS TERMS

Process terms are recursive terms over a certain signature of operator symbols taken from Lauer's COSY [LTS 79, Be 87], Milner's CCS [Mi 80] and Hoare's CSP as in [Ho 85]. More specifically, we take the parallel composition || from COSY, prefix a., choice + and action morphism [φ] from CCS, and deadlock *stop* : A , divergence *div* : A and the idea of using communication alphabets to state certain context-sensitive restrictions on process terms from CSP.

To the set Comm of communication we add an element $\tau \in$ Comm yielding the set Act = Comm $\cup \{\tau\}$ of *actions*. The element τ is called *internal* action and the communications are also called *external* actions. We let u,v range over Act. As before let a,b range over Comm and A,B over communication alphabets. The set of *(process) identifiers* is denoted by Idf; it is partitioned into sets Idf:A $\subseteq$ Idf of *identifiers with alphabet* A, one for each communication alphabet A. We let X,Y,Z range over Idf. By an *action morphism* we mean a mapping φ: Act ⟶ Act with $\varphi(\tau) = \tau$ and $\varphi(a) \neq a$ for only finitely many $a \in$ Comm. Communications a with $\varphi(a) = \tau$ are said to be hidden via φ and communications a with $\varphi(a) = b$ for some $b \neq a$ are said to be *renamed* into b via φ.

Definition. The set Rec of *(recursive) terms*, with typical elements P,Q,R, consists of all terms generated by the following context-free production rules:

P ::=	*stop* : A	(deadlock)
	| *div* : A	(divergence)

| a.P	(prefix)
| P + Q	(choice)
| P || Q	(paralellism)
| P [φ]	(morphism)
| X	(identifier)
| μX.P	(recursion)

□

An occurence of an identifier X in a term P is said to be *bound* if it occurs in P within a subterm of the form μX.Q. Otherwise the occurence is said to be *free*. A term P ∈ Rec without free occurences of identifiers is called *closed*. P{Q/X} denotes the result of *substituting* Q for every free occurence of X in P.

A term P is called *action-guarded* if in every recursive subterm μX.Q of P every free occurence of X in Q occurs within a subterm of the form a.R of Q. E.g. μX. a. X is action-guarded, but a.μX. X is not.

To every term P we assign a communication alphabet α(P) defined inductively as follows :

$$\alpha(stop : A) = \alpha(div : A) = A ,$$
$$\alpha(a.P) = \{a\} \cup \alpha(P) ,$$
$$\alpha(P+Q) = \alpha(P \,||\, Q) = \alpha(P) \cup \alpha(Q) ,$$
$$\alpha(P[\varphi]) = \varphi(\alpha(P)) - \{\tau\},$$
$$\alpha(X) = A \text{ if } X \in Idf(A),$$
$$\alpha(\mu X.P) = \alpha(X) \cup \alpha(P).$$

Definition. A *process term* is a term P∈Rec which satisfies the following context-sensitive restrictions:

(1) P is action-guarded,
(2) every subterm a.Q of P satisfies a∈α(Q),
(3) every subterm Q+R of P satisfies α(Q) = α(R),
(4) every subterm μX.Q of P satisfies α(X) = α(P).

Let Proc denote the set of all process terms and CProc the set of all closed process terms. □

The semantics of a process term P will be defined as a certain Petri net $\mathfrak{N}[\![P]\!]$. As nets we consider here *labelled place/transition nets* with arc weight 1 and place capacity ω [Re 85] but we will mainly work in the subclass of safe Petri nets. We deviate slightly from the standard definition and use the following one which is inspired by [Go 88].

Definition. A *Petri net* or simply *net* is a structure $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$ where
(1) A is a communication alphabet;
(2) Pl is a possibly infinite set of *places*;
(3) $\longrightarrow \subseteq \mathfrak{P}_{nf}(Pl) \times (A \cup \{\tau\}) \times \mathfrak{P}_{nf}(Pl)$ is the *transition relation*;
(4) $M_0 \in \mathfrak{P}_{nf}(Pl)$ is the *initial marking*. □

Here $\mathfrak{P}_{nf}(Pl)$ denotes the set of all non-empty, finite subsets of Pl. An element (I, u, O) $\in$ —> is called a *transition (labelled with the action u)* and will usually be written as

$$I \xrightarrow{u} O .$$

For a transition $t = I \xrightarrow{u} O$ its *preset* or *input* is given by pre(t) = I, its *postset* or *output* by post(t) = O and its action by act(t) = u.

The graphical representation of a net $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$ is as follows. We draw a rectangular box subdivided into an upper part diaplaying the alphabet A and a lower part displaying the remaining components Pl, —> and M_0 in the usual way. Thus places $p \in Pl$ are represented as *circles* with the name "p" outside and transitions

$$t = \{ p_1, \ldots, p_m \} \xrightarrow{u} \{ q_1, \ldots, q_n \}$$

as *boxes* carrying the label "u" inside and connected via directed arcs to the places in pre(t) and post(t). Since pre(t) and post(t) need not be disjoint, some of the outgoing arcs of u actually point back to places in pre(t) and thus introduce *cycles*. The initial marking is represented by putting a token into the circle of each $p \in M$.

The dynamic behaviour of a Petri net is defined by its *token game*; it describes which transitions are concurrently enabled at a given marking and what the result of their concurrent execution is. Though the initial marking of a net is defined to be a set of places, the token game can result in more general markings, viz. multisets.

Consider a net $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$. A *marking* or *case* or *global state* of $\mathfrak{N}$ is a *multiset (over Pl)*, i.e. a mapping M: Pl —> $\mathbb{N}_0$. Graphically, such a marking M is represented by putting M(p) tokens into the circle drawn for each $p \in Pl$. For simplicity any set $N \subseteq Pl$, e.g. the initial marking M_0, will be identified with the multiset given by the characteristic function of N: N(p)=1 for $p \in N$ and N(p)=0 otherwise. For multisets M and N let

$$M \subseteq N,\ M \cup N \text{ and } M - N$$

denote *multiset inclusion, union* and *difference*. If M and N are sets then $M \subseteq N$ an M - N are just set inclusion and difference whereas $M \cup N$ in general differs from set-theoretic union. We write $p \in M$ if $M(p) \geq 1$.

A *global transition* of $\mathfrak{N}$ is any non-empty, finite set $\mathfrak{T}$ of transitions of $\mathfrak{N}$. Define by using multiset union

$$pre(\mathfrak{T}) = \bigsqcup_{t \in \mathfrak{T}} pre(t)$$

and analogously for post($\mathfrak{T}$) and act($\mathfrak{T}$).

Definition. Let $\mathfrak{N}$ be a net, $\mathfrak{T}$ be a global transition of $\mathfrak{N}$ and M be a marking of $\mathfrak{N}$. Then

(1) the transitions in $\mathfrak{T}$ are *concurrently enabled at M* or simply $\mathfrak{T}$ is *enabled at M* if $pre(\mathfrak{T}) \subseteq M$,

(2) if enabled at M, the *concurrent execution* of the transitions in $\mathfrak{T}$ transforms M into a new marking M' of $\mathfrak{N}$; this is also called a *step from M to M' in (the token game of)* $\mathfrak{N}$. In symbols:

$$M \xrightarrow{\mathfrak{T}} M' \text{ in } \mathfrak{N}$$

if $pre(\mathfrak{T}) \subseteq M$ and $M' = (M - pre(\mathfrak{T})) \cup post(\mathfrak{T})$. For $\mathfrak{T} = \{ t \}$ we write $M \xrightarrow{t} M'$ instead. □

We distinguish two notions of reachability for nets $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$:

A *(dynamically) reachable marking* of $\mathfrak{N}$ is a marking M for which there exist intermediate markings $M_1, ..., M_n$ and global transitions $\mathfrak{T}_1, ..., \mathfrak{T}_n$ with

$$(*) \qquad M_0 \xrightarrow{\mathfrak{T}_1} M_1 \xrightarrow{\mathfrak{T}_2} \cdots \xrightarrow{\mathfrak{T}_n} M_n = M$$

Let $mark(\mathfrak{N})$ denote the set of *reachable* markings of $\mathfrak{N}$. Note that the set $mark(\mathfrak{N})$ does not change if in (*) we consider only singleton transitions $\mathfrak{T}_i = \{ t_i \}$.

