FUNCTIONAL SPECIFICATION OF TIME SENSITIVE COMMUNICATING SYSTEMS[1]

Manfred Broy
Institut für Informatik
Technische Universität München
Postfach 20 24 20, D-8000 München 2, Germany

Abstract

A formal model and a logical framework for the functional specification of time sensitive communicating systems and their interacting components is outlined. The specification method is modular with respect to sequential composition, parallel composition, and communication feedback. Nondeterminism is included by underspecification. The application of the specification method to timed communicating functions is demonstrated. Abstractions from time are studied. In particular a rational is given for the chosen concepts of the functional specification technique. The relationship between system models based on nondeterminism and system models based on explicit time notions is investigated. Forms of reasoning are considered. The alternating bit protocol is used as a running example.

[1]This work was supported by the ESPRIT BASIC RESEARCH ACTION "DEMON" and by the Sonderforschungs-bereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Architekturen"

1. Introduction

The behaviour of distributed systems that communicate by exchanging messages can be represented by functional models. By such a modelling, systems that operate embedded into some environment are described by functions that map histories of input communications onto histories of output communications. Communications of a system can be internal, i.e. between the components of a system and therefore not necessarily important nor even observable from the environment, or external, i.e. lead to an exchange of messages between the system and its environment. In the latter case we speak of input and output of the system. The causal or logical relationship between input messages and output messages is called the (extensional) behaviour of a system. A simple way for representing the behaviour of interacting systems is a *trace*. A trace is a sequence of communication actions providing the input/output history of a particular instantiation of a system. Since traces do not emphasize compositional forms and the conceptual difference between input and output for interacting system components, we use functional models for representing the behaviour of systems instead of traces.

In a functional specification of a component that is part of a distributed system, interaction and communication are essential. The behaviour of system components can be understood by mappings that relate the input received from the environment to output sent to the environment (cf. [Kahn, MacQueen 77], [Broy 85]). Formally this can be modelled by prefix monotonic functions mapping histories ("streams") of input communications onto histories of output communications.

Prefix monotonicity reflects a basic property of communicating systems: assume we have observed a finite sequence of output messages for a corresponding finite sequence of input messages. Then if we observe additional input (thus the old input sequence is a prefix of the extended one) we may just observe additional output (thus the old output sequence is a prefix of the extended one). Prefix monotonicity provides a notion of causality between input and output. It reflects the stepwise consumption of input and production of output and guarantees the existence of least fixed points, which is mandatory for giving meaning to communication feedback loops. Using streams and stream processing functions for modelling interacting computations has been proposed many times before in the literature (see also data flow approaches, such as [Kahn, MacQueen 77]).

A specification of a system component can be given by a predicate that characterizes the component's behaviour. Technically such a behaviour is described by a class of stream processing functions. For every input x and every choice of a function f from this class, a system behaviour with output f(x) is obtained. It seems methodologically fruitful to specify systems by giving a set of properties. Following this concept the functional behaviour of a system is described by a number of properties represented by logical formulas. The properties can be classified into *safety* properties and *liveness* properties. Safety properties describe system characteristics that can be falsified by finite observations. Accordingly liveness properties correspond to characteristics that can be falsified only by infinite observations.

Agents the behaviours of which depend on the timing of the input messages can be modelled also along the lines described above by introducing particular time messages. An interesting question that we want to discuss in the following concerns the relationship between models of communicating systems including explicit notions of time and abstractions from time by the concept of nondeterminism.

In our functional setting a system component is called *nondeterministic,* if there are several output streams possible for a given input stream. Nondeterminism generally causes a number of complications for the semantic treatment of communicating systems. Nevertheless, since nondeterminism is an important methodological concept, it has to be incorporated into a specification formalism. The reasons are in particular as follows:

- the abstraction from certain only operationally relevant details (such as time, storage etc.) keeps the specification more understandable and more manageable,
- leaving open certain properties in specifications that are not relevant or should be determined only later in the design process avoids overspecification and brings additional flexibility.

Accordingly, nondeterministic specifications of systems with time dependent behaviour are of interest.

Nondeterminism of systems is easily incorporated into functional specifications by underspecification: there may be several functions that meet the predicate specifying an agent and any of them can be chosen for generating system behaviours. Strictly speaking it does not make any difference whether a choice is done throughout the design of a system component (removing underspecification) or left open in the implementation as nondeterminism in the execution.

However, for particular agents at particular levels of abstraction the input/output behaviour cannot be modelled simply by a set of prefix monotonic functions. Examples are system components that produce output as long as there is input on at least one of its input lines. For those agents particular, in principle possible, behaviours should only be chosen for input histories with certain properties. In particular, when abstracting from explicit time, there may exist conflicts between time oriented choice conditions and the requirement of monotonicity. These conflicts can be solved by either introducing an explicit notion of time into the proposed model or by considering so-called *input choice specifications*. Input choice specifications allow the expression of liveness properties of agents that depend on certain time properties of input histories. For such agents the nondeterministic choice of a behaviour function may depend on the timing of the specific input. Input choice specifications allow one to handle even such time dependent agents by nondeterminism without including necessarily explicit time notions into the semantic model.

The presented approach is powerful enough to deal with the specification of liveness properties that are considered especially difficult in specifications such as fairness or responsiveness in the absence of input. The particular formal model for specification has been chosen carefully for avoiding anomalies (such as the one described in [Brock, Ackermann 81]).

Timing aspects are of high importance in computing science. In many safety critical applications of process control systems the timing of the system actions is decisive. Therefore, if formal methods are to be applied for increasing the reliability of such safety critical applications by formal specification and verification techniques the formal modelling of timing aspects and a logical formalism for reasoning about them is necessary.

Most early work on the formal treatment of interactive system mostly did not consider timing aspects at all. Instead of explicit notions of time some nondeterministic scheduling was assumed. Fairness was used to incorporate some properties of scheduling that are not appropriately represented when using just straightforward nondeterminism.

The papers of treatment of time can roughly be divided into three categories. In the first largest category we may see all the work giving logical calculi for the verification of time properties using assertion logic or temporal logic (cf. [Bernstein, Harter 81], [Zwarico, Lee 85], [Jahanian, Mok 86], [Bernstein 87], [Shankar, Lam 87], [Davies, Schneider 89], [Hooman, Widom 89], [Joseph, Goswami 89], [Ostroff 89], [Koymans 90], [Schneider et al. 91], [Hooman 91]).

In the second category we see all the work on given formal models for real time systems as well as semantics to real time languages (cf. [Broy 83], [Koymans et al. 83], [Reed, Roscoe 86], [Huizing et al. 87], [Reed 89], [Yi 90], [Berry, Beneviste 91]). Reed and Roscoe relate timed and untimed models.

In a third category we see more recent papers on algebraic properties of real time languages (cf. [Baeten, Bergstra 90], [Nicollin et al. 90]).

In the following we concentrate rather on two aspects that are not so directly present in the literature mentioned above: the specification of timing properties and the relation between timed and untimed models for interacting systems.

The remaining part of the paper is organized as follows: in section 2 the basic mathematical structures for functional system models are introduced such as streams, the basic operations on streams, and the notion of stream processing functions.

In section 3 functional system specifications are introduced and related to system models using sets of action traces.

In section 4 the alternating bit protocol is treated as an example for a modular system specification. A particular problem in functional system specification is identified that is caused by the fact that the relative timing of messages on different channels is not used for determining the output.

In section 5 a functional model of timed components is introduced.

In section 6 this model is related to the nontimed functional model and abstractions for a certain class of time insensitive components are defined.

In section 7 input choice specifications are introduced that allow to abstract from certain components in the timed model that are not time insensitive in the strict sense.

In section 8 the concept of input choice specifications is related to safety and liveness properties.

In section 9 it is shown how input choice specification can be seen as time free abstraction of certain timed components.

In section 10 properties are given that allow to guarantee fixpoints for modelling communication feedback for input choice specifications.

In section 11 it is shown how the correctness of the alternating bit protocol is proved formally based on the functional specification.

In section 12 he full abstractness of the functional specification technique is defined and an equivalence relation of full abstractness is introduced. Finally it is shown how timed and nontimed models of components can be used side by side.

2. Basic Structures

For a given set M of messages a stream over M is a finite or infinite sequence of elements from M. By M^* we denote the finite sequences over the set M. M^* includes the empty sequence which is denoted by $\langle\rangle$.

By M^∞ we denote the infinite sequences over the set M. M^∞ can be understood to be represented by the set of total mappings from the natural numbers $\mathbb{N}$ into M. We denote the set of streams over the set M by M^ω. Formally we have

$$M^\omega =_{def} M^* \cup M^\infty.$$

For simplicity we write also a^n for the finite stream consisting of n copies of the element a.

Streams can be understood to represent the history of communications (on channels) between components of interactive systems. We introduce a number of functions on streams that are useful in system descriptions. For every stream s we may define its *length*. The length of a stream is infinite or a natural number. It will be denoted by #s. Formally we have

$$\#: M^\omega \to \mathbb{N} \cup \{\infty\}$$

A classical operation on streams is the *concatenation* which we denote by ˆ. The concatenation is a function that takes two streams (say s and t) and produces a stream as result starting with s and continuing with t . If s is infinite then the result of concatenating s with t yields s again. Formally we have for the concatenation the following functionality:

$$.\hat{}. : M^\omega \times M^\omega \to M^\omega$$

On the set M^ω of streams we define a *prefix ordering* $\sqsubseteq$. We write $s \sqsubseteq t$ for streams s and t if s is a *prefix* of t. Formally we have for streams s and t

$$s \sqsubseteq t \quad \text{iff} \quad \exists\, r: s\hat{}r = t.$$

The prefix ordering defines a partial ordering on the set M^ω of streams. If $s \sqsubseteq t$, then we also say that s is an *approximation* of t. The set of streams ordered by $\sqsubseteq$ is even complete in the sense that every directed set $S \subseteq M^\omega$ of streams has a *least upper bound* denoted by lub S. A nonempty set S which is subset of a partially ordered set is called *directed*, if

$$\forall\, x, y \in S: \exists\, z \in S: x \sqsubseteq z \wedge y \sqsubseteq z\ .$$

With the technique of least upper bounds of directed sets of finite streams we are able to describe infinite streams. Infinite streams are also of interest as (and can also be described by) fixed points of prefix monotonic functions. Note, the streams associated with feedback loops in interactive systems can be seen as such fixed points.

A *stream processing function* is a function

$$f: M^{\omega} \to N^{\omega}$$

that is *prefix monotonic* and *continuous*. The function f is called *monotonic*, if for all streams s and t we have

$$s \sqsubseteq t \Rightarrow f.s \sqsubseteq f.t \,.$$

For better readability we often write f.x for the function application instead of f(x). The function f is called *continuous*, if for all directed sets $S \subseteq M^{\omega}$ of streams we have

$$\text{lub}\ \{f.s: s \in S\} = f(\text{lub}\ S) \,.$$

If a function is continuous, then its results for infinite input can be predicted from its results on all finite approximations of this input.

By $\perp$ (called "bottom") we denote the pseudo element which represents the result of diverging computations. One has to deal with diverging computations at the level of the processes computing the individual messages (such as nontermination of recursive calls or nonterminating while loops without send or receive actions). This is the reason for introducing $\perp$. We write $M^{\perp}$ for $M \cup \{\perp\}$. Of course, we assume that $\perp$ is not an element of M. On $M^{\perp}$ we define a simple partial ordering by:

$$x \sqsubseteq y \quad \text{iff} \quad x = y \vee x = \perp$$

We use the following functions on streams

$ft: M^{\omega} \to M^{\perp}$,	*first*
$rt: M^{\omega} \to M^{\omega}$,	*rest*
$.\&.: M^{\perp} \times M^{\omega} \to M^{\omega}$,	*cons*

The function & (called "prefixing") uses as arguments a message m and a stream s and produces as result a stream that has as its first element x and as its rest the stream s, if the element x is defined (i.e. $x \neq \perp$), otherwise the stream collapses to the empty stream. Note, the empty stream $\langle\rangle$ as being the least element in the domain of streams corresponds to the indication "output undefined" as long as is further input unspecified and not to the message "end of transmission".

The properties of the functions can be expressed by the following axiomatic equations:

$\perp \& s = \langle\rangle$,	$rt.\langle\rangle = \langle\rangle$,
$ft(x \& s) = x$,	$x \neq \perp \Rightarrow rt(x \& s) = s$,
$\langle\rangle\hat{\ }s = s = s\hat{\ }\langle\rangle$,	$x \neq \perp \Rightarrow (x \& s)\hat{\ }r = x \& (s\hat{\ }r)$,
$\#\langle\rangle = 0$,	$x \neq \perp \Rightarrow \#(x \& s) = 1 + \#s$.

Sometimes it is useful to work with a filter function on streams. Given a set $S \subseteq M$ and a stream $x \in M^{\omega}$ we write S©x for the substream of x consisting only of the elements of x contained in S. Formally we define

$S©\langle\rangle = \langle\rangle$,	
$S©(d \& x) = d \& (S©x)$	if $d \in S$,
$S©(d \& x) = S©x$	if $\neg(d \in S)$.

We denote the function space of (n,m)-ary prefix continuous stream processing functions by:

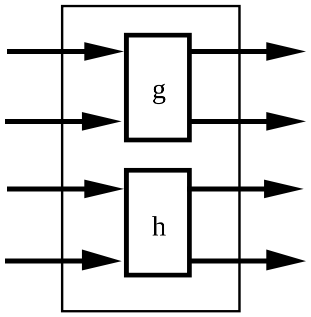
$[(M^{\omega})^n \to (M^{\omega})^m]$

The prefix ordering $\sqsubseteq$ induces an ordering on tuples of streams and on this function space as well.

Note, the operations ft, rt, and & are prefix monotonic and continuous, but the concatenation ˆ as defined above is not prefix monotonic. Prefix monotonicity reflects a characteristic property of interactive systems: communicated data cannot be changed after being shown as output. In addition, monotonicity gives the formal platform for handling communication in feedback loops: feedback is translated to fixed point equations, which are known to have solutions, if the involved functions are monotonic.

