Fundamenta Informaticae 45 (2001) 1–21 1
IOS Press

Process Algebras for Network Communication*

Damas P. Gruska†
Institute of Informatics
Comenius University
Bratislava, Slovakia
gruska@fmph.uniba.sk

Andrea Maggiolo-Schettini‡
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
maggiolo@di.unipi.it

Abstract. Critical issues that arise when process algebras are used for protocol specifications are discussed. To overcome some of these problems, a process algebra for protocol specifications is presented. It is based on Milner's Calculus of Communicating Systems, which is enriched by time and network reasoning. Several bisimulation based semantics for the calculus are defined and their properties are discussed.

Keywords: Concurrency, Specification, Process Algebras, Protocols.

1. Introduction

Formal methods play a growing role in the design of (mainly critical) software applications and hardware components. Among their main benefits, are clear design specification and support of verification methods.

*Work partially supported by CNR - Progetto Strategico "Modelli e Metodi per la Matematica e l'Ingegneria", by CNR - GNIM, and by the grant VEGA 1/7654/20 and by UCM Trnava.
†Address for correspondence: Institute of Informatics, Comenius University, Bratislava, Slovakia
‡Address for correspondence: Dipartimento di Informatica, Università di Pisa, Pisa, Italy

In practice, various formalisms are used. They range from graphical languages such as Petri Nets or Statecharts to algebraic, logic or special programming languages/models. They permit the description of concurrent systems at different levels of abstraction. Moreover, almost all of the formalisms are supported by special software tools that allow further investigation of system descriptions.

Process algebras are an important class of such formalisms. They successfully describe the behavior of systems as "communicating systems". *Communicating Sequential Processes, CSP* [9], *Algebra of Communicating Processes, ACP* [2], *Calculus of Communicating Systems, CCS* [13] are typical examples of this approach. At this level of description, the activity of a whole system is reduced to atomic (synchronous) communications, either inside the system or between the system and its environment. This paradigm predetermines the main area of application, namely specification of protocols. In the case of the above mentioned calculi, the protocols for communications via networks with static topology. These protocols might be of very different levels, ranging from the application level to the physical level.

Now, standard process algebras abstract away many real properties of existing systems, such as durations and structure of actions, properties of communication networks, distribution of system components, and so on. This leads to a relatively simple syntax for the language, but it might result in inadequate, too complicated or also incorrect specifications.

More precisely, the standard process algebras are relatively simple tools for describing and to studying "principles" of behavior of concurrent processes. They are universal in the sense that practically all features of concurrency can be modeled. For example, Milner's CCS is based on one-to-one synchronous communications of messages via communication channels. But if needed, thanks to its universality, one can model (by means of CCS) also one-to-all communications (broadcasting), asynchronous communications, communications of names of channels (as messages) etc. In those cases new auxiliary processes have to be introduced. They imitate a broadcaster, an asynchronous communication medium, passing of channels names, and so on. This approach generates two kinds of problems. On a syntactical level, resulting descriptions might be very complex. A number of auxiliary processes can increase dramatically and overall system analysis can be very difficult. Not only due to much higher complexity (of resulting descriptions) but also due to problems with distinguishing fundamental and auxiliary processes. This happens if one wants to analyze such a specification with respect to number of communications or number of processes needed to perform a particular task.

On a semantical level, some important aspects cannot be treated at all. For example, semantics which take into account localities or distribution of processes cannot be used.

It turns out that in many cases it is more advantageous to modify a "standard" process algebra instead of indirectly modeling all features which cannot be expressed directly. Following the above mentioned arguments many specialized process algebra are suggested in the literature. They are tailored for particular needs (timing, way of communication, system architecture etc.).

The aim of this paper is to discuss some problems that arise when process algebras are used to specify lower level communication protocols, and to propose a more adequate specification language.

A basic skeleton of many communication protocols is as follows: a *sender* sends a message to a *receiver*; after receiving the message the receiver sends an acknowledgment back to the sender. So, the sender knows that data have arrived safely and is ready for a new transmission. The message might consist of just one bit or it can be a very complex data structure. The acknowledgment might have various forms as well. Typical CCS terms describing the sender and the receiver might be as follows:

$$\begin{aligned} Sender &= \sum_{x \in M} input_x.S_x \\ S_x &= \overline{trans_x}.(ack.Sender + S_x) \text{ for } x \in M \\ Receiver &= \sum_{x \in M} trans_x.(\overline{ack}.\overline{output_x}.Receiver + Receiver) \end{aligned}$$

Above, actions are composed with the operators "+" and ".", standing for choice and sequential composition respectively. To distinguish between input and output actions, output actions are overlined. The set M represents all possible messages to be transmitted. Process *Sender* can input any message $x \in M$ from an external medium, and then it behaves as process S_x. Choices within brackets in processes S_x and *Receiver* express the possibilities that a message may get lost or be corrupted during transmission. So, S_x performs an output action $\overline{trans_x}$ and, afterwards, it can either receive an acknowledgment (input action ack) and be ready for a further transmission, or do nothing (it behaves as S_x). Process *Receiver*, after performing an input action $trans_x$, can either transmit an acknowledgment and output the message to an external medium, and then be ready for further receiving, or do nothing (namely behave as *Receiver*). The protocol skeleton can now be described as a parallel run of processes Sender and Receiver which can synchronize on complementary input-output actions, called also *actions* and *coactions* (for example, ack and $\overline{ack}$). Actions ack and $trans_x$ are intended to be internal to the system, namely they can be used only for a communication among system components. So, the complete process is

$$CoSy = (Receiver \mid Sender) \setminus \{trans_x, x \in M\} \cup \{ack\}.$$

From this skeleton concrete protocols are derived, tailored to different properties of the medium by means of which messages are sent, different length of packages of information to be sent, and so on.

The above specification can be proved to be correct in the following sense. The system takes an input message, then it makes two or more internal actions (depending on the number of errors attributable to the medium) and eventually (if the medium cannot lose a message infinitely many times) it makes an output action. If one does not consider internal actions, process $CoSy$ behaves like process $CoSy' = input_x.\overline{output_x}.CoSy'$.

There is at least one not very well specified aspect of the specification, namely that it would be more realistic to allow the sender to retransmit a message only in the case that an acknowledgment has not arrived in time. Earlier transmissions have no sense. Now several timed extensions of process algebras have been introduced in the literature (see the overview article [1] and [15, 16]), which usually offer the ability to specify a *timeout* property. So, by means of an enriched calculus, the protocol skeleton could be specified as follows:

$$\begin{aligned} Sender &= \sum_{x \in M} input_x.S_x \\ S_x &= \overline{trans_x}.\,\texttt{Timeout}(ack.Sender, S_x, rt) \text{ for } x \in M \\ Receiver &= \sum_{x \in M} trans_x.(\overline{ack}.\overline{output_x}.Receiver + Receiver) \\ CoSy &= (Receiver \mid Sender) \setminus \{trans_x, x \in M\} \cup \{ack\} \end{aligned}$$

where $\texttt{Timeout}(ack.Sender, S_x, rt)$ indicates that S_x is performed only if during time rt action ack has not been performed. (It might be a new operator or just a property expressible in a time-enriched language.) Response time rt corresponds to time needed for both message and acknowledgment transmissions plus time for error recovery plus time needed to form the acknowledgment, etc.

But things are not so simple. Every timed process algebra has to deal with a *minimal idling property*. This property means that if two components can communicate they have to communicate immediately. In general, there are two possibilities, namely either to allow minimal idling or not. The first case corresponds to a system whose components are fully connected or better, that can be considered to be such at a given level of abstraction. In this case, if two components are willing to communicate there is no obstacle in their way. On the other hand, if the components may have to wait for a communication channel, then non-minimal idling must be allowed. In fact, such idling should be allowed if the number of free channels is less than the number of communications which might be performed at any given moment.

