European Review, Vol. 18, No. 4, 507–563 © 2010 Academia Europæa
doi:10.1017/S1062798710000281

Relating Time and Causality in Interactive Distributed Systems

MANFRED BROY

Institut für Informatik, Technische Universität München, D-80290 München, Germany. E-mail: broy@in.tum.de

For digital interactive distributed systems, the timing of their events and the causality between their events are key issues. In real-time applications of embedded software systems, timing properties are essential, such as the response times of reactions depending on the precise timing of the input events. In a highly abstract view, a digital system can be represented by a set of events annotated by their timing. Sets of timed events labeled by actions represent the observations about systems. An essential property that helps to understand distributed interactive systems and a way to reason about their event flow is *causality*. Causality addresses the questions under which conditions certain events must, may or must not happen. Causality addresses the logical dependencies between the events in systems. Strictly speaking, causality reflects the logical essence in the event and action flow of systems. Causality is closely related to time. In particular, we study in the following the relationship between causality and the timing of input and output events, as well as its relationship to the granularity of time. We deal, in particular, with the problem of time abstraction and the precise timing of events. We show how causality and time form the basis of inductive reasoning, in particular in the case of dependencies in communication cycles ('feedback') and how we can work with time in models of distributed systems with a flexible choice of local clocks and local timing.

1. Time and causality

We study the information flow in relation to the time flow in a system of events and components in the following. Information flow evolves through events in physical or technical systems within a global time frame.

There are many ways to model and represent time, events, and information and the way in which they evolve in physical or technical systems. Even more, there are many ways to interpret certain phenomena related to events carrying or

disseminating information inside a system or over the system boundaries between a system and its environment.

Since we consider models of systems that are the result of abstractions we are very critical not to confuse observations in models with observations about real-world systems. Only if the abstractions are chosen carefully do the models reflect properties of real world systems faithfully.

In this paper, we study issues of timing and causality of discrete event systems and how they are related. The behavior of systems is modeled by sets of events. Causality addresses the logical dependencies between the events of systems and thus the relationship between their cause and their effect. It refers in philosophy to the principle that nothing can happen without causes. In distributed systems, it addresses the logical dependencies between events, related actions and messages generated by the different parts (often called 'components') of systems.

For realistic models of distributed interactive systems in terms of their flow of events, data or signals as found in interactive distributed systems, we assume that there is a source and a cause for each (communication) event and its transmitted information. A reasonable way to model such systems is sets of communication events with a causality relation and a timing relation where each event consists of sending or receiving a message by one of the components of the system. This approach models the behavior of systems in terms of their communication and event flows.

We are interested here basically in a number of fundamental questions of system modeling and design, such as:

- How can the principle of causality be formalized?
- Are there universal laws of causality and what are they?
- How do causality and time relate?
- How does causality help to reason about systems?
- How does time help to reason about systems and their causality?

We first discuss notions of time and causality independently and then study their relationship.

1.1. Discrete event systems

In physics, system behaviors are often modeled by continuous functions depending on time. There, the dynamics of a system is captured by a set of variables that change their values continuously over time. Dependencies between these variables are captured by continuous functionals and expressed by formulas of differential calculus and integration theory, where time is represented by real numbers. The values of the continuous functions represent the states of the system at the corresponding points in time.

Digital event systems provide a more abstract logical model of technical or economical systems than do continuous functions. Discrete steps of the systems

modeled by events capture the dynamics of digital systems and discrete state changes. A discrete event is a quite general notion. An event may represent the execution of a certain statement, and thus a state change − such as the change of the value of some system state attribute, a certain system predicate becoming true, the arrival or the sending of a certain message, or reaching a certain time boundary.

A digital system shows a behavior that can be modeled by a set of digital processes. A digital process is a finite or even an infinite set of discrete events. These events are causally dependent and take place within a time frame. The time model may be continuous or discrete. In the following, we introduce a formal model taking these considerations into account.

1.2. Time

Timing properties of systems can be classified into quantitative and qualitative aspects. The quantitative aspects aim at measuring time and talk about time distances between discrete events, leading to questions such as:

- How much time does it take until event A happens?
- How much time does it take until event B happens after event A had happened?

If we are not interested in measures of time distances between events, as in hard real-time applications, there remain questions addressing the qualitative nature of timing properties, expressed by the after/before relation. Given such relations we can ask questions such as:

- Does event A happen after event B?
- Do events A and B happen simultaneously?
- Does event A always happen only after event B?

Qualitative timing relates events in a partial or even linear order while quantitative time refers to measures and distances between events in terms of time.

In this section, we introduce an abstract model of time. For our purpose a simple model of time is sufficient. Time is represented by a set called TIME with a linear order $\leqslant$.

For each process formed by a set of events E their timing is a mapping

$$\text{time} : \text{E} \rightarrow \text{TIME}$$

If such a mapping is defined for the events of a process we call the process a *timed* process.

In timed processes, we can relate events by their timing. In our approach each event happens at a certain point in time. In other words, we assume that an event represents a point in time and thus has no time duration. If we want to model

system activities with some time duration we can do that by referring to two events associated with the activity, the beginning of the activity and its termination.

For two events, *e1* and *e2*, in a timed process we say *e1* takes place before *e2* if time(*e1*) < time(*e2*) holds, and then we write $e1 < e2$, and we say that the events *e1* and *e2* are *simultaneous*, if time(*e1*) = time(*e2*). This gives a simple but quite useful notion for the timing of events.

We carefully distinguish between the timing of events, which gives us 'absolute' values in a time frame, and values measuring the duration of time. Given two events *e1* and *e2* we can ask about the duration of time between the occurrence of event *e1* and event *e2*. The time duration is not a member of the set TIME but of another set called DURATION, which is an additive semi-group with a zero element.

For the set DURATION we assume an additive structure (which does not make sense for time points). More precisely we assume an operation

$$+ : \text{DURATION} \times \text{DURATION} \to \text{DURATION}$$

as well as a mapping

$$\text{dur} : \text{TIME} \times \text{TIME} \to \text{DURATION}$$

This function maps two time points $t1, t2 \in$ TIME with $t1 \leqslant t2$ onto a duration dur($t1$, $t2$) $\in$ DURATION which represents the time period that it takes until the second time point is reached from the first time point. We can also introduce negative time durations by considering dur(*e2*, *e1*) where $e1 < e2$.

We, moreover, assume that there is a null element $0 \in$ DURATION and that

$$\text{dur}(t, t) = 0$$

for all time points $t \in$ TIME and

$$0 + d = d + 0 = d$$

for all durations d.

The operation dur can easily be extended to events *e1*, *e2* given a timing:

$$\text{dur}(e1, e2) = \text{dur}(\text{time}(e1), \text{time}(e2))$$

This function introduces a relationship between the events and the duration of time between them. For all simultaneous events their time is identical and thus the duration of time between the two events is zero.

Also on durations we assume a linear ordering $\leqslant$ which faithfully reflects the timing, such that the following law holds:

$$e1 \leqslant e2 \leqslant e3 \Rightarrow \text{dur}(e1, e2) \leqslant \text{dur}(e1, e3)$$

The ordering on the set of durations is assumed to be consistent with the addition operator. In particular, we assume that the operator + is monotonic

$$d1 \leqslant d1' \wedge d2 \leqslant d2' \Rightarrow d1 + d2 \leqslant d1' + d2'$$

So far, and in the following, we assume that time is linear (one-dimensional). This seems to be obvious. However, when selecting the model of time, it is

worthwhile considering the difference between a linear (one-dimensional) space of time and time spaces where time might have several dimensions. We consider only 'one-dimensional', linear time in the following.

A one-dimension linear space of time is characterized by the law

$$\text{time}(e1) \leqslant \text{time}(e2) \leqslant \text{time}(e3) \Rightarrow$$
$$\text{dur}(e1, e3) = \text{dur}(e1, e2) + \text{dur}(e2, e3)$$

or expressed in terms of time points $t1$, $t2$, $t3 \in$ TIME:

$$t1 \leqslant t2 \leqslant t3 \Rightarrow$$
$$\text{dur}(t1, t3) = \text{dur}(t1, t2) + \text{dur}(t2, t3)$$

Duration allows us to refer to the relative timing of different time points. So we can ask for time points $t1$, $t2$, $t3 \in$ TIME with $t1 \leqslant t2 \leqslant t3$ whether

$$\text{dur}(t1, t2) < \text{dur}(t2, t3)$$

A further question concerns the model of time. Basically there are two essentially different classes of models of time, *discrete* (*digital*) time and *continuous, dense* (*analog*) time.

For each set of time points we assume the existence of a least upper bound (lub) and a greatest lower bound (glb). Examples of time domains are $\mathbb{N} \cup \{\infty\}$ the natural numbers $\mathbb{N}$, including ∞ for infinity, as well as the non-negative real numbers, including infinity $\mathbb{R}_+ \cup \{\infty\}$ where $\mathbb{R}_+ = \{r \in \mathbb{R}: 0 \leqslant r\}$.

In a discrete model we find time points t1, t2 $\in$ TIME with $t1 < t2$ which are 'direct neighbors' (that cannot be strictly separated by a third time point that lies in between both points), such that for all t3 $\in$ TIME:

$$t1 \leqslant t3 \leqslant t2 \Rightarrow t1 = t3 \vee t2 = t3$$

Thus, each time point (except the maximal one, if it exists) has a unique successor. This property of discreteness does not hold for dense time. In dense time we can always find a time point separating two given different time points. This density of time results in Zenon's problem, where there is an infinite sequence of time points that step by step increase in time but do not reach or go beyond a particular point in a finite number of steps. For an extensive discussion on ways to represent digital and analog time see Ref. 1.

1.3. Discrete timed processes

Based on the idea of timed events, the concept of a timed process is quite straightforward. Let E be a set of timed events; then each subset $P \subseteq E$ defines a discrete timed process. If the set of events is finite then P is called a *finite* process; otherwise it is called *infinite*.

A timed process P has a start time specified by the time value

$$\text{glb } \{\text{time}(e) : e \in P\}$$

As its termination time we define

$$\text{lub}\{\text{time}(e)\colon e \in P\}$$

that may be infinite. A process has also a duration

$$\text{lub}\,\{\text{dur}(e, e')\colon e, e' \in P\}$$

and a structure in terms of a partial order according to the relation $e \leqslant e'$ for events e, $e' \in P$ provided $\text{time}(e) < \text{time}(e')$ or $e = e'$ hold.

By TPROCESS we denote the set of all timed processes.

1.4. Selecting models of time

As a first critical question for system models we address the nature of time and its models:

- For which engineering tasks, and when is it necessary or at least more convenient, to work with continuous, dense time and when is a model of discrete time good enough or even more appropriate?