For composing stream processing functions we may use the three classical forms of composition, namely sequential and parallel composition and feedback. Let g be a (n,m)-ary function and h be a (n',m')-ary stream processing function.

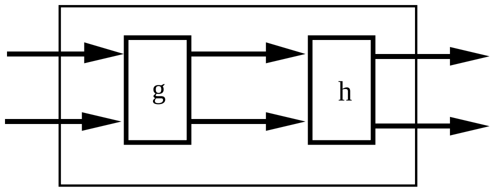
We denote *parallel composition* of g and h by g||h. g||h is a (n+n',m+m')-ary stream processing function. Parallel composition can be visualized by a diagram:



We define for $x \in (M^{\omega})^n$, $z \in (M^{\omega})^{n'}$

$(g||h).(x,z) = (g.x, h.z)$.

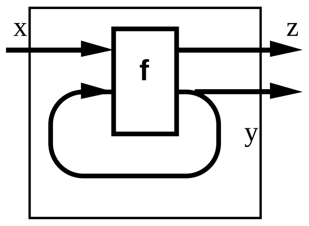
Let m = n'; we denote *sequential composition* by (g ; h). (g ; h) is a (n,m')-ary stream processing function. Sequential composition can be visualized by a diagram:



We define for $x \in (M^{\omega})^n$:

$(g;h).x = h.g.x$.

We define the feedback operator μ^k for a (n+k,m)-ary stream processing function f where $m \geq k$. $\mu^k f$ is a (n,m)-ary stream processing function derived from f by k feedback loops. This can be visualized by a diagram:



We define for $x \in (M^{\omega})^n$

$(\mu^k f).x = \textbf{fix}\ \lambda\ z, y: f(x, y)$

where $z \in (M^{\omega})^{m-k}$, $y \in (M^{\omega})^k$. By **fix** we denote the least fixed point operator. We write μf for $\mu^1 f$.

In a more readable version we may specify the μ-operator as follows: we have

$(z, y) = (\mu^k f).x,$

if (z, y) is the least fixed point of the equation

$(z, y) = f(x, y).$

Formally $(\mu^k f).x$ denotes a fixed point relative to x, i.e. for every given tuple x of streams we obtain (z, y) by a fixed point construction.

Note, the fixed point concept reflects the characteristics of feedback of communications in an appropriate manner. The output of the agent f is sent back to its own input line. If further output requires more input than available this way no more output is generated. This is properly modelled by the assumption that the fixed point is the least one.

3. Functional System Specification

A simple example of a specification of a system component is an interactive stack. It can be modelled by a function

$f: (D \cup \{?\})^{\omega} \rightarrow D^{\omega}.$

Here data messages in the input stream are understood as to be pushed onto the stack and messages "?" are understood as messages indicating requests for data. We specify the behaviour of the interactive stack by the formula (for $x \in D^*$, $d \in D$, $y \in (D \cup \{?\})^{\omega}$):

$f.x = \langle\rangle,$

$f(x \hat{\ } \langle d \rangle \hat{\ } \langle ? \rangle \hat{\ } y) = d \ \& \ f(x \hat{\ } y)$.

The first equation indicates that the interactive stack does not produce any output as long as no request signals are received as input. The second equation indicates that the interactive stack, after having received a finite sequence x of data messages followed by the data message d and a request signal, produces the data message d, which is the last data message received before the request signal, as output and then produces the same output as an interactive stack on the input without the data message d and the request signal.

Similarly, an interactive queue can be specified by replacing the second equation by:

$f(\langle d \rangle \hat{\ } x \hat{\ } \langle ? \rangle \hat{\ } y) = d \ \& \ f(x \hat{\ } y)$.

Note, in both cases the function f is not uniquely determined by the given equation, since nothing is said about its output for an input of the form ? & y.

A system component with n input lines and m output lines accepts n streams as input and produces m streams as output. It is called *(n,m)-ary agent*. An agent specification describes the behaviour of an agent.

For a deterministic agent its behaviour is represented by a stream processing function. For a nondeterministic agent its behaviour is specified by a *class* of stream processing functions. Several instances of an agent may occur in a system. In every instance (every execution) one function is chosen from that class for determining one particular output. An agent specification thus can be represented by a predicate

$$Q: [(M^{\omega})^n \to (M^{\omega})^m] \to \mathbb{B}$$

where $\mathbb{B} = \{\mathbb{1}, \mathbb{0}\}$ represents the set of truth values.

A simple example for a highly nondeterministic agent is a fork that receives a stream as input and produces two streams as output by splitting the input stream:

$$\text{FORK}.f = \forall\ x, y, z, d: f.x = (y, z) \Rightarrow \#(\{d\}©x) = \#(\{d\}©y)+\#(\{d\}©z).$$

Another basic example for a nondeterministic agent is the merge agent that nondeterministically merges two streams. It can be specified by the predicate MERGE:

$$\text{MERGE}.f \equiv \forall\ x, y: \exists\ \text{oracle} \in \{\mathbb{1}, \mathbb{0}\}^{\omega}: f(x, y) = \text{sched}(x, y, \text{oracle}) \wedge \#\text{oracle} = \infty$$

$$\begin{aligned} \textbf{where}\ \forall\ a, b, c: \quad & \text{sched}(a, b, \mathbb{0}\ \&\ c) = \text{ft}.a\ \&\ \text{sched}(\text{rt}.a, b, c) \wedge \\ & \text{sched}(a, b, \mathbb{1}\ \&\ c) = \text{ft}.b\ \&\ \text{sched}(a, \text{rt}.b, c)\ . \end{aligned}$$

A function f with MERGE.f need not to be fair and it may be strict, i.e. we may have (even if $x = \langle\rangle$ and $y \neq \langle\rangle$) :

$$f(x, y) = x\ .$$

Strictness is an issue here, because for may practical applications we prefer a merge that does guarantee correct transmission on one of its input lines even in cases where the component providing input for the other line runs into a diverging computation (i.e. the merge should be nonstrict).

A version of a specification of a (on infinite input) fair merge agent reads as follows:

$$\begin{aligned} \text{FAIR\_MERGE}.f \equiv \forall\ x, y: \exists\ \text{oracle} \in \{\mathbb{1}, \mathbb{0}\}^{\omega}: f(x, y) = \text{sched}(x, y, \text{oracle}) \wedge \\ \#(\{\mathbb{1}\}©\text{oracle}) = \infty \wedge \#(\{\mathbb{0}\}©\text{oracle}) = \infty\ . \end{aligned}$$

Again we may have

$$f(x, y) = x$$

for finite streams x, but not for infinite ones. A specification of a nonstrict and fair version of merge is given later.

Stream processing functions

$$f: (M^{\omega})^n \to (M^{\omega})^m$$

can be seen as a special case of an agent specification, i.e. as a deterministic agent specification. For notational convenience we write

$$f$$

also for a deterministic agent specification Q with Q.h = (f = h).

The combining forms for stream processing functions carry over to system specifications by pointwise application. Let Q and Q' be specifications for (n, m)-ary and (n',m')-ary agents resp. We define:

$(Q \| Q').f \equiv_{df} \exists\ g, h: Q.g \wedge Q'.h \wedge f = g \| h,$

$(Q\ ;\ Q').f \equiv_{df} \exists\ g, h: Q.g \wedge Q'.h \wedge f = g\ ;\ h,$

$(\mu^k Q).f \equiv_{df} \exists\ g: Q.g \wedge f = \mu^k g.$

A decomposition of specifications into safety and liveness properties as suggested in [Lamport 83] and [Schneider 87] may help in the understanding of behaviours of agents. It is defined as follows:

- *safety properties* correspond to characteristics of a system that can be falsified by finite observations. A finite observation for an input x consists of some finite output which is an approximation for some possibly infinite totally correct output produced for input x. This observation does include the causality between input and observed output. For every prefix $x' \sqsubseteq x$ we may observe which is the maximal prefix $y' \sqsubseteq y$ that can be caused by input x' (when assuming that the output is a prefix of y).
- *liveness properties* correspond to characteristics of a system that can be falsified only by infinite observations.

Since a specification of an agent should provide enough information about its safety properties to determine the safety properties of composed systems in which the agent is used as component we do not specify a simple input/output relation, but a set of prefix continuous (and thus monotonic) functions. This way the causality between input and output in the sense of safety properties is properly indicated. This will be demonstrated by an example below. Each of the functions specifies information about system behaviours in the sense of possible courses of computation.

The notion of observation has proved to be central in the understanding of interacting systems. What can be observed about a system is what we consider to be the relevant properties of the system. Therefore the notion of observability determines the notion of semantic identity. The observation of the causality between input and output as being part of the safety properties is a subtle point and therefore requires further explanation.

We use a very basic notion of observability here. Consider a system component with one input line and one output line. Assume, furthermore, we may select a message and send it via the input line to the component. After having sent a message we observe a sequence of messages (which may be empty, of course). Then we may continue sending messages and observing output. If we stop our observation after a finite amount of time, the observation is called finite.

An observation can be represented by a trace, i.e. a stream of messages labelled as input or output. Mathematically then an observation is a partially ordered set of events labelled by input and output actions. For simplicity we represent finite observations by finite traces of messages M labelled by "in" for input and "out" for output. Then a (finite) trace is a (finite) sequence t

$$t \in (I \cup O)^{\omega}$$

where I and O denote the set of input and output actions resp.

$$I = \{in.m: m \in M\},$$

$$O = \{out.m: m \in M\}\ .$$

By

$$strip: (I \cup O)^{\omega} \rightarrow M^{\omega}$$

we denote the function that derives the underlying pure message stream from a given trace. It is specified by the following equations:

$\text{strip}.\langle\rangle = \langle\rangle$,

$\text{strip}(\text{in}.m \ \&\ t) = \text{strip}(\text{out}.m \ \&\ t) = m \ \&\ \text{strip}.t$.

Note, that for simplicity we model observations by traces, i.e., just use sequential observations. It is also possible to use partial ordered event structures for representing observations about the input/output causality of stream processing functions. However, for our purpose traces seem to be sufficient.

For formally defining the concept of observations by traces we introduce a predicate obs. A trace t is called a *partial observation* for a continuous stream processing function

$f: M^{\omega} \to M^{\omega}$,

if we have

$\text{obs}(t, f)$

where the predicate

$\text{obs}: (I \cup O)^{\omega} \times (M^{\omega} \to M^{\omega}) \to \mathbb{B}$

specifies that the sequence of actions t corresponds to possible observations of a sequence of input actions for the function f. Formally the predicate obs is specified by the most liberal (weakest) predicate that fulfils the following equations:

$\text{obs}(\langle\rangle, f) \equiv \text{true},$

$\text{obs}(\text{in}.m \ \&\ t, f) \equiv \text{obs}(t, g)$ where $\forall x: g.x = f(m \ \&\ x),$

$\text{obs}(\text{out}.m \ \&\ t, f) \equiv (\text{ft}.f.\langle\rangle = m \wedge \text{obs}(t, \text{rt} \ ;\ f))$

The concept of trace observations is closely related to the notion of approximation. This can be seen by the following lemma:

Lemma: $f \sqsubseteq h \Rightarrow (\text{obs}(t, f) \Rightarrow \text{obs}(t, h))$

Proof: For finite traces t we prove the proposition by induction on #t.

For #t = 0 the proposition is obvious. Now let us assume that the proposition holds for all traces of length ≤ n. For a trace t' of length n+1 we consider two cases:

(1) $t' = \text{in}.m \ \&\ t$; from obs(t', f) we obtain by definition of obs also obs(t, g) where

$\forall x: g.x = f(m \ \&\ x).$

For g' with

$\forall x: g'.x = h(m \ \&\ x)$

we have $g \sqsubseteq g'$ if $f \sqsubseteq h$; by induction hypothesis we have

$\text{obs}(t, g) \Rightarrow \text{obs}(t, g')$ and thus

$\text{obs}(t', f) \Rightarrow \text{obs}(t', h).$

(2) $t' = \text{out}.m \ \&\ t$; from obs(t', f) we obtain $\text{ft}.f\langle\rangle = m \wedge \text{obs}(t, (\text{rt} \ ;\ f))$. By $f \sqsubseteq h$ we obtain $\text{ft}.f\langle\rangle = m$ and $(\text{rt} \ ;\ f) \sqsubseteq (\text{rt};\ h)$; by induction hypothesis we obtain

$\text{obs}(t, (\text{rt};\ f)) \Rightarrow \text{obs}(t, (\text{rt};\ h))$

and thus obs(t', h). ◊

A trace t is called *total observation* for f, if obs(t, f) and f.strip(I©t) = strip(O©t).

Lemma: Alternative characterisation of traces

(a) t is partial observation for f, iff

$$\forall t': t' \sqsubseteq t \Rightarrow \text{strip}(O©t') \sqsubseteq f.\text{strip}(I©t') .$$

(b) t is total observation for f, iff in addition

$$f.\text{strip}(I©t) = \text{strip}(O©t) .$$

Proof: (a) By induction on the length of t. If t = ‹›, then the proposition (a) trivially holds. Assume t is finite and nonempty and the formula holds for finite streams of length #rt.t. By definition we have:

$$\text{obs}(t, f) = \text{obs}(rt.t, g)$$

where g is defined as follows

$$ft.t \in I \Rightarrow \forall x: g.x = f(ft.strip.t \,\&\, x),$$

$$ft.t \in O \Rightarrow \forall x: ft.strip.t \,\&\, g.x = f.x .$$

For rt.t and g we apply the induction hypothesis and obtain (a) for t and f.

This completes the proof for finite traces by induction on the number of elements in t. To infinite traces the proof is extended by the continuity of f and g. Note, we have for a chain of finite traces t.i :

$$(\forall i: \text{obs}(t.i, f)) \Rightarrow \text{obs}(\text{lub}\ \{t.i: i \in \mathbb{N}\}, f) .$$

(b) is just the definition. ◊

Every predicate Q on functions defines a property of functions and also specifies a set of finite and infinite observations represented by finite and infinite traces. We say that a trace t *is an observation about the property* Q, if

$$\exists f: Q.f \wedge \text{obs}(t, f) .$$

We write then also obs(t, Q). We say that property Q is falsified for a specification R, if there is a trace t such that

$$\text{obs}(t, R) \wedge \neg\, \text{obs}(t, Q)$$

i.e. if we can make an observation about R that falsifies property Q.