The set $place(\mathfrak{N})$ of *statically reachable places* of $\mathfrak{N}$ is the smallest subset of Pl satisfying

(1) $M \subseteq place(\mathfrak{N})$,

(2) If $I \subseteq place(\mathfrak{N})$ and $I \xrightarrow{u} O$ for some $u \in A \cup \{ \tau \}$ and $O \subseteq Pl$ then also $O \subseteq place(\mathfrak{N})$.

The term "statical" emphasizes that, by (2), the set $place(\mathfrak{N})$ is closed under the execution of any transition $t = I \xrightarrow{u} O$ independently of whether t is ever enabled at some dynamically reachable marking of $\mathfrak{N}$. Thus $place(\mathfrak{N}) \subseteq \{ p \mid \exists M \in mark(\mathfrak{N}): p \in M \}$ and in general this inclusion is proper.

In the following we shall mainly work with safe nets where multiple tokens per place do not occur. Formally, a net $\mathfrak{N}$ is *safe* if

$$\forall M \in mark(\mathfrak{N}) \ \forall p \in Pl: M(p) \leq 1.$$

Thus in a safe net all reachable markings are sets.

Moreover, we mostly wish to ignore the identity of places and forget about places that are not statically reachable. We do this by introducing suitable notions of isomorphism and abstract net.

Definition. Two nets $\mathfrak{N}_i = (A_i, Pl_i, \longrightarrow_i, M_{0i})$, i=1,2, are *weakly isomorphic*, abbreviated

$$\mathfrak{N}_1 =_{isom} \mathfrak{N}_2,$$

if $A_1 = A_2$ and there exists a bijection $\beta : place(\mathfrak{N}_1) \longrightarrow place(\mathfrak{N}_2)$ such that

$$\beta(M_{01}) = M_{02}$$

and for all $I, O \subseteq place(\mathfrak{N}_1)$ and all $u \in A \cup \{ \tau \}$

$$I \xrightarrow{u}_1 O \text{ iff } \beta(I) \xrightarrow{u}_2 \beta(O)$$

where $\beta(M_{01})$, $\beta(I)$, $\beta(O)$ are understood elementwise. The bijection β is called an *weak isomorphism between $\mathfrak{N}_1$ and $\mathfrak{N}_2$* . □

Clearly, $=_{isom}$ is an equivalence relation. An *abstract net* is defined as the isomorphism class

$$[\mathfrak{N}]_{=_{isom}} = \{ \mathfrak{N}' \mid \mathfrak{N} =_{isom} \mathfrak{N}' \}$$

of a net $\mathfrak{N}$. It will be written shorter as $[\mathfrak{N}]$. For abstract nets, we use the same graphical representation as for nets; we only have to make sure that all places are statically reachable and eliminate their names. Most concepts for nets can be lifted in a straightforward way to abstract nets. For example, we shall call an abstract net $[\mathfrak{N}]$ safe, if $\mathfrak{N}$ is safe. Let Net denote the set of nets and ANet the set of abstract nets.

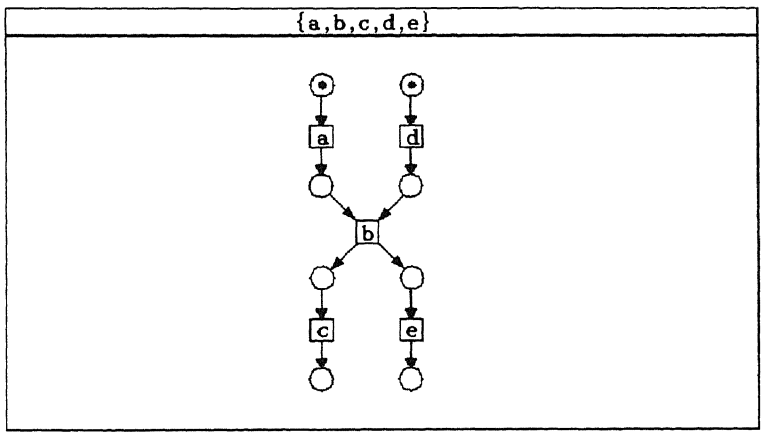
The semantics of process terms is a mapping $\mathfrak{N}[\![\cdot]\!] : CProc \longrightarrow ANet$ which assigns to every $P \in CProc$ an safe abstract net of the form

$$\mathfrak{N}[\![P]\!] = [(\alpha(P), Pl, \longrightarrow, M_0)].$$

For the definition of the components Pl, $\longrightarrow$ and M_0 we refer to [Ol 89]. Here we have space only for an example.

Example. Let P = a.b.c.*stop* : {a, b, c} || d.b.e.*stop* : {d, b, e}. Then

$\mathfrak{N}[\![P]\!]$ =



4. PROCESS CORRECTNESS

In this section we define our notion of process correctness P *sat* S. Let us begin with an informal explanation by considering once more the user interface of the process P shown in the introduction. Consider now a communication trace $\eta = a_1 \ldots a_n$ over $\alpha(P)$. We say that P *may engage* in η if there exists a transition sequence of the process where the user was able to depress the communication buttons $a_1 \ldots a_n$ in that order. We say that P *must engage* in η if the following holds: When started the process eventually becomes stable. Then it is possible for the user to communicate a_1 by depressing the corresponding communication button. Now the process may engage in some internal activity, but eventually it becomes stable again. Then it is ready for the next communication a_2 with the user, etc. for a_3, ..., a_n. Also after the last communication a_n the process eventually becomes stable again. Summarising, in every transition sequence of the process the user is able to depress

the communication buttons a_1, ..., a_n in that order after which the process eventually becomes stable. Stability can be viewed as an acknowledgement of the process for a successful communication with the user. We say that P is *stable immediately* if the stability light goes on immediately after switching the process on. These explanations should suffice to appreciate the following definition of process correctness.

Definition. Consider a closed process term P and a trace specification S. Then

$$P \; sat \; S$$

if $\alpha(P) = \alpha(S)$ and the following conditions hold:

(1) *Safety.* For every trace $\eta \in \alpha(P)^*$ whenever P may engage in η then $\eta \models S$.

(2) *Liveness.* For every trace $\eta \in \alpha(S)^*$ whenever *pref* $\eta \models S$ then P must engage in η. The notation *pref* $\eta \models S$ means that η and all its prefixes satisfy S.

(3) *Stability.* P is stable immediately. □

The distinction between safety and liveness properties of concurrent processes is due to Lamport (see e.g. [OL 82]). Following Lamport, a safety property states that nothing bad ever happens and a liveness property states that something good eventually happens. In our context, a bad thing is a trace η not satisfying S and a good thing is the successful engagement in all communications of a trace η. Note that the notion of safety is different from safeness defined for nets in Section 3: safeness can be viewed as a specific safety property of the token game of a net. Stability is also a safety property, but it is singled out here because its rôle is more technical. Its presence allows a more powerful verification rule for the choice operator [OI 88/89]. For mathematical characterisations of safety and liveness properties see [AS 85].

In the following we give formal definitions of the notions of "may" and "must engage" and of initial stability by looking at the Petri net denoted by P. The intuition behind these definitions is as follows. Whereas transitions labelled by a communication occur only if the user participates in them, transitions labelled by τ occur autonomously at an unknown, but positive speed. Thus τ-transitions give rise to unstability and divergence.

Definition. Consider a net $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$ and let $M, M' \in mark(\mathfrak{N})$ and $\eta \in Comm^*$.

(1) *Progess properties.* The set of *next possible actions* at M is given by

$$next(M) = \{u \in Act \mid \exists t \in \longrightarrow : pre(t) \subseteq M \text{ and } act(t) = u\}.$$

M is called *stable* if $\tau \notin next(M)$ otherwise it is called *unstable*. M is *ready* for a communication b if M is stable and $b \in next(M)$. M is ready for the communication set A if M is stable and next(M) = A. $\mathfrak{N}$ is *stable immediately* if M_0 is stable. We write

$$M \stackrel{\eta}{\Longrightarrow} M'$$

if there exists a finite transition sequence $M \xrightarrow{t_1} M_1 \; ... \; M_{n-1} \xrightarrow{t_n} M_n = M'$ such that $\eta = (act(t_1) \; ... act(t_n)) \setminus \tau$, i.e. η results from the sequence of actions $act(t_1)...act(t_n)$ by deleting all internal actions τ.