We work with discrete time in the following. Discrete time is typical for digital systems that are pulse driven and proceed in discrete time steps. We want to demonstrate in the following that discrete time also allows for time models that are as flexible as continuous time.

We work with the following idea of observations that we can make about a system. We assume that at each time t we observe a finite family of events. In each run of the system we make a sequence of observations, one at each time point. The timing introduces a structure on the events of a process. For each time $t \in$ TIME we define the sub-process (the partial process)

$$\{e \in P\colon\ \text{time}(e) \leqslant t\}$$

that consists of all events that take place until time t. This process is denoted by $P \downarrow t$.

A logical property of a timed process is represented by a predicate

$$Q\colon \text{TPROCESS} \to \mathbb{B}$$

A property Q that holds at time t is a predicate on (finite) processes applied to the set $P \downarrow t$ of events. Each predicate Q of that type applied to a process $P \downarrow t$ is called an observation about a system or about a process until time t.

We write also $Q_t(P)$ for $Q(P \downarrow t)$. We get a kind of temporal logic if we define (for a given process P)

$$\Diamond Q \equiv \exists t \in \text{TIME}\colon Q_t(P)$$
$$\Box Q \equiv \forall t \in \text{TIME}\colon Q_t(P)$$

By these formulas we can introduce the temporal operators in a rather straightforward manner.

An observation (a predicate) Q is called *stable*, if the following proposition

$$Q_t(P) \wedge t < t' \Rightarrow Q_{t'}(P)$$

holds for all processes P and all times t, t'.

1.5. System behaviors

A discrete system may show a set of behaviors represented by processes. In the literature a discrete system is often described by a state transition machine, for instance, or by a set of concurrent cooperating state machines, or by a Petri-Net. When executing such a state machine, actions are carried out that correspond to state transitions. Each instance of an action is an event. Thus, we associate for an *interpreted* process P (generated by the system) an action with each of its events.

Let E_p be the set of events forming a process P. The concept of a discrete interpreted process is defined by a mapping

$$\text{act: } E_p \rightarrow \text{Action}$$

where Action is the set of actions of the system. Each action defines a state change, a timing action or a communication action. This leads to a slightly more sophisticated notion of a system behavior represented by a set of processes where each event is timed and corresponds to an action.

1.6. Causality

Causality is a notion that is more sophisticated than that of time. Time is used to capture straightforward observations about systems. Referring to causality we speak about the rules ('the logics') for the occurrence of events and actions in the executions (processes) of systems.

Timing is an observation about a single process representing a run of a system. Causality speaks about the properties and rules that hold for all processes of a system and therefore deals with the logical properties of the system. Causality addresses the rules and thus the logics of systems.

But what is causality precisely and how is it related to the observations with respect to time?

Causality addresses the logical dependencies between the events and actions in the digital processes of a concurrent interactive system S. Causality actually deals both with liveness and safety in observations:

- *Liveness*: Does observation A about the system S guarantee another observation B to follow later eventually: we say 'in system S observation A leads_to observation B'.
- *Safety*: Does observation B for the system S occur only if another observation A was observed before: we may write 'in system S observation B $\Rightarrow$ observation A'; however, this expression using

implication does not properly express the relation between the two observations stating that B follows only after A, since the causal relationship is expressed rather implicitly this way.

For the concept of causality, the notion of the flow of actions and events in each process of the system is decisive, which reflects implicitly or explicitly a qualitative notion of time − more precisely an ordering of the events according to time.

Example: Airbag

Let us look at a simple example. An airbag in a car is activated only if the crash sensor indicates a crash, and whenever the crash sensor indicates a crash it is activated. Both observations are stable. Therefore, if we do not consider time flow, the two propositions: 'crash sensor indicates crash' (csic) and 'airbag is activated' (aia) are logical equivalent for completed processes.

The asymmetry of these two events with respect to the event flow of a system only gets modeled if we take the time flow into account. This can be done by studying sub-processes of the processes P representing the behaviors of cars. Then we require that

$$\forall t\colon \mathrm{csic}(\mathrm{P} \downarrow t) \Leftarrow \mathrm{aia}(P \downarrow t)$$
$$\forall t\colon \mathrm{csic}(\mathrm{P} \downarrow t) \Rightarrow \exists\, t'\colon t \leqslant t' \wedge \mathrm{aia}(P \downarrow t')$$

Here csic($\mathrm{P}\downarrow t$) stands for 'crash sensor indicates crash' as one of the actions of an event in $\mathrm{P}\downarrow t$ and aia stands for 'airbag is activated' as one of the actions of an event in $\mathrm{P}\downarrow t$.

These are examples of properties addressed by the notion of causality.

This example leads to a delicate question that addresses the difference between causality and logical implication. If we talk about the processes of a system and their sets of events without timing or causality information, we may easily write implications.

Example: Air Bag (continued)

For instance, in the case of an airbag we may write

$$(^*)\ \mathrm{crash\_sensor\_indicates\_crash} \quad \Rightarrow \mathrm{airbag\_is\_activated}$$

This system property expresses that for each (complete) process of the airbag system this implication is valid. This seems fine. We are even tempted to read (*) as follows:

if crash_sensor_indicates_crash, this leads to airbag_is_activated.

This is an assertion that can be appropriately formalized in temporal logic, but as a translation into the statement (*) above it is misleading, however, since it is an over-interpretation of implication. What we have written in (*) is simply logical

implication. For the airbag we also assume the validity of the formula:

$$\text{airbag\_is\_activated} \Rightarrow \text{crash\_sensor\_indicates\_crash}$$

This is an implication like the one before. But now the 'leads-to' interpretation is no longer appropriate. A more accurate interpretation is: 'If the air bag is activated then the crash sensor has indicated a crash before.'

In fact, one way to look at causality is to see it as a principle that can be observed in any form of information processing systems independently of their technical or physical representation. Causality addresses the logical dependencies between the events (and in turn the actions) of a system. Causality is a way to 'understand' a system. Certain events and actions may take place or not (such as input). Other events and actions can be caused by these events and actions (such as output). If a system has no rules of causality it is chaotic. Then all actions and events may occur at all times in a completely unrelated random manner.

Causality addresses both safety and liveness properties of systems. An event A may guarantee an event B to happen later. This relationship is what is called a *liveness property*. An event B may only happen if event A has happened before. This relationship is what is called a *safety property*.

There are basically two aspects of causality between two events (or actions) A and B. If A is causal for B, we may assume that

- A enforces (leads_to) B: this means that whenever event A occurs event B eventually occurs (later),
- B requires A: this means that event B does not occur if event A did not take place before.

Of course, there are many generalizations of the notion of causality beyond these two basic principles.

Example: Air Bag (continued)

For the airbag we obviously get the rules

$$\text{crash\_sensor\_indicates\_crash enforces}$$

$$\text{airbag\_is\_activated}$$

as well as

$$\text{airbag\_is\_acivated requires}$$

$$\text{crash\_sensor\_indicates\_crash.}$$

Here we have a typical example of a strong relation of causality.

Causality can be discussed not only in terms of events but also in terms of system properties. A property Q is called *causal* for a property Q' if, for each system run,

whenever we observe Q at some time t at some time later $t' > t$ we observe Q'. Moreover, whenever we observe Q at some time t' there exists some time with $t < t'$ such that Q holds.

We say 'observation A is *causal* for observation B in system S' if for each process $P \in$ S we have (for all times $t \in$ TIME)

$$\mathrm{A}(P \downarrow t) \Rightarrow \exists\, t' \in \mathrm{TIME}\colon t \leqslant t' \wedge \mathrm{B}(P \downarrow t')$$
$$\mathrm{B}(P \downarrow t) \Rightarrow \exists\, t' \in \mathrm{TIME}\colon t' \leqslant t \wedge \mathrm{A}(P \downarrow t')$$

The first formula is capturing what is called the 'leads-to'-property in temporal logic. It can be expressed by temporal logic as follows:

$$\Box(\mathrm{A} \Rightarrow \Diamond \mathrm{B})$$

If both formulas shown above hold we speak of causality and write

$$\mathrm{A} > \approx > \mathrm{B}$$

From the definition of causality we easily prove the transitivity of the causality relation:

$$\mathrm{A} > \approx > \mathrm{B} \wedge \mathrm{B} > \approx > \mathrm{C} \Rightarrow \mathrm{A} > \approx > \mathrm{C}$$

An observation is called *stable* for a process P, if

$$t \leqslant t' \wedge \mathrm{A}(P \downarrow t) \Rightarrow \mathrm{A}(P \downarrow t')$$

or in terms of temporal logic

$$\Box(\mathrm{A} \Rightarrow \Box \mathrm{A})$$

In other words, if A holds at time t it remains valid from thereon. For stable observations causality is very close to logical equivalence. If A is causal for B and if A and B are stable then for each complete process P system causality boils down to implication:

$$\mathrm{A}(P) \Rightarrow \mathrm{B}(P) \text{ and } \mathrm{B}(P) \Rightarrow \mathrm{A}(P)$$

Here, logically there is no asymmetry between A and B, which indicates that by simple implication the causal relationship between A and B is not modeled appropriately. Causality is reduced to logical equivalence when abstracting from timing. If we include timing the asymmetry between A and B becomes observable. There exist times t such that $\mathrm{A}(P \downarrow t)$ holds but not $\mathrm{B}(P \downarrow t)$. We get for all times t (recall that we assume that A and B are stable)

$$\mathrm{B}(P \downarrow t) \Rightarrow \mathrm{A}(P \downarrow t)$$

and

$$\mathrm{A}(P \downarrow t) \Rightarrow \exists t'\colon t \leqslant t' \wedge \mathrm{B}(P \downarrow t)$$

This shows that if A is causal for B we get B implies A which sounds counterintuitive since sometimes 'implies' may be confused with 'leads to'.

A sophisticated question asks about the very nature of causality. If we assume in systems that there are mechanisms ('algorithms') that enforce that certain

actions are executed by some control flow, this gives us a very concrete ('operational') idea of causality. On the other hand, we may develop some idea about causality only by observing systems and their generated actions and try to conclude causality relations from that.

Actually there are, along these lines, two ways to look at the causality of a system:

- in an *intentional* approach ('glass box view') we study the internal structure and mechanisms of a system to recognize that certain events A are causal for certain events B due to the technical mechanisms (the 'algorithmics') that control the behavior and the flow of actions and events of the system.
- in an *extensional* approach ('black box view') to causality we study only observations about a system in terms of its events and actions and their temporal relationships without any knowledge of the internal structure and mechanisms of the system. In the set of all observations we may recognize certain regularities that we then call causal dependencies.