For an agent specification Q and some input $x \in (M^{\omega})^{n}$ an output $y \in (M^{\omega})^{m}$ is called *correct*, if there exists a stream processing function f such that

$$Q.f \wedge f.x = y .$$

Partial correctness corresponds to safety properties and finite observations. For an input x partial correctness corresponds to observations restricted to finite prefixes of the output streams, but it includes observations of maximal output for approximations (prefixes) of x. For modelling this we introduce the notion of output finite function. A function f is called *output finite*, if its output always is a tuple of finite streams, i.e. if

$$f: (M^{\omega})^{n} \rightarrow (M^{*})^{m}$$

An output finite prefix monotonic function f is called *partially correct* or *safe* for the specification Q, if we have:

$\exists$ g: Q.g $\wedge$ f $\sqsubseteq$ g .

Generally a function is called *safe* (or partially correct) with respect to specification Q if all its output finite approximations are safe with respect to Q. According to this definition, if for a function f which is partially correct for Q we have f.x = y where y is infinite, this does not imply that there exists a function g with Q.g and g.x = y. It only indicates that for all finite prefixes y' of y there exists a function g with Q.g and y' $\sqsubseteq$ g.x.

Lemma: A function f is safe with respect to Q iff all finite observations about f are observations about Q.

Proof: Assume f is safe with respect to Q; then for every output finite approximation f' $\sqsubseteq$ f there exists g with

Q.g $\wedge$ f' $\sqsubseteq$ g.

Moreover, for every finite trace t with obs(t, f) there exists an output finite function f' such that

obs(t, f) $\Rightarrow$ obs(t, f').

By this we obtain obs(t, Q). Now assume

obs(t, f) $\Rightarrow$ obs(t, Q)

for all finite traces. An output finite function f' is an approximation of f iff for all finite traces t

obs(t, f') $\Rightarrow$ obs(t, f).

This shows that all output finite functions that are approximations of f are safe and thus f is safe by definition. ◊

By this definition of safety we can be sure that the following property of compositionality holds: if f is partially correct with respect to Q, then μ^kf is partially correct w.r.t μ^kQ. The same holds for parallel and sequential composition. Also with respect to liveness our approach is compositional.

This scenario of input and output shows a strong relationship between I/O-automata as used by many researchers for modelling interactive components and functional system specifications (cf. [Lynch, Stark 89]). Stream processing functions can be seen as functional minimal (with respect to the size of the state space) representations of deterministic input/output–automata. Specifications of components including underspecification can be seen as nondeterministic I/O-automata.

As a result of the monotonicity of the considered function we have

obs(tˆ‹out.m1 in.m2›, T) $\Rightarrow$ obs(tˆ‹in.m2 out.m1›, T).

This indicates that once an output is enabled, this enabledness cannot be changed by further input. This shows a very schematic property of functional system specification which will be discussed in detail in the following section.

There are two reasons why we work with sets of functions rather than set-valued functions or relations when modelling nondeterminism in communicating systems:

- we can use the classical functional calculus for proving properties of specifications,

- for determining the meaning of feedback loops least fixed points have to be considered, which cannot be characterized uniquely, in general, in the framework of set-valued functions or relations.

We illustrate the second remark by an example.

Example: *Agent specifications with identical input/output relations*

We consider two (1,1)-ary agents. Let the continuous stream processing functions

$g, h, i, j: [\{\mathbb{1}\}^{\omega} \to \{\mathbb{1}\}^{\omega}]$

be specified (for all $x, y \in \{\mathbb{1}\}^{\omega}$) by the following equations

$g.\langle\rangle = h.\langle\rangle = \langle\mathbb{1}\rangle$, $\quad i.\langle\rangle = j.\langle\rangle = \langle\rangle$,

$g.\langle\mathbb{1}\rangle = j.\langle\mathbb{1}\rangle = \langle\mathbb{1}\ \mathbb{1}\rangle$, $\quad h.\langle\mathbb{1}\rangle = i.\langle\mathbb{1}\rangle = \langle\mathbb{1}\rangle$,

$y = \langle\mathbb{1}\ \mathbb{1}\rangle\hat{\ }x \Rightarrow g.y = h.y = i.y = j.y = \langle\mathbb{1}\ \mathbb{1}\rangle$.

We define the specifying predicates Q1 and Q2 by

$Q1.f \equiv (f = g \vee f = i)$,

$Q2.f \equiv (f = h \vee f = j)$.

Obviously for every input the sets of totally correct output of Q1 and Q2 coincide. The behaviours μQ1 and μQ2 for the agent specifications Q1 and Q2 under feedback are given by the least fixed points z of functions f with Q1.f and Q2.f resp. The least fixed point of the functions i and j is $\langle\rangle$. The least fixed points of g is $\langle\mathbb{1}\ \mathbb{1}\rangle$. The least fixed points of h is $\langle\mathbb{1}\rangle$. So μQ1 is distinct from μQ2. Note, for Q1 the trace

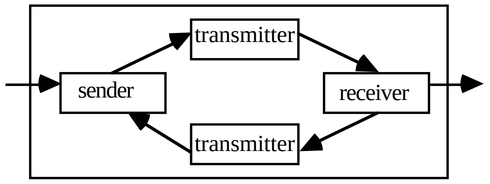
$\langle out.\mathbb{1}\ \ in.\mathbb{1}\ \ out.\mathbb{1}\ \ in.\mathbb{1}\rangle$

is a possible observation, but not for Q2. ◊

For explaining the adequacy of the given approach let us for a moment refer to operational considerations: for an agent specification Q given some input x every finite prefix y of f.x for some function f with Q.f represents some approximation for the totally correct output f.x . The stream y can be seen as the output produced for the input x after a certain finite amount of computation time. It corresponds to a "finite observation". If we "wait long enough" then the output should be increased eventually as long as "more" output is guaranteed by all functions f with Q.f and $y \sqsubseteq f.x$ regardless what further inputs are added.

4. The Alternating Bit Protocol

For illustrating the proposed functional specification techniques we consider the example of the alternating bit protocol. Let D stand for a set of data elements. The alternating bit protocol can be graphically represented by the following diagram:



From this diagram the number of input lines and output lines for the involved agents are obvious. The diagram represents the following composed specification

$$AB = \mu(CS ; CT ; CR ; (ID \| CT)) ; CP$$

where CP specifies the projection function

$$CP.f \equiv \forall\ x, y: f(x, y) = x,$$

and ID specifies the identity function

$$ID.f \equiv \forall\ x: f.x = x.$$

The function that is to be realized by the alternating bit protocol is simply the identity.

We use for the specification of the sender and the receiver the following auxiliary continuous function:

$$alt: \{\mathbb{1}, \mathbb{0}\}^{\omega} \to \{\mathbb{1}, \mathbb{0}\}^{\omega}.$$

The function alt records the alternations from $\mathbb{1}$ to $\mathbb{0}$ in its input stream :

$$alt.\langle b\rangle = \langle b\rangle,$$

$$alt(b \,\&\, b \,\&\, x) = alt(b \,\&\, x),$$

$$alt(b \,\&\, \neg b \,\&\, x) = b \,\&\, alt(\neg b \,\&\, x)\ .$$

The continuity of alt implies $alt(\mathbb{1}^{\infty}) = \langle\mathbb{1}\rangle$. The continuous function

$$res: (D \times \{\mathbb{1}, \mathbb{0}\})^{\omega} \to D^{\omega}$$

produces as output those data that are labelled by an alternating bit:

$$res.\langle\rangle = \langle\rangle,$$

$$res.\langle(d, a)\rangle = \langle d\rangle,$$

$$res((d, a) \,\&\, (d, a) \,\&\, x) = res((d, a) \,\&\, x),$$

$$res((d, a) \,\&\, (d', a') \,\&\, x) = d \,\&\, res((d', a') \,\&\, x) \qquad \text{if } (d, a) \neq (d, a').$$

The continuity of res implies $res((d,\mathbb{1})^{\infty}) = \langle d\rangle$. The continuous function

$$ack: (D \times \{\mathbb{1}, \mathbb{0}\})^{\omega} \to \{\mathbb{1}, \mathbb{0}\}^{\omega}$$

drops the data and produces a stream of bits:

$$ack.\langle\rangle = \langle\rangle,$$

$$ack((d, a) \,\&\, x) = a \,\&\, ack.x\ .$$

We may specify the agent receiver in the alternating bit protocol by the following predicate CR:

$$CR.f \equiv \forall\ x: f.x = (res.x, ack.x)\ .$$

Note, the receiver in the specification above yields pairs of result streams (r, y) where r is the output of the system and y is the stream of alternating bits that is produced as feedback for the sender.

For a specification technique it is essential that we can specify also the behaviour of highly (unboundedly) nondeterministic components for representing the behaviour of physical devices. As an illustration for such components we specify the transmitter. We give a specification by the predicate CT:

$$\mathrm{CT.f} \equiv \forall x: \exists s \in \mathbb{B}^{\omega}: f.x = h(x, s) \wedge \#(\mathbb{1} © s) = \infty$$

$$\textbf{where } \forall x, s: h(x, \mathbb{1} \& s) = ft.x \,\&\, h(rt.x, s) \wedge h(x, \mathbb{0} \& s) = h(rt.x, s).$$

A further example for a specification is the sender. The sender should send its current message again and again as long as there is no (correct) feedback from the receiver. However, there is a clear conflict between the prefix monotonicity and the intended behaviour: if there is never feedback the message should be repeated infinitely often, if there is correct feedback the message should be sent no longer (i.e. at most finitely often after all). We may try to give a specification CS for the sender as follows:

$$\mathrm{CS.f} \equiv \forall x, y: res.f(x,y) \sqsubseteq x \wedge \#res.f(x,y) = \min(\#x, 1+\#alt.y) \wedge ft.ack.f(x,y) \sqsubseteq \mathbb{1} \wedge (\#x > \#alt.y \Rightarrow \#f(x,y) = \infty).$$

This specification asserts in particular by the last clause that if not enough acknowledgements are sent back to the sender then the first nonacknowledged message is repeated infinitely often.

From CS.f we may conclude in particular:

$$f(d\&d'\&x, \langle\rangle) = (d, \mathbb{1})^{\infty},$$

$$\#(\{(d',\mathbb{0})\} © f(d\&d'\&x, \mathbb{1}\&y)) \geq 1.$$

These formulas show that there does not exist a prefix monotonic function that fulfils the predicate CS. This problem has to do with timing. Informally speaking, the sender repeats its message *as long as* it does not get the expected acknowledgement. However, without explicit time information we cannot express the notion "as long as" in the considered model.

For fixing this gap we have either to introduce explicit time considerations (see below) or to regard the specification either as contradictory with respect to monotonicity or we have to give up the requirement of monotonicity which makes it difficult to ensure the existence of least fixed points. Least fixed points, however, are badly needed to give meaning to feedback loops such as the one occurring in the alternating bit protocol.

This example seems to show that functional system specifications show less expressiveness than for instance I/O–automata (cf. [Lynch 89]), which allow one to model the alternating bit protocol in a straightforward way. However, the reason for that is that I/O-automata are somewhat less abstract. They describe the behavior of system components in a more operational way. By this the relative *timing* of input on different channels can be used for determining the output. This way an additional "pulse" is introduced into the computational model that makes I/O-automata more similar to the "timed" stream processing functions treated in the following section.

For solving this trade off we switch to real time models and later to so-called input choice specifications as an abstraction of timing as introduced in the following sections.

5. Agents with Time Dependent Behaviour

For many applications of communicating systems the timing of the messages on their input history is essential: the behaviour of a communicating agent may critically depend on information about the relative or absolute timing of received messages. Such a situation can be formally modelled by introducing a notion of time explicitly into a semantic model.

Time is already modelled in the formalism as introduced so far as a total order on events. Now we add a quantitative notion of time in the form of a discrete, rather than a continuous ("dense") time model. We work in the following with a fairly simple but nevertheless sufficiently expressive model of time by using a special element $\surd$ called "tick" for representing the situation that no message has arrived at some input line (or no message has been produced at some output line) within a certain time interval. The fact that no actual message has arrived within such an interval of time can be seen as the specific information represented by $\surd$.

Note, we assume that the arrival of an actual message takes the same interval of time as $\surd$. We use $\surd$ as a message and write $M^{\surd}$ for $M \cup \{\surd\}$. A *timed stream* is an element from the set

$$(M^{\surd})^{\omega} .$$

It represents the communication over a channel with additional time information, i.e. information about the time that it takes between the transmission of two messages on some input or output line. Then a *time annotated* (or *timed* for short) *(n, m)-ary stream processing function* f is a continuous function of the following functionality:

$$f: [((M^{\surd})^{\omega})^{n} \to ((M^{\surd})^{\omega})^{m}]$$

Timed streams and time annotated stream processing functions can be specified by the same techniques as introduced and used for streams without explicit time information above.

Example: *Specification of a time sensitive function*
A time sensitive stream processing function f that produces some input message as output only if the input is repeated within a certain amount of time can be specified by the following equations (let $x \in (M^{\surd})^{\omega}$, $a, b \in M$, $a \neq b$):

$$f.\langle\rangle = \langle\rangle,$$

$$f.\langle a\rangle = \langle\surd\rangle,$$

$$f(\surd \,\&\, x) = \surd \,\&\, f.x,$$

$$f(a \,\&\, a \,\&\, x) = \surd \,\&\, a \,\&\, f.x,$$

$$f(a \,\&\, b \,\&\, x) = \surd \,\&\, f(b \,\&\, x),$$

$$f(a \,\&\, \surd \,\&\, a \,\&\, x) = \surd \,\&\, \surd \,\&\, a \,\&\, f.x,$$

$$f(a \,\&\, \surd \,\&\, b \,\&\, x) = \surd \,\&\, \surd \,\&\, f.(b \,\&\, x),$$

$$f(a \,\&\, \surd \,\&\, \surd \,\&\, x) = \surd \,\&\, \surd \,\&\, \surd \,\&\, f.x.$$

Note, we have $\#f.x = x$ for all input streams x. The function f can be understood as a simplified version of a login function with a password: if the correct password is not given within a certain amount of time the login is aborted. This shows that ticks can be used to model a timeout. ◊

From every timed stream $s \in (M^{\surd})^{\omega}$ we may derive its time information free stream of actual messages by M©s. Note, an infinite timed stream $s \in (M^{\surd})^{\omega}$ provides a complete timing information about the communication history on an input or output line.