(2) *Divergence properties.* $\mathfrak{N}$ *can diverge from* M if there exists an infinite transition sequence

$$M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots$$

such that $\tau = act(t_1) = act(t_2) = act(t_3) = \ldots$ $\mathfrak{N}$ *can diverge immediately* if $\mathfrak{N}$ can diverge from M_0. $\mathfrak{N}$ *can diverge after* σ if there exists a marking M with

$$M_0 \stackrel{\sigma}{\Longrightarrow} M$$

such that $\mathfrak{N}$ can diverge from M. $\mathfrak{N}$ *can diverge only after* σ if whenever $\mathfrak{N}$ can diverge after some trace σ' then $\sigma \leq \sigma'$. $\mathfrak{N}$ *can diverge* if there is a marking $M \in mark(\mathfrak{N})$ from which $\mathfrak{N}$ can diverge. $\mathfrak{N}$ is *divergence free* if $\mathfrak{N}$ cannot diverge.

(3) *Deadlock properties.* $\mathfrak{N}$ *deadlocks* at M if $next(M) = \Phi$. $\mathfrak{N}$ *deadlocks immediately* if $\mathfrak{N}$ deadlocks at M_0. $\mathfrak{N}$ *can deadlock after* σ if there exists a marking M with

$$M_0 \stackrel{\sigma}{\Longrightarrow} M$$

such that $\mathfrak{N}$ deadlocks at M. $\mathfrak{N}$ *can deadlock only after* σ if whenever $\mathfrak{N}$ can deadlock after some trace σ' then $\sigma \leq \sigma'$. $\mathfrak{N}$ *can deadlock* if there is a marking $M \in mark(\mathfrak{N})$ at wich $\mathfrak{N}$ deadlocks. $\mathfrak{N}$ is *deadlock free* if $\mathfrak{N}$ cannot deadlock. □

We now turn to process terms.

Definition. Consider a closed process term P, a representative $\mathfrak{N}_0 = (\alpha(P), Pl, \longrightarrow, M_0)$ of the abstract net $\mathfrak{N}[\![P]\!]$, and a trace $\sigma \in Comm^*$

(1) P is *stable immediately* if $\mathfrak{N}$ is so.

(2) P *can diverge (immediately* or *after* σ or *only after* σ) if $\mathfrak{N}_0$ can do so. P is *divergence free* if $\mathfrak{N}$ is so.

(3) P *deadlocks immediately* if $\mathfrak{N}_0$ does so. P *can deadlock (after* σ or *only after* σ) if $\mathfrak{N}_0$ can do so. P is *deadlock free* if $\mathfrak{N}$ is so.

(4) P *may engage* in σ if there exists a marking $M \in mark(\mathfrak{N}_0)$ such that $M \stackrel{\sigma}{\Longrightarrow} M$.

(5) P *must engage* in $\sigma = a_1 \ldots a_n$ if the process term $P \parallel a_1 \ldots a_n$. *stop*: $\alpha(P)$ is divergence free and can deadlock only after σ. □

Clearly, the above definitions are independent of the choice of the representative $\mathfrak{N}_0$. The formalisations of immediate stability and "may engage" capture the intuitions earlier, but the formalisation of "must engage" requires some explanation. The process term $a_1 \ldots a_n$. *stop*: $\alpha(P)$ models a user wishing to communicate the trace $a_1 \ldots a_n$ to P and stop afterwards. Communication is enforced by making the alphabet of user and process identical. Thus the parallel composition $P \parallel a_1 \ldots a_n$. *stop*: $\alpha(P)$ can behave only as follows: it can engage in some prefix $a_1 \ldots a_k$ of σ with $0 \leq k \leq n$ and then either diverge (i.e. never become stable again) or deadlock (i.e. become stable, but unable to engage in any further communication). The user's wish to communicate σ is realised if and only if $P \parallel a_1 \ldots a_n$. *stop*: $\alpha(P)$ never diverges and if it deadlocks only after σ. A final deadlock is unavoidable because the user wishes to stop. This is how we formalise the notion of "must engage".

The terminology of "may" and "must engage" originates from DeNicola and Hennessy's work on testing of processes [DH 84, He 88]. There it is used to define several so-called testing equivalences on processes, among them one for the "may" case and one for the "must" case. Here we make different use of theses two notions. Also, our definition of "must engage" is stronger than in [DH 84, He 88] because we require stability after each communication. This will result in an equivalence which differs from their testing equivalences (see Section 6).

We can show that P *sat* S has very strong consequences for P.

Proposition. Consider a closed process term P and a trace specification S. Then P *sat* S implies the following:

(1) "May" is equivalent to "must", i.e. for every trace η the process P may engage in η if and only if P must engage in η.

(2) P is divergence free.

(3) P is externally deterministic. □

Intuitively, a process is externally deterministic if the user cannot detect any nondeterminism by communicating with it. Formally, we define this notion as follows:

Definition. Consider a closed process term P and some representative $\mathfrak{N}_0 = (\alpha(P), Pl, \longrightarrow, M_0)$ of $\mathfrak{M}[\![P]\!]$. Then P is called *externally deterministic* if for all traces $\eta \in Comm^*$ and all markings $M_1, M_2 \in mark(\mathfrak{N}_0)$ whenever

$$M_0 \stackrel{\eta}{\Longrightarrow} M_1 \text{ and } M_0 \stackrel{\eta}{\Longrightarrow} M_2$$

such that M_1 and M_2 are stable then

$$next(M_1) = next(M_2).$$

That is: every communication trace η uniquely determines the next stable set of communications. □

Thus trace formulas specify only divergence free and exernally deterministic processes. This is a clear restriction of our approach, but it yields an interesting class of processes with many applications and simplest verification rules (see Section 7).

Examples. Let us consider the trace specification

$$S = 0 \le up\#h - dn\#h \le 2$$

which is an abbreviation for $dn\#h \le up\#h \le 2 + dn\#h$, and examine how a process P satisfying S should behave. Since P *sat* S implies $\alpha(P) = \alpha(S) = \{ up, dn \}$, P should engage only in the communications up and dn. By the safety condition, in every communication trace η that P may engage in, the difference of the number of up's and the number of dn's is between 0 and 2. If P has engaged in such a trace η and the extension η.dn still satisfies S, the liveness condition of P *sat* S requires that after η the process P must engage in the communication dn. The same is true for up.

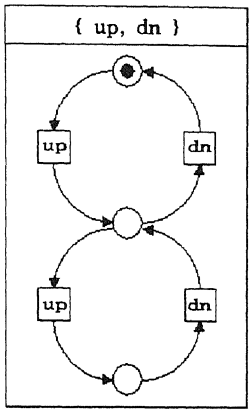
Thus S specifies that P should behave like a *bounded counter of capacity 2* which can internally store a natural number n with $0 \leq n \leq 2$. After a communication trace η, the number stored is $n = up^{*}\eta - dn^{*}\eta$. Initially, when η is empty, n is zero. Communicating up increments n and communicating dn decrements n. Of course, these communications are possible only if the resulting changes of n do not exceed the counter bounds.

A process term satisfying S is

$$P = \mu X.\ up.\ \mu Y.\ (\ dn.\ X + up.\ dn.\ Y\ )$$

denoting the following abstract net

$\mathfrak{N}[\![P]\!]$ =

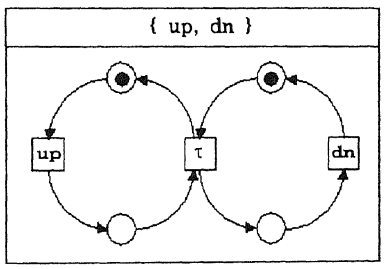


This net is purely sequential, i.e. every reachable marking contains at most one token, and there are no internal actions involved. Another process term satisfying S is

$$Q = (\ (\ \mu X.\ up.\ dn.\ X\ )\ [\ lk/dn\ ]\ \|\ (\mu X.\ up.\ dn.\ X\ )\ [\ lk/up\ ]\ )\ \backslash lk$$

denoting the following abstract net.