In the first case we can speak about the causality within a single process or a single instance. In the second case we speak about the causality within the system considering all its processes. Both approaches lead to concepts and notions of causality. Extensional causality can be seen as an abstraction of intentional causality. By this abstraction some intentional causality may get hidden. We get a universal notion of extensional ('observable') causality for a system S modeled by its set of timed processes.

A simple example that illustrates this distinction between intentional and extensional is a non-deterministic process, where we have one run that first shows the action d that leads, due to its underlying mechanism, causally to the action b, and a behavior where the system can non-deterministically choose between the actions d or b. Then, intentionally in the first run, there is a causal relationship between d and b, while extensionally for the system this does not hold. Observationally, the intentional causality is hidden by the non-determinism of the system. An extensional causality is only valid if it is intentionally valid for all executions of the system.

If we can only observe runs of a system with the events and their timing we get processes that are sets of events with their timing. From the set of all observations we cannot, in general, derive the intentional causality. Intentional causality allows us to talk about the causality within a single execution represented by a process, while extensional causality considers a system as a whole with all its processes.

In spite of the example above, causality and non-determinism are independent notions. We do not even have to have a notion of non-determinism to introduce the concept of causality and vice versa.

2. Causality and time

Of course, we can also model causality for event structures that model systems by concurrent traces. There, causality is a logical relationship between the events of a system. We work with a universal notion of extensional causality for a system S consisting of a set of timed processes.

Time, or at least a time-based ordering of the events, is an indispensable instrument to capture and measure causality.

If event A is causal for event B then it certainly holds that B happens after A. The reverse does not hold. If in an observation an event A is before an event B this does not mean necessarily that they are causally related. However, if A occurs in all processes (all observations) before B (whenever A occurs) then we may assume a causal relationship between the events A and B.

Note that there are many possibilities how causality is realized operationally. Even time can be a mechanism to establish causality between events. Assume that a system starts a clock under certain conditions such that, after some time, A happens, and after some later time B happens: then A is observably causal for B.

Time is definitely a way to observe causality since time is one way to observe the 'happens before' relation. But the ability to observe this relationship requires that the time granularity is fine enough.

We call a system model *strongly causal* if all events that are intentionally causal to each other are strictly separated by their timing; more precisely the time scale is fine enough that whenever an event e1 is causal for an event e2 then we have

$$\text{time}(e1) < \text{time}(e2)$$

Of course, in any case, we assume for causal events

$$\text{time}(e1) \leqslant \text{time}(e2)$$

We speak of *weak causality* if this property at least holds for a system model.

Weak causality is what we at least expect for any faithful model of a real-world system. Only if we include 'incorrect' observations does the 'law of weak causality' not hold any longer.

2.1. What is a nondeterministic system

In the literature, there is a careful distinction between deterministic and non-deterministic systems. When looking at concrete system models, this distinction seems obvious, at a first sight. In fact, however, it is often not so clear how to give a definition of what it means that a system is 'deterministic' or 'non-deterministic'. Often, abstractions turn deterministic system models into non-deterministic ones and vice versa.

The notion of non-determinism is inherently related to the notion of input to a system. If we consider systems without input from the outside, either the system

has only one behavior (one run or one process) or it has to be considered as being non-deterministic. Input to a system comes from its environment and cannot be controlled by the system itself. Systems with a variety of choices for input are not considered non-deterministic if several processes depending on different input exist. Only if the system shows at least two different processes for the same input events do we call it non-deterministic.

We take here a very fundamental point of view. For us, a system is non-deterministic if its behavior includes choices that cannot be controlled by input from the outside but do influence its visible observable behavior.

2.2. Causality between input and output

For a system, there are events that are only internal and others that affect or are affected by the environment. Typical events of the second class are *input* and *output*. If we are aiming at an interface view of systems then we study the events at the border of a system, these being events that carry input or output across the system borders.

Of course, we may study models of systems where certain events carry both input and output; however, in the following we deal only with systems that show a clear separation between events providing input and those providing output in their externally visible events. Note that there exist system models (specific instances of so-called 'process algebras') like CSP or CCS (see Refs 2 and 3) with the concept of synchronous communication where certain events are 'shared' between systems and their environments and thus may carry both input and output. Then there is no straightforward clear notion of causality between input and output.

We study a specific form of causality for systems in the following, namely the causality between their input and output events. We work under following hypothesis:

- there is a canonical notion of extensional causality between input and output.

This idea of causality captures the essential relationship between input and output. For a system we assume that we have a clear notion of input, which means input events can be chosen arbitrarily by the environment (and must be accepted by the system). The input events form a sub-process called the input process. The system may or must react to such an input by an output. This manifests the fundamental principle of causality between input and output. Of course, in addition, there may be a causal relationship between the output events observable, which manifests some additional logics of the system.

In a deterministic system for each input process, exactly one output process is generated. Therefore, there is a fixed causality between the output events given

some input process. In a non-deterministic system, several output processes may be possible for a given input.

Extensional causality is observed by studying all possible system processes or traces to find out whether an event is always present and guaranteed only after another event. This also applies for the causality between input and output. Such observations about the flow of events and their relationships are closely related to a model of time.

We work in the following with the simplifying assumption that we do not know anything about or at least do not take advantage of any causality in the input history. We are mainly interested in studying causality between input and output and the causality in the output history. This in turn also leads to a notion of causality in the output.

2.3. Information flow and causality

Whenever two subsystems of a system interact, and in this way mutually influence each other's behavior within the context of the larger system, this requires a form of information exchange between the subsystems. In general, the information flows in both directions. This leads to a fundamental question:

- Can we always model information exchange as a directed activity (information flow) with an explicit sender and one or several receivers (modeling mutual information exchange as two steps of directed information exchange)? The events of information exchange are then the output of one system and the input to the other.

The basic principle of causality in the information flow of components with input and output is as simple as the following axiom (here we assume that a system cannot predict its future input and thus there is no 'anticipating logical gate' – no logical gate that can predict its future input and thus produce a corresponding output before the input actually arrived; therefore, in real life systems, an output depending on such a future input can be produced only after that input had been received):

- Information can only be evaluated, processed and forwarded as soon as/after it has been received.

This hypothesis leads to essential questions about the relationship between time, granularity, and causality. If the time granularity is fine enough there is always a delta, the 'reaction time', between the timing of the input and the output caused by it. If the time granularity is not fine enough, this does not hold, in general. In this case, input and output might occur in the same time interval and thus have the same time stamp.

2.4. Time granularity and causality

In this section we study changes and transformations of the time granularity of a system and its consequences. We are, in particular, interested in time abstractions that make the time scale coarser. Given a timed process P with the event set E and the timing function

$$\text{time: E} \rightarrow \text{TIME}$$

we can change ('transform') the timing of the process P by a function

$$\text{trans: TIME} \rightarrow \text{TIME}'$$

where we assume the following monotonicity property for all times $t1, t2 \in$ TIME:

$$t1 \leqslant t2 \Rightarrow \text{trans}(t1) \leqslant \text{trans}(t2)$$

Given a time transformation function we get a new timing for process P by the function

$$\text{time}'\text{: E} \rightarrow \text{TIME}'$$

specified by the formula (for all events $e \in$ E)

$$\text{time}'(e) = \text{trans}(\text{time(e)})$$

As a result of a time transformation, the new timing may be coarser. Events *e1* and *e2* with the timing property $\text{time}(e1) < \text{time}(e2)$ may become simultaneous events under time'. In other words, we may get $\text{time}'(e1) = \text{time}'(e2)$. We speak of a *time coarsening* in this case.

We discuss in the remainder of this paper the notions of strong and of weak causality. Given strong causality, the time model strictly separates all events that are causal for other events from those for which they are causal. If we coarsen the timing of a process we may map a strongly causal process onto a process that is no longer strongly but only weakly causal.

We are interested to study in the following the effects of the time granularity and its coarsening onto the notion of causality. We study systems and their models under several time granularities.

Basically, for systems and their models we assume the following principles:

- For the time granularity we find:
  - If the time scale of the input events is chosen fine enough then there is no non-determinism/underleft in the system model that is due to missing information about the timing of the input.
  - If the time scale is finer than the minimal delay of reactions to events, then causal events are separated by time and the behavior is *strongly causal*.
  - Every system behavior is weakly causal in appropriately constructed system models. Strong causality, however, may be abstracted away due to time models that are too coarse and hence do not separate some events that are in the causality relation.

- Every system behavior is 'physically' strongly causal. This means that even for a given weakly causal model of a real-world system there exists a strongly causal behavior model such that the given weakly causal system behavior model is a time coarsening of this strongly causal model.

If we choose the time granularity not fine enough then we get a system behavior that is only weakly causal, although implicitly strongly causal since it is an abstraction of a strongly causal system.

3. Streams and stream processing systems

In this section we briefly introduce a simple, but very fundamental model of systems called FOCUS (see Ref. 4). It is based on streams representing interaction histories.

3.1. Streams

A *stream* is a finite or infinite sequence of elements. In interactive systems streams are built over sets of messages, signals, or actions. Streams are used in that way to represent communication histories for sequential communication devices such as channels or for sequential histories of activities.

Let M be a given set of messages. A stream over the set M is a finite or an infinite sequence of elements from M.

We use the following notation:

M^* denotes the set of finite sequences over M including the empty sequence $\langle\,\rangle$,

M^∞ denotes the set of infinite sequences over M (that are represented by the total mappings $\mathbb{N}\backslash\{0\} \to M$).

A stream is a member of the set M^ω that is defined by the equation

$$M^\omega = M^* \cup M^\infty$$

We introduce the prefix ordering $\sqsubseteq$ on streams, which is a partial order specified for streams $x, y \in M^\omega$ by the formula

$$x \sqsubseteq y \equiv \exists z \in M^\omega : x^\frown z = y$$

Here, $x^\frown z$ denotes the well-known concatenation of sequences; by concatenation the stream z is appended to the stream x. To extend the concatenation to infinite streams we define: if x is infinite then $x^\frown z = x$.

Throughout this paper, we do not work with this simple concept of a stream as introduced above. Since we want to deal with the timing of systems we find it more appropriate to use so-called timed streams. An infinite timed stream represents an infinite history of communications over a channel or an infinite

history of activities that are carried out sequentially in a discrete time frame. The discrete time frame represents time as an infinite chain of time intervals of equal finite duration. In each time interval a finite number of messages can be communicated or a finite number of actions can be executed. Since we do not assume anything about the speed of communication the sequence of messages communicated within a time interval can be arbitrarily long but it is always finite.