Again, we may think about traces of timed agents. Then time ticks occur as elements in the traces. Note, in principle time ticks in the input stream may refer to an interval of different time duration than a time tick in an output stream. Such assumptions, however, may lead to conceptual anomalies when considering feedback. The concept of time ticks can be applied for different modelings of time such as quantitative or relative time models. The different models can be reflected by different additional properties that are assumed for the stream processing functions (see next section).

One of the most interesting aspects in connection with models for interaction including timing aspects are questions of abstracting away time. So in the following a short explanation is given of how streams with explicit timing informations are related to streams without explicit timing information. A timed partial stream $s \in (M^{\surd})^*$ of length n can be understood to provide incomplete (i.e. finite) information about the communication history on a channel just representing the behaviour on the channels in the first n time intervals. In contrast to this a nontimed partial stream $s \in M^*$ of length n represents information more ambiguously. It may correspond to the complete information about the communication history on an input channel where, after all, only a finite number of messages is sent (i.e. it is the abstraction from an infinite timed stream s' by s = M©s') or it may just represent the behaviour on the channels in the first n+k time intervals (i.e. it is the abstraction from a finite timed stream s' of length n+k by s = M©s'). The identification of these two essentially different situations into one when abstracting away time information is the reason for the problems with monotonicity (and continuity) in the sender specification of the alternating bit example above. The sender should behave differently on a finite timed stream (which of course contains only a finite number of messages) of acknowledgement feedbacks without the expected bit (where it should send only a "sufficient" number of messages) and on an infinite timed stream without the expected bit (where it should send an infinite number of messages).

If an infinite timed stream s contains only a finite number of proper messages, i.e. if

$$\#(M©s) < \infty,$$

then after a finite prefix the stream s consists of an infinite stream of time ticks √. However, there is a remarkable difference between s and M©s (and also between s and every finite stream $s' \in (M^{\surd})^{\omega}$ with M©s' = M©s). The stream s is a total element with respect to prefix ordering, while M©s (and similarly s') is a partial element, i.e. an element representing (possibly) incomplete information about the input.

Note, there does not exist a prefix continuous function end_of_transmission with

$$\text{end\_of\_transmission}.s = \text{true} \qquad \text{iff } s = \surd^{\infty}.$$

Clearly, even with explicit time information we can never test algorithmically, i.e., in a program if end_of_transmission is valid, because we cannot predict the future. However, we may define functions that for instance repeat a message as often as there is no new actual message in the next time intervals. Such a repeater (rep.m) starting with message m can be defined on timed streams as follows (for $x \in M$):

$$(\text{rep.m}).(\surd \ \&\ s) = m \ \&\ (\text{rep.m}).s,$$

$$(\text{rep.m}).(x \ \&\ s) = x \ \&\ (\text{rep.x}).s \ .$$

Of course there is no way to define a monotonic function g.m in analogy to the repeater on the level of nontimed streams such that

$$(\text{rep.m}).s = (\text{g.m}).(M©s).$$

This simple observation throws some light onto the difference between modelling systems with or without explicit time notions. In an infinite stream s including time ticks with #(M©s) < ∞ (i.e. s is the representation of the partial stream M©s with full time information) we cannot test (by monotonic predicates) whether the stream M©s is empty, but we can test again and again whether a stream is (so far) not continued by actual messages and according to this react to this situation by producing, for instance, more and more output. We get this way a weak (stepwise) test for the availability of further input on timed streams which does not exist (by a monotonic predicate) for the nontimed partial stream M©s.

The behaviour of a timed agent with n input lines and m output lines can be specified by a predicate over the space of timed stream processing functions. A nontimed agent may be obtained as an abstraction from a timed agent. If the timing information for the input streams does not influence the actual messages in the output streams, but only their timing, then an agent is called *time insensitive*. A more formal definition of time insensitivity is given later.

The use of time notions in specifications may make specifications longer and more detailed, since we have to deal with all kinds of time considerations, but it may also make the specification of certain agents simpler, since using the explicit notion of time may allow us to express certain liveness aspects more directly.

Example: *Timed specifications for the alternating bit example*
Assuming time information in the input streams the sender specification of the alternating bit example above for instance is rather simple. We give such a specification by the predicate TCS for the sender:

TCS.f ≡ (Q.𝟙).f **where** ∀ f, b: (Q.b).f ≡ ∀ x, y, a:

(a) f(a & x, ‹›) = ‹(a,b)› ∧

(b) f(a & x, √ & y) = (a,b) & f(a & x, y) ∧

(c) f(a & x, ¬b & y) = (a,b) & f(a & x, y) ∧

(d) f(a & x, b & y) = (a,b) & g(x, y) **where** (Q.¬b).g

In this specification of the sender a timed function is specified to be a sender function if it is a sender function (specified by (Q.b).f) with initial bit b = 𝟙. (Q.b).f is specified by four lines that, informally speaking, indicate that for the current bit b:

(a) if the acknowledgement stream is empty and the first message to transmit is then the sender outputs just (a, b), i.e., the message a labelled by bit b;

(b)+(c) if the sender receives a time tick or the wrong bit on the acknowledgement channel, then the current message and labelled by the current bit is repeated;

(d) if the current bit is received on the acknowledgement line, then the (after sending the message (a, b)) the actual message is dropped and the remaining input streams are processed with a behaviour g using the negation of the current bit.

In contrast to the time free specification above there do not arise any conflicts w.r.t monotonicity here. Of course, we may give more nondeterministic versions of the sender (repeating a message an arbitrary - but finite - number of times on nonpositive feedback).

For being able to use such a timed specification of the sender for the alternating bit protocol we need of course also specifications dealing with explicit time notions for the transmitter and the receiver. We specify the agent receiver by the timed agent specification TCR:

TCR.f ≡ ∀ x: f.x = h(x, 𝟙)

where ∀ x, b, a: h(‹›, b) = (‹›, ‹›),

$$h(\surd \ \&\ x, b) = (\surd \ \&\ r, \surd \ \&\ y) \ \ \textbf{where}\ (r, y) = h(x, b),$$
$$h((a,b) \ \&\ x, b) = (a\&r, b\&y) \ \ \textbf{where}\ (r, y) = h(x, \neg b),$$
$$h((a,\neg b) \ \&\ x, b) = (r, \neg b\&y) \ \ \textbf{where}\ (r, y) = h(x, b).$$

We give a specification with explicit time notions for the transmitter by the predicate TCT:

$$TCT.f \equiv \forall\ x: \exists\ s \in \mathbb{B}^{\omega}: f.x = h(x, s) \wedge \#(\mathbb{1}©s) = \infty$$
$$\textbf{where}\ \forall\ x, s: h(x, \mathbb{1}\&s) = ft.x \ \&\ h(rt.x, s) \wedge h(x, \mathbb{0}\&s) = \surd \ \&\ h(rt.x, s) \ .$$

Again we may give more nondeterministic versions of the transmitter where a lost message may lead to an arbitrary but finite (and nonempty) number of time ticks. ◊

The specifications of the timed agents given here are rather simple and straightforward. Our model of time is discrete. It is similar to the one proposed in [Ostroff 89b]. Models of nondiscrete, "dense" time, where between two finite time points infinitely many time points can be found, are used, for instance, in [Lynch 89]. The concept of "dense" time, as it is proposed, for instance, in [Alur, Dill 90], [Schneider et al. 91], and [Baeten, Bergstra 90] allows one to study processes where infinitely many actions may occur in finite time intervals such as in Zeno's paradox (cf. [Abadi, Lamport 91]). Such models of time may be of interest when specifying physical phenomena of hardware. We concentrate rather on software aspects and on discrete time.

In the next section we study the abstraction from time for agents with time dependent behaviour.

6. Abstracting from Time

In principle, the message $\surd$ can be used and treated in specifications like any other message. However, looking at timed systems we may assume additional properties that are characteristic for timing. We may assume that for every agent it takes some time until arriving input messages trigger output messages. This can be modelled by time ticks in the output streams.

In the following we take a very simple view towards timing. We assume that every message takes an equal amount of time. A time tick indicates that there is no actual message in a time interval. This is a very simple view. More sophisticated models using time ticks are feasible. If we understand a timed agent as a function or a set of functions mapping input histories including time information onto output histories, then it is suggestive to add the following assumptions to our model:

– infinite input histories are mapped on infinite output histories, i.e. on a complete timing of the output streams,

– finite input histories of length n lead to output histories of length $\geq n+1$.

The second assumption implies the first by the continuity of the functions. According to these assumptions of the first k elements of an output stream an agent is determined by the first k elements of its input streams. A timed stream processing function f is called *pulse driven* (or also *time synchronous*), if we have (for all i, $1 \leq i \leq m$, $x \in ((M^{\surd})^{\omega})^{n}$):

$$\#(f.x).i \geq \min \{1+\#x.j: 1 \leq j \leq n\} \ .$$

This formula expresses that for each output channel at least the first k+1 output elements are determined by the first k elements on the input channel.

We may then ask for the time, the *delay*, that it takes until messages received at input lines show effects at output lines. The most remarkable observations about such a modelling of timed system, however, can be stated as follows

(a) Pulse driven agents produce infinite output streams on infinite input streams.

(b) Streams that are (components of) least fixed points of pulse driven functions are infinite.

(c) Parallel and sequential composition of pulse driven agents as well as feedback lead to pulse driven agents.

Observation (b) allows one to conclude that fixed points of pulse driven functions are unique. Observation (a) shows that for pulse driven functions every input history with a complete timing is mapped onto an output history with a complete timing.

Pulse driven functions in particular form an adequate functional model for switching circuits.

Studying certain properties of timed communicating systems by abstracting away time information is a methodologically attractive idea. Therefore it is useful to classify those timed functions and timed agents that do not use timing information in an essential way and thus allow one to study message flow independent from timing. In general, however, the message flow of communicating systems may depend on timing. The order (the relative timing) in which messages are received on different input lines may influence the behaviour of systems. Nevertheless for some agents this timing is not relevant for the produced actual messages or it influences the produced actual messages not in an essential way: this can be modelled by nondeterminism. Some other situations cannot be mapped so simply onto nondeterminism. The respective agents are called *time critical.*

A timed stream processing function f is called *time insensitive*, if the time information on the input streams does not influence the produced message streams at all, i.e. if

$$M©x = M©x' \;\Rightarrow\; M©f.x = M©f.x'.$$

Here we write for $x \in (S^{\omega})^n$ simply $M©x$ instead of $(M©x_1, ..., M©x_n)$.

A time insensitive timed stream processing function f uniquely determines a nontimed stream processing function g by the formula

$$g(M©x) = M©f.x\ .$$

Moreover g is prefix monotonic and continuous, if f is so. This is immediately seen by the fact that $(M^{\omega})^n \subseteq ((M^{\surd})^{\omega})^n$, and, since $M©x = x$ for $x, y \in (M^{\omega})^n$ with $x \sqsubseteq y$ we have:

$$g.x = M©f.x \sqsubseteq M©f.y = g.y\ .$$

Here we used the fact that the function λx: $M©x$ is prefix monotonic. Note, moreover, this function is continuous.

We define a time abstraction function

$$\varphi: ((M^{\surd})^{\omega})^n \to (M^{\omega})^n$$

that deletes the ticks in a stream by

$$\varphi.x = M©x\ .$$

A timed function f is time insensitive if φ is homomorphic with respect to a function g called the *time abstraction* for f where:

$$\varphi.f.x = g.\varphi.x$$

or in a more combinatorial notation if

$$f ; \varphi = \varphi ; g.$$

For a time insensitive function abstracting away time does not introduce any nondeterminism. We obtain a uniquely determined function on nontimed pure message streams.

Returning to the specification of the alternating bit protocol, functions specified by the predicate TCR in the timed specification of the alternating bit protocol characterize timed functions that are time insensitive and their time abstractions coincide with the functions specified by CR given for nontimed functions above.

Given a (n, m)-ary timed agent specification T the relation

$$W: (M^{\omega})^n \times (M^{\omega})^m \rightarrow \mathbb{B}$$

defined by

$$W(x, y) \equiv \exists\, x' \in ((M \cup \{\surd\})^{\omega})^n, f: T.f \wedge x = M©x' \wedge y = M©f.x',$$

is called *time free input/output relation.*

To keep specifications abstract we would like to be able to abstract away the exact time aspects in specifications. This is certainly not reasonable for arbitrary timed agents or stream processing functions. However, for certain agents this abstraction is adequate. Of course, we require some fundamental properties for making the abstraction mapping methodological reasonable. These requirements are classical:

(a) the abstraction enables the deduction of the time free input/output relation of the original agent,

(b) the abstraction is *modular* with respect to parallel and sequential composition as well as with respect to feedback.

The requirement (b) essentially means that the abstraction function is required to be a homomorphism with respect to the compositional forms. Note, if Φ is a homomorphism with respect to all functions occurring in a compositional form describing a network, then the network can be refined by just refining the nodes of the network. We define the time abstraction function Φ mapping time insensitive stream processing functions f onto nontimed functions by

$$\Phi.f =_{df} f' \quad \text{if} \qquad f ; \varphi = \varphi ; f'$$

Φ is modular, i.e. we have

$$\Phi(h \| g) = \Phi.h \;\|\; \Phi.g,$$

$$\Phi(h;g) = \Phi.h \;;\; \Phi.g,$$

$$\Phi(\mu^k f) = \mu^k\, \Phi.f.$$

as long as f, h, g are time insensitive. The proof is straightforward.