$\mathfrak{N}[\![Q]\!]$ =



Here, after each up-transition the net has to engage in an internal action τ before it is ready for the corresponding dn-transition. Since τ-actions occur autonomously, readiness for the next dn is guaranteed, as required by the specification S. This leads in fact to a marking where up and dn are concurrently enabled.

The examples of P and Q demonstrate that presence or absence of concurrency or intervening internal activity are treated here as properties of the implementation (process term and net), not of the specification.

It is easy to generalise the above trace specification. For $k \geq 1$ a *bounded counter of capacity k* is specified by

$$S_k = 0 \leq up\#h - dn\#h \leq k \ .$$

If we drop the upper bound k, we obtain a trace specification S_∞ for an *unbounded counter* that can store an arbitrary large natural number:

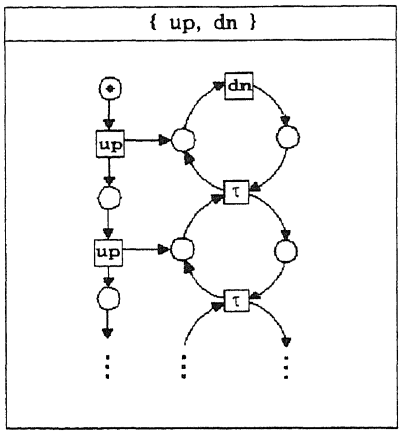
$$S_\infty = dn\#h \leq up\#h \ .$$

In a process satisfying S_∞ the communication up may and must occur after every trace. One such process is given by the term

$$P_\infty = \mu X.\ up.\ (X\ [\ lk/dn\ ]\ \|\ \mu Y.\ dn.\ lk.\ Y)\ \backslash\ lk$$

which denotes the infinite abstract net

$\mathfrak{N}[\![P_\infty]\!] =$



After the n-th communication up the net will engage in n-1 internal actions τ before being ready for the corresponding n-th communication dn. But again, these intervening internal actions do not impair the user's view of the specified behaviour. □

5. MODIFIED READINESS SEMANTICS

The liveness condition of the satisfaction relation P *sat* S is difficult to check when only the net semantics of P is available. To simplify matters, we introduce now a second, more abstract semantics for process terms. It is a variation of the readiness semantics $\mathfrak{R}$ introduced in [OH 86]. The main idea of $\mathfrak{R}$ is to record information about the process behaviour in the form of pairs $(\mathfrak{h}, \mathfrak{F})$ consisting of a trace $\mathfrak{h}$ and a so-called *ready set* $\mathfrak{F}$. This is a set of communications in which the process is ready to engage when it has become stable after the trace $\mathfrak{h}$ [Ho 81, FLP 84, BMOZ 88]. Additionally, $\mathfrak{R}$ records information about divergence and applies a certain closure operator known as "chaotic closure" and due to [BHR 84]. The semantics $\mathfrak{R}$ is modified here in three ways:

(1) Information about initial unstability is recorded. This is needed because we use here Milner's choice operator + instead of Hoare's two operators □ and *or* distinguishing external and internal choice as in [OH 86].

(2) The "acceptance closure" due to [DH 84] is enforced on the ready sets.

(3) A new "radiation closure" on ready sets is enforced; it will be explained below.

To avoid confusion, we shall write $\mathfrak{R}^*$ for the modified readiness semantics. Formally, it is a mapping

$$\mathfrak{R}^*[\![\cdot]\!] : CProc \longrightarrow DOM_{\mathfrak{R}}$$

which assigns to every $P \in CProc$ an element $\mathfrak{R}^*[\![P]\!]$ in the readiness domain $DOM_{\mathfrak{R}}$. This domain consists of pairs (A,Γ) where A is a communication alphabet and Γ is a set of *process informations*. We consider three types of process information:

(1) The element τ indicating initial *unstability*.
(2) *Ready pairs* $(\mathfrak{h}, \mathfrak{F})$ consisting of a trace $\mathfrak{h} \in A^*$ and a ready set $\mathfrak{F} \subseteq A$.
(3) *Divergence points* $(\mathfrak{h}, \uparrow)$ consisting of a trace $\mathfrak{h} \in A^*$ and a special symbol $\uparrow$ standing for divergence.

The set of these process informations can be expressed as follows:

$$Info_{\mathfrak{R}}:A = \{ \tau \} \cup A^* \times \mathfrak{P}(A) \cup A^* \times \{ \uparrow \}.$$

Define

$$DOM_{\mathfrak{R}}:A = \{ (A,\Gamma) \mid \Gamma \subseteq Info:A \}.$$

The readiness domain is then given by

$$DOM_{\mathfrak{R}} = \bigcup DOM_{\mathfrak{R}}:A$$

where the union is taken over all communication alphabets A.

For a pair $(A, \Gamma) \in DOM_{\mathfrak{R}}$ we define its alphabet by $\alpha(A, \Gamma) = A$ and its *set of process informations* by $\pi(A, \Gamma) = \Gamma$. We adopt the following notational conventions: letters γ, δ range over $Info_{\mathfrak{R}}:A$, letters Γ, Δ over subsets of $Info_{\mathfrak{R}}:A$ and hence pairs (A, Γ), (B, Δ) over $DOM_{\mathfrak{R}}$, letters $\mathfrak{F}$, $\mathfrak{G}$ range over ready sets and the letter $\mathfrak{X}$ can either be a ready set or the symbol $\uparrow$.

The mapping $\mathfrak{R}^*[\![\cdot]\!]$ retrieves the relevant process information from the operational Petri net semantics. Hence we talk of an operational readiness semantics. First we consider individual nets.

Definition. The *readiness semantics of a Petri net* $\mathfrak{N} = (A, Pl, \longrightarrow, M_0)$ is given by

$$\begin{aligned}\mathfrak{R}^*(\mathfrak{N}) = close(A, \ & \{ \ \tau \ \mid M_0 \text{ is unstable } \} \\ & \cup \{ (\mathfrak{h}, \mathfrak{F}) \mid \exists\, M \in mark(\mathfrak{N}) : \\ & \qquad M_0 \stackrel{\mathfrak{h}}{\Longrightarrow} M \text{ and } M \text{ is stable and } \mathfrak{F} = next(M) \} \\ & \cup \{ (\mathfrak{h}, \uparrow) \mid \exists\, M \in mark(\mathfrak{N}) : \\ & \qquad M_0 \stackrel{\mathfrak{h}}{\Longrightarrow} M \text{ and } \mathfrak{N} \text{ can diverge from } M \} \quad)\end{aligned}$$

where the *closure operator* close: $DOM_{\mathfrak{R}} \longrightarrow DOM_{\mathfrak{R}}$ is defined as follows:

$$\begin{aligned}close\ (A,\Gamma) = (A, \ & \Gamma \cup \{ (\mathfrak{h}, \mathfrak{G}) \mid \exists\, \mathfrak{F}: \ (\mathfrak{h}, \mathfrak{F}) \in \Gamma \text{ and } \mathfrak{F} \subseteq \mathfrak{G} \subseteq succ(\mathfrak{h}, \Gamma) \} \\ & \cup \{ (\mathfrak{h}', \mathfrak{X}) \mid \exists\, \mathfrak{h} \leq \mathfrak{h}' : (\mathfrak{h}, \uparrow) \in \Gamma \text{ and } \mathfrak{h}' \in A^* \\ & \qquad\qquad \text{and } (\mathfrak{X} \subseteq A \text{ or } \mathfrak{X} = \uparrow) \} \\ & \cup \{ (\mathfrak{h}, \mathfrak{G}) \mid \exists\, a: \ (\mathfrak{h}.a, \uparrow) \in \Gamma \text{ and } \mathfrak{G} \subseteq succ(\mathfrak{h}, \Gamma) \})\end{aligned}$$

Here succ($\mathfrak{h}$, Γ) denotes the set of all *successor communications* of $\mathfrak{h}$ in Γ:

$$\mathrm{succ}(\mathfrak{h}, \Gamma) = \{\, a \mid \exists\, \mathfrak{G}\colon (\mathfrak{h}.a, \mathfrak{G}) \in \Gamma \,\}\ .$$

The *readiness semantics of an abstract net* $[\mathfrak{N}]$ is given by $\mathfrak{R}^*([\mathfrak{N}]) = \mathfrak{R}^*(\mathfrak{N})$ and the *(operational) readiness semantics of a closed process term* P is given by $\mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*(\mathfrak{N}[\![P]\!])$. □

Let us now investigate the basic properties of the readiness semantics. First of all, it is an interleaving semantics, i.e. it is insensitive to concurrency. This is demonstrated by the law

$$\mathfrak{R}\ [\![\ a.\ \mathit{stop}.\{a\} \parallel b.\ \mathit{stop}.\{b\}\ ]\!] = \mathfrak{R}^*[\![\ a.\ b.\ \mathit{stop}.\{a, b\} + b.\ a.\ \mathit{stop}.\{a, b\}\ ]\!]$$

which is easily established by retrieving the readiness information from the corresponding nets. Secondly, the readiness semantics enjoys a number of structural properties which we summarise under the notion of being well-structured.