Therefore, we represent a communication history over a sequential communication medium in a system model executed in a discrete time frame by an infinite sequence (a 'stream') of finite sequences of messages or actions. By

$$(M^*)^\infty$$

we denote the set of timed streams. Note that the elements of $(M^*)^\infty$ are infinite streams of finite sequences.

The idea of a timed stream reflects directly the concept of an observation in a discrete time scale. For every time $t \in \mathbb{N}$ and every stream $s \in (M^*)^\infty$ the prefix of s of length t (which is a sequence of length t of finite sequences from M*) reflects the observation until time t.

For a function f we often write $f.z$ for function applications instead of $f(z)$ to avoid unnecessary brackets.

Throughout this paper we work with a couple of simple basic operators and notations for streams that are summarized below:

$\langle\,\rangle$	empty sequence or empty stream,
$\langle m \rangle$	one-element sequence containing m as its only element,
$x.t$	tth element of the stream x, which is a sequence in the case of a timed stream
$\#x$	length of the stream x,
$x\hat{}z$	concatenation of the sequence x to the sequence or stream z,
$x \downarrow t$	prefix of length t of the stream x (which is a sequence with t elements; in the case of a timed stream, a sequence with t sequences as elements), provided x has at least t elements (otherwise $x \downarrow t = x$),
$S©x$	stream obtained from x by deleting all its messages that are not elements of the set S.

For a timed stream we denote by

$\bar{x}$	the finite or infinite stream that is the result of concatenating all sequences in the timed stream x. Note that $\bar{x}$ is finite if x carries only a finite number of non-empty sequences.

A timed stream $x \in (M^*)^\infty$ carries the information at which times which messages are transmitted. As long as the timing is not relevant for a system it does not matter if a message is transmitted somewhat later (scheduling messages

earlier may make a difference with respect to causality – see later). To take care of this we introduce the concept of a 'delay closure'.

For a timed stream $x \in (M^*)^\infty$ we define the delay closure of x by the set $x\uparrow$ of timed streams that carry the same stream of messages but perhaps with some additional time delays as follows:

$$x\uparrow = \{x' \in (M^*)^\infty : \forall t \in \mathbb{N}: \overline{x' \downarrow t} \sqsubseteq \overline{x \downarrow t} \wedge \bar{x} = \overline{x'}\}$$

Obviously, we have $x \in x\uparrow$ and for each $x' \in x\uparrow$ we have

$$x'\uparrow \subseteq x\uparrow$$

as well as

$$\bar{x} = \overline{x'}$$

The set $s\uparrow$ is called the *delay closure* for the stream s. The delay closure is easily extended from streams to sets of streams by pointwise application (let $S \subseteq (M^*)^\infty$)

$$S\uparrow = \bigcup_{s \in S} s\uparrow$$

Throughout this paper, we use streams exclusively to model the communication histories of sequential communication media that we call *channels*. In general, in a system, many communication streams occur. Therefore, we work with channels to name the individual communication streams. Accordingly, in FOCUS, a channel is simply an identifier in a system that identifies a communication line and in every execution of the system it evaluates a stream of messages communicated over that line.

3.2. *Components: syntactic and semantic interfaces*

In this section we introduce a mathematical notion of components and their interfaces. Components interact with their environment via channels. A channel is a communication link and is uniquely identified by a channel identifier.

3.2.1. I/O-behaviors. Types (or sorts) are useful concepts to describe interfaces. We work with a simple notion of types where each type represents a set of data elements. These data elements are used as messages or as values of state attributes.

Let a set S of types of messages be given. By M where

$$M = \bigcup_{s \in S} s$$

we denote the set (the 'universe') of all data messages.

In FOCUS a typed channel is an identifier for a sequential directed communication link for transmitting messages of that type. By C we denote a typed channel set. We assume that a type assignment for the channels in the set C is given by the mapping:

$$\text{type: } C \to S$$

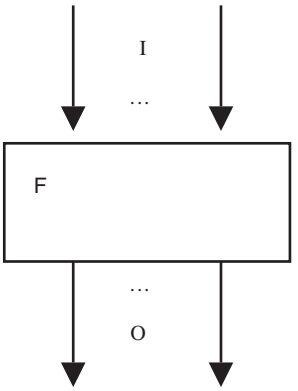

Figure 1. Graphical Representation of a Component F with the Set of Input Channels I and the Set of Output Channels O.

Given a set C of typed channels, a *channel valuation* is an element of the set $\vec{C}$ defined as follows:

$$\vec{C} = \{x\colon C \to (M^*)^\infty\colon \forall c \in C\colon x.c \in (\text{type}(c)^*)^\infty\}$$

A channel valuation $x \in \vec{C}$ associates a stream of elements of type type(c) with each channel $c \in C$. In this way the channel valuation x defines a communication history for each of the channels in the set C. The operators on streams induce operators on channel valuations and furthermore on sets of streams as well as sets of channel valuations by pointwise application. In this way all our operators introduced on streams generalize to channel valuations.

Given a set of typed input channels I and a set of typed output channels O we introduce the notion of a *syntactic interface* of a component:

(I, O)	syntactic interface,
I	set of typed input channels and,
O	set of typed output channels.

A graphical representation of a component and its interface as a data flow node is shown in Figure 1. We do not require that the channel sets I and O are disjoint. If a channel c occurs both in I and O it denotes two different channels: one channel in the set of input channels and one channel in the set of input channels.

For a component-oriented approach to system development in addition to the syntactic interface, a concept for describing the behavior of a component is needed. We work with a simple and straightforward notion of a behavior. Relations between input histories and output histories represent behaviors of systems. Input histories are represented by valuations of the input channels and output histories are represented by the valuations of the output channels.

In FOCUS, we represent the black box or interface behavior of systems by set-valued functions (we also speak of the semantic or behavioral interface of the component):

$$F: \vec{I} \to \wp(\vec{O})$$

As is well known, such a set-valued function is isomorphic to a relation that is a subset of $\vec{I} \times \vec{O}$. We prefer set-valued functions in the case of system behaviors to emphasize the different roles of input and output. We call the function F an I/O-behavior. Given an input history $x \in \vec{I}$, F.x denotes the set of all output histories. A system with behavior F may exhibit anyone of these in reaction to the input x.

An I/O-behavior F is called deterministic, if F.x is a one-element set for each $x \in \vec{I}$. A deterministic I/O-behavior represents a function $\vec{I} \to \vec{O}$.

It is well known that a naive modeling of the behavior of systems by relations on streams leads into the so-called merge anomaly (also called Brock-Ackermann anomaly, see Ref. 6). In FOCUS, this anomaly is avoided by the notion of strong causality (see later).

3.3. Specification of I/O-behaviors

An I/O-behavior represents a model of the behavior of a system. Using logical means, an I/O-behavior F can be described by a logical formula, called a specifying assertion relating the streams on the input channels to the streams on the output channels. In such a formula, channel identifiers occur syntactically as identifiers (variables) for streams of the respective type. The specifying formulas are interpreted in the standard way of typed higher-order predicate logic (see Ref. 7).

An abstract specification of a system provides the following information:

- its syntactic interface, describing the input and output channels by which the system interacts with its environment,
- its behavior by a specifying formula Φ relating input and output channel valuations.

This leads to a specification technique for systems (see Ref. 4 for lots of examples). In FOCUS we specify a system by a scheme of the following form:

⟨ name ⟩

in ⟨ input channels ⟩ **out** ⟨ output channels ⟩
⟨ specifying formula ⟩

The shape of the scheme is inspired by well-known specification approaches like Z (see Ref. 8).

Example. Transmission, merge and fork

As simple but quite fundamental examples of systems we specify a merge component MRG, a transmission component TMC, and a fork component FRK. In the examples, let T1, T2, and T3 be types (recall that in our case types are simply sets) where T1 and T2 are assumed to be disjoint and T3 is the union of the sets of elements of type T1 and T2. The specification of the merge component MRG (actually the specification relies on the fact that T1 and T2 are disjoint, which should be made explicit in the specification in a more sophisticated specification approach) reads as follows:

MRG

in x: T1, y: T2 **out** z: T3
$\bar{x} = \mathrm{T1}\copyright\, z \wedge \bar{y} = \mathrm{T2}\copyright\, \bar{z}$

In this specification we do not specify the quantitative timing (since the output does not refer to or depend on the timing of the input of the output streams) and therefore we refer only to the time abstractions of the involved streams. The causality of the time and message flow is considered in detail in the following subsection.

We specify the proposition $x \sim y$ for timed streams x and y of arbitrary type T; $x \sim y$ is true if the messages in x are a permutation of the messages in y. Formally, we define this operator by the following logical equivalence relation:

$$x \sim y \equiv (\forall m \in \mathrm{T}: \{m\}\copyright\, \bar{x} = \{m\}\copyright\, \bar{y})$$

Based on this definition we specify the component TMC below.

Often it is helpful to use some channel identifiers both for input channels and for output channels. These are then actually two different channels, which, of course, may have different types. To distinguish these channels in the specifying formulas, we use a well-known notational convention. In a specification, it is sometime convenient to use the same channel name for an input as well as for an output channel. Since these are different channels with identical names we have to distinguish them in the body of a specification. Hence, in the body of a specification, we write, for a channel c − which occurs both as an input and as an output channel − simply c to denote the stream on the input channel c, and c' to denote the stream on the output channel c. Thus, in the following specification, z is the outside name of the output channel z, and z' is its local name used in the specifying formula.

TMC

in z: T3 **out** z: T3
$z \sim z'$

This simple specification says that for component TMC every input message occurs eventually also as an output message, and vice versa. Nothing is specified about the timing of the messages. In particular, messages may be arbitrarily delayed and overtake each other. Output messages may even be produced earlier than they are received. This paradox is excluded by causality in the following section.

The following component, FRK, is just the 'inversion' of the component MRG.

FRK

in z: T3 **out** x: T1, y: T2
$\bar{x} = \mathrm{T1} \copyright \bar{z}$ $\bar{y} = \mathrm{T2} \copyright \bar{z}$

Note that the merge component MRG as well as the TMC component and the fork component FRK as they are specified here are 'fair'. Every input is eventually processed and reproduced as output.

Based on the specifying formula given in a specification of an I/O-behavior, F, we may prove properties about the function F in classical (higher order) predicate logic.

3.4. Causality and timing in I/O-behaviors

For input/output information processing devices, the notion of causality is crucial. Certain output depends causally on certain input. Causality indicates dependencies between the input and output actions of information exchange of a system. So far I/O-behaviors are nothing but relations represented by set-valued functions. In the following we introduce and discuss the notion of causality for I/O-behaviors.