Similar to the definition of time insensitive functions we define time insensitivity of agents represented by specifications. Consider the specification

$$T: [((M^{\surd})^{\omega})^n \to ((M^{\surd})^{\omega})^m] \to \mathbb{B}$$

T is called *time insensitive*, if there exists a specification

$$Q: [(M^{\omega})^n \to (M^{\omega})^m] \to \mathbb{B}$$

such that

$$T ; \varphi = \varphi ; Q.$$

If a stream processing function is not time insensitive, then abstracting away time can be seen as introducing some kind of nondeterminism.

Given a (not necessarily time insensitive) timed function f, a nontimed function g is called a *strong time abstraction for f,* and we write

$$g \textbf{ abs } f,$$

if for every input x to g there exists a total timing y of the input such that the behaviour of g on x coincides with the time abstraction of the behaviour of f on y, i.e.

$$\forall x \in (M^{\omega})^n: \exists y \in ((M^{\surd})^{\infty})^n: x = M©y \wedge (\varphi ; g)|_{(y\downarrow)} = (f ; \varphi)|_{(y\downarrow)},$$

where we use the following abbreviations: $f|_S$ denotes the restriction of the function f to S (where S is a subset of the domain of f) and:

$$y\downarrow = \{x: x \sqsubseteq y\} .$$

A timed function f is called *weakly time insensitive*, if for every input $y \in (M^{\surd})^{\omega}$ there exists a prefix continuous strong time abstraction g for f such that:

$$g(M©y) = M©f.y .$$

When abstracting from time we may try to represent the behaviour (apart from timing aspects) of a weakly time insensitive function by its set of strong time abstractions. For a time insensitive function all its strong time abstractions coincide.

Example: *Weakly time insensitive functions*
Let us consider a function that reproduces its input messages provided there is "sufficient time" between two messages:

$$f: (M^{\surd})^{\omega} \to (M^{\surd})^{\omega}$$

is specified by (let $x \in M$, i.e. $x \neq \surd$, $y \in M^{\surd}$):

$$f.\langle\rangle = \langle\rangle,$$

$$f.\langle y\rangle = \langle\surd\rangle,$$

$$f(\surd \& s) = \surd \& f.s,$$

$$f(x \& y \& s) = \surd \& x \& f.s.$$

f is not time insensitive, since

$$M©f.\langle x\ \surd\ x\ \surd\rangle = \langle x\ x\rangle \neq \langle x\rangle = M©f.\langle x\ \ x\rangle .$$

Nevertheless f is weakly time insensitive. All strong time abstractions for f fulfil the nontimed transmitter specification. ◊

The transmitter specification TCT defines functions that are weakly time insensitive. The sender specification TCS given above is not even weakly time insensitive. Since the input of an infinite stream $\surd^\infty$ of time ticks leads to an infinite number of copies of the message (a,b) in the output stream, there are, due to lack of monotonicity, no strong time abstractions for the timed functions specified by TCS.

For a weakly time insensitive function its output may depend on the timing of the input, but only in a very weak ("nondeterministic") form. In particular, the function f must not react to infinite streams of time ticks by an infinite number of actual messages (but with different messages in reaction to proper messages) since this would generally be in conflict with the monotonicity requirement. Instead it must respond with different messages in reaction to proper messages.

A further example of a timed function that is not weakly time insensitive is the repeater rep.m. We have:

$$(\text{rep.m}).(\surd^\infty) = m^\infty$$

and (for $a \in M$):

$$(\text{rep.m}).(\surd^n \hat{}\langle a\rangle\hat{}\ \surd^\infty) = (m^n)\hat{}(a^\infty)$$

which contradicts the required monotonicity for strong time abstractions.

So far we have considered time abstractions for stream processing functions. Now we study time abstractions for specifications. A nontimed agent specified by the predicate Q on nontimed stream processing functions is called *time abstraction* for T and we denote Q by Φ.T, if for every nontimed input x for Q every nontimed behaviour shown by Q is a reflection of a timed behaviour of T:

$$Q.g \Rightarrow \exists\ f: T.f \wedge g\ \textbf{abs}\ f\ .$$

This generalizes the notion of time insensitivity to timed agents. Q is called *full time abstraction* for T, if for every timed input y to T all behaviours possibly generated by T are mirrored by strong time abstractions specified by Q:

$$T.f \Rightarrow \forall\ y \in ((M^{\surd})^\infty)^n: \exists\ g: g\ \textbf{abs}\ f \wedge Q.g \wedge (\varphi\ ;\ g)|(_{y}\downarrow) = (f\ ;\ \varphi)|(_{y}\downarrow)\ .$$

For a time insensitive nondeterministic agent specified by the predicate T either all functions f with T.f are time insensitive or the time sensitivity is not observable due to the nondeterminism in T. Technically speaking, whenever we have a timed input y and some behaviour function f with T.f, then for every timed input y' with M©y = M©y' there is a function f' with T.f' such that M©f.y = M©f'.y'. So for the output M©f.y for a function f for which we only know T.f it cannot be predicted more by knowing the timing of y.

The requirement of the existence of strong time abstractions g is very strong. We require for a strong time abstraction g that g is continuous and for every input x the output g.x is the time abstraction M©f.y for an output f.y of a function f with T.f for some infinite timed input y. Monotonicity (which is part of continuity) enforces that for certain distinct infinite streams y and y' with M©y ⊑ M©y' we have functions f and f' such that

$$T.f \wedge T.f' \wedge (\varphi\ ;\ g)|(_{y}\downarrow) = (f\ ;\ \varphi)|(_{y}\downarrow) \wedge (\varphi\ ;\ g)|(_{y'}\downarrow) = (f'\ ;\ \varphi)|(_{y'}\downarrow)\ .$$

This requirement is rather strong. In particular, it implies

$$M©f.y \sqsubseteq M©f'.y'$$

which is not at all true if, for instance, when $n = 1$ both M©f.y and M©f'.y' are infinite and distinct. So for certain agents such as fair timed merge such strong time abstractions do not exist.

Moreover, time insensitivity of agents T and T' does not allow the conclusion of time insensitivity of

$$T ; T',$$

since the timing in f.x (where T.f holds) may such that for the function f' with T'.f' only very particular output f'.f.x is produced, due to the timing of f.x. So we cannot conclude time insensitivity for T ; T' from the time insensitivity of T and T'. Nevertheless we may introduce a further notion that expresses that for given input y all timings of the output are possible.

An agent specification T is called *time unspecific*, if for every infinite input y and all pulse driven functions f and f" with T.f and $(f ; \varphi)|_{(f.y)\downarrow} = (f'' ; \varphi)|_{(f.y)\downarrow}$ there exists a function f' such that

$$T.f' \wedge f'|_{(f.y)\downarrow} = f''|_{(f.y)\downarrow} .$$

This means that the timing in the output is not influenced by the timing in the input. If an agent is time unspecific then the timing in the input does not influence the data messages in the output nor the timing in the output.

7. Input Choice Specifications

Even for not weakly time insensitive agents we may look for a nondeterministic nontimed modelling under certain assumptions. For certain time sensitive agents such as the sender in the example of the alternating bit protocol the choice of the (function generating its) output may depend on the timing of the input. In particular an infinite stream of time ticks may be mapped onto an infinite stream of proper messages. Looking for strong time abstractions for those function we run into conflicts with monotonicity. Nevertheless particular time abstractions for such functions can be represented at the level of nondeterminism by an input choice specification.

Formally an *input choice specification* of an (n,m)-ary agent is given by a predicate

$$R: [(M^{\omega})^{n} \rightarrow (M^{\omega})^{m}] \times (M^{\omega})^{n} \rightarrow \mathbb{B}.$$

The proposition R is used to specify the behaviour of an agent. R(f, x) is supposed to express that f and also all g such that $g \sqsubseteq f$ are partially correct stream processing functions with respect to the specified agent, i.e. correctly reflect safety properties. Moreover, for the input x the output f.x is correct output, i.e. correct also with respect to liveness conditions. In other words R(f, x) is supposed to stand for the following two logical statements:

- f is a function that is safe (fulfils the safety requirements),
- the output f.x is live (fulfils the liveness requirements) for input x and the behaviour (the causality between input and output) indicated by f.

This form of a specification is called input choice specification, since in contrast to the forms of functional specifications considered so far, where the choice of the actual function in a computation is free from considerations of the actual input, in an input choice specification the choice of the function can definitely depend on the input.

For a programming language that allows one to write programs satisfying such specifications cf. [Broy 86] and [Broy 87a]. For a logical calculus for relating the programs written in such a language to input choice specifications see [Broy 87b].

Every agent specification

$$Q: [(M^{\omega})^n \to (M^{\omega})^m] \to \mathbb{B}$$

can be translated schematically into an input choice agent specification

$$Q': [(M^{\omega})^n \to (M^{\omega})^m] \times (M^{\omega})^n \to \mathbb{B}$$

by

$$Q'(f, x) = \exists\, f': Q.f' \land f \sqsubseteq f' \land f.x = f'.x \,.$$

For notational convenience we write often Q instead of Q'.

Seen as a time abstraction of an agent T specifying a set of timed pulse driven functions g we may understand R(f, x) as the logical statement "f is a partially correct time abstraction with respect to some timed function g with T.g and f.x is totally correct with respect to a timed stream y with complete time information, i.e. y is infinite and x = M©y holds". This way input choice specifications provide a concept for abstractions from time for functions that are time sensitive only in a restricted manner.

For composing input choice specifications of system components we may again use the three classical concepts of sequential and parallel composition and feedback. For input choice specifications R and R' we define the compositional forms as follows:

$$(R\|R').(f, (x, z)) \equiv \exists\, g, h: R(g, x) \land R'(h, z) \land f = g\|h,$$

$$(R \,;\, R').(f, x) \equiv \exists\, g, h: R(g, x) \land R'(h, g.x) \land f = (g \,;\, h),$$

$$(\mu^k R).(f, x) \equiv \exists\, y, z, g: R(g, (x,y)) \land (z, y) = f.x \land f = \mu^k g.$$

We again write μR for $\mu^1 R$.

For an input choice specification R and an input $x \in (M^{\omega})^n$ an output $y \in (M^{\omega})^m$ is called *correct*, if there exists a stream processing function f such that

$$R(f, x) \land f.x = y.$$

An output finite function g is called *partially correct* or *safe* with respect to the specification R, if for all input elements x:

$$\exists\, f: R(f, x) \land g \sqsubseteq f.$$

Generally a function f is called *safe* with respect to R, if all its approximations by output finite functions are safe with respect to R.

We call two input choice specifications R and R' *relationally equivalent*, if for every input x the sets of totally correct output for R and R' coincide. We then write

$$R \sim R'.$$

Unfortunately the equivalence relation ~ is not a congruence.

Example: *The equivalence ~ is not a congruence.*

We consider two (1,1)–ary agents. Let the continuous stream processing functions

$$f, g, h: [\mathbb{B}^{\omega} \to \mathbb{B}^{\omega}]$$

be defined (for all $x \in \mathbb{B}^{\omega}$) by the following equations

$$f.x = \langle \mathbb{1}\ \mathbb{1} \rangle, \qquad g.\langle\rangle = \langle \mathbb{1} \rangle, \qquad h.\langle\rangle = \langle\rangle,$$

$$g(\mathbb{1}\&x) = h(\mathbb{1}\&x) = \langle \mathbb{1}\ \mathbb{0} \rangle,$$

$$g(\mathbb{0}\&x) = h(\mathbb{0}\&x) = \langle \mathbb{1}\ \mathbb{0} \rangle .$$

Note $f.\langle\rangle = \langle \mathbb{1}\ \mathbb{1} \rangle$.We define the specifying predicates R1 and R2 by

$$R1(q, x) \equiv (q = f \vee (x \neq \langle\rangle \wedge q = g)),$$

$$R2(q, x) \equiv (q = f \vee (x \neq \langle\rangle \wedge q = h)).$$

If we consider feedback $\mu R1$ and $\mu R2$ for the agents R1 and R2 we define its meaning by the least fixed points z of functions q with R(q, z). The least fixed points of f and g are $\langle \mathbb{1}\ \mathbb{1} \rangle$ and $\langle \mathbb{1}\ \mathbb{0} \rangle$ resp. and fulfil R1. The least fixed point of f also fulfils R2. So R1 exhibits two behaviours under feedback while R2 exhibits only one, since the least fixed point of h is $\langle\rangle$ which is ruled out since it does not satisfy the input choice condition. ◊

The example in particular shows that under feedback relationally equivalent agents may lead to agents that are not relationally equivalent.

This trivial example shows further aspects of safety properties. If we give a partial input x (in our example $\langle\rangle$) to an agent and we observe some finite output y, which of course has to be an approximation for some totally correct output, then this may indicate that some particular decisions have been taken inside the agent (in the sense of nondeterministic choices). If we observe in our example the output $\langle \mathbb{1} \rangle$ for the input $\langle\rangle$, then agent R1 still may be free to choose between f and g, while R2 certainly has already chosen f. Partial output may indicate that certain choices have taken place. This is relevant for determining the behaviour of agents within feedback loops. Note, the chosen examples correspond directly to the example provided in [Brock, Ackermann 81].

The power of this specification method can be demonstrated by the specification of an agent which represents fair nonstrict merge:

$$\text{FAIR\_NONSTRICT\_MERGE}(f, (x, y)) \equiv \text{FAIR\_MERGE}.f \wedge \#f(x, y) = \#x + \#y.$$

The sender specification for the alternating bit protocol now can be written as an input choice specification as follows:

$$\begin{aligned} CS(f, (x', y')) \equiv\ & (\forall\, x, y: && res.f(x, y) \sqsubseteq x \wedge \\ & && \#y < \infty \Rightarrow \#res.f(x,y) \leq 1+\#alt.y \wedge \\ & && \#y = \infty \Rightarrow \#res.f(x,y) \leq \#alt.y) \wedge \\ & (\#x' > \#alt.y' \Rightarrow \#f(x', y') = \infty\ ) \wedge \\ & \#res.f(x', y') = \min(\#x', 1+\#alt.y') . \end{aligned}$$

The first part of the specification defines the safety properties (which are just general properties for function f) and the second part defines the liveness properties.