Let us now return to our specification. Time *rt* represents the only problem. If processes have a full interconnection network at their disposal, with constant link speeds and such that other delays can also be reasonably estimated, the protocol specification works. But let us assume a more realistic environment. Suppose that there are three pairs of Senders and Receivers which have to share a full-duplex communication line. We can assume that (following an underlying protocol) the duplexer fairly assigns lines according to requests. Now, our specification might fail completely. If a sender sends its message and immediately another does the same, then both the links are busy. If the third sender also wants to send its message, it has to wait until one of the previous transmissions finishes before being able to transmit. But suppose that on the way back an acknowledgment is delayed. The sender, waiting for that acknowledgment, understands that it has to send its message again. This may cause the network to become overloaded by redundant messages. Each of them increases traffic and delays others. Finally one could get an infinite sequence of transmissions without any of the three messages being both sent and acknowledged allowing the next transmission. And this happens even when no message gets lost or corrupted. This is the consequence of the shared communication network, or more precisely of the fact that its capacity (number of independent communication paths) is lower than the number of transmissions to be carried on. The transmission time represents the main part of *rt* while the rest can be usually neglected. If we abstract away the communication network, any *rt* time that we take as a parameter of the timeout operator might cause unwanted system behaviors. So, since typical networks are shared, we conclude that there is a need to develop a more suitable specification calculus. This calculus should follow a typical communication architecture scheme, but the scheme should be general enough to cover various existing networks.

To illuminate the design of the language we are going to present, let us start with a typical example where the above mentioned protocol is used. We assume a system consisting of DTE's, their DCE's and a communication network (DCE stands for a *data circuit-terminating equipment* and DTE for a *data terminal equipment*, respectively (see [7])). An interconnection network is schematically depicted in Fig. 1. Requests for data transmission are sent from a DTE to its DCE; for these transmissions there are fast local physical links (Physical Communications) ([7]). They can be abstracted as a full interconnection network. Connections between DCE's are usually via a network of logical links (Logical Communications). For communications between DTE's and their DCE's and communications between DTE's and their environment we accept the handshaking paradigm (a complete interconnection network), for communications among DCE's we assume a network with a limited capacity.

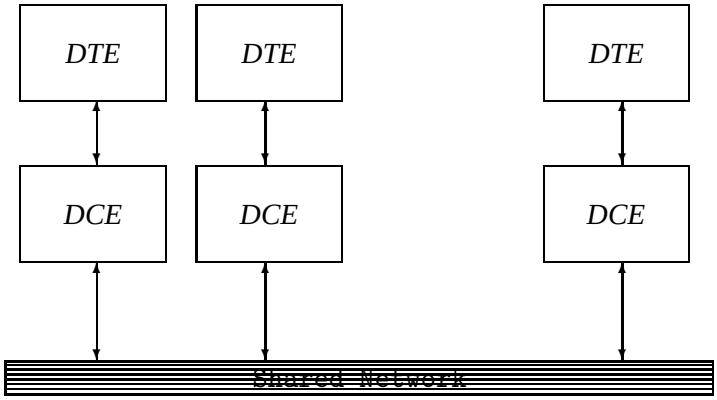

Figure 1. Network Scheme

We assume that performing communications via a limited capacity network decreases the number of free communication links. It means that a communication of that kind consumes a communication link, and therefore the communication is possible only if there is free link in the network. After completion of the communication the link is free again. At the description level presented, we do not distinguish between active interconnection networks, such as crossbars, n-dimensional cubes, omega networks, and passive ones, such as multiple global buses, rings, trees, multiplexing, optical networks with wavelength-divison multiplexing and so on (see [7]). Similarly, we do not distinguish between physical and logical connections. We only require that n links can transmit n pieces of information at the same time. We assume that the current number of free links depends on the current number of communicating processes. This means that, during an execution, a process may have to wait for a free communication link. Now, if there are no free links, a networked process cannot perform any communication via the corresponding (sub) network. At this point we depart from a common assumption of timed calculi that communications are instantaneous (minimal idling property, see [8]), which simplifies problems of communications to an unrealistic level, and instead we assume a *restricted minimal idling property*. This means that, in case of "limited" communications, an internal communication can idle if there is no free communication link in a network. In general, we assume that the duration of an action might be different from the duration of its communication part, i.e. the time for which the action occupies a link. So, this does not lead to a restriction on the maximal number of concurrently running actions or processes.

In general, the presented calculus allows to model two types of networks and any combination of them. In this way it covers any possible network architecture. Fig. 2 shows a network of subsystems $P_1, \ldots P_n$ which are connected via an "ethernet like" link of a limited capacity. That means any communication between P_i and P_j occupies one link (not just a path between P_i and P_j). Communications inside each component P_i are considered to be made by means of a fully connected network.

A network where a communication between two subsystems has no influence on other commnucations is depicted on Fig. 3

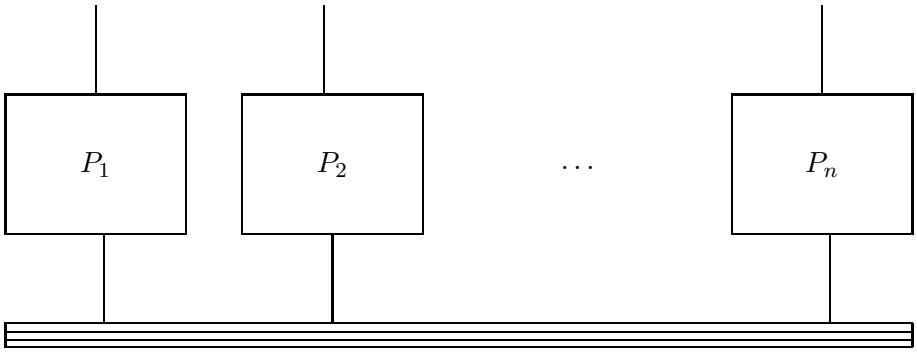

Figure 2. Network Scheme 1

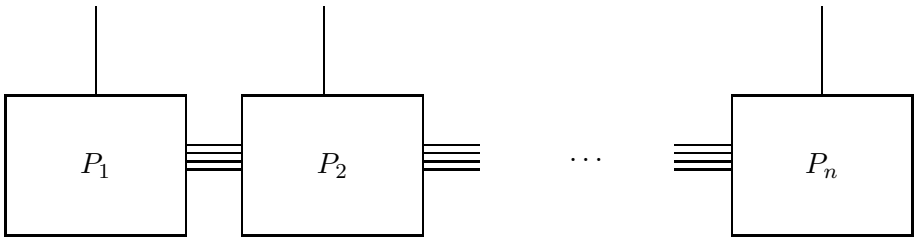

Figure 3. Network Scheme 2

As a basis of our specification language we take CCS, which is simple and yet powerful enough to express parallelism, non-determinism and synchronization, and we enrich it with the capacity of expressing time and network properties. First we introduce into the calculus an auxiliary action t that represents a run of discrete time. Actually, this approach is rather common, and since it seems to be convenient also for our purposes we adopt it. Action t will allow us to model timeout and action duration. To express durations, actions will be enveloped by sequences of t. This enveloping will be called a *time refinement*. Actually we will distinguish between duration of an action a ($d(a)$) and duration of a "communication" part of the action ($c(a)$). With this distinction process behaviour may be specified in better detail. We assume that the time needed for a communication might be shorter than the time needed to complete the whole action. For example, if $c(a) = 2$ and $d(a) = 4$ then every occurrence of action a will be followed by four t actions. In this sense the duration is syntactically expressed in process description by t actions while the duration of communication part is expressesed only by the function c. We prefer this solution instead of employing a new auxiliary action to express ends of communications.