Definition. An element $(A,\Gamma) \in \mathrm{DOM}_{\mathfrak{R}}$ is called *well-structured* if the following holds:

(1) *Initial ready pair*: $\exists\ \mathfrak{G} \subseteq A : (\varepsilon, \mathfrak{G}) \in \Gamma$.
(2) *Prefix closure*: $(\mathfrak{h}.\ a, \mathfrak{F}) \in \Gamma$ implies $\exists\ \mathfrak{G} \subseteq A : (\mathfrak{h}, \mathfrak{G}) \in \Gamma$ and $a \in \mathfrak{G}$.
(3) *Extensibility*: $(\mathfrak{h}, \mathfrak{F}) \in \Gamma$ and $a \in \mathfrak{F}$ imply $\exists\ \mathfrak{G} \subseteq A : (\mathfrak{h}.\ a, \mathfrak{G}) \in \Gamma$.
(4) *Acceptance closure*: $(\mathfrak{h}, \mathfrak{F}) \in \Gamma$ and $\mathfrak{F} \subseteq \mathfrak{G} \subseteq \mathrm{succ}(\mathfrak{h}, \Gamma)$ imply $(\mathfrak{h}, \mathfrak{G}) \in \Gamma$.
(5) *Chaotic closure*: $(\mathfrak{h}, \uparrow) \in \Gamma$ and $\mathfrak{h} \preceq \mathfrak{h}'$ and ($\mathfrak{X} \subseteq A$ or $\mathfrak{X} = \uparrow$) imply $(\mathfrak{h}', \mathfrak{X}) \in \Gamma$.
(6) *Radiation closure*: $(\mathfrak{h}.\ a, \uparrow) \in \Gamma$ and $\mathfrak{G} \subseteq \mathrm{succ}(\mathfrak{h}, \Gamma)$ imply $(\mathfrak{h}, \mathfrak{G}) \in \Gamma$.
(7) *Unstability closure*: $(\varepsilon, \uparrow) \in \Gamma$ implies $\tau \in \Gamma$. □

Proposition. For every closed process term P the readiness semantics $\mathfrak{R}^*[\![P]\!] \in \mathrm{DOM}_{\mathfrak{R}}$ is well-structured. □

Properties (1), (3), (5) and (2) without the condition "and $a \in \mathfrak{G}$" are as in the original readiness semantics $\mathfrak{R}$ in [OH 86]. Property (4) stems from the semantic models studied by DeNicola and Hennessy [DH 84, He 88]; it implies the condition "and $a \in \mathfrak{G}$" in (2). Property (7) is motivated by [DH 84] and [BKO 87]. Property (6) is completely new: it states that divergence affects the ready sets one level up; we therefore say that divergence "radiates up". Note that the closure properties (4) - (6) add ready sets and divergence points to $\mathfrak{R}^*[\![P]\!]$ which are not justified by the token game of $\mathfrak{N}[\![P]\!]$. These additions make the semantics $\mathfrak{R}^*[\![\cdot]\!]$ more abstract so that less process terms can be distinguished under $\mathfrak{R}^*[\![\cdot]\!]$. In Section 6 we shall see that the resulting level of abstraction is in perfect match with the distinctions that we can make among process terms under the satisfaction relation P *sat* S. Technically speaking, $\mathfrak{R}^*[\![\cdot]\!]$ is fully abstract with respect to this relation.

Here we notice that with the readiness semantics we can easily express the process properties relevant for the satisfaction relation P *sat* S. Recall that $\pi(\mathfrak{R}^*[\![P]\!])$ is the set of process informations collected by $\mathfrak{R}^*[\![P]\!]$.

Proposition. For every divergence free, closed process term P and trace $\mathfrak{h} = a_1 \ldots a_n$:

(1) P may engage in $\mathfrak{h}$ iff $(\mathfrak{h}, \mathfrak{F}) \in \pi(\mathfrak{R}^*[\![P]\!])$ for some ready set $\mathfrak{F}$.
(2) P can deadlock after $\mathfrak{h}$ iff $(\mathfrak{h}, \Phi) \in \pi(\mathfrak{R}^*[\![P]\!])$.

(3) P must engage in $\mathfrak{h}$ iff for every prefix $a_1 \ldots a_k$ of $\mathfrak{h}$ with $0 \leq k < n$ and every ready set $\mathfrak{F}$

$$(a_1 \ldots a_k, \mathfrak{F}) \in \pi(\mathfrak{R}^*[\![P]\!]) \text{ implies } a_{k+1} \in \mathfrak{F},$$

i.e. whenever P becomes stable, it is ready to engage in the next communication of $\mathfrak{h}$.

(4) P is externally deterministic iff for every trave $\mathfrak{h}$ there is at most one ready set $\mathfrak{F}$ with $(\mathfrak{h}, \mathfrak{F}) \in \pi(\mathfrak{R}^*[\![P]\!])$. □

With these preparations, we can now approach the main objective of this section: a direct comparison of process terms and trace specifications on the basis of the readiness domain. To this end, we extend now the readiness semantics $\mathfrak{R}^*[\![\cdot]\!]$ to cover trace specifications as well, i.e. to a mapping

$$\mathfrak{R}^*[\![\cdot]\!]: \text{CProc} \cup \text{Spec} \longrightarrow \text{DOM}_{\mathfrak{R}}.$$

Definition. The *readiness semantics of a trace specification* S is given by

$$\mathfrak{R}^*[\![S]\!] = (\alpha(S), \{ (\mathfrak{h}, \mathfrak{F}) \mid \mathfrak{h} \in \alpha(S)^* \text{ and } \mathit{pref}\ \mathfrak{h} \models S \text{ and } \mathfrak{F} = \{ a \in \alpha(S) \mid \mathfrak{h}.\, a \models S \} \})$$

where, as before, $\mathit{pref}\ \mathfrak{h} \models S$ means that $\mathfrak{h}$ and all its prefixes satisfy S. □

Since trace specifications S specify only processes which are stable immediately and divergence free, it is understandable that $\mathfrak{R}^*[\![S]\!]$ does not contain elements of the form τ and $(\mathfrak{h}, \uparrow)$ indicating unstability and divergence. Note that $\mathfrak{R}^*[\![S]\!]$ satisfies the properties (2) - (7) of being well-structured, but not (1) because $\mathfrak{R}^*[\![S]\!]$ may be empty. Thus the readiness semantics of trace specifications S is closed, i.e. close($\mathfrak{R}^*[\![S]\!]$) = $\mathfrak{R}^*[\![S]\!]$ but need not be well-structured. However, if $\varepsilon \models S$ then $\mathfrak{R}^*[\![S]\!]$ is well-structured.

The main result of this section is the following theorem which is proved in [Ol 88/89].

Correctness Theorem. For every closed process term P and trace specification S we have

$$P \mathit{\ sat\ } S \quad \text{iff} \quad \mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![S]\!],$$

i.e. in the readiness semantics process correctness reduces to semantics equality. □

The Correctness Theorem simplifies, at least conceptually, the task of proving that a process term P satisfies a trace specification S.