I/O-behaviors generate their output and consume their input in a global time frame. This time frame is useful to characterize causality between input and output. Output that depends causally on certain input cannot be generated before this input has actually been received.

Let an I/O-behavior

$$\mathrm{F}: \vec{\mathrm{I}} \to \wp(\vec{\mathrm{O}})$$

be given. In the following we define a couple of fundamental notions to characterize specific properties of F that relate to causality and the timing of the input and output messages.

Definition: Weak causality for component behavior

An I/O-behavior F is called *weakly causal* (*properly timed*), if for all times $t \in \mathbb{N}$ the following formula is valid

$$x \downarrow t = z \downarrow t \Rightarrow (\mathrm{F}.x) \downarrow t = (\mathrm{F}.z) \downarrow t$$

F is properly timed if the output in the tth time interval does not depend on input that is received after time t. This ensures that there is a proper time flow for the

component modeled by the behavior function F. This is a property that manifests the essential asymmetry between input and output.

If F is not weakly causal then there exists a time t and input histories x and x' such that $x \downarrow t = x' \downarrow t$ but $(\mathrm{F}.x) \downarrow t \neq (\mathrm{F}.x') \downarrow t$. A difference between the input histories x and x' occurs only after time t, but at time t the reactions of F in terms of its output messages until time t are already different.

Nevertheless, proper timing in terms of weak causality does not exclude instantaneous reaction (see Ref. 9): the output at time t may depend on the input at time t. This may, however, lead into problems with causality if we consider, in addition, delay free feedback loops (such problems also occur in Esterel[9]). To avoid these problems we better strengthen the concept of proper time flow to the notion of strong causality.

Definition: Strong causality for component behavior
An I/O-behavior F is called *strongly causal* (or *time guarded*), if for all times $t \in \mathbb{N}$ (and all input histories x and z) we have

$$x \downarrow t = z \downarrow t \Rightarrow (\mathrm{F}.x) \downarrow t + 1 = (\mathrm{F}.z) \downarrow t+1$$

If F is time guarded then the output in the tth time interval does not depend on input that is received after the $(t-1)$th time interval. Then F is certainly properly timed and, in addition, reacts to input received in the $(t-1)$th time interval, and not before the tth time interval. In this way, causality between input and output is guaranteed and explicitly visible according to the sufficiently fine time granularity.

Definition: Delay by n time units
We write delay(F, n), if F is a behavior with a delay by (at least) n time units. More precisely we define:

$$\mathrm{delay}(\mathrm{F}, n) \equiv [\forall x, z, t\colon x \downarrow t = z \downarrow t \Rightarrow (\mathrm{F}.x) \downarrow t + n = (\mathrm{F}.z) \downarrow t + n]$$

In other words, F is (weakly) causal if delay(F, 0) holds and strongly causal if delay(F, 1) holds.

Obviously we have for all $n, m \in \mathbb{N}$ (the proof is straightforward):

$$n \leqslant m \wedge \mathrm{delay}(\mathrm{F}, m) \Rightarrow \mathrm{delay}(\mathrm{F}, n)$$

If delay (F, ∞) holds then the output does not depend on the input at all.

For a weakly causal component there is always a maximal number $n \in \mathbb{N} \cup \{\infty\}$ such that delay(F, n) holds. This number is called the *guaranteed delay*.

Our concept of delay is easily extended to individual output channels. Of course, there may be different delays and different guaranteed delays valid for the different output channels of an I/O-behavior. We will return to this.

A function $\mathrm{f}\colon \vec{\mathrm{I}} \to \vec{\mathrm{O}}$ is called weakly or strongly causal respectively if the deterministic I/O-behavior F: $\vec{\mathrm{I}} \to \wp(\vec{\mathrm{O}})$ specified by $\mathrm{F}.x = \{\mathrm{f}.x\}$ for all $x \in \vec{\mathrm{I}}$ has the required properties.

By [F] we denote the set of time guarded total functions f: $\vec{I} \to \vec{O}$, with f.$x \in$ F.x for all input histories $x \in \vec{I}$.

A non-deterministic specification F defines the set [F] of total deterministic behaviors. A specification is only meaningful ('consistent') if the set [F] is not empty. This idea leads to the following definition.

Definition: Realizability
An I/O-behavior F is called *realizable*, if [F]$\neq \emptyset$; this means that there exists a time guarded total function f: $\vec{I} \to \vec{O}$ such that

$$\forall x \in \vec{I} : \text{f}.x \in \text{F}.x$$

Given an input $x \in \vec{I}$ an output $y \in \vec{O}$ is called realizable for F with input x, if there is a function f $\in$ [F] such that $y =$ f.x.

A time guarded function f: $\vec{I} \to \vec{O}$ provides a deterministic *strategy* to calculate for every input history x a particular output history y which is correct for F, i.e. such that $y \in$ F.x holds. Every input $x \downarrow t$ until time point t fixes the output until time point $t+1$ and in particular the output at time $t+1$. Actually the function f essentially defines a deterministic automaton with input and output.

If an I/O-behavior F is not realizable there does not exist a state machine that implements it, not even a state machine that implements a refinement of F (for a proof, see Ref. 11).

There are sophisticated examples of behaviors that are strongly causal, but not realizable. Consider for instance the following example of a behavior F: $\vec{I} \to \wp(\vec{O})$ that is not realizable (here the proof of this fact is left to the reader, a proof can be found in Ref. 4):

$$\text{F}.x = \{x' \in \vec{I} : x \neq x'\}$$

Note that this behavior F is strongly causal but not realizable.

Definition: Full realizability
An I/O-behavior F is called *fully realizable*, if it is realizable, and if for all input histories $x \in \vec{I}$:

$$\text{F}.x = \{\text{f}.x : \text{f} \in [\text{F}]\}$$

holds.

Full realizability guarantees that for all output histories there is a strategy (a deterministic state machine implementation) that computes this output history. For fully realizable behavior there is a close relationship between F and the set [[F]]. Both define the behavior of a system. In fact, non-deterministic state machines with input and output are not more powerful or more expressive than sets of deterministic state machines with input and output. An extensive definition of realizability and its relation to computability can be found in Ref. 11.

3.5. Property refinement of interfaces

Property refinement allows us to replace an interface behavior by one having additional properties. In this way, interface behaviors are replaced by a more restricted ones. An interface

$$F : \vec{I} \to \wp(\vec{O})$$

is refined by a behavior

$$\hat{F} : \vec{I} \to \wp(\vec{O})$$

If

$$\forall x \in \vec{I} : \hat{F}(x) \subseteq F(x)$$

We then write

$$\hat{F} \subseteq F$$

Obviously, property refinement is a partial order and therefore reflexive, asymmetric, and transitive. Note that the paradoxical system, with empty sets of output histories for each of its input histories, is logically a refinement for every system with the same syntactic interface.

A property refinement is a basic refinement step, adding requirements as is done step by step in requirements engineering.

3.6. Computations

Our model describes behavior in terms of the relationship between input and output histories. In this section, we show how to associate computations with behaviors. The key idea is that we calculate the output in each time interval step by step from the input given in the previous time interval according to the progress of time.

We show how to calculate stepwise inductively an output history for a given strongly causal realizable behavior F and an input history $x \in \vec{I}$ that is only provided stepwise. We construct the output histories $y \in \vec{O}$ defining $y{\downarrow}t$ iteratively from input $x{\downarrow}t$ for $t = 0, 1, 2, \ldots$ The key idea is that we can select the output $y.t+1$ given $x{\downarrow}t$ without considering $x.t + 1$.

We define a computation for a realizable behavior F for a given input history inductively as follows: we choose a deterministic behavior $f \in [F]$.

We start with $t = 0$; by definition $x{\downarrow}0$ and $y{\downarrow}0$ are both empty histories. Given $x{\downarrow}t$ and $y{\downarrow}t \in (F.x){\downarrow}t$ we construct from $x{\downarrow}t$ and $y{\downarrow}t$ the sequence $y.t + 1$ as follows.

We construct a chain of partial output histories $y_t \in (F.x){\downarrow}t$ such that

$$y_t \in \{(f.x') \downarrow t : x' \downarrow t = x \downarrow t\}$$

The construction works as follows: we start the iteration with the empty history:

$$y_0' c = \langle\,\rangle \quad \forall c \in O$$

Given y_t we construct y_{t+1} by the following rules:

(1) choose some $y' \in \{\mathrm{f}.x' : x' \downarrow t = x \downarrow t\}$
(2) define $y_{t+1} = y_t{}^\frown \langle y'.t+1 \rangle$

Note that by strong causality $y_t = y' \downarrow t$ and thus $y_{t+1} = y' \downarrow t+1$. This shows that our construction essentially relies on the strong causality of f.

If the behavior is not strongly causal, then our construction does not work, in general. Given some input $x \downarrow t$ we may select some output

$$y \downarrow t \in \{y' \downarrow t : \exists x' : y' \in \mathrm{F}.x' \wedge x' \downarrow t = x \downarrow t\}$$

such that

$$y \downarrow t \notin (\mathrm{F}.x) \downarrow t$$

The partial output $y \downarrow t$ therefore cannot be necessarily completed into a complete infinite output history (for details, see Ref. 11).

The same applies if the output y is not realizable. Then a function $\mathrm{f} \in [\mathrm{F}]$ does not exist with $y = \mathrm{f}.x$. If we would choose in step (1) the formula $y' \in \{\mathrm{F}.x' : x' \downarrow t = x \downarrow t\}$ then it is not guaranteed that $y \in \mathrm{F}.t$ holds with this construction. For a behavior function $\mathrm{F} : \vec{\mathrm{I}} \to \wp(\vec{\mathrm{O}})$ there may exist a history $y \in \vec{\mathrm{O}}$ such that for some input history $x \in \vec{\mathrm{I}}$ we have $y \downarrow t \in (\mathrm{F}.x) \downarrow t$ for all t but $y \notin \mathrm{F}.x$ (for details see Ref. 11).

3.7. Fixpoints

To compose two or more components into a system architecture these components are connected by their channels. Some of these connecting channels typically form feedback loops, in general. Feedback corresponds to recursive definitions of the streams in the channel valuations. Recursion on streams is treated, as usual, in terms of 'fixpoints'.

As is well-known, deterministic strongly causal functions always have fixpoints. Strongly causal function are guarded and thus the fixpoints can be 'inductively' defined. The existence proof basically is given by an inductive construction of the fixpoint along the lines of the definition of the concept of a computation as shown above. Thus, realizable strongly causal behaviors always have fixpoints that reflect feasible computations of the system.