Now an atomic action will not represent a communication but just its beginning. We will distinguish two kinds of communications. For communications (of concurrently running processes) via a complete network we will use the standard CCS parallel operator $|$. For communications via a network with a limited number of links we will use the new parallel operators $[. \parallel ..... \parallel .]^n_L$. To count the number of busy links, we will indicate the total number of links and the current number of busy links. As regards busy links, we need two pieces of information. The number of busy links and the time needed to finish communications. We will indicate this by a multi-set of positive integers $L = \{n_1, n_2, \ldots, n_k\}$. A process $[P_1 \parallel P_2 \ldots \parallel P_m]^n_L$ has n links at its disposal but at the moment k of them are busy. Number n_i

indicates how many time units are needed to complete a communication and to release a link.

For the specification language presented we give an operational semantics in terms of labeled transition systems. This is sufficient for analysis of process behaviour, but if one needs to check whether two specifications represent the "same" behaviour, different - usually equivalence based - semantics are required. We consider both strong and weak bisimulation, and we prove properties about them. By means of weak bisimulation a testing equivalence is defined. This equivalence is based on the usual observational scenario, but also networked testing processes are considered which interact with tested processes via the net. The bisimulations mentioned allow comparing networked timed processes, but one may have a specification as a CCS term and want to compare it with a networked process. Therefore the need arises of introducing a concept of abstracting time and network properties. For this purpose two equivalences are defined and their properties are discussed.

The paper is organized as follows. In Section 2 we describe the language NTiCCS and give an operational semantics for processes in terms of labeled transition systems. In Section 3 we present several algebraic semantics and their properties. In Section 4 we discuss our approach and compare it with related works.

2. The language

The language presented - NTiCCS - is based on CCS, whose philosophy is maintained even though some new features are introduced. These will enable the modeling of duration of actions and network exploitation, thus allowing specification and analysis of time properties of concurrent systems running in a concrete network. Similarly to CCS, in NTiCCS atomic actions represent basic execution units. A run-of-time unit is expressed by a special time action t.

To define the language NTiCCS, we first presuppose the set of atomic action symbols A not containing symbols τ and t, and such that for every $a \in A$ there exists $\overline{a} \in A$ and $\overline{\overline{a}} = a$. We define $Act = A \cup \{\tau\}, Actt = Act \cup \{t\}$. We assume that $a, b, \ldots$ range over A, $u, v, \ldots$ range over Act, and $x, y \ldots$ range over $Actt$. We suppose that L is a finite multi-set of positive integers. By $|L|$ we will indicate the cardinality of L. Let $L = \{n_1, \ldots, n_k\}$. By $L - 1$ we will mean a multi-set $\{n_i - 1 | 1 \leq i \leq k, n_i \in L, n_i - 1 > 0\}$. If L is empty then $L - 1$ is the empty set. Let $c, d : A \to N$ such that $d(a) \geq c(a)$. By $d(a)$ we will indicate the duration of action a and by $c(a)$ we will indicate the duration of a communication part of a.

The set of NTiCCS terms over the signature Σ is defined by the following BNF notation:

$$P \ ::= \ X \ \mid \ op(P_1, P_2, \ldots P_n) \ \mid \ \mu X P$$

where $X \in Var$, Var is a set of process variables, $P, P_1, \ldots P_n$ are NTiCCS terms, $\mu X -$ is the binding construct, $op \in \Sigma$. Assume the signature $\Sigma = \bigcup_{n \geq 0} \Sigma_n$ where

$$\begin{aligned}
\Sigma_0 &= \{Nil\} \\
\Sigma_1 &= \{x \mid x \in A \cup \{t\}\} \cup \{[S] \mid \text{where } S \text{ is a relabeling function over } Actt\} \cup \{\backslash M \mid M \subseteq A\} \\
\Sigma_2 &= \{|, +, [. \parallel .]_L^n\} \\
\Sigma_j &= [. \parallel . \parallel \ldots \ldots \parallel .]_L^n, j > 2
\end{aligned}$$

with the agreement to write unary action operators in prefix form, the unary operators $[S], \backslash L$ in postfix form, and the rest of operators in infix form. To have terms shorter sometimes the term Nil will be omitted.

Relabeling functions, $S : Actt \rightarrow Actt$ are such that $\overline{S(a)} = S(\bar{a})$ for $a \in A, S(\tau) = \tau$ and $S(t) = t$. Moreover, $M \subseteq A$, $n \in N^+$ and L is a multi-set of positive integers.

The set of CCS terms consists of NTiCCS terms without t action and $[. \parallel . \parallel \ldots . \parallel .]^n_L$ operators.

A structural operational semantics of terms will be defined in terms of labeled transition systems. The set of terms represents a set of states, labels are actions from $Actt$. The transition relation $\rightarrow$ is a subset of NTiCCS $\times$ $Actt$ $\times$ NTiCCS. We write $P \xrightarrow{x} P'$ instead of $(P, x, P') \in \rightarrow$ and $P \not\xrightarrow{x}$ if there is no P' such that $P \xrightarrow{x} P'$. The meaning of the expression $P \xrightarrow{x} P'$ is that the term P can evolve to P' by performing action x, by $P \xrightarrow{x}$ we will denote that there exists a term P' such that $P \xrightarrow{x} P'$. We define the network transition relation as the least relation satisfying the following inference rules:

$$\frac{}{x.P \xrightarrow{x} P} \quad A1$$

$$\frac{}{u.P \xrightarrow{t} u.P} \quad A2$$

$$\frac{}{Nil \xrightarrow{t} Nil} \quad A3$$

$$\frac{P \xrightarrow{u} P'}{P \mid Q \xrightarrow{u} P' \mid Q} \quad Pa1$$

$$\frac{P \xrightarrow{u} P'}{Q \mid P \xrightarrow{u} Q \mid P'} \quad Pa2$$

$$\frac{P \xrightarrow{a} P', Q \xrightarrow{\bar{a}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \quad Pa3$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q', P \mid Q \not\xrightarrow{\tau}}{P \mid Q \xrightarrow{t} P' \mid Q'} \quad Pa4$$

$$\frac{P \xrightarrow{u} P'}{P + Q \xrightarrow{u} P'} \quad S1$$

$$\frac{P \xrightarrow{u} P'}{Q + P \xrightarrow{u} P'} \quad S2$$

$$\frac{P \xrightarrow{t} P', Q \xrightarrow{t} Q'}{P + Q \xrightarrow{t} P' + Q'} \quad S3$$

$$\frac{P_i \xrightarrow{u} P_i', \text{for some } i, 1 \leq i \leq m}{[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m]_L^n \xrightarrow{u} [P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_m]_L^n} \quad N1$$

$$\frac{P_i \xrightarrow{a} P_i', P_j \xrightarrow{\overline{a}} P_j', \text{for some } i, j, 1 \leq i, j \leq m, |L| < n}{[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_j \parallel \ldots \parallel P_m]_L^n \xrightarrow{\tau} [P_1 \parallel \ldots \parallel P_i' \parallel \ldots \parallel P_j' \parallel \ldots \parallel P_m]_{L \cup \{c(a)\}}^n} \quad N2$$

$$\frac{P_i \xrightarrow{t} P_i', \text{for every } i, 1 \leq i \leq m \text{ and } [P_1 \parallel \ldots \parallel P_m]_L^n \not\xrightarrow{\tau}}{[P_1 \parallel \ldots \parallel P_m]_L^n \xrightarrow{t} [P_1' \parallel \ldots \parallel P_m']_{L-1}^n} \quad N3$$

$$\frac{P \xrightarrow{x} P'}{P \setminus M \xrightarrow{x} P' \setminus M}, (x, \overline{x} \notin M) \quad Res$$

$$\frac{P[\mu X P/X] \xrightarrow{x} P'}{\mu X P \xrightarrow{x} P'} \quad Rec$$

$$\frac{P \xrightarrow{x} P'}{P[S] \xrightarrow{S(x)} P'[S]} \quad Rl$$

Here we mention rules which are new with respect to CCS. Axioms $A2, A3$ allow arbitrary idling. Concurrent processes connected via full connection network can idle only if there is no possibility of an internal communication ($Pa4$). A run of time is deterministic ($S3$). If concurrent processes are connected via a network of a limited capacity, they behave similarly except internal communications and t action. Any internal communication increases the number of busy links. It adds to L the duration of the communication part of the corresponding actions ($N2$). By any t step every member of L is decreased by 1. If any number was equal to 1 then after that step it is removed from L. This means that a link gets free. The action t can be performed ($N3$) if there is no possibility of an internal communication either between processes P_i, P_j (there are no two processes ready to communicate or there is no free link for such communication) or there is not a possibility of an internal communication "inside" P_i for every i (it can be easily proved by structural induction that $P \xrightarrow{t}$ implies $P \not\xrightarrow{\tau}$).