Example. In Section 4 we considered the trace specification

$$S = 0 \leq \text{up}\#h - \text{dn}\#h \leq 2$$

and argued informally that the process terms

$$P = \mu X.\ \text{up}.\ \mu Y.\ (\ \text{dn}.\ X + \text{up}.\ \text{dn}.\ Y\ )$$

and

$$Q = (\ (\ \mu X.\ \text{up}.\ \text{dn}.\ X\ )[\ \text{lk/dn}\ ] \parallel (\ \mu X.\ \text{up}.\ \text{dn}.\ X\ )[\ \text{lk/up}\ ]\ ) \backslash \text{lk}$$

both satisfy S. We can now prove this claim by comparing the readiness semantics of S with that of P and Q:

$$\mathfrak{R}^*[\![S]\!] = \{\, \{ up, dn \}, \{ (\mathfrak{h}, \mathfrak{F}) \mid \forall\, \mathfrak{h}' \le \mathfrak{h}\colon\ 0 \le up*\mathfrak{h}' - dn*\mathfrak{h}' \le 2$$
$$\text{and } (\text{ if } 0 = up*\mathfrak{h} - dn*\mathfrak{h} \quad \text{then } \mathfrak{F} = \{ up \} \quad)$$
$$\text{and } (\text{ if } 0 < up*\mathfrak{h} - dn*\mathfrak{h} < 2 \quad \text{then } \mathfrak{F} = \{ up, dn \})$$
$$\text{and } (\text{ if } \quad up*\mathfrak{h} - dn*\mathfrak{h} = 2 \quad \text{then } \mathfrak{F} = \{ dn \} \quad) \,\} \,)$$

By an exhaustive analysis of the reachable markings of the nets $\mathfrak{M}[\![P]\!]$ and $\mathfrak{M}[\![Q]\!]$ displayed in Section 4 we see that

$$\mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![S]\!] = \mathfrak{R}^*[\![Q]\!].$$

Thus indeed P *sat* S and Q *sat* S. □

6. FULL ABSTRACTION

Process terms denote Petri nets describing all details of the process behaviour many of which are irrelevant from the viewpoint of trace specifications. We therefore investigate the following question:

> Under what circumstances can we replace a closed process term P by a closed process term Q without ever noticing this change by the satisfaction relation *sat* ?

Since replacement can take place within a larger process term, we use the notion of a context to make this question precise. A *context* is a term $\mathfrak{C}(X) \in$ Rec with one free identifier X. To simplify notation, we shall write $\mathfrak{C}(R)$ instead of $\mathfrak{C}(X)\{R\backslash X\}$ for the substitution of a process term R for X in $\mathfrak{C}(X)$. For example, the "must" condition of P *sat* S can be viewed as a condition on $\mathfrak{C}(P)$ where the context is

$$\mathfrak{C}(X) = X \parallel a_1 \ldots a_n.\ stop : \alpha(P)$$

Equivalence under the satisfaction relation *sat* is covered by the following *satisfaction equivalence* $\equiv_{sat}$ on closed process terms:

$$P \equiv_{sat} Q$$

if for every trace specification S the following holds: P *sat* S iff Q *sat* S. Now the above question becomes: Under what condition on P and Q do we have $\mathfrak{C}(P) \equiv_{sat} \mathfrak{C}(Q)$ for every context $\mathfrak{C}(X)$ with $\mathfrak{C}(P), \mathfrak{C}(Q) \in$ CProc ? Milner's notion of *full abstraction* [Mi 77] (see also [Pl 77, HP 79]) can be seen as looking for a sufficent and necessary condition that solves this type of question.

Definition. A semantics (or semantic model) $\mathfrak{M}$: CProc —> $DOM_{\mathfrak{M}}$ is called *fully abstract* for an equivalence relation $\equiv$ on CProc if the following holds for all closed process terms P and Q:

$\mathfrak{M}[\![P]\!] = \mathfrak{M}[\![Q]\!]$ iff $\mathfrak{C}(P) \equiv \mathfrak{C}(Q)$ holds for every context $\mathfrak{C}(X)$ with $\mathfrak{C}(P), \mathfrak{C}(Q) \in$ CProc.

□

Intuitively, a fully abstract model $\mathfrak{M}$ optimally fits the equivalence $\equiv$ in the sense that $\mathfrak{M}$ just makes the identifications on process terms that are forced by $\equiv$. For a given semantic model $\mathfrak{M}$: CProc —> $DOM_{\mathfrak{M}}$ let the *model equivalence* $\equiv$ be defined as follows:

$$P \equiv_{\mathfrak{M}} Q \quad \text{if} \quad \mathfrak{M}[\![P]\!] = \mathfrak{M}[\![Q]\!].$$

Then we can state the following consequence of the definition of full abstraction.

Proposition. For every equivalence relation $\equiv$ on CProc there exists a fully abstract model $\mathfrak{M}$ for $\equiv$ which is compositional w.r.t. the process operators in CProc and unique up to model equivalence $\equiv_{\mathfrak{M}}$. □

This proposition provides an attractive method of *specifying* the semantics of processes. Starting from an equivalence relation $\equiv$ that captures the kind of distinctions or observations on processes one is interested in, the proposition guarantees the existence of a compositional semantics $\mathfrak{M}$ that is optimal for $\equiv$ and unique up to model equivalence $\equiv_{\mathfrak{M}}$. Then $\mathfrak{M}$ is the semantics specified by $\equiv$. More generally, this specification method is used for programming languages with and without concurrency (see e.g. [As 84]) and in the area of algebraic specifications (see e.g. [ST 87]).

The existence of a fully abstract semantics $\mathfrak{M}$ is an interesting fact, but its implicit definition via contexts does not give us any ides about the explicit structure of $\mathfrak{M}$. Often it is a very difficult or even unsolved problem to find such an explicit structure [Mi 77, Pl 77, HP 79, MS 88]. Fortunately, for the satisfaction equivalence $\equiv_{sat}$ we will be able to exhibit this structure: it is the modified readiness semantics $\mathfrak{R}^*[\![\cdot]\!]$ discussed in the previous sections.

Full Abstraction Theorem. The modified readiness semantics $\mathfrak{R}^*[\![\cdot]\!]$: CProc —> $\mathrm{DOM}_{\mathfrak{R}}$ is fully abstract for the satisfaction equivalence $\equiv_{sat}$, i.e. for all closed process terms P and Q the following holds

$$\mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![Q]\!]$$

if and only if for all contexts $\mathcal{C}(X)$ with $\mathcal{C}(P)$, $\mathcal{C}(Q) \in$ CProc and all trace specifications S

$$\mathcal{C}(P) \; sat \; S \quad \text{iff} \quad \mathcal{C}(Q) \; sat \; S.$$

Proof. *"only if"* : see [Ol 88/89]. *"if"* : Suppose $\mathfrak{R}^*[\![P]\!] \neq \mathfrak{R}^*[\![Q]\!]$, say $\mathfrak{R}^*[\![P]\!] \not\subseteq \mathfrak{R}^*[\![Q]\!]$. We will exhibit a context $\mathcal{C}(X)$ with $\mathcal{C}(P)$, $\mathcal{C}(Q) \in$ CProc and a trace specification S with

$$\mathcal{C}(P) \; sat \; S \quad \text{but} \quad \mathcal{C}(Q) \; \not sat \; S.$$

Let $A=\alpha(P)$. If $\alpha(P) \neq \alpha(Q)$, we can take $\mathcal{C}(X) = X$ and $S = h\restriction A \leq h\restriction A$. If $\alpha(P) = \alpha(Q)$, we distinguish three cases depending on the structure of process informations in $\pi(\mathfrak{R}^*[\![P]\!])$ and $\pi(\mathfrak{R}^*[\![Q]\!])$.

Case 1: $\tau \in \pi(\mathfrak{R}^*[\![Q]\!])$ and $\tau \notin \pi(\mathfrak{R}^*[\![P]\!])$. Then P cannot diverge immediately. Take

$$\mathcal{C}(X) = X \parallel stop{:}A \quad \text{and} \quad S = h\restriction A \leq \varepsilon$$

Then $\mathcal{C}(P)$, $\mathcal{C}(Q) \in$ CProc and $\tau \in \pi(\mathfrak{R}^*[\![\mathcal{C}(Q)]\!])$, but $\tau \notin \pi(\mathfrak{R}^*[\![\mathcal{C}(P)]\!])$. In fact, $\mathcal{C}(P)$ is stable immediately, divergence free and can engage only in the empty trace. Thus $\mathcal{C}(P)$ *sat* S. On the other hand, $\mathcal{C}(Q)$ $\not sat$ S because $\mathcal{C}(Q)$ is unstable as the τ in its readiness semantics indicates.