Recall, that every strongly causal function

$$\mathrm{f} : \vec{\mathrm{I}} \to \vec{\mathrm{O}}$$

has a unique fixpoint $x = \mathrm{f}(x)$. The proof is quite straightforward by an inductive construction of the fixpoint as in the previous section on computations. Therefore, every realizable function

$$\mathrm{F} : \vec{\mathrm{I}} \to \wp(\vec{\mathrm{O}})$$

has a fixpoint $x \in \mathrm{F}(x)$. (Note that, for a function F that maps histories onto sets of histories, x is called a fixpoint if $x \in \mathrm{F}(x)$.) Moreover, for a fully realizable behavior, every fixpoint $x \in \mathrm{F}(x)$ is a fixpoint of a function $\mathrm{f} \in [\mathrm{F}]$.

Using weakly causal functions we run into difficulties, here, however, to separate fixpoints $x \in F(x)$ that correspond to actual computations from those that do not. In the case of strongly causal, realizable functions, every fixpoint corresponds to a proper computation. For a weakly causal function

$$f : \vec{I} \to \vec{O}$$

we do not even have a guarantee that fixpoints actually exist.

Example: Fixpoints for weakly and strongly causal functions
Consider as an example the following specification:

Succ

in a: $\mathbb{N}$ **out** b: $\mathbb{N}$
$\forall\ t$: $b.t =$ **if** $a.t = \langle\ \rangle$ **then** $\langle 1 \rangle$ **else** succ*$(a.t)$ **fi** **where** $\forall\ n \in \mathbb{N}, s \in \mathbb{N}^*$: succ*$(\langle\ \rangle) = \langle\ \rangle$ $\wedge$ succ*$(\langle n \rangle \hat{}\, s) = \langle n+1 \rangle \hat{}$ succ*(s)

Succ is obviously weakly causal, but not strongly causal since the output at time t depends exclusively on the input at time t. However, Succ does not have a fixpoint $x \in \text{Succ}.x$; this is proved by contradiction as follows; there does not exist a history x and y with

$$y \in \text{Succ}.x$$
$$x.a = y.b$$

since this would immediately result in a contradiction because then for all times $t \in \mathbb{N}$ we would get (with $a.t = b.t$):

$$b.t = \text{if } b.t = \langle\ \rangle \text{ then } \langle 1 \rangle \text{ else succ}^*(b.t) \text{ fi}$$

which is an equation impossible to fulfill since if $b.t = \langle\ \rangle$ then $b.t = \langle 1 \rangle$ would follow as well as $b.t \neq \langle\ \rangle$ and $b.t = \text{succ}^*(b.t)$ cannot hold. If we slightly change the definition by writing $b.t + 1 = \ldots$ instead of $b.t = \ldots$ then the specified behavior is obviously strongly causal and fixpoints do exist. One fixpoint (in fact the only one) is then given by the timed stream on channel b:

$$b.0 = \langle\ \rangle$$
$$b.1 = \langle 1 \rangle$$
$$b.2 = \langle 2 \rangle\ \ldots$$

This example demonstrates the complications when working with only weakly causal in contrast to strongly causal functions and behaviors. Such problems are also encountered in approaches called 'perfect synchrony' as found in Esterel,[9] leading to the well-known problem of certain Esterel programs called causal loops that are without proper computations.

4. Composition operators

In this section we introduce an operator for the composition of components.

We prefer to introduce only one general powerful composition operator and later show how to define a number of other operators as special cases.

4.1. Composing components

Given I/O-behaviors

$$F_1 : \vec{I}_1 \to \wp(\vec{O}_1), \quad F_2 : \vec{I}_2 \to \wp(\vec{O}_2)$$

where the sets of output channels are disjoint

$$O_1 \cap O_2 = \varnothing$$

we define the parallel composition with feedback as it is illustrated in Figure 2 by the I/O-behavior

$$F_1 \otimes F_2 : \vec{I} \to \wp(\vec{O})$$

with a syntactic interface as specified by the equations:

$$I = (I_1 \cup I_2) \backslash (O_1 \cup O_2), O = (O_1 \cup O_2).$$

The resulting function is specified by the following equation (here we assume $z \in \vec{C}$ where the set of all channels C is given by $C = I_1 \cup I_2 \cup O_1 \cup O_2$):

$$(F_1 \otimes F_2).x = \{z|O : z|I = x|I \wedge z|O_1 \in F_1(z|I_1) \wedge z|O_2 \in F_2(z|I_2)\}$$

Here for a channel set $C' \subseteq C$ we denote for $z \in \vec{C}$ by $z|C'$ the restriction of y to the channels in C'. The equation defining composition includes a fixpoint construction for all channels in

$$Z = (I_1 \cup I_2) \cap (O_1 \cup O_2).$$

For these channels for given input history $x \in \vec{I}$ we get, by the notation above, some $y|Z \in \vec{Z}$ which is a fixpoint of the function

$$\lambda z.[(F_1 \otimes F_2).(x \oplus z)]|Z$$

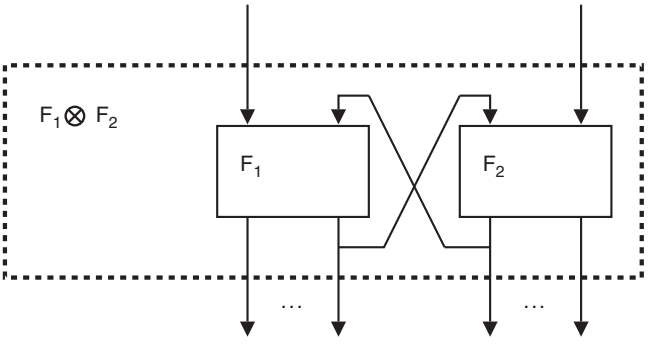

Figure 2. Parallel composition with feedback.

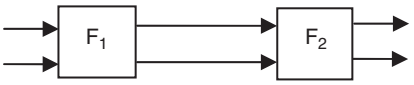

Figure 3. Pipelining.

where, for $x_1 \in \vec{I}_1$, $x_2 \in \vec{I}_2$ where $(I_1 \cap I_2) = \varnothing$

$$x_1 \oplus x_2 \in \overrightarrow{I_1 \cup I_2}$$

is defined as follows:

$$(x_1 \oplus x_2)|I_1 = x_1 \wedge (x_1 \oplus x_2)|I_2 = x_2$$

As long as F_1 and F_2 have disjoint sets of input and output channels the composition is simple. Given $x_1 \in \vec{I}_1$ and $x_2 \in \vec{I}_2$ we get the equation

$$(F_1 \otimes F_2).(x_1 \oplus x_2) = \{y_1 \oplus y_2 : y_1 \in F_1.x_1 \wedge y_2 \in F_2.x_2\}$$

Now assume

$$I_1 = O_1 \text{ and } I_2 = O_2 = \varnothing$$

Note F_2 is then the component without input and output. We write then $\mu.F_1$ for $F_1 \otimes F_2$. We get $I = \emptyset$ ($\mu.F_1$ has no input channels) and

$$\mu.F_1 = \{y : y \in F_1.y\}$$

This somewhat special construction shows once more that composition with feedback loops corresponds to a kind of fixpoint equation. We call $y \in F_1.y$ a fixpoint of F_1. Note in the case of a deterministic function $f_1 : \vec{O}_1 \to \vec{O}_1$ we get $y = f_1.y$.

The operator $\otimes$ is a rather general composition operator that can be easily extended from two components to a family of components.

A more specific operation is sequential composition also called *pipelining*. It is a special case of the composition operator where $O_1 = I_2$ and the sets I_1 and O_2 are disjoint. In this case we define

$$F_1 \circ F_2 = F_1 \otimes F_2$$

where the composition is illustrated by Figure 3.

Pipelining is the special case of composition without feedback. It can easily be generalized to the case where the channel sets I_1 and O_2 are not disjoint. The definition reads as follows

$$(F_1 \circ F_2).x = \{z \in F_2.y : y \in F_1.x\}$$

This composition is also called *relational composition* if F_1 and F_2 are represented as relations or *functional composition* if F_1 and F_2 are deterministic and thus functions.

4.2. Granularity refinement: changing levels of abstraction

In this section we show how to change the levels of abstractions by refinements of the interfaces, state machines and processes. Changing the granularity of interaction and thus the level of abstraction is a classical technique in software system development.

Interaction refinement is the refinement notion for modeling development steps between levels of abstraction. Interaction refinement allows us to change for a component

- the number and names of its input and output channels,
- the types of the messages on its channels determining the granularity of the communication.

An *interaction refinement* is described by a pair of functions

$$A : \vec{C}' \to \wp(\vec{C}) \qquad R: \vec{C} \to \wp(\vec{C}')$$

that relate the interaction on an abstract level with corresponding interaction on the more concrete level. This pair specifies a development step that is leading from one level of abstraction to the other, as illustrated by Figure 4. Given an abstract history $x \in \vec{C}$ each $y \in R(x)$ denotes a concrete history representing x. Calculating a representation for a given abstract history and then its abstraction yields the old abstract history again. Using sequential composition, this is expressed by the requirement:

$$R \circ A = Id$$

Let Id denote the identity relation, and '$\circ$' the sequential composition is defined as follows:

$$(R \circ A)(x) = \{y \in A(z) : z \in R(x)\}$$

A is called the *abstraction* and R is called the *representation*. R and A are called a *refinement pair*. For untimed systems we weaken this requirement by requiring $R \circ A$ to be a property refinement of the untimed identity, formally expressed by the following equation:

$$\overline{(R \circ A)(x)} = \{\vec{x}\}$$

This defines an identity under time abstraction.

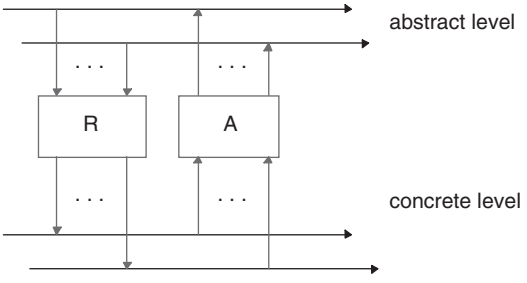

Figure 4. Communication history refinement.

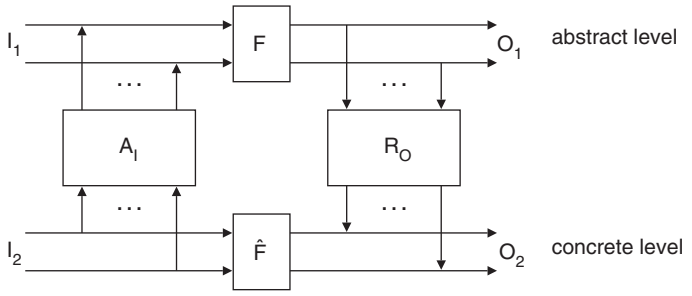

Figure 5. Interaction refinement (U^{-1}-*simulation*).