In the definition of the labeled transition system we have used negative premises (see $Pa4, N3$). In general this may lead to problems, for example with consistency of the defined system. We avoid these dangers by making derivations of τ independent of derivations of t. For an explanation and details see [3].

In CCS, terms correspond to abstract states of a system. But not every NTiCCS term (which is more concrete) corresponds to a CCS one. Some of them can be viewed as intermediate states and some have no meaning at all. For example, let us consider the CCS process $a.Nil$ and its timed version $a.t.t.Nil$, which indicates that the action a takes two time units, and more precisely that its communication part takes one time unit and its post communication part takes another time unit, i.e. $d(a) = 2$ and $c(a) = 1$. The timed process can perform a and then it behaves like $t.t.Nil$. Now, this process has no counterpart

in CCS; the action a has started but has not finished yet, which is against CCS philosophy of atomic and instantaneous actions.

Example 2.1. Let us return to the protocol considered in the introduction. Now we can specify three Sender-Receiver pairs running in a duplex network, i.e. a network of capacity 2. Let us estimate the response time to be just one time unit. The specification is the following:

$$System = [CoSy[S_1] \parallel CoSy[S_2] \parallel CoSy[S_3]]^2_{\{\emptyset\}}$$

where S_1, S_2 and S_3 are relabeling functions which index all actions by 1,2 and 3, respectively, and $CoSy$ is as follows:

$$\begin{aligned} Sender &= \sum_{x \in M} input_x.S_x \\ S_x &= \overline{trans_x}.t.(ack.t.Sender + t.S_x) \text{ for } x \in M \\ Receiver &= \sum_{x \in M} trans_x.t.(\overline{ack}.t.\overline{output_x}.Receiver + Receiver) \\ CoSy &= (Receiver \mid Sender) \setminus \{trans_x, x \in M\} \cup \{ack\}. \end{aligned}$$

This new specification permits a finer description of the desired protocol behaviour. In fact, it allows us to distinguish between delays in delivery caused by network traffic and fundamental errors in the protocol.

3. Algebraic network semantics

In this section we define an algebraic semantics by means of a bisimulation, i.e. as an equivalence relation on the set of processes. Informally, we require that if two processes "behave in the same manner", then both of them can perform the same actions, and resulting processes again "behave in the same manner". A formal definition follows.

Definition 3.1. A binary relation $\Re \subseteq$ NTiCCS $\times$ NTiCCS is called a *(strong) bisimulation* if $P \Re Q$ implies, for all $x \in Actt$,

1. Whenever $P \xrightarrow{x} P'$ then, for some Q', $Q \xrightarrow{x} Q'$ and $P' \Re Q'$
2. Whenever $Q \xrightarrow{x} Q'$ then, for some P', $P \xrightarrow{x} P'$ and $P' \Re Q'$.

Two terms P, Q are *bisimilar*, abbreviated $P \sim Q$, if there exists a bisimulation relating P and Q.

Compositionality is an important property of any semantics. It allows bottom-up design of specifications, i.e. a composition of larger systems from smaller ones. Now we show that bisimulation supports this specification technique. In other words, we show that the relation $\sim$ is a congruence. Since the recursion operator handles open terms, i.e. terms with free variables, we have to extend the definition of bisimulation to the set of all terms. Let X be the only free variable occurring in P or Q or both. Then $P \sim Q$ iff, for every NTiCCS closed and guarded term R, $P[R/X] \sim Q[R/X]$.

Theorem 3.2. Let $P_i, Q_i,$ for $1 \leq i \leq m$, and R be NTiCCS terms, and let $P_i \sim Q_i$. Then

$$\begin{array}{rcl} x.P_i & \sim & x.Q_i \\ P_i + R & \sim & Q_i + R \\ P_i \mid R & \sim & Q_i \mid R \\ [P_1 \parallel \ldots \parallel P_m]^n_L & \sim & [Q_1 \parallel \ldots \parallel Q_m]^n_L \\ P_i[S] & \sim & Q_i[S] \\ P_i \setminus L & \sim & Q_i \setminus L \\ \mu X P_i & \sim & \mu X Q_i. \end{array}$$

Proof: The proof is a slight modification of the proof of the analogous property of CCS (see e.g. [13]). We shall give the proof for the$[. \parallel \ldots . \parallel .]^n_L$ operators. We need to show that $\bar{\Re}$ is a bisimulation, where $\bar{\Re}$ is the symmetric closure of

$$\Re = \{([P_1 \parallel \ldots \parallel P_m]^n_L, [Q_1 \parallel \ldots \parallel Q_m]^n_L) | P_i \sim Q_i, 1 \leq i \leq m\}.$$

Now suppose $([P_1 \parallel \ldots \parallel P_m]^n_L, [Q_1 \parallel \ldots \parallel Q_m]^n_L) \in \Re$. Let $[P_1 \parallel \ldots \parallel P_m]^n_L \xrightarrow{x} R$. There are three cases:

1. $[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m]^n_L \xrightarrow{u} [P_1 \parallel \ldots \parallel P'_i \parallel \ldots \parallel P_m]^n_L$, with $u \in Act$. Then, as $P_i \sim Q_i$, we have $[Q_1 \parallel \ldots \parallel Q_i \parallel \ldots \parallel Q_m]^n_L \xrightarrow{u} [Q_1 \parallel \ldots \parallel Q'_i \parallel \ldots \parallel Q_m]^n_L$ and $P'_i \sim Q'_i$.

2. $[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_j \parallel \ldots \parallel P_m]^n_L \xrightarrow{\tau} [P_1 \parallel \ldots \parallel P'_i \parallel \ldots \parallel P'_j \parallel \ldots \parallel P_m]^n_{L \cup \{c(a)\}}$, then since $P_i \sim Q_i, P_j \sim Q_j$, we have $[Q_1 \parallel \ldots \parallel Q_i \parallel \ldots \parallel Q_j \parallel \ldots \parallel Q_m]^n_L \xrightarrow{\tau} [Q_1 \parallel \ldots \parallel Q'_i \parallel \ldots \parallel Q'_j \parallel \ldots \parallel Q_m]^n_{L \cup \{c(a)\}}$ and $P'_i \sim Q'_i, P'_j \sim Q'_j$.

3. $[P_1 \parallel \ldots \parallel P_m]^n_L \xrightarrow{t} [P'_1 \parallel \ldots \parallel P'_m]^n_{L-1}$. By using similar arguments as in the previous case we get $[Q_1 \parallel \ldots \parallel Q_m]^n_L \xrightarrow{t} [Q'_1 \parallel \ldots \parallel Q'_m]^n_L$ and $P'_i \sim Q'_i$ for $1 \leq i \leq m$.