Case 2: $(\mathfrak{h}, \mathfrak{F}) \in \pi(\mathfrak{R}^*[\![\mathcal{C}(Q)]\!])$ and $(\mathfrak{h}, \mathfrak{F}) \notin \pi(\mathfrak{R}^*[\![P]\!])$. Suppose $\mathfrak{h} = a_1 \ldots a_n$ where $n \geq 0$ and $a_1, \ldots, a_n \in A$. Since $\mathfrak{R}^*[\![P]\!]$ is well-formed (cf. Section 5), we conclude that

(1) $\quad \neg\ \exists\ \mathfrak{h}' \le \mathfrak{h} : (\mathfrak{h}', \uparrow) \in \pi(\mathfrak{R}^*[\![P]\!])$

because otherwise the chaotic closure would force $(\mathfrak{h}, \mathfrak{F}) \in \pi(\mathfrak{R}^*[\![P]\!])$. Let $\mathfrak{h}'$ be the longest prefix of $\mathfrak{h}$ such that

(2) $\quad \exists\ \mathfrak{G} : (\mathfrak{h}', \mathfrak{G}) \in \pi(\mathfrak{R}^*[\![P]\!])$

Such trace $\mathfrak{h}'$ exists because there is an initial ready pair $(\varepsilon, \mathfrak{G})$ in $\pi(\mathfrak{R}^*[\![P]\!])$. Take some fresh $d \notin A$. Such a communication d exists because Comm is infinite whereas $A \subset$ Comm is finite.

Subcase 2.1: $\mathfrak{h}' < \mathfrak{h}$. Then $\mathfrak{h}' = a_1 \ldots a_k$ for some $k < n$. As context we consider the term

$$\begin{aligned}
\mathfrak{C}(X) = \quad & d.\ (\ X \\
& \|\ (\ d^k\ .\ \mathit{stop}{:}A \cup \{d\} \\
& \quad +\ a_1\ .\ (\ d^{k-1}\ .\ \mathit{stop}{:}A \cup \{d\} \\
& \qquad +\ a_2\ .\ (\ \ldots\ (\ d\ .\ \mathit{stop}{:}A \cup \{d\} \\
& \qquad\qquad +\ a_k\ .\ a_{k+1}\ .\ \mathit{stop}{:}A \cup \{d\}\ )\ \ldots\ ))) \\
&)\ [\varphi]
\end{aligned}$$

where the renaming morphism φ : Act ⟶ Act is given by $\varphi(u) = d$ for $u \in A \cup \{d\}$ and $\varphi(u) = u$ otherwise. The notation d^m . *stop*:A∪{d} abbreviates $\underbrace{d\ .\ \ldots\ .\ d}_{m \text{ times}}$. *stop*:A∪{d}.

Clearly, $\mathfrak{C}(P)$, $\mathfrak{C}(Q) \in$ CProc. The initial communication d of $\mathfrak{C}(X)$ serves to absorb possible unstabilities of P and Q in $\mathfrak{C}(P)$ and $\mathfrak{C}(Q)$. Since $d \notin A$, the communications d occuring in the right-hand operand of the parrallel composition of $\mathfrak{C}(P)$ and $\mathfrak{C}(Q)$ do not require synchronisation with the left-hand operand P or Q. Thus both $\mathfrak{C}(P)$ and $\mathfrak{C}(Q)$ can deadlock only after engaging in k + 1 communications.

In fact, $\mathfrak{C}(P)$ must engage in k + 1 communications because, by property (1) above, $\mathfrak{C}(P)$ is divergence free. Hence we consider as specification the trace formula

$$S =_{df} d\#h \le k+1$$

Then $\mathfrak{C}(P)$ *sat* S, but $\mathfrak{C}(Q)$ *sat* S becuase $\mathfrak{C}(Q)$ may engage in the trace

$$(\ d\ .\ a_1\ \ldots\ a_{k+1}\ )\ \{\ \varphi\ \} = d\ .\ d\ \ldots\ d$$

of the length k + 2.

Subcase 2.2: $\mathfrak{h}' = \mathfrak{h}$

Case 3: $(\mathfrak{h}, \uparrow) \in \pi(\mathfrak{R}^*[\![Q]\!])$ and $(\mathfrak{h}, \uparrow) \notin \pi(\mathfrak{R}^*[\![P]\!])$.

Full details for these cases are given in [Ol 88/89]; we omit them here. □

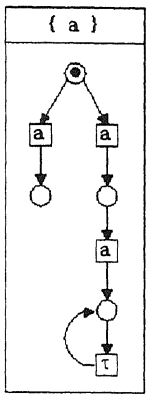
The full abstraction proof exploits that the modified readiness semantics $\mathfrak{R}^*[\![\cdot]\!]$ incorporates three extra closure conditions: chaotic closure, acceptance closure, and radiation closure. The chaotic closure, dealing with divergence, was introduced by Brookes, Hoare and Roscoe in their *failure semantics* $\mathfrak{F}[\![\cdot]\!]$ for CSP [BHR 84]. The acceptance closure on ready sets was introduced by DeNicola and Hennessy for a process semantics of CCS that is fully abstract for their strong "must" version of testing equivalence [DH 84, He 88]. For simplicity we call this semantics here *strong testing semantics* and denote it by $\mathfrak{T}$. The radiation closure, however, is new.

Consider for example the process terms

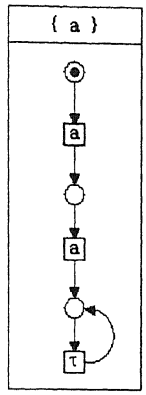
$$P = a \,.\, \mathit{stop}\colon \{a\} + Q \quad \text{and} \quad Q = a \,.\, a \,.\, \mathit{div}\colon \{a\}.$$

Then P and Q, or better their syntactic equivalents in CSP and CCS, are distinguished by failure and testing semantics: $\mathfrak{F}[\![P]\!] \neq \mathfrak{F}[\![Q]\!]$ and $\mathfrak{T}[\![P]\!] \neq \mathfrak{T}[\![Q]\!]$. But in the modified readiness semantics they are identified. Indeed, look at the abstract nets denoted by P and Q:

$\mathfrak{N}[\![P]\!]$ =



$\mathfrak{N}[\![Q]\!]$ =



Thus the ready pair (a,Φ) belongs to $\mathfrak{R}^*[\![P]\!]$ by the token game of $\mathfrak{N}[\![P]\!]$, but it also belongs to $\mathfrak{R}^*[\![Q]\!]$ by the radiation closure. Informally, the divergence point (a . a, ↑) of $\mathfrak{R}^*[\![Q]\!]$ "radiates up" and thus forces (a, Φ) to be present in $\mathfrak{R}^*[\![Q]\!]$. Hence

$$\mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![Q]\!]$$

This identification is justified by the idea of full abstraction because in every context $\mathfrak{C}(X)$ both $\mathfrak{C}(P)$ and $\mathfrak{C}(Q)$ satisfy exactly the same trace specification S.

The example demonstrates that the *modified readiness equivalence* $\equiv_{\mathfrak{R}^*}$ on process terms, given by

$$P \equiv_{\mathfrak{R}^*} Q \quad \text{if} \quad \mathfrak{R}^*[\![P]\!] = \mathfrak{R}^*[\![Q]\!],$$

differs from the corresponding *failure equivalence* $\equiv_{\mathfrak{F}}$ and *strong testing equivalence* $\equiv_{\mathfrak{T}}$. It has to be different because of the satisfaction relation *sat* which uniquely determines $\equiv_{\mathfrak{R}^*}$ via the notion of full abstraction.