8. Safety and Liveness

The concept of input choice specification strongly reflects on the concepts of safety and liveness properties. An input choice specification

$$R: [(M^{\omega})^n \to (M^{\omega})^m] \times (M^{\omega})^n \to \mathbb{B}$$

may in particular always be decomposed into safety and liveness properties.

Partial correctness or safety properties of communicating systems indicate which communications may occur as output caused by particular input. The predicate

$$S: [(M^{\omega})^n \to (M^{\omega})^m] \to \mathbb{B}$$

is called the *safety predicate* for the specification R, if a function f is partially correct with respect to R iff S.f. According to our definition of partial correctness we can expect that if f is least upper bound of a chain of functions fulfilling S, then S.f. Safety predicates are downward closed and closed with respect to least upper bounds of chains of functions. Note, S specifies more than just a relation between input and output: it specifies a set of functions. Therefore as shown in section 3 additional information about the causality between input, output and nondeterministic choice is provided by S, too.

For the specification R the predicate

$$L: [(M^{\omega})^n \to (M^{\omega})^m] \times (M^{\omega})^n \to \mathbb{B}$$

is called the *liveness predicate* for R, if

$$L(f, x) \equiv (S.f \Rightarrow R(f, x)) .$$

The liveness predicate specifies if for a partially correct function and some input history enough output is provided. Obviously an input choice specification R can be reconstructed from its safety and its liveness predicates if we assume R(f, x) ⇒ S.f.

9. Abstracting from Time (continued)

Similar to the definition of time insensitivity functions we define time insensitivity of agents described by input choice specifications. Let an agent be specified by the predicate

$$T: [((M^{\surd})^{\omega})^n \to ((M^{\surd})^{\omega})^m] \to \mathbb{B}$$

T is called *time insensitive on infinite timed streams* if there exists an input choice agent specification

$$Q: [(M^{\omega})^n \to (M^{\omega})^m] \times (M^{\omega})^n \to \mathbb{B}$$

such that

$$\#x = \infty \Rightarrow (T ; \varphi).x = (\varphi ; Q).x .$$

In a time insensitive nondeterministic agent specified by the predicate T either all functions f with T.f are time insensitive or the time sensitivity is not observable due to the nondeterminism in T.

Technically speaking, as in the case of general time abstractions, whenever we have a timed input y and some behaviour function f with T.f, then for every timed input y' with M©y = M©y' there is a function f' with T.f' such that M©f.y = M©f'.y'. So the output M©f.y for f for which we only know T.f cannot be predicted by the timing of y.

When considering abstractions from time, the question becomes crucial, whether timed agents behave essentially differently on timed partial (finite) input streams s and timed total (infinite) input streams t with M©s = M©t. A special problem arises, in particular, for completely timed streams s and r (i.e. $\#s = \#r = \infty$) where M©s $\sqsubseteq$ M©r. Without assuming $s \sqsubseteq r$ we cannot assume for timed stream processing functions f any particular relationship between f.s and f.r and therefore nothing specific can be said, in general, about the relationship between M©f.s and M©f.r (in particular M©f.s $\sqsubseteq$ M©f.r does not hold, in general).

From M©s $\sqsubseteq$ M©r we may only conclude that there exists a finite time transformation t such that for some finite stream v we have

$$v \sqsubseteq t.s \wedge v \sqsubseteq r \wedge M©v = M©s.$$

In general, time abstraction leads to a loss of monotonicity or putting it some other way strong time abstractions do not always exist. An abstraction from time can be obtained also for certain agents that are not time insensitive in the strict sense by input choice specifications as follows.

Let

$$f: ((M^{\surd})^{\omega})^{n} \to ((M^{\surd})^{\omega})^{m}$$

be an arbitrary time dependent function. A prefix continuous function

$$g: (M^{\omega})^{n} \to (M^{\omega})^{m}$$

is called *safe time abstraction* for f, if for every $x \in (M^{\omega})^{n}$ there exists $y \in ((M^{\surd})^{\infty})^{n}$ such that

$$x = M©y \wedge (\varphi ; g)|(y\downarrow) \sqsubseteq (f ; \varphi)|(y\downarrow) .$$

We then write g **abs_safe** f.

For $x \in M^{\omega}$ the function g is called *live for f and input y*, if in addition for $y \in ((M^{\surd})^{\infty})^{n}$ we have

$$x = M©y \Rightarrow (\varphi ; g)|(y\downarrow) = (f ; \varphi)|(y\downarrow) .$$

We then write (g, x) **abs_live** (f, y). Let T be a specifying predicate for a timed agent with n input lines and m output lines. We define the input choice specification R, called *the time abstraction* for T, as follows:

$$R(g, x) \equiv_{df} \exists\, f: T.f \wedge g \textbf{ abs\_safe } f \wedge \exists\, y \in ((M^{\surd})^{\infty})^{n}: (g, x) \textbf{ abs\_live } (f, y) .$$

For a timed agent specification T an input choice specification R is called a time abstraction if

$$T.f \equiv \forall\, y \in ((M^{\surd})^{\infty})^{n}: \exists\, g: R(g, M©y) \wedge (g, M©y) \textbf{ abs\_live } (f, y) .$$

Time abstractions in the form of input choice specifications by definition always exist. However, in the time abstraction the choice of a particular behaviour that may be influenced by the timing using the specification T cannot be influenced any more and is tuned into pure nondeterminism.

Of course, for arbitrary time dependent functions f and f' (or similarly for specifications T and T') this concept of time abstraction is not modular in the following sense: let R and R' be input choice specifications such that R(g, x) (and R'(g, x)) if g is a safe time abstraction for f (and for f' resp.)

that is life with respect to to input x. For the safe time abstraction h for the function (f ; f') that is life with respect to x we might consider

$$(R\ ;\ R')(h,\ x).$$

However, from (R ; R')(h, x) we cannot conclude that h is a safe time abstraction for (f ; f') that is life with respect to x. The individual use of time information in the composition of f and f' cannot be expressed sufficiently well at the level of nondeterminism, if the involved agents are time sensitive.

10. Proper Specifications and Least Fixed Points

As in every powerful specification mechanism it is possible to write inconsistent input choice specifications. But apart from consistency there are further properties we require for an input choice specification to make sure that we always get consistent specifications when we combine it with other proper specifications. We formalize such properties in the following. An input choice specification R

$$R: [(M^{\omega})^{n} \rightarrow (M^{\omega})^{m}] \times (M^{\omega})^{n} \rightarrow \mathbb{B}$$

is called *consistent*, if

$$\forall\ x: \exists\ f: R(f, x)\ .$$

From the viewpoint of a time abstraction consistency just means that for a nearly time insensitive agent T from which R is the time abstraction there is at least a timed stream-processing function f with T.f. So the definition coincides with the classical notion of consistency.

The consistency of specifications R and R' implies the consistency of R;R' and R||R'. But the consistency of R does not imply the consistency of $\mu^k R$. Therefore we are especially interested in conditions guaranteeing that an input choice specification (with n = m) has a least fixed point. We therefore ask the question under which conditions for an input choice specification R we have:

$$\exists\ f: R(f,\ \mathbf{fix}\ f)\ .$$

A stream z is called *least fixed point* of an input choice specification, if there exists a function f with z = **fix** f, i.e. z is least fixed point of f and R(f, z).

In addition to the straightforward notion of consistency mentioned above add the following requirement for input choice specifications which is a generalisation of the property of monotonicity. If we have R(f, x) then for every output finite function $g \sqsubseteq f$ and every input x' with $x \sqsubseteq x'$ we expect that there exists a behaviour function f' such that R(f', x') and $g \sqsubseteq f'$. Accordingly R is called *weakly monotonic*, if for every output finite prefix continuous function g we have

$$R(f, x) \wedge g \sqsubseteq f \wedge x \sqsubseteq x' \Rightarrow \exists\ f': R(f', x') \wedge g \sqsubseteq f'.$$

With this assumption we may think about computations of agents as follows: let $\{x.i: i \in \mathbb{N}\}$ be chain (i.e. $x.i \sqsubseteq x.(i+1)$ for all i) of input streams. The output of the agent specified by R is computed for the input stream lub $\{x.i: i \in \mathbb{N}\}$ by the following strategy: for every i we choose functions f.i and output finite approximations g.i for f.i such that f.i is a correct choice for the input x.i, the output (g.i).x.i is an approximation (a prefix) for the output generated by f.i on x.i and the function f(i+1) is approximated by f.i . Formally we have (for all $i \in \mathbb{N}$):

$$R(f.i,\ x.i),$$

$$g.i \sqsubseteq f.i,$$

$$g.i \sqsubseteq g(i+1) \sqsubseteq f(i+1)\ .$$

For being able to guarantee that the least upper bound of the chain $\{g.i: i \in \mathbb{N}\}$ fulfils required liveness properties, however, we need one additional assumption. The idea is to choose the chain $\{g.i : i \in \mathbb{N}\}$ such that (for all $i \in \mathbb{N}$):

$$R(\text{lub}\ \{g.i: i \in \mathbb{N}\}, \text{lub}\ \{x.i: i \in \mathbb{N}\}).$$

For modelling this choice we require the existence of a function δ that indicates for given x.i and g.i how large we should choose g(i+1) to guarantee both liveness and safety conditions. The function δ therefore allows the construction of such a chain of (g.i).x.i as approximations for the output.

Operationally one may think about a computation of an agent specified by the input choice specification R as follows: the agent gets step by step input and produces step by step output. The input can be modelled by a chain $\{x.i: i \in \mathbb{N}\}$ of finite elements, the corresponding output is also modelled by a chain $\{(g.i).x.i: i \in \mathbb{N}\}$ of finite elements. Whenever the agent gets additional input, which can be modelled by increasing input x.i to x(i+1), it may add messages to its output streams by increasing output (g.i).x.i to (g.i).x(i+1). However, the strategy has to be chosen carefully so that the stepwise produced output is large enough, and the least upper bound of $\{(g.i).x.i: i \in \mathbb{N}\}$ is correct in the sense of liveness requirements, but it has to be small enough so that the output produced so far is still partially correct for any possible additional input.

A function

$$\delta: [(M^*)^n \rightarrow (M^*)^m] \times [(M^\omega)^n \rightarrow (M^\omega)^m] \rightarrow [(M^*)^n \rightarrow (M^*)^m]$$

is called *computation strategy*, if the following formulas hold:

(1) $g \sqsubseteq f \Rightarrow g \sqsubseteq \delta(g, f) \sqsubseteq f,$

(2) $g.0 \sqsubseteq f \wedge \forall i: g(i+1) = \delta(g.i, f) \Rightarrow \text{lub}\ \{g.i: i \in \mathbb{N}\} = f\ .$

An input choice specification R is called *weakly continuous,* if there exists a computation strategy δ that fulfils the following requirements: it allows for some input x and some safety correct output y to produce a chain $\{g.i: i \in \mathbb{N}\}$ of approximations with the least upper bound f such that for every i, if we increase the input x to x' (i.e. $x \sqsubseteq x'$), then g(i+1) produces safe output for the input x'.

More formally expressed we require for R the following properties: For all chains $\{x.i: i \in \mathbb{N}\} \subseteq (M^\omega)^n$ we have:

$$(\forall i: \exists f: R(f, x.i) \wedge g(i+1) = \delta(g.i, f)) \Rightarrow R(\text{lub}\ \{g.i: i \in \mathbb{N}\}, \text{lub}\ \{x.i: i \in \mathbb{N}\}).$$

Intuitively speaking, the function δ gives for every input x and every approximation g for a function f (i.e. $g \sqsubseteq f$) a better approximation $\delta(g, f)$ for f such that for every continuation of the input stream x into a stream x' (where $x \sqsubseteq x'$) the approximation is safe with respect to x' and can be continued to a live output by using function generated by using functions generated by successively applying δ.

By the assumptions above we may prove the existence of least fixed points. An input choice specification is called *proper*, if it is consistent, weakly monotonic and weakly continuous. Note, the input specification CS for the sender in the alternating bit example is proper.

Theorem: A proper input choice specification has a least fixed point.

Proof: Let R be a proper input choice specification. We define chains of tuples of streams $\{x.i: i \in \mathbb{N}\}$ and functions $\{g.i: i \in \mathbb{N}\}$ such that

$$x.0 = \langle\rangle^n ,$$

$$(g.0).x = \langle\rangle^m , \qquad \text{for all } x.$$

g.0 is safe according to consistency of R. Given x.i, g.i we define

$$x(i+1) =_{df} (g.i).x.i,$$

$$g(i+1) =_{df} \delta(g.i, f.i), \text{ where f.i is arbitrary, such that } R(f.i, x.i) \wedge g.i \sqsubseteq f.i .$$

Such a function f.i always exists since R is consistent and weakly monotonic and δ is the computation strategy that exists according to weak continuity. According to weak continuity we have:

$$R(\text{lub } \{g.i: i \in \mathbb{N}\}, \text{lub } \{x.i: i \in \mathbb{N}\}).$$

By the construction of the x.i and f.i we obtain

$$\text{lub } \{g.i: i \in \mathbb{N}\}.\text{lub}\{x.i: i \in \mathbb{N}\} =$$

$$\text{lub } \{(g.i).x.i: i \in \mathbb{N}\} =$$

$$\text{lub } \{x(i+1): i \in \mathbb{N}\} =$$

$$\text{lub } \{x.i: i \in \mathbb{N}\} .$$

So lub $\{x.i: i \in \mathbb{N}\}$ is a fixed point of lub $\{g.i: i \in \mathbb{N}\}$. According to the construction we have

$$\text{lub } \{g.i: i \in \mathbb{N}\}.x(i+1) \sqsupseteq (g.i).x.i .$$

So it is the least fixed point. We obtain:

$$\textbf{fix } \text{lub } \{g.i: i \in \mathbb{N}\} = \text{lub}\{x.i: i \in \mathbb{N}\} \qquad \Diamond$$

The existence of least fixed points is essential for proofs that rely on fixed point arguments. An example is the alternating bit protocol again.

What is also important is the invariance of the properties used in the theorem above.

Theorem: The sequential and parallel composition and feedback applied to proper input choice specifications yields proper input choice specifications.