□

Processes are designed to model concurrent systems when actions are non-atomic, in the sense that they might have a duration. Usually with such assumptions the interleaving paradigm must be abandoned. In fact, if an execution of actions takes some time, then a parallel run of two actions is easily distinguished from the nondeterministic choice of their sequentializations. Processes describe "system actions" by means of sequences of "language actions". These indicate start of "system actions", their duration, and so on. And, what is important, they can be interleaved. Hence we can formulate two kinds of expansion theorem, one for $\mid$ and one for $[. \parallel \ldots \parallel .]^n_L$ operators. With the help of these theorems processes can be sequentialized, i.e. parallel operators can be removed. This turns out to be particularly important for a proof system, as we shall see later. In this way parallelism is replaced by nondeterminism. For example, process $[a.Nil \parallel b.Nil]^n_L$ can be replaced by the non-deterministic sequential process $a.b.Nil + b.a.Nil$. On the one hand, the resulting (sequential) processes are syntactically more complex, but, on the other hand, as we shall show, there is an axiomatic characterization of bisimulation for such processes. Proofs of the both theorems come directly from the inference rules.

Theorem 3.3. (Expansion Theorem 1) Let $P \equiv (P_1[S_1]| \ldots |P_m[S_m]) \setminus M$.

$$\begin{aligned} P \sim & \sum_{P_i \xrightarrow{u} P_i', S_i(u) \notin M \cup \bar{M}} S_i(u).(P_1[S_1]| \ldots |P_i'[S_i]| \ldots |P_m[S_m]) \setminus M \\ & + \sum_{P_i[S_i] \xrightarrow{a} P_i'[S_i], P_j[S_j] \xrightarrow{\bar{a}} P_j'[S_j]} \tau.(P_1[S_1]| \ldots |P_i'[S_i]| \ldots |P_j'[S_j]| \ldots |P_m[S_m]) \setminus M \\ & + \sum_{P_i \xrightarrow{t} P_i', \text{ for every } i, P \not\xrightarrow{\tau}} t.(P_1'[S_1]| \ldots |P_m'[S_m]) \setminus M \end{aligned}$$

Theorem 3.4. (Expansion Theorem 2) Let $P \equiv [P_1[S_1] \parallel \ldots \parallel P_m[S_m]]_L^n \setminus M$. For every n, m and L

$$\begin{aligned} P \sim & \sum_{P_i \xrightarrow{u} P_i', S_i(u) \notin M \cup \bar{M}} S_i(u).[P_1[S_1] \parallel \ldots \parallel P_i'[S_i] \parallel \ldots \parallel P_m[S_m]]_L^n \setminus M \\ & + \sum_{P_i[S_i] \xrightarrow{a} P_i'[S_i], P_j[S_j] \xrightarrow{\bar{a}} P_j'[S_j], |L| < n} \tau.[P_1[S_1] \parallel \ldots \parallel P_i'[S_i] \parallel \ldots \parallel P_j'[S_j] \parallel \ldots \parallel P_m[S_m]]_{L \cup \{c(a)\}}^n \setminus M \\ & + \sum_{P_i \xrightarrow{t} P_i' \text{ for every } i, P \not\xrightarrow{\tau}} t.[P_1'[S_1] \parallel \ldots \parallel P_m'[S_m]]_{L-1}^n \setminus M \end{aligned}$$

As regards a proof system for bisimulation we focus our attention on finite processes, since, due to the Turing-power of CCS, there is no effective complete proof system for arbitrary processes. Such a system helps to answer the fundamental question for a system designer, namely whether a formal system description satisfies a system specification. For example, in the case of a communication protocol, whether a description of the protocol (detailed formalization of all components as senders, receivers, mediums and their cooperation) satisfies desired behaviour, i.e.that data are transmitted "correctly" (specified in some language, possibly in the same language).

With the help of the expansion theorems (Theorem 3.3 and 3.4) we can gradually remove parallel operators and replace them by $+$. Then to show that two finite processes P, Q are equivalent we can use the proof system given by Tab. 1.

Let E denote the set of equations in Tab. 1 together with the equation given by the Expansion Theorems and, let $P =_E Q$ mean that P can be transformed into Q by application of the equations in E. Then we have the following theorem.

Theorem 3.5. For finite processes P, Q, $P \sim Q$ holds if and only if $P =_E Q$.

Proof: We use a technique of normal forms. With the help of the Expansion Theorems and axioms $A9 - 15$ we can safely transform any term into an equivalent sequential one, i.e. into the term which does not contain parallel operators, renaming and restriction operators. Then using axioms $A1 - 8$ terms are simplified. For the rest of the proof it is easy to show that two equivalent terms have the same normal forms. A normal form differs from that of CCS only in that a new kind of summands (given by "distributivity" of t prefixing) is added. □

So, if we have a protocol specification and the specification of desired protocol properties, both given in terms of our calculus, we can use the equations to check equivalence, in the sense of bisimilarity.

$P + P$	$=$	P	$A1$
$P + Q$	$=$	$Q + P$	$A2$
$P + (Q + R)$	$=$	$(P + Q) + R$	$A3$
$P + Nil$	$=$	P	$A4$
$t.Nil$	$=$	Nil	$A5$
$t.(P + Q)$	$=$	$t.P + t.Q$	$A7$
$t.u.P + u.P$	$=$	$u.P$	$A8$
$Nil \setminus M$	$=$	Nil	$A9$
$(x.P) \setminus M$	$=$	Nil if $x, \overline{x} \in M$	$A10$
$(x.P) \setminus M$	$=$	$x.(P \setminus M)$ otherwise	$A11$
$(P + Q) \setminus M$	$=$	$P \setminus M + Q \setminus M$	$A12$
$Nil[S]$	$=$	Nil	$A13$
$(x.P)[S]$	$=$	$S(x).(P[S])$	$A14$
$(P + Q)[S]$	$=$	$P[S] + Q[S]$	$A15$

Table 1. The equational characterization of $\sim$.

For processes we have also developed a model checker that allows verifying properties expressed in a logical language, *tick*-ACTL, a timed variant of ACTL [12].

Strong bisimulation is usually considered to be too discriminating. It distinguishes also processes which cannot be distinguished by an observer who does not see internal communications. A week bisimulation overcomes this.

Definition 3.6. If $x \in Actt$ then $\hat{x} = x$ if $x \neq \tau$ and $\hat{\tau} = \epsilon$ (empty sequence). We will write $P \stackrel{x}{\Rightarrow} P'$ if $P(\stackrel{\tau}{\rightarrow})^* \stackrel{x}{\rightarrow} (\stackrel{\tau}{\rightarrow})^* P'$.

Definition 3.7. A binary relation $\Re \subseteq \text{NTiCCS} \times \text{NTiCCS}$ is called a *weak bisimulation* if $P \Re Q$ implies, for all $x \in Actt$, that

1. whenever $P \stackrel{x}{\rightarrow} P'$ then, for some Q', $Q \stackrel{\hat{x}}{\Rightarrow} Q'$ and $P' \Re Q'$
2. whenever $Q \stackrel{x}{\rightarrow} Q'$ then, for some P', $P \stackrel{\hat{x}}{\Rightarrow} P'$ and $P' \Re Q'$.

Two terms P, Q are *weakly bisimilar*, abbreviated $P \approx Q$, if there exists a weak bisimulation relating P and Q.

One cannot expect that weak bisimulation would be congruence because it is not a congruence in CCS, which is actually a sub-calculus of NTiCCS. But, similarly to CCS, it is a congruence with respect to all the operators but nondeterministic choice. A relation stronger than weak bisimulation and which is congruence can be obtained by similar technique as for CCS.

Theorem 3.8. Let $P_i, Q_i,$ for $1 \leq i \leq m$, and R be NTiCCS terms, and let $P_i \approx Q_i$. Then

$$\begin{aligned} x.P_i &\approx x.Q_i \\ P_i \mid R &\approx Q_i \mid R \\ [P_1 \parallel \ldots \parallel P_m]^n_L &\approx [Q_1 \parallel \ldots \parallel Q_m]^n_L \\ P_i[S] &\approx Q_i[S] \\ P_i \setminus L &\approx Q_i \setminus L \\ \mu X P_i &\approx \mu X Q_i. \end{aligned}$$

Proof: Again the proof is a slight modification of the proof of the analogous property of CCS (see e.g. [13]). We shall give the proof for the $[. \parallel \ldots . \parallel .]^n_L$ operators. We need to show that $\bar{\Re}$ is a weak bisimulation, where $\bar{\Re}$ is the symmetric closure of

$$\Re = \{([P_1 \parallel \ldots \parallel P_m]^n_L, [Q_1 \parallel \ldots \parallel Q_m]^n_L) | P_i \approx Q_i, 1 \leq i \leq m\}.$$

Now suppose $([P_1 \parallel \ldots \parallel P_m]^n_L, [Q_1 \parallel \ldots \parallel Q_m]^n_L) \in \Re$. Let $[P_1 \parallel \ldots \parallel P_m]^n_L \xrightarrow{x} R$. There are four cases.