However, the differences appear only for processes which can diverge. On divergence free process terms $\equiv_{\mathfrak{R}^*}$ and $\equiv_{\mathfrak{T}}$ coincide and on divergence free process terms which are stable immediately also $\equiv_{\mathfrak{R}^*}$ and $\equiv_{\mathfrak{F}}$ coincide. This can be easily seen by comparing the definitions of the semantics $\mathfrak{R}^*[\![\cdot]\!]$ with $\mathfrak{T}[\![\cdot]\!]$ and $\mathfrak{F}[\![\cdot]\!]$.

7. CONCLUSION

Based on the notion of process correctness P *sat* S defined in this paper we have developed compositional transformation rules for the systematic construction of process terma from given trace specifications [Ol 88/89]. Most rules turn out to be very simple. For example, parallel composition P || Q of process terms P and Q is reflected by the logical

conjunction of trace formulas. Soundness of these rules is proved by using an equivalent *denotational* definition of the modified readiness semantics. Applications of our notion of process correctness and the transformation rules can be found in [BDF 88, DB 89, OI 88/89].

8. REFERENCES

[As 84] E.Astesiano, Combining an operational with an algebraic approach to the specification of concurrency, in: D. Bjørner (Ed.), Proc. Workshop on Combining Methods (Nyborg, Denmark, 1984)

[AS 85] B. Alpern, F.B. Schneider, Defining liveness, Inform. Proc. Letters 21 (1985) 181-185.

[BMOZ 88] J.W. de Bakker, J.-J. Meyer, E.-R. Olderog, J.I. Zucker, Transition systems, metric spaces and ready sets in the semantics of uniform concurrency, J. Comput. System Sci. 36 (1988) 158-224.

[Be 87] E. Best, COSY: its relation to nets and CSP, in: W. Brauer, W. Reisig, G. Rozenberg (Eds.), Petri Nets: Applications and Relationships to Other Models of Concurrency, Lecture Notes in Comput. Sci. 255 (Springer-Verlag, 1987) 416-440.

[BKO 87] J.A. Bergstra, J.W. Klop, E.-R. Olderog, Failures without chaos: a new process semantics for fair abstraction, in: M. Wirsing (Ed.), Proc. IFIP Working Conference on Formal Description of Programming Concepts III (North-Holland, 1987) 77-101.

[BDF 88] M. Bretschneider, M. Duque Antón, A. Fink, Constructing and verifying protocols using TCSP, in: S. Aggarwal, K. Sabnani (Ed.), Proc. IFIP Working Conference on Protocol Specification, Testing and Verification (North-Holland, 1988).

[BHR 84] S.D. Brookes, C.A.R. Hoare, A.W. Roscoe, A theory of communicating sequential processes, J. ACM 31 (1984) 560-599.

[CHo 81] Z. Chaochen, C.A.R. Hoare, Partial correctness of communicating processes, in: Proc. 2nd Intern. Conf. on Distributed Comput. Systems, Paris, 1981.

[DH 84] R. DeNicola, M. Hennessy, Testing equivalences for processes, Theoret. Comput. Sci. 34 (1984) 83-134.

[Di 76] E.W. Dijkstra, A Discipline of Programming (Prentice-Hall, Englewood Cliffs, NJ, 1976).

[DB 89] M. Duque Antón, M. Bretschneider, Formulas, processes and Petri-nets applied to the specification and verification of a HDLC protocol, in J. Diaz, F. Orejas (Eds.), Proc. TAPSOFT '89, Vol. 2 Lecture Notes in Comput. Sci. 352 (Springer-Verlag, 1989) 140-154.

[Go 88] U. Goltz, Über die Darstellung von CCS-Programmen durch Petrinetze, Doctoral Diss., RWTH Aachen, 1988.

[Hen 88] M. Hennessy, Algebraic Theory of Processes (MIT Press, Cambridge, Mass., 1988).

[HP 79] M. Hennessy, G.D. Plotkin, Full abstraction for a simple programming language, in: J. Becvar (Ed.), 8th Symp. on Math. Found. of Comput. Sci., Lecture Notes in Comput. Sci. 74 (Springer-Verlag, 1979) 108-120.

[Ho 78] C.A.R. Hoare, Some properties of predicate transformers, J. ACM 25 (1978) 461-480.

[Ho 81] C.A.R. Hoare, A calculus of total correctness for communicating processes, Sci. Comput. Progr. 1 (1981) 44-72.

[Ho 85] C.A.R. Hoare, Communicating Sequential Processes (Prentice-Hall, London, 1985).

[Jo 87] B. Jonsson, Compositional Verification of Distributed Systems, Ph.D. Thesis, Dept. Comput. Sci., Uppsala Univ., 1987.

[LTS 79] P.E. Lauer, P.R. Torrigiani, M.W. Shields, COSY - A system specification language based on paths and processes, Acta Inform. 12 (1979) 109-158.

[Mz 77] A. Mazurkiewicz, Concurrent program schemes and their interpretations, Tech. Report DAIMI PB-78, Aarhus Univ., 1977.

[MS 88] A.R. Meyer, K. Sieber, Towards fully abstract semantics for local variables, Preliminary Report, in: Proc. 15th ACM Symp. Principles of Program. Lang. (San Diego, California, 1988) 191-203.

[Mi 77] R. Milner, Fully abstract models of typed λ-calculi, Theoret. Comput. Sci. 4 (1977) 1-22.

[Mi 80] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Comput. Sci. 92 (Springer-Verlag, 1980).

[MC 81] J. Misra, K.M. Chandy, Proofs of networks of processes, IEEE Trans. Software Eng. 7 (1981) 417-426.

[Ol 88/89] E.-R. Olderog, Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship, Habilitationsschrift, Univ. Kiel, 1988/89.

[Ol 89] E.-R. Olderog, Strong bisimilarity on nets: a new cvoncept for comparing net semantics, in: J.W. de Bakker, W.P. de Roever, G. Rozenberg (Eds.), Linear Time/Branching Time/Partial Order in the Semantics of Concurrency, Lecture Notes in Comput. Sci. 354 (Springer-Verlag, 1989) 549-573.

[OH 86] E.-R. Olderog, C.A.R. Hoare, Specification-oriented semantics for communicating processes, Acta Inform. 23 (1986) 9-66.

[Os 83] M. Ossefort, Correctness proofs of communicating processes: three illustrative examples from the literature, ACM TOPLAS 5 (1983) 620-640.

[OL 82] S. Owicki, L. Lamport, Proving liveness properties of concurrent programs, ACM TOPLAS 4 (1982) 199-223.

[Pl 77] G.D. Plotkin, LCF considered as a programming language, Theoret. Comput. Sci. 5 (1977) 223-255.

[Re 85] W. Reisig, Petri Nets, An Introduction, EATCS Monographs on Theoret. Comput. Sci. (Springer-Verlag, 1985).

[Rm 87] M. Rem, Trace theory and systolic computation, in: J.W. de Bakker, A.J. Nijman, P.C. Treleaven (Eds.), Proc. PARLE Conf., Eindhoven, Vol. I, Lecture Notes in Comput. Sci. 258, (Springer-Verlag, 1987) 14-33.

[ST 87] D.T. Sanella, A. Tarlecki, On observational equivalence and algebraic specification, J. Comput. System Sci. 34 (1987) 150-178.

[Sn 85] J.L.A. van de Snepscheut, Trace Theory and VLSI Design, Lecture Notes in Comput. Sci. 200 (Springer-Verlag, 1985).

[Sti 87] C. Stirling, Modal logics for communicating systems, Theoret. Comput. Sci. 49 (1987) 311-347.

[WGS 87] J. Widom, D. Gries, F.B. Schneider, Completeness and incompleteness of trace-baced network proof systems, in: Proc. 14th ACM Symp. on Principles of Progr. Languages, München, 1987, 27-38.

[Zw 89] J. Zwiers, Compositionality, Concurrency and Partial correctness, Lecture Notes in Comput. Sci. 321 (Springer-Verlag, 1989).

[ZRE 85] J. Zwiers, W.P. de Roever, P. van Emde-Boas, Compositionality and concurrent networks, in: W. Brauer (Ed.), Proc. 12th Coll. Automata, Languages and Programming, Lecture Notes in Comput. Sci. 194 (Springer-Verlag, 1985) 509-519.