Proof: For sequential and parallel composition the proof of the theorem is rather straightforward. Therefore we concentrate only on feedback. Let R be a proper specification. According to the theory on the existence of least fixed points $\mu^k R$ is consistent. Weak monotonicity and weak continuity follow from the continuity of the fixed point operator. ◊

The notion of proper input choice specifications replaces the classical monotonicity and continuity requirements for functions.

11. Correctness Proof of the Alternating Bit Protocol

We return to the alternating bit protocol and give a proof of its correctness. This proof uses just straightforward reasoning based on fixed point properties and does not use induction. The proof is due to Birgit Schieder (cf. [Schieder 91]). It has to be shown that the function specified by the alternating bit agent is the identity, i.e. that the following equation holds for every input stream x:

$(\mu\ (fs\ ;\ f\ ;\ fr\ ;\ (id \parallel g))\ ;\ pr1).x = x$

where $CS\ (fs, (x, (\mu\ (fs\ ;\ f\ ;\ fr\ ;\ (id \parallel g))\ ;\ pr2).x)) \wedge CR.fr \wedge CT.f \wedge CT.g \wedge ID.id \wedge$

$CP.pr1 \wedge (\forall\ s1, s2{:}\ pr2(s1, s2) = s2 \wedge pr1(s1, s2) = s1)$

We need some properties of the involved functions for our proof:

(0)	$\forall y{:}\ \#alt.y \geq \#alt.g.y$	(by CT)
(1)	$\forall y{:}\ \#alt.ack.y \geq \#alt.pr2.(id \parallel g).fr.f.y$	(by (0) and CR)
(2)	$\forall y{:}\ \#res.y \geq \#alt.ack.y$	(by res)
(3)	$\forall y{:}\ \#y = \infty \Rightarrow \#pr2.(id \parallel g).fr.f.y = \infty$	(by CT and CR)
(4)	$\forall y{:}\ (\#res.y < \infty \wedge \#res.y = \#alt.ack.y \wedge \#alt.ack.y = \#alt.ack.f.y) \Rightarrow$	
	$res.y = res.f.y$	(by CT)

We abbreviate the second component of the least fixed point by v:

$v =_{def} (\mu\ (fs\ ;\ f\ ;\ fr\ ;\ (id \parallel g))\ ;\ pr2).x$

From this definition it follows immediately that in particular v is a fixed point:

(5) $v = pr2.(id \parallel g).fr.f.fs(x,v)$

At first we show that

$\#x = \#alt.v$.

We know that

$\#alt.v \leq \#alt.ack.fs(x,v)$	(by (1) and (5))
$\leq \#res.fs(x,v)$	(by (2))
$\leq \#x$	(by the 5th axiom of CS)

In the other direction it holds that

	$\#res.fs(x,v) = \min\ (\#x,\ 1+\#alt.v)$	(by the 5th axiom of CS)
$\Rightarrow$	$\#x < 1+\#alt.v \ \vee\ \#res.fs(x,v) = 1+\#alt.v$	
$\Rightarrow$	$\#x \leq \#alt.v \ \vee\ (\#alt.v = \infty \ \vee\ \#v < \infty)$	(by the 3rd axiom of CS)
$\Rightarrow$	$\#x \leq \#alt.v \ \vee\ \#alt.v = \infty \ \vee\ \#x \leq \#alt.v$	(by the 4th axiom of CS and (3))
$\Rightarrow$	$\#x \leq \#alt.v$	

So we have shown the above equality.

From this equality we may conclude the following equations on lengths:

(6) $\#x = \#res.fs(x,v)$

(7) $\#res.fs(x,v) = \#alt.ack.fs(x,v)$

(8) $\#alt.ack.fs(x,v) = \#alt.ack.f.fs(x,v)$

Let #x be finite now. Then we get

$pr1.fr.f.fs(x,v)$	$= res.f.fs(x,v)$	(by CR)
	$= res.fs(x,v)$	(by $\#x < \infty$, (6), (7), (8) and (4))
	$= x$	(by (6) and the 1st axiom of CS)

Hence the correctness is proved for finite input streams. For infinite input it follows by continuity.

12. Full Abstractness

The three combining forms of sequential and parallel composition as well as feedback define a notion of contexts for agent specifications. Now we are looking for relations ≈ on input choice specifications such that we may derive relational equivalence from it (remember that ~ denotes relational equivalence):

$R1 \approx R2 \Rightarrow R1 \sim R2$

and moreover the relation ≈ forms a congruence with respect to the combining forms of parallel and sequential compositions as well as feedback:

$R1 \approx R2 \Rightarrow R||R1 \approx R||R2 \wedge R1||R \approx R2||R,$

$R1 \approx R2 \Rightarrow R \,;\, R1 \approx R \,;\, R2 \wedge R1 \,;\, R' \approx R2 \,;\, R',$

$R1 \approx R2 \Rightarrow \mu^k R1 \approx \mu^k R2.$

A relation ≈ with such properties is called *compositional*.

The concept for the specification of distributed system components as outlined in the previous sections has been carefully chosen to be compositional. However, our specifications carry too much information, in general. In this section we answer the question under which circumstances two logically distinct input choice specifications are relational equivalent in all contexts. This leads to the concept of full abstractness. A compositional relation is called *fully abstract*, if there is no weaker compositional relation (cf. also [Kok 87], [Jonsson 89]).

For defining a fully abstract relation we introduce the notion of computation. A pair of chains $(\{x.i: i \in \mathbb{N}\}, \{y.i: i \in \mathbb{N}\})$ is called a *computation* for an input choice specification R, if there exists a function f with

$R(f, \text{lub } \{x.i: i \in \mathbb{N}\}),$

$\forall\, i \in \mathbb{N}: y.i \sqsubseteq f.x.i,$

$f.\text{lub } \{x.i: i \in \mathbb{N}\} = \text{lub } \{y.i: i \in \mathbb{N}\}.$

We define a relation ≈ on input choice specifications as follows:

$R \approx R'$ iff the sets of computations of R and R' coincide.

Now we prove that the relation ≈ is fully abstract.

Theorem: The relation $\approx$ is fully abstract.

Proof: We start by proving that the relation cannot be weakened without losing compositionality: R and R' are not relational equivalent for some context, if they have different sets of computations: Assume there is a function

$$f: [(M^{\omega})^{n} \to (M^{\omega})^{m}]$$

and a chain $\{x.i: i \in \mathbb{N}\}$ with $x.0 = (\langle\rangle)^{n}$ and

$$R(f, \text{lub } \{x.i: i \in \mathbb{N}\})$$

and there does not exist a function f' with

$$f.\text{lub } \{x.i: i \in \mathbb{N}\} = f'.\text{lub } \{x.i: i \in \mathbb{N}\}$$

and $f.x.i \sqsubseteq f'.x.i$ for all $i \in \mathbb{N}$ and $R'(f', \text{lub } \{x.i: i \in \mathbb{N}\})$. W.l.o.g. we may assume $n = m$ (add dummy arguments or dummy results) and that either $f.x.i \neq f.x(i+1)$ for all i or there exists an index j with $f.x.i \neq f.x(i+1)$ for all $i < j$ and $f.x.i = f.x(i+1)$ for all $i \geq j$.

We define a specification R'' for (m, n)-ary functions g by

$$R''(g, y) = \forall z: g.z = \text{lub } \{x.i: f.x.i \sqsubseteq z\}$$

We consider the least fixed point y of f ; g where R''(g, f.y). This corresponds to an element of $\mu^{n}(R;R'')$. We show that y is not totally correct with respect to $\mu^{n}(R';R'')$. We define the chain $\{z.i: i \in \mathbb{N}\}$ by

$$z.0 = (\langle\rangle)^{n}$$

$$z(i+1) = g.f.z.i$$

We prove that $\text{lub } \{z.i: i \in \mathbb{N}\} = \text{lub } \{x.i: i \in \mathbb{N}\}$. If there exists an index j with $f.x.i \neq f.x(i+1)$ for all $i < j$ and $f.x.i = f.x(i+1)$ for all $i \geq j$, then this is obvious; otherwise for all $k \in \mathbb{N}$: $x.k = z(k+1)$:

$z(k+1) =$

$g.f.x.k =$

$\text{lub } \{x.i: f.x.i \sqsubseteq f.x.k\} =$

$x.k$

Thus the least upper bounds of $\{z.i: i \in \mathbb{N}\}$ and of $\{x.i: i \in \mathbb{N}\}$ coincide (and $y = \text{lub } \{z.i: i \in \mathbb{N}\}$).

For every function f' with not $f.x.k \sqsubseteq f'.x.k$ for some k we obtain for the chain z'.i defined by

$$z'.0 = (\langle\rangle)^{n}$$

$$z'(i+1) = g.f'.z.i$$

the equation

$$z'.i = z.k \text{ for all } i \geq \min \{j: \neg(f.x.j \sqsubseteq f'.x.j)\}$$

since with $k = \min \{j: \neg(f.x.j \sqsubseteq f'.x.j)\}$ we obtain

$z'.k+1 =$

$g.f'.x.k =$

$\text{lub } \{x.i: f.x.i \sqsubseteq f'.x.k\} =$

z'.k

Thus y cannot be a behaviour of $\mu^n(R' ; R'')$. This concludes the first part of the proof. Next we prove that the relation ≈ is a congruence, i.e. it is invariant under our compositional forms. For every continuous operator

$$\tau: [(M^\omega)^n \to (M^\omega)^m] \times (M^\omega)^{n'} \to [(M^\omega)^{n'} \to (M^\omega)^{m'}] \times (M^\omega)^n$$

we have if $R \approx R'$, then $Q \approx Q'$ where

$$Q(f,x) = \exists\ f', x': R(f', x') \wedge (f, x') = \tau(f', x)$$

(and in analogy Q' is defined based on R'). Since parallel and sequential composition and feedback operator correspond to such continuous operators, the condition is invariant under the considered compositional forms. ◊

Based on this theorem we may prove equivalences of the form $R \approx R'$ for given specifications R and R'. Moreover we may derive a fully abstract model.

In terms of input choice specifications R and R' we have

$$R \approx R' \quad \text{iff} \quad (R \approx> R' \wedge R' \approx> R)$$

where

$$R \approx> R' \quad \text{iff} \quad \forall\ f, x: R(f, x) \Rightarrow \exists\ g: R'(g, x) \wedge f|((f.x)\downarrow) = g|((f.x)\downarrow)$$

This leads to a unique normal form for proper specifications. An input choice specification R is normal form, if for every function f that is partially correct with respect to R we have

$$R(g, x) \wedge f|((f.x)\downarrow) = g|((f.x)\downarrow) \Rightarrow R(f, x) .$$

For proper input choice specifications R and R' in this normal form we have

$$R \approx R' \text{ iff } \forall\ f, x: R(f, x) \Leftrightarrow R'(f, x)$$

So on proper input choice specifications in normal form the logical equivalence is a fully abstract relation.

13. Timed Agents Revisited

Even for timed agents that are not time insensitive we may find ways to abstract away the timing information and still retain a sufficient representation of their behaviour. For making this more precise we first introduce the notion of finite time transformation.

A *finite time transformation* is a continuous function

$$t: [((M^{\surd})^\omega)^n \to ((M^{\surd})^\omega)^n]$$

that causes only finite time-shifts, i.e. we require

(i) $\varphi.t.x = \varphi.x$,

(ii) $\#(\surd©x).i = \infty \Leftrightarrow \#(\surd©t.x).i = \infty$ for all i, $1 \leq i \leq n$.

We write FTT.t, iff a function t fulfils the requirements (i) and (ii). Finite time transformations formalize the concept of independent timing (or speed) of system components. This is often used informally when speaking about time dependent systems and programs when it is stated that "nothing is assumed about the relative speed of two units running in parallel".

An (n,m)-ary timed agent specification T is called *timing stable,* if for all finite time transformations t

$$T.f \Rightarrow T(t ; f)$$

Otherwise the agent specification is called *time critical.* In the specification of a timing stable agent there are no sharp timing requirements. Either the time information in the input is not relevant for the output or it is not exploited effectively due to the nondeterminism of the agent.

Having time information available in the input it is rather simple to specify certain behaviours which are hard to specify without notions of timing. For instance using timed agents it is rather simple to specify timed nondeterministic "strict" fair merge:

$$TNFM.f \equiv_{df} \forall\ x, y: \exists\ r, t: FTT.t \wedge f(x, y) = t.scd(x, y, r) \wedge \#(\mathbb{1}©r) = \infty \wedge \#(\mathbb{0}©r) = \infty$$

$$\textbf{where } \forall\ x, y, r: \ scd(x, y, \mathbb{1}\&r) = ft.x\ \&\ scd(rt.x, y, r) \wedge$$
$$scd(x, y, \mathbb{0}\&r) = ft.y\ \&\ scd(x, rt.y, r)$$

The agent specified by the predicate TNFM in the example above formalizing timed nondeterministic fair merge is time insensitive. This also means that the associated input choice specification fulfils the requirements that will be given for input choice specifications below.

The agent specified by TNFM is timing stable. The timing influences the output only in a nondeterministic way such that finite time shifts do not change the essential behaviour according to the nondeterminism contained in T.

However, this agent behaves essentially differently on finite timed streams and infinite timed streams. If the input on both of the input lines is infinite, then all the input on both lines is guaranteed to appear as output. So for infinite input the merge is fair. This is not true for finite input. This indicates that the specification TFNM is not time insensitive as defined so far. Otherwise there would exist a nontimed function g such that (for certain i and j):

$$g(M©(\surd^{i}\hat{}\langle 1\rangle\hat{}\surd^{\infty}), M©(\surd^{j}\hat{}\langle 2\rangle\hat{}\surd^{\infty})) = \langle 1\rangle\hat{}\langle 2\rangle,$$

and

$$g(\langle\rangle, M©(\surd^{j}\hat{}\langle 2\rangle\hat{}\surd^{\infty})) = \langle 2\rangle.$$

This, however, is in conflict with our monotonicity assumptions for g.