1. $[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_m]^n_L \xrightarrow{u} [P_1 \parallel \ldots \parallel P'_i \parallel \ldots \parallel P_m]^n_L$, with $u \in Act$. Then, as $P_i \approx Q_i$, we have $[Q_1 \parallel \ldots \parallel Q_i \parallel \ldots \parallel Q_m]^n_L \stackrel{u}{\Rightarrow} [Q_1 \parallel \ldots \parallel Q'_i \parallel \ldots \parallel Q_m]^n_L$ and $P'_i \approx Q'_i$.

2. $[P_1 \parallel \ldots \parallel P_i \parallel \ldots \parallel P_j \parallel \ldots \parallel P_m]^n_L \xrightarrow{\tau} [P_1 \parallel \ldots \parallel P'_i \parallel \ldots \parallel P'_j \parallel \ldots \parallel P_m]^n_{L \cup \{c(a)\}}$, then since $P_i \approx Q_i, P_j \approx Q_j$, we have $[Q_1 \parallel \ldots \parallel Q_i \parallel \ldots \parallel Q_j \parallel \ldots \parallel Q_m]^n_L \stackrel{\tau}{\Rightarrow} [Q_1 \parallel \ldots \parallel Q'_i \parallel \ldots \parallel Q'_j \parallel \ldots \parallel Q_m]^n_{L \cup \{c(a)\}}$ and $P'_i \approx Q'_i, P'_j \approx Q'_j$.

3. $[P_1 \parallel \ldots \parallel P_m]^n_L \xrightarrow{t} [P'_1 \parallel \ldots \parallel P'_m]^n_{L-1}$. By using similar arguments as in the previous case, we get $[Q_1 \parallel \ldots \parallel Q_m]^n_L \stackrel{t}{\Rightarrow} [Q'_1 \parallel \ldots \parallel Q'_m]^n_L$ and $P'_i \approx Q'_i$ for $1 \leq i \leq m$.

□

Note that according to derivation rules Pa4 and N3 an execution of t actions of a "parallel" term depends on the possibility of an execution of a τ action. Since weak bisimulation does not take into account τ actions it has no influence on being a congruence with respect to parallel operators.

Quite a different situation occurs when we take even identical terms and let them communicate via two different limit capacity networks. Then the resulting behaviour strongly depends on network properties.

Theorem 3.9. For every n, with $n > 0$, there exist processes P, Q such that $[Q \parallel P]^n_\emptyset \not\approx [Q \parallel P]^{n+1}_\emptyset$.

Proof: Let

$$\begin{aligned} P &= a_1.\bar{x}_1.t.b_1.Nil|\ldots|\bar{a}_{n+1}.\bar{x}_{n+1}.t.b_{n+1}.Nil \\ Q &= x_1.t.Nil|\ldots|x_{n+1}.t.Nil \end{aligned}$$

and $c(x_i) = d(x_i) = 1$. It is easy to check that $[Q \parallel P]_\emptyset^{n+1} \stackrel{s}{\Rightarrow}$ but $[Q \parallel P]_\emptyset^n \stackrel{s}{\not\Rightarrow}$ where $s = a_1 \ldots a_{n+1} t b_1 \ldots b_{n+1}$.

□

Moreover, if we think about an observer as an environment which interacts with processes, we have to specify the way in which the environment and the processes communicate. For example, in the case of the ethernet like network (see Fig. 2), i.e. the network represented by term $[P_1 \parallel \ldots \parallel P_n]_L^k$, it would be natural to suppose that also the observer communicates via the same limited capacity network which interconnects processes. In this case, a more appropriate semantics than weak bisimulation would be a variant of testing semantics. An observer will employ a testing NTiCCS term which will communicate with tested processes via and only via a limited capacity network. The resulting semantics is sensitive to the number of possible communications which can be concurrently performed by the process under test since communications (τ actions) are not visible to an observer.

Definition 3.10. Let $P = \{P_1, \ldots, P_n\}, Q = \{Q_1, \ldots, Q_m\}$ be two multisets of NTiCCS terms and let $M_P = \bigcup_{i=1}^n \mathcal{M}(P_i)$ and $M_Q = \bigcup_{i=1}^m \mathcal{M}(Q_i)$ where $\mathcal{M}(R)$ is a set of actions from A which occur in a term R. We say that P and Q are testing equivalent via a network with capacity k, abbreviated $P \approx_k Q$, if for any NTiCCS term T it holds $[T \parallel P_1 \parallel \ldots \parallel P_n]_\emptyset^k \setminus M_P \approx [T \parallel Q_1 \parallel \ldots \parallel Q_m]_\emptyset^k \setminus M_Q$.

In general, relations $\approx_k$ are defined between multisets of terms but they can be restricted to relations on terms as $\approx_k \cap$ NTiCCS $\times$ NTiCCS. From now on we will consider these restricted relations. The following theorem shows that two processes which are equivalent in a network offering k links may not be such anymore in a network offering more than k links but they remain equivalent in a network offering less than k links.

Theorem 3.11. $\approx \neq \approx_k$ for every $k, k > 0$ and $\approx_1 \supset \approx_2 \supset \approx_3 \supset \ldots$.

Proof: Let $P \approx_{k+1} Q$ i.e. $[T \parallel P]_\emptyset^{k+1} \setminus M_P \approx [T \parallel Q]_\emptyset^{k+1} \setminus M_Q$. It is easy to check that then also $P|w.Nil \approx_{k+1} Q|w.Nil$, where w is a new action which does not occur in P, Q. Let us suppose that $c(w)$, with $c(w) = d(w)$, is infinite. Practically, we can choose a value which is big enough, as it will be seen later. Since both the processes have to pass any internal test T they should pass also the test $T|\bar{w}.t^{d(w)}.Nil$. If testing starts with a communication by action w, one link in the communication network will be occupied forever (for time $c(w)$) and hence the system will behave as a system with just k links. This means that $P \approx_k Q$. To show that $\approx_k \neq \approx_{k+1}$ let us consider the following processes:

$$\begin{aligned} P &= x_1.t|\ldots|x_{k+1}.t \\ Q &= (x_2.t.x_1.t|x_3.t|\ldots|x_{k+1}.t) + \cdots + (x_2.t|x_3.t|\ldots|x_{k+1}t.x_1.t) + \\ &\vdots \\ &\quad +(x_1.t.x_{k+1}.t|x_2.t|\ldots|x_k.t) + \cdots + (x_1.t|x_2.t|\ldots|x_k.t.x_{k+1}.t) \end{aligned}$$

It can be checked that $P \approx_k Q$ but $P \not\approx_{k+1} Q$ and $P \not\approx Q$. As a testing process we can take $T = a_1.\bar{x}_1.t.b_1 | \ldots | a_{k+1}.\bar{x}_{k+1}.t.b_{k+1}$ with $c(x_i) = 1$, and the rest is the same as in the proof of Theorem 3.9.

□

Another possibility of testing processes might be a combination of a testing term and direct observations. In this case we would allow examinations of processes by the testing term as well by an observer. This means that we would remove the restriction operators from Definition 3.10. But this would not result into new equivalences - all of them would be equal to the weak bisimulation.

The above mentioned notion is useful if one needs to compare the specification of a concurrent system with the specification of its desired behaviour. But what happens very often is that there is no full specification of the desired behaviour available, or it is not of interest.