Interaction refinement allows us to refine systems, given appropriate refinement pairs for their input and output channels. The idea of an interaction refinement is visualized in Figure 5 for the so-called U^{-1}-simulation. Note that here the components (boxes) A_I and A_O are no longer definitional in the sense of specifications, but rather methodological, since they relate two levels of abstraction.

Given refinement pairs

$$A_I : \vec{I}_2 \to \wp(\vec{I}_1) \qquad R_I : \vec{I}_1 \to \wp(\vec{I}_2)$$
$$A_O : \vec{O}_2 \to \wp(\vec{O}_1) \quad R_O : \vec{O}_1 \to \wp(\vec{O}_2)$$

for the input and output channels we are able to relate abstract to concrete channels for the input and for the output. We call the interface

$$\hat{F} : \vec{I}_2 \to \wp(\vec{O}_2)$$

an *interaction refinement* of the I/O-behavior

$$F : \vec{I}_1 \to \wp(\vec{O}_1)$$

if the following proposition holds: $\approx>$

$$A_I \circ F \circ R_O \approx> \hat{F} \quad U^{-1}\text{-}simulation$$

This formula essentially expresses that $\hat{F}$ is a property refinement of the system $A_I \circ F \circ R_O$. Thus, for every 'concrete' input history $\hat{x} \in \vec{I}_2$ every concrete output $\hat{y} \in \vec{O}_2$ can also be obtained by translating $\hat{x}$ onto an abstract input history $x \in A_I \cdot \hat{x}$ such that we can choose an abstract output history $y \in F(x)$ such that $\hat{y} \in R_O(y)$.

5. Variations on time granularity

In this section we show how to change the time granularity in system models and study the effect of those changes. We start by defining operators that change the time scale. Then we study algebraic properties of these operators.

5.1. Changing the time scale

Let $n \in \mathbb{N}$ with $n > 0$ and C be a set of typed channels; to make, for a channel history (or a stream)

$$x \in \vec{C}$$

its time scale coarser by the factor n we introduce the coarsening function

$$\mathrm{COA(n)}\colon \vec{C} \to \vec{C}$$

defined by (for all $t \in \mathbb{N}$):

$$\mathrm{COA}(n)(x).t{+}1 = x.(t^*n + 1)^\wedge \ldots ^\wedge x.(t^*n + n)$$

$\mathrm{COA}(n).x$ yields a history from history x where for each stream associated with a channel the sequences for n successive time intervals are concatenated ('abstracted') into one. In that way we forget about some of the time distributions of x. The time scale is made coarser that way.

Time coarsening, obviously, is an instance of abstraction. We forget some information about the timing of a history. The mapping is not injective. Distinct histories may be mapped onto the same history by time scale coarsening.

It is not difficult to allow even a coarsening factor $n = \infty$ in time coarsening. Then, the infinite number of time intervals in a timed stream is mapped into one. The infinite stream of sequences is concatenated into a non-timed stream. Timed streams are abstracted into non-timed streams in that way if we define:

$$\mathrm{COA}(\infty).x = \bar{x}$$

On histories, coarsening is a function that is not injective and thus there does not exist an inverse.

We generalize the coarsening of the time scale from channel histories to behaviors. We coarsen both the input and the output histories. To make a behavior

$$\mathrm{F}\colon \vec{I} \to \wp(\vec{O})$$

coarser by the factor n, we define the coarsening operator that maps F onto a function

$$\mathrm{COA}(\mathrm{F}, n)\colon \vec{I} \to \wp(\vec{O})$$

which is defined as follows

$$\mathrm{COA}(\mathrm{F}, n)(x) = \{\mathrm{COA}(n).y : \exists x'\colon x = \mathrm{COA}(n).x' \wedge y \in \mathrm{F}(x')\}$$

Coarsening maps I/O-behaviors onto I/O-behaviors. On one hand, coarsening may introduce further non-determinism and underspecification into behaviors due to the coarser time scale of the input histories. Certain different input histories are mapped by the time coarsening onto the same coarser input histories. Then their sets of output histories are defined by the union of all their coarsened output histories. In this way non-determinism may grow.

On the other hand, some non-determinism and underspecification may be removed in behaviors by coarsening since some different output histories may be mapped by the time coarsening onto the same coarser output history.

A special case is the coarsening COA(F, ∞), which abstracts completely away all time information. If the output of F depends on the timing of the input, then the coarsening COA(F, ∞) introduces a lot of non-determinism, in general. However, if the output produced by F does not depend on the timing of the input messages at all but only on their values and the order in which they arrive, COA(F, ∞) will rather be more deterministic.

If F is weakly causal, the behavior COA(F, n) is obviously also weakly causal. However, strong causality is not maintained by coarsening the time scale. We will return to more explicit laws of causality and coarsening later. Reactions to input at later time intervals may be mapped onto one time interval.

We can also map a history as well as a behavior onto a finer time granularity. Let $n \in \mathbb{N}$; to make for a history (or a stream)

$$x \in \vec{C}$$

its time scale finer by the factor n we use the function

$$\text{FINE}(n)\colon \vec{C} \to \wp(\vec{C})$$

defined by the equation:

$$\text{FINE}(n)(x) = \{x' \in \vec{C} : \forall t \in \mathbb{N} : x.t + 1 = x'.(n^* t + 1)^\wedge \ldots ^\wedge x'.(n^* t + n)\}$$

FINE(n).x yields the set of histories where for each time interval the sequences of messages in this interval are arbitrarily subdivided into n sequences that are associated with n successive time intervals. Thus, the sequence on each time interval for each channel is non-deterministically divided into n sequences. The time scale is made finer that way.

Making the time scale finer is a form of concretization in contrast to abstraction. Each history is mapped onto a number of histories by making its time scale finer. Each of these histories represents one version of the history with a finer time granularity.

Another way to define the function FINE is demonstrated by the following formula

$$\text{FINE}(n)(x) = \{x' \colon \text{COA}(n).x' = x\}$$

This equation shows more explicitly the relationship between making the time scale coarser and making the time scale finer. They are 'inverse' operations (for detailed discussion see the following section). Changing the time scale represents an abstraction if we make the time scale coarser, and a concretization if we make it finer.

The idea of making a time scale finer can be applied also to behaviors. We specify

$$\text{FINE}(\text{F}, n)(x) = \{y' \in \text{FINE}(n).y \colon \exists x' \colon x = \text{FINE}(n)(x') \wedge y \in F(x')\}$$

Due to the underspecification that is involved in the way we make the time scale finer, there is no guarantee that we get a higher number of delays in the behaviors when moving to a finer time scale.

Nevertheless, we can introduce an 'artificial' operator DFINE that makes the time scale finer in a way that guarantees a larger delay. This operator is easily defined in a brute force way by defining DFINE as follows for $t \in \mathbb{N}\backslash\{0\}$

$$(\text{DFINE(n).x).t} = \begin{cases} x.(t/n) & \text{if } t \textbf{ mod } n = 0 \\ \langle\,\rangle & \text{otherwise} \end{cases}$$

Then every input is delayed by the factor n.

Such an operator is artificial, however, since it resolves some of the non-determinism introduced by the operator FINE in a brute force way by making the time grain finer in a way that guarantees the largest delay.

Going from a finer to a coarser time scale we generally lose some information about the timing that cannot be recovered properly when making the time scale finer again.

5.2. *Rules for time scale refinement*

Changing the time scale is an operation on histories and behaviors. In this section we study laws and rules for changing the time scale.

Our first rules for changing the time scale show that the functions COA(n) and FINE(n) form refinement pairs in the sense of[4]

$$\text{COA}(n).\text{FINE}(n).x = \{x\}$$
$$x \in \text{FINE}(n).\text{COA}(n).x$$

In other words, coarsening is the inverse of making the time scale finer. The proof of the equation is quite straightforward by the definition of COA and FINE.

We observe, in particular, the following equations (here $F_1 \circ F_2$ denotes the pipeline composition of F_1: $\vec{I}_1 \to \wp(\vec{O}_1)$ and F_2: $\vec{I}_2 \to \wp(\vec{O}_2)$ where $O_1 = I_2$ and $(F_1 \circ F_2).x = \{y: \exists z: z \in F_1.x \wedge y \in F_2.z\}$)

$$\text{COA}(F, n) = \text{FINE}(n) \circ F \circ \text{COA}(n)$$
$$\text{FINE}(F, n) = \text{COA}(n) \circ F \circ \text{FINE}(n)$$

The proof is again quite straightforward. The equations show that time refinement in fact is a special case of interaction granularity refinement (see Ref. 12).

Both time granularity abstractions and refinements by factors $n*m$ can be seen as two consecutive refinements by the factor n followed by a refinement with factor m or vice versa.

Figure 6 shows such an iterative refinement of the time scale.

We get the following obvious rules:

$$\text{FINE}(n * m) = \text{FINE}(n) \circ \text{FINE}(m)$$
$$\text{COA}(n * m) = \text{COA}(n) \circ \text{COA}(m)$$

We are, in particular, interested to analyze how time refinement relates to causality. This relationship is illustrated by the following equation (let $m, n \in \mathbb{N}$

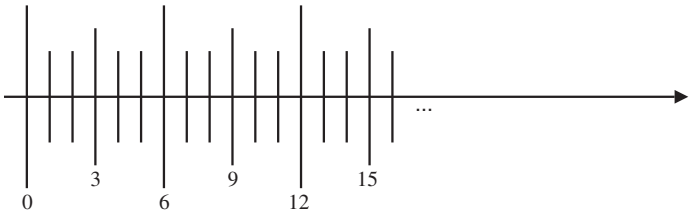

Figure 6. Refinement of the time scale by 6: shown as a refinement by the factor 3 followed by a refinement by the factor 2.

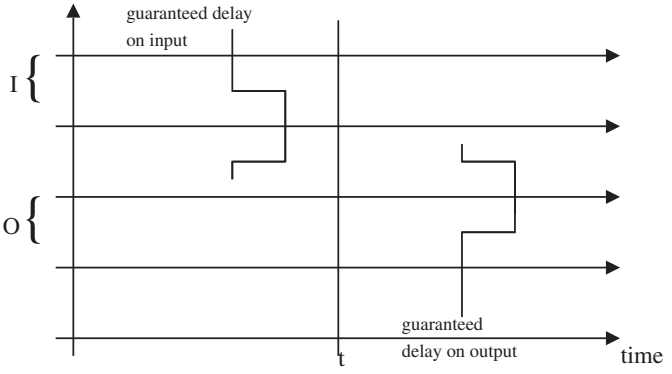

Figure 7. Relationship between input with guaranteed delay and output with guaranteed delay at time t.

with $m \geqslant 1$):

$$\text{delay}(\text{F}, n * m) \Rightarrow \text{delay}(\text{COA}(\text{F}, m), n)$$

The proof of this rule is quite straightforward and uses the fact that

$$(\text{COA}(m).x) \downarrow t = \text{COA}(m).(x \downarrow (t * m))$$

holds.