The inclusion of explicit time considerations into the specification of communicating systems leads to a trade off. Certain properties are easier and more explicitly expressible, however, the detailed consideration of timing may lead to problems with overspecification. This trade off can be partly solved by techniques that allow us to specify parts of a system with explicit timing and other parts without explicit timing. For those techniques, however, we need interfaces between timed and nontimed parts of system specifications. The interfaces can be again described by appropriately specified agents. Basically, we may use agents that filter out time informations from streams as well as agents that (re-)introduce time information in a nondeterministic manner. The first is simple. The later is considerably more difficult.

A function

$$t: \ (M^{\omega})^{n} \ \rightarrow \ ((M^{\surd})^{\omega})^{n}$$

is called *complete timing function*, if

(i) $\varphi.t.x = x$,

(ii) $\#(t.x).i = \infty$ for all i, $1 \le i \le n$.

Of course, a complete timing function t is not prefix monotonic as can be seen by the fact that all images of t are infinite. The monotonicity is demonstrated by the following equations

$t.\langle\rangle = \surd^{\infty}$ and $t.\langle a\rangle = \surd^{k}\,\hat{}\,\langle a\rangle\hat{}\,\surd^{\infty}$.

Nevertheless, we may give an input choice specification τ for complete timing functions

$$\tau(t, x) \equiv_{df} (\forall\, z \in (M^{\omega})^{n}\!: \varphi.t.z \sqsubseteq z) \wedge \#t.x = \infty \wedge \varphi.t.x = x.$$

The input choice specification τ fulfils all the requirements necessary for guaranteeing the existence of least fixed points. This shows that by the technique of input choice specification we may combine timed and nontimed models of communicating systems.

Lemma: The input choice specification

$\tau ; \varphi$

represents the identity.

Proof: For $f.x = y$ with $(\tau ; \varphi).(f, x)$ we have

$\exists\, t\!: f = t ; \varphi \wedge \tau(t, x) \wedge f.x = y$

which is by definition of τ equivalent to

$\exists\, t\!: f = t ; \varphi \wedge (\forall\, z \in (M^{\omega})^{n}\!: \varphi.t.z \sqsubseteq z) \wedge \#t.x = \infty \wedge \varphi.t.x = x \wedge f.x = y$

From this we obtain:

$\exists\, t\!: \varphi.t.x = x \wedge \varphi.t.x = y$

which yields

$x = y.$ ◊

Note, the behaviours specified by τ correspond to system components sometimes called *arbiters*. Based on the input choice specification τ we may easily program fair nonstrict merge by timed merge:

τ ; TNFM ; φ .

We may furthermore specify the sender by the predicate CS by the timed sender specification:

CS = τ ; TCS ; φ .

where we include complete time information in the stream of acknowledgements and later filter out the time information again.

A specification

$T\!: [((M^{\surd})^{\omega})^{n} \to ((M^{\surd})^{\omega})^{m}] \to \mathbb{B}$

T is called *timing stable* if there exists an agent specification

$Q\!: [(M^{\omega})^{n} \to (M^{\omega})^{m}] \to \mathbb{B}$

such that

$Q = \tau ; T ; \varphi.$

Lemma: Every time insensitive agent is timing stable.

Proof: For a time insensitive agent T we have by definition

$T ; \varphi = \varphi ; Q$

This allows us to derive

$\tau ; T ; \varphi = \tau ; \varphi ; Q$

and since is the identity we have

$\tau ; T ; \varphi = Q$ ◊

If a stream processing function is not time insensitive, then abstracting away time can be seen as introducing some kind of nondeterminism or underspecification. Certain more sophisticated relationships between the timing and the message flow are no longer visible in the abstraction. In particular, if parts of a system are time sensitive, such that the correct behaviour of one part strongly depends on the particular timing of the other parts time abstraction does not work for proving certain properties.

14. Conclusion

Functional techniques comprising fixed point theory with concepts like monotonicity and continuity together with logical techniques can be seen as the backbone of formal methods for program construction. When considering communicating systems a straightforward application of such techniques seems difficult. This is due to the very particular abstractions by information hiding (such as hiding time information) and the nondeterminism and underspecification introduced thereby. Fortunately these difficulties can be mastered by a careful choice of the specification formalism. This way a simple and powerful formal framework for the functional treatment of communicating systems is obtained. For the functional formalism a verification calculus at hand is the functional calculus that can be used in a straightforward way.

For many applications time aspects are important. According to the principle of abstraction computing scientists are interested in techniques that allow one to talk about behaviours in terms of timing wherever this seems appropriate and to avoid time consideration wherever possible. A rigorous formal foundation for time sensitive communicating systems therefore is of high methodological importance in system design.

After all we have identified the following classifications of timed functions and specifications of time functions. These classifications has to do with the following two questions relevant in connection with timed behaviour:

- Does the timing of the input streams influence the output streams only with respect to their timing?
- Is it possible to capture the timed behaviour by nontimed behaviour using the concept of nondeterminism, i.e. can every time abstract output be seen as an output for a particular timing of the input?

Regarding the first aspect a timed function or a timed specification T is called

- *time insensitive*, if the time information of the input streams does not influence the produced message streams, i.e. if there exists a nontimed specification Q such that

  $$T ; \varphi = \varphi ; Q$$

  where φ is the time abstraction function on timed streams. In particular, T is time insensitive if

  $$T = \varphi ; Q ; \tau$$

- *timing stable*, if the timing does influence the output, but this influence gets unobservable by the nondeterminism in Q and so is an abstraction. This is expressed by

  $$Q = \tau ; T ; \varphi$$

Note, these two formal characterisations just represent notions of refinements for communicating agents as treated in [Broy 91]. This way introducing explicit timing and time abstraction can just be seen as particular concepts of refinement.

Acknowledgement

Discussions with Leslie Lamport have motivated large parts of this work. I am grateful to Frank Dederichs, Thomas Streicher, and Rainer Weber for helpful remarks on draft versions of this paper. I thank Birgit Schieder for careful reading and for the equational proof for the alternating bit protocol.

References

[Abadi, Lamport 91]
M. Abadi, L. Lamport: An old-fashioned receipt for real time. To appear in the REX Workshop on Real Time, 1991

[Alur et al 90]
R. Alur, C. Courcoubetis, D. Dill: Model-checking for real-time systems. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science, 1990, 414-425

[Alur, Dill 90]
R. Alur, D. Dill: Automata for modeling real-time systems. In: ICALP 1990, Lecture Notes in Computer Science, 1990, 322-335

[Baeten, Bergstra 90]
J.C.M. Baeten, J.A. Bergstra: Real time process algebra. Technical Report P8916b, University of Amsterdam, 1990

[Benveniste, Berry 91]
A. Benveniste, G. Berry: The synchronous approach to reactive and real-time systems. INRIA Rapports de Recherche No 1445, 1991

[Bernstein 87]
A.J. Bernstein: Predicate transfer and timeout in message passing. Information Processing Letters, 24, 1987, 43-52

[Bernstein, Harter 81]
A. Bernstein, P.K. Harter, Jr: Proving real-time properties of programs with temporal logic. In: Proceedings of the 8th Annual ACM Symposium on Operating System Principles, 1981, 1-11

[Brock, Ackermann 81]
J.D. Brock, W.B. Ackermann: Scenarios: A model of nondeterminate computation. In: J. Diaz, I. Ramos (eds): Lecture Notes in Computer Science 107, Springer 1981, 225-259

[Broy 83]
M. Broy: Applicative real time programming. In: Information Processing 83, IFIP World Congress, Paris 1983, North Holland Publ. Company 1983, 259-264

[Broy 85]
M. Broy: Specification and top down design of distributed systems. In: H. Ehrig et al. (eds.): Formal Methods and Software Development. Lecture Notes in Computer Science 186, Springer 1985, 4-28, Revised version in JCSS 34:2/3, 1987, 236-264

[Broy 86]
M. Broy: A theory for nondeterminism, parallelism, communication and concurrency. Habilitation, Fakultät für Mathematik und Informatik der Technischen Universität München, 1982, Revised version in: Theoretical Computer Science 45 (1986) 1-61

[Broy 87a]
M. Broy: Semantics of finite or infinite networks of communicating agents. Distributed Computing 2 (1987), 13-31

[Broy 87b]
M. Broy: Predicative specification for functional programs describing communicating networks. Information Processing Letters 25 (1987) 93-101

[Broy 91]
M. Broy: Interaction Refinement - the Easy Way. Forthcomming paper

[Davies, Schneider 89]
J. Davies, S. Schneider: Factorizing proofs in timed CSP. In Mathematical Foundations of Programming Semantics. Lecture Notes in Computer Science 442, Springer-Verlag, 1989, 129-159

[Dybier, Sander 88]
P. Dybier, H. Sander: A functional programming approach to the specification and verification of concurrent systems. Chalmers University of Technology and University of Göteborg, Department of Computer Sciences 1988

[Haase 90]
V.H. Haase: Real-time behaviour of programs. IEEE Transactions on Software Engineering, SE-7(5), 1981, 494-501

[Hooman 91]
J. Hooman: A denotational real-time semantics for shared processors. In: Parallel Architectures and Languages Europe. LNCS, Springer-Verlag, 1991

[Hooman, Widom 89]
J. Hooman, J. Widom: A temporal-logic based compostional proof system for real-time message passing. In Parrallel Architectures and Languages Europe, volume II. LNCS 336, Springer-Verlag, 1989, 424-441

[Huizing et al. 87]
C. Huizing, R. Gerth, W. P. de Roever: Full abstraction of a real-time denotational semantics for an OCCAM-like language. In: Proceedings of the 14th ACM Symposium on Principles of Programming Languages , 1987, 223-237

[Jahanian, Mok 86]
F. Jahanian and A. Mok. Safety analysis of timing properties in real-time systems. Transactions on Software Engineering, SE-12(9), 1986, 890-904

[Jonsson 89]
B. Jonsson: A fully abstract trace model for dataflow networks. 16th POPL 1989, 155 - 165

[Joseph, Goswami 89]
M. Joseph, A. Goswami: Relating computation and time. Research Report RR138, Department of Computer Science, University of Warwick, 1989.

[Kahn, MacQueen 77]
G. Kahn and D. MacQueen, Coroutines and networks of processes, Proc. IFIP World Congress 1977, 993-998

[Kok 87]
J.N. Kok: A fully abstract semantics for data flow networks. Proc. PARLE, Lecture Notes in Computer Science 259, Berlin-Heidelberg-New York: Springer 1987, 214-219

[Koymans 90]
R. Koymans: Specifiying real -time properties with metric temporal logic. Real-Time Systems, 2(4), 1990, 255-299

[Koymans at al. 88]
R. Koymans, R.K. Shyamasundar, W.P. de Roever, R. Gerth, S. Arun-Kumar: Compositional semantics for real-time distributed computing. Information and Computation, 79(3), 1988, 210-256

[Koymans et al. 83]
R. Koymans, J. Vytopyl, W.P. de Roever: Real-time programming and asynchronous message-passing. In: Proceedings of the 2nd ACM Symposium Principles of Distributed Computing, 1983, 187-197

[Kurshan 89]
R. P. Krushan: Analysis of descrete event coordination. In: J. W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): Stepwise Refinement of distributed Systems. Lecture Notes In Computer Science 430, 1989, 414-453

[Lamport 83]
L. Lamport: Specifying concurrent program modules. ACM Toplas 5:2, April 1983, 190-222

[Lewis 90]
H. R. Lewis: A logic of concrete time intervals. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science , 1990, 280-289

[Lynch 89]
N. A. Lynch: Multivalued possibilities mappings. In: J. W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.): Stepwise Refinement of Distributed Systems. Lecture Notes in Computer Science 430, 1989

[Lynch, Stark 89]
N. Lynch, E. Stark: A proof of the Kahn principle for input/output automata. Information and Computation 82, 1989, 81-92

[Lynch, Tuttle 87]
N. A. Lynch, M. R. Tuttle: Hierarchical correctness proofs for distributed algorithms. In: Proceedings of the Sixth ACM Symposium on Principles of Distributed Computing, 1987

[Nicollin et al. 90]
X. Nicollin, J.-L. Richier, J, Sifakis, J. Voiron: ATP: an algebra for timed processes. In Proceedings IFIP Working Group Conference on Programming Concepts and Methods, 1990, 402-429

[Ostroff 89a]
J. Ostroff: Temporal logic for real-time systems. Advanced Software Development Series. Research Studies Press, 1989

[Ostroff 89b]
J. S. Ostroff: Automated verification of timed transition models. In: J. Sifakis (ed.): Automatic Verification Methods for Finite State Systems. International Workshop, Grenoble, France, Lecture Notes in Computer Science 407, 1989, 247-256

[Park 80]
D. Park: On the semantics of fair parallelism. In: D. Björner (ed.): Abstract Software Specification. Lecture Notes in Computer Science 86, Berlin-Heidelberg-New York: Springer 1980, 504-526

[Schieder 91]
B. Schieder: Private Communication, 1991

[Schneider 87]
F.B. Schneider: Decomposing properties into safety and liveness using predicate logic. Cornell University, Department of Computer Science, Technical Report 87-874, October 1987

[Schneider et al. 91]
F.B. Schneider, B. Bloom, K. Marzullo: Putting time into proof outlines. Unpublished manuscript

[Reed 89]
G.M. Reed: A hierarchy of domains for real-time distributed computing. In Mathematical Foundations of Programming Semantics. LNCS 442, Springer-Verlag, 1989, 80-128

[Reed, Roscoe 86]
G. Reed, A. Roscoe: A timed model for Communicating Sequential Processes. In Proceedings of ICALP ´86, LNCS 226, Springer-Verlag, 1986, 314-323

[Shankar, Lam 87]
A.U. Shankar, S.S. Lam: Time-dependent distributed systems: proving safety, liveness and real-time properties. Distributed Computing 2, 1987, 61-79

[Yi 90]
Wang Yi: Real-time behaviour of asynchronous agents. In: CONCUR ´90. LNCS 458, Springer-Verlag, 1990, 502-520

[Zwarico, Lee 85]
A. Zwarico, I. Lee: Proving a network of real-time processes correct. In: Proceedings IEEE Real-Time Systems Symposium, 1985, 169-177