For instance, we may have, on one side, a "networked" process P and, on the other side, an abstract CCS process Q. We would like to know whether these two processes behave in the same manner - except for network traffic and time, of course, since this is not described by Q. In this case bisimulation as it has been defined cannot be used. So we reformulate the original definition. We define a transition relation that abstracts t actions, namely we take $\stackrel{x}{\rightarrow_t} = (\stackrel{t}{\rightarrow})^* \stackrel{x}{\rightarrow} (\stackrel{t}{\rightarrow})^*$.

Definition 3.12. A relation $\Re \subseteq$ NTiCCS $\times$ NTiCCS is called a *t-abstracting bisimulation* if $P \Re Q$ implies, for all $x \in Actt$,

1. Whenever $P \stackrel{x}{\rightarrow_t} P'$ then, for some Q', $Q \stackrel{x}{\rightarrow_t} Q'$ and $P' \Re Q'$
2. Whenever $Q \stackrel{x}{\rightarrow_t} Q'$ then, for some P', $P \stackrel{x}{\rightarrow_t} P'$ and $P' \Re Q'$.

Two processes P, Q are *t-abstracting bisimilar*, abbreviated $P \sim_t Q$, if there exists a t-abstracting bisimulation relating P and Q.

Contrary to weak bisimulation, t-abstracting bisimulation is a congruence.

Theorem 3.13. Let $P_i, Q_i, P_i \sim Q_i, 1 \leq i \leq m$ and R be NTiCCS terms. Then

$$\begin{array}{rcl} x.P_i & \sim_t & x.Q_i \\ P_i + R & \sim_t & Q_i + R \\ P_i \mid R & \sim_t & Q_i \mid R \\ [P_1 \parallel \ldots \parallel P_m]^n_L & \sim_t & [Q_1 \parallel \ldots \parallel Q_m]^n_L \\ P_i[S] & \sim_t & Q_i[S] \\ P_i \setminus L & \sim_t & Q_i \setminus L \\ \mu X P_i & \sim_t & \mu X Q_i. \end{array}$$

Proof: We shall give the proof for the most interesting case, namely that of the choice operator. Let us suppose that $x \neq t$ and $R + P \stackrel{x}{\rightarrow_t} W$, which means there exists a sequence $s = t^i.x.t^j$ such that $R + P \stackrel{s}{\rightarrow} W$. This means that there are processes R', P' such that $R + P \stackrel{t^i}{\rightarrow} R' + P'$, $R' + P' \stackrel{x}{\rightarrow} W'$ and $W' \stackrel{t^j}{\rightarrow} W$. And this means that either $R \stackrel{s}{\rightarrow} W$ or $P \stackrel{s}{\rightarrow} W$. Since $P \sim_t Q$ we get $R + Q \stackrel{x}{\rightarrow_t} W''$, where $W \sim_t W''$. In case of $x = t$ the proof is similar.

□

Now let us consider an abstract CCS process P and suppose we know time properties of its elementary actions (i.e. a duration refinement given by functions d and c). We can get a timed version of P by syntactical substitution of every action a by the sequence $a.t^{d(a)}$. The resulting process can be denoted by $d(P)$.

But an observer who cannot see elapsing of time can distinguish between P and $d(P)$.

Example 3.14. Let

$$\begin{aligned} P &= (\bar{x}.Nil|x.a.Nil|b.c.Nil) \setminus \{x\} \\ d(P) &= (\bar{x}.Nil|x.a.Nil|b.t.c.Nil) \setminus \{x\}. \end{aligned}$$

It is easy to see that $P \stackrel{bc\tau a}{\rightarrow}_t$ but $d(P) \stackrel{bc\tau a}{\not\rightarrow}_t$.

The problem why these two processes behave in a different manner even when time is abstracted is rooted in minimal idling property. If there is a possibility of an internal communication (x) this has to be performed immediately, which means before b action is completed. To get a behaviour of the timed process $d(P)$ similar (up to time elapsing) we would need to change the rule $Pa4$ in the definition of labeled transition system for NTiCCS. The new rule should be as follows.

$$\frac{P \stackrel{t}{\rightarrow} P', Q \stackrel{t}{\rightarrow} Q'}{P \mid Q \stackrel{t}{\rightarrow} P' \mid Q'} \quad Pa'4$$

Transitions based on this rule will be denoted by $\rightarrow_{\prime}$ and $\rightarrow_{t'}$ and corresponding bisimulations will be denoted by $\sim_{\prime}$ and $\sim_{t'}$. In the case of the parallel operator, a t action can be performed regardless the possibility of internal communications. There is no information about free links so also the possibility that all of them are busy has to be admitted. In this case an internal communication cannot be performed. Hence time may elapse. If we would consider t-abstracting bisimulation based on the labeled transition system with the rule $Pa4$ replaced by $Pa'4$ (abbreviated $\sim_{t'}$) we will get that processes P and $d(P)$ are t-abstracting bisimilar. This means that an assumption about minimal idling property is not appropriate if there is no exact information about interconnection network properties.

Theorem 3.15. Let P be a CCS process. Then for any d it holds $P \sim_{t'} d(P)$.

Proof: By structural induction. We give the proof for the most interesting case when $P = P_1|P_2$ and when an action different from t is performed. Let $d(P) \stackrel{x}{\rightarrow}_{t'} P'$ i.e.without lost of generality we can assume that there exists a sequence $s = t^i.u.t^j$ such that $d(P_1)|d(P_2) \stackrel{s}{\rightarrow}_{\prime} P_1'|P_2'$. This means that there are processes X_i such that $d(P_1)|d(P_2) \stackrel{t^i}{\rightarrow}_{\prime} X_1|X_2$. Then there are two possibilities. Action u is performed just by one of processes X_i or it is an internal communication between them. Let us suppose that $X_1|X_2 \stackrel{u}{\rightarrow}_{\prime} X_1'|X_2$ and then $X_1'|X_2 \stackrel{t^j}{\rightarrow}_{\prime} P_1'|P_2'$. This means that $d(P_1) \stackrel{u}{\rightarrow}_{t'} P_1'$ and $d(P_2) \stackrel{t}{\rightarrow}_{t'} P_1'$. By induction assumption we get $P \stackrel{x}{\rightarrow}_{t'} P''$ and $P' \sim_{t'} P''$. Other cases are similar. □

The previous theorem allows us to define a semantics for CCS processes which is sensitive to duration properties of actions, hence it has to be parametrized by d.

Definition 3.16. Let P, Q be CCS processes and d a duration refinement. We define P, Q to be d-equivalent, abbreviation $P \sim_d Q$, if $d(P) \sim_{\prime} d(Q)$.

These equivalences are non-interleaving ones. It is easy to check that $(x_1.Nil|x_2.Nil) \not\sim_d (x_1.x_2.Nil + x_2.x_1.Nil)$ if $d(x_1) = d(x_2) = 1$. But all of them are refinements of strong bisimulation on CCS. Actually, if $P \sim_d Q$ then $d(P) \sim_{'} d(Q)$. It is easy to check that $\sim_{'} \subset \sim_{t'}$, and hence, since $P \sim_{t'} d(P)$ and $Q \sim_{t'} d(Q)$, we get $P \sim_{t'} Q$. It is clear that $\sim_{t'}$ and $\sim$ coincide on CCS terms, and hence $P \sim Q$.

Let us return to the original system NTiCCS. Process $d(P)$ does not contain network information, neither which processes communicate via a full connection or a limited capacity network nor the capacity of its subnetworks. This information can be added by replacing some $|$ operators by operators $[. \parallel \ldots\ldots \parallel]^n_L$. In this way a more concrete NTiCCS process, say P', is obtained from the abstract process P. At this moment we have to stress again that process P and P' are not t-abstracting bisimilar in general (see Example 3.14). There are several ways how to solve this dilemma. Either we change the embedding of process P into an interconnection network or we change the network architecture or we consider new action durations or we admit that the original abstract specification given by P was not correct.

We remark that in [4] similar bisimulations are studied. They relate CCS processes with timed CCS ones running in an environment which offers a limited number of processors. In case that there are more processes than processors they have to either execute only as many actions as are the processors at disposal, or they can run more actions but some of them have to be (temporarily) interrupted. These two possibilities lead to two different bisimulations on CCS processes, parametrized by time refinements and by the number of processors. It turns out that one of them coincides with t-abstracting bisimulation in case that all processes are fully connected, and another coincides with the case that all processes are connected via a network with a limited capacity and communication and duration parts of actions are equal.

4. Discussion and related works

In this paper we have defined a timed extension of CCS which takes into account properties of an interconnection communication network of limited capacity through which processes communicate.

Modeling a network behaviour within "pure" CCS would face the following obstacles. Specification on the low level considered requires to express time properties, such as that the network is occupied for a certain amount of time units. So we need to use a timed variant of CCS. Suppose we have one. To model the network we have to employ new processes which emulate it. Now, sending a message through the limited capacity network will be represented by sending the message to an auxiliary process. This process will keep the message for the time needed and then it will send it to a final destination. Moreover, the number of such "re-senders" has to be limited by the capacity of the network and they have to run in parallel.

For instance, let us model in such a way the simplest possible architecture consisting of two groups of processors communicating via a network of capacity n. In this case the number of auxiliary parallel running processes (modeling the network) goes up to $2n$. The number of additional communications is doubled as well. But not only system descriptions become much more complex. Suppose that the system described is an implementation of a communication protocol and one wants to analyze also its efficiency. This might depend on the number of communications needed to transmit data, number of processes involved, etc. All such analyses are complicated by auxiliary processes and communications. In fact, these should not be taken into account. And it does not matter whether we use a syntactical, semantical, dynamic or static analysis. Moreover, some methods do not allow to differentiate between

fundamental processes/communications and auxiliary ones at all. Note that more complicated is the network architecture we emulate more are the problems we have.

We have argued that it is more practical to develop directly a "networked" calculus, called NTiCCS. The more complicated syntax with respect to that of CCS is balanced by shorter, clearer and easier to analyze specifications containing also information on networking and distributivity. For NTiCCS we have developed operational semantics in Plotkin style and bisimulation semantics. An abstracting bisimulation relates a CCS specification with a NTiCCS more concrete one (containing information on a network) that can realize the wanted behaviour. The basic concepts of CCS have been maintained so that many of existing tools developed for CCS can be (with some minor modifications) used for NTiCCS as well. The network we assume in the present paper is a generalization of those assumed in preliminary versions [5] and [6]. Optical networks with WDM are expected to be the key technology for the future. They allow not only high speeds and multiplexing but they can be tuned for different types of data transmitted at the moment. Now, such tuning changes level of multiplexing and number of communication links. To model such networks the presented calculus can be modified in such a way that some communications (audio/video streams etc) either lower the original capacity of the network or, whenever they start, they decrease the number of free links, possibly by more than one, as several links at once may be occupied.

There are many examples how process algebras can be used for protocol descriptions and subsequent verifications. But, naturally, such descriptions are on such level of a protocol which can be captured by the specification language used. For example, in [10] a sequential consistency protocol for shared memory multiprocessor systems is specified in CSP and verified. The role of the protocol is to guarantee consistency among the shared memory and processors caches. The systems considered have architecture similar to the one of Fig. 1 but instead of DTE's and DCE's there are processors and their caches, respectively. The shared memory is connected directly to a shared network. While processors and their caches are directly connected, the shared network might be anything from a local bus to an optical network, in case of distributed computing. Actually CSP allows to describe only a high level of the protocol and does not permit taking into account time and network architecture, even though this is known.

In [11] a specification language TCOZ (Timed Communicating Object-Z) is presented. It extends the Object-Z notation with Timed CSP constructs and with a new network topology operator. This allows specification of complex networks of communicating processes. It can describe all possible networkings of single/multiple processes over a single/multiple channels. Authors argue that the operator simplifies and clarifies specifications, and therefore it should be included directly in the basic CSP notation. NTiCCS includes network operators but gives also an operational meaning to them, which is lacking in the case of TCOZ.

We want to mention also a paper ([14]) where a process algebra for a polymorphic subset of concurrent ML is presented. Using the process algebra a static analysis for communicating behaviour of CML programs is developed. Such analysis predicts the number of channels (and processes) required during the execution of the CML program. In case that a program creates only a finite number of channels (finite communication topology) we have information on how many links (at most) are needed to connect appropriate processes. But such upper bound is much above a reasonable trade-off between performance and cost.

References

[1] Baeten, J. C. M. and Bergstra, J. A.: "Discrete Time Process Algebra", *Formal Aspects of Computing*, **VolNo**(8), 1996, 188-208.

[2] Bergstra J. A. and Klop, J. W.: "Algebra of Communicating Processes with Abstraction", *Theor. Comput. Sci.*, **VolNo**(37), 1985, 77- 21.

[3] Groote, J. F.: "Transition Systems Specification with Negative Premises". Baeten, J.C.M. and Klop, J.W. (eds.), *CONCUR'90*, Springer Verlag, Berlin, LNCS 458, 1990, 332-341.

[4] Gruska, D. P.: "Bounded Concurrency". Chlebus, B. S. and Czaja, L. (eds.), *FCT'97*, Springer LNCS 1279, 1997, 198-209.

[5] Gruska, D. P. and Maggiolo-Schettini, A.: "Timed Network Semantics for Communicating Processes". Burkhard, H-D., Czaja, L. and Starke, P. (eds.), *Proceedings of CSP 93*, Warsaw University Press, 1994, 25-143.

[6] Gruska, D.P. and Maggiolo-Schettini, A.: "Process Communication Environment". Purushothaman S. and Zwarico A. (eds.), *Proceedings First North American Process Algebra Workshop*, Springer Verlag, Berlin, Workshops in Computing Series, 1993, 1-12.

[7] Halsall, F.: *Data Communications, Computer Networks and Open Systems.* Addison-Wesley, Reading, Massachusetts, 1998.

[8] Hansson, H. and Jonsson, B.: "A Calculus for Communicating Systems with Time and Probabilities". IEEE Computer Society Press, *Proceedings of 11th IEEE Real - Time Systems Symposium*, Orlando, 1990, 278-287.

[9] Hoare, C. A. R.: *Communicating Sequential Processes.* Prentice-Hall International, New York, 1985.

[10] Lowe, G. and Davies, J.: "Using CSP to verify sequential consistency", *Distributed Computing*, **VolNo**(12), 1999, 91-103.

[11] Mahony, B. and Dong, J.S.: "Network Topology and a Case Study in TCOZ". Bowen, J. P., Fett, A. and Hinchey, M. G. (eds.) *ZUM'98: The Z Formal Specification Notation*, Springer Verlag, Berlin, LNCS 1493, 1998, 308-327.

[12] Martinelli, F.: "McTACTL: A Tool to Verify NTiCCS Specifications", *Technical Report TR-21/94*, Dipartimento di Informatica, Università di Pisa, 1994.

[13] Milner, R.: *Communication and Concurrency.* Prentice-Hall International, New York, 1989.

[14] Nielson, H. R. and F. Nielson, F.: "Communication analysis for concurrent ML", *ML with Concurrency*, Springer Verlag, Berlin, Monographs in Computer Science, 1997, 185-235.

[15] Schneider, S.: *Concurrent and real-time systems: The CSP approach*, John Wiley & Sons, New York, 1999.

[16] Yi, W.: "CCS+Time = an Interleaving Model for Real Time Systems". Leach Albert, J., Monien B., and Rodriguez Artalejo, M. (eds), *ICALP'91*, Springer Verlag, Berlin, LNCS 510, 1991, 217-228.