This rule is, in particular, interesting for our discussion of time, time abstraction, and causality. As long as we coarsen a behavior F for which delay(F, n) holds at most by factor n, causality is still guaranteed for the resulting component.

In fact, there exists a kind of monotonicity of coarsening with respect to delay properties:

$$n < m \wedge \text{delay}(\text{COA}(\text{F}, m), k) \Rightarrow \text{delay}(\text{COA}(\text{F}, n), k)$$

Again the proof is straightforward and left to the reader. If we coarsen the time scale we may, in general, lose strong causality. In other words, the property of strong causality depends on a fine enough granularity of the time scale.

The following equation does not, in general, hold for the operator FINE. It holds only for the version DFINE of FINE that increases the delay.

$$\text{delay}(\text{F}, n) \Rightarrow \text{delay}(\text{DFINE}(\text{F}, m), m^* n)$$

To keep this formula valid we cannot replace the operator DFINE by FINE.

5.3. *Fractions of time changes*

For a given behavior F we can change its timing by fractions m/n for $m, n \in \mathbb{N}$ using the time change operator TCH.

$$\text{TCH}(\text{F}, m/n) = \text{COA}(\text{FINE}(\text{F}, n), m)$$

Note that all the rules introduced so far for COA and FINE obviously carry over to this case, since we expressed TCH in terms of COA and FINE.

5.4. *Choosing the appropriate time scale*

We have discussed how closely the time scale is related to the property of causality. It depends very much on the time scale whether a behavior is strongly causal as well as on the delay properties of the components. In a large system with many components, different time scales may be appropriate for different subsystems. In the remainder of this section we therefore study an idea of flexible timing.

Complex hierarchical distributed systems require a flexible time model such that its time granularity can be adapted individually to the needs of its various subsystems.

This leads to the following idea: we establish and model different time scales for the subsystems of a composed system. Then we can choose the time scales in a flexible way, according to the following observations:

- for each system composed of strongly causal components its time delay is greater than the length of the shortest path of channels through the system of components;
- therefore we can coarsen the interface abstraction of the system by the factor k without losing strong causality provided the shortest path is $\geqslant k$.

This leads to hierarchical system models that support local islands ('subsystems') of finer granularity of time. A system may be composed of many subsystems with their own finer time scales. To discuss this idea in detail we have first to introduce a notion of composition.

5.5. *Causal fixpoints and causal loops*

Given a strongly causal function

$$\text{F}: \vec{\text{C}} \to \vec{\text{C}}$$

each fixpoint $y \in \text{F}.y$ is called causally faithful or, for short, *causal*. In a causal fixpoint, each sequence of values in a time interval t is triggered by the values that occurred in the history before t.

Now we consider a coarsening of the time for the function F

$$\mathrm{COA}(\mathrm{F}, \mathrm{n}) \colon \vec{\mathrm{C}} \to \vec{\mathrm{C}}$$

We get for each fixpoint

$$y \in \mathrm{F}.y$$

that COA(n).y is a fixpoint of COA(F, n), too. But there may be fixpoints COA(F, n) that do not correspond to fixpoints of F.

A fixpoint $y' \in (\mathrm{COA}(\mathrm{F}, n)).y'$ is called *causal w.r.t.* F if there is a fixpoint $y \in \mathrm{F}.y$ such that $y' = \mathrm{COA}(n).y$. Otherwise y' is called a *causal loop*.

We will show that considering COA(F, n) alone without knowing F we cannot distinguish, in general, causal fixpoints from causal loops. In other words, if we choose the time scale too coarse such that the resulting function is not strongly causal we may lose the faithful notion of causality and compositionality and cannot distinguish causal from not causal fixpoints.

Let us consider a simple function

$$\mathrm{IDS} \colon \vec{\mathrm{C}} \to \wp(\vec{\mathrm{C}})$$

with the specification (for all channels $\mathrm{c} \in \mathrm{C}$)

$$(\mathrm{IDS}.x).c = \{\langle\langle\rangle\rangle^{\wedge}(x.c)\}$$

IDS is the identity on the data streams shifted by one time interval.

Now we consider the history

$$x.c = \langle\langle 1\rangle\langle\rangle\rangle^{\infty} \text{ for all } c \in \mathrm{C}$$

we get

$$(\mathrm{IDS}.x).c = \langle\langle\rangle\langle 1\rangle\rangle^{\infty} \text{ for all } c \in \mathrm{C}$$

This shows that x is not a fixpoint of IDS. Now we consider COA(2).IDS. We get:

$$\mathrm{COA}(2).\langle\langle 1\rangle\langle\rangle\rangle^{\infty} = \{\langle\langle 1\rangle\rangle^{\infty}\}$$

$$\mathrm{COA}(2).\langle\langle\rangle\langle 1\rangle\rangle^{\infty} = \{\langle\langle 1\rangle\rangle^{\infty}\}$$

This proves the fixpoint property

$$\langle\langle 1\rangle\rangle^{\infty} \in (\mathrm{COA}(2).\mathrm{IDS}).\langle\langle 1\rangle\rangle^{\infty}$$

So $\langle\langle 1\rangle\rangle^{\infty}$ is a fixpoint of COA(2).IDS. However, this fixpoint is not causal, since $\langle\langle\rangle\rangle^{\infty}$ is the only fixpoint of IDS as well as COA(IDS, 2).

This shows that coarsening a behavior F leads to functions F′ with additional fixpoints that are not causal in the case that F′ is, in contrast to F, not strongly but only weakly causal. Moreover, we cannot distinguish, in general, for these functions between causal and non-causal fixpoints.

5.6. Delay calculus for system architectures

By composition of a family of components we can form a network (representing an architecture) of components with delays. This network is a directed graph with channels as arcs and components as nodes.

With each path in the graph leading from an input channel to an output channel c, we can associate a delay, which is the sum of all delays on that path (each component on the path adds to the delays). For an output channel c the (guaranteed) delay in the system for c is the minimum over the sum of delays on each of the paths from some input channel to the output channel c.

In a behavior function we can define the guaranteed delay between an input channel and an output channel. Consider

$$\mathrm{F}: \vec{\mathrm{I}} \to \wp(\vec{\mathrm{O}})$$

and an output channel $c' \in \mathrm{O}$ and an input channel $c \in \mathrm{I}$. We define the guaranteed delay for the output channel c in F by the following formula (Figure 7 illustrates the relationship between delayed input and guaranteed delay of the output):

$$\begin{aligned}&\mathrm{gardelay}(\mathrm{F}, c') = \\ &\max\{k \in \mathbb{N} \cup \{\infty\} : \forall x, x' \in \vec{\mathrm{I}}, t \in \mathbb{N} : \\ &\quad x \downarrow t = x' \downarrow t \\ &\Rightarrow \ ((\mathrm{F}.x').c) \downarrow t + k = ((\mathrm{F}.x).c') \downarrow t + k\}\end{aligned}$$

In a composed system we define the delay length of a path from an input channel c' to an output channel c as follows:

A path is a sequence of channels $p = \langle c_0\, c_1\, \ldots\, c_k \rangle$ such that for each index $i < k$ there exists a component F_i in the system such that c_i is an input channel of F_i and c_{i+1} is an output channel of that component. c_0 is called the source of p and c_k is called the target of p. By $\mathrm{Path}(c, c')$ we denote the set of all paths with source c and target c'.

The delay length of the path is given by the formula

$$\mathrm{dl}(p) = \sum_{i=1}^{k} \mathrm{gardelay}(\mathrm{F}_i, c_i)$$

For each output channel c' for the system F represented by the considered network we obtain:

$$\mathrm{gardelay}(\mathrm{F}, c') \geqslant \min\{\mathrm{dl}(p) : p \in \mathrm{Path}(c, c')\}$$

Given lower bounds for the delays between the components we can calculate lower bounds for the delays for output channels in composed systems.

5.7. Composition and delay

Using our composition operators we can construct system architectures represented by data flow nets. In this section, we study how to calculate the delay of

composed systems from the delay profile of their components. For each component of the considered system, we can define a (maximal) guaranteed delay for each output channel. Given a component

$$F : \vec{I} \to \wp(\vec{O})$$

we introduce mappings

$$d : I \to \mathbb{N} \cup \{\infty\}, e : O \to \mathbb{N} \cup \{\infty\}$$

that associate a delay with every input and output channel. To calculate the guaranteed delay of a network of components, we have to be able to determine the delay of F if for each of its input channels a delay is given.

We are interested in the delay of a component that is inside a large architecture given by a data flow graph (with feedback loops). If we give input to the data flow graph up to time $t \in \mathbb{N}$ then we assume that for each channel $c \in I$ the stream $(x.c)\downarrow(t + \mathrm{d}.c)$ determined by that input to the network up to time t.

From this delay of the input we guarantee a delay for the output channels at time t, if $\mathrm{d}.b$ is the guaranteed delay on input channel $b \in I$; thus the input $(x.b)\downarrow(t + \mathrm{d}.b)$ is fixed at time t. We write

$$\mathrm{isdelay}(F, d, e)$$

for the logical proposition that holds, if for all times $t \in \mathbb{N}$ we have (here we assume that for a stream s the partial stream $s\downarrow j$ is empty if $j \leqslant 0$)

$$\forall x, z \in \vec{I} : [\forall b \in I : (x, b) \downarrow (t - \mathrm{d}.b) = (z.b) \downarrow (t - \mathrm{d}.b)] \Rightarrow$$
$$[\forall c \in O : \{(y.c) \downarrow t + \mathrm{e}.c : y \in F.x\} = \{(y.c) \downarrow t + \mathrm{e}.c : y \in F.z\}]$$

This formula expresses that the input on channel b affects the output on channel c at most after $\mathrm{e}.c$ time steps, provided fresh input arrives on the input channels only after the times as described by d. We call then the functions

$$d : I \to \mathbb{N} \cup \{\infty\}, e : O \to \mathbb{N} \cup \{\infty\}$$

a *delay profile*. A delay profile can be graphically represented by an annotated data flow node as show in Figure 8.

Let all the definitions be as in the section on composition. For a composition we get graphical representations of the delay profiles of the components as shown in Figure 9.

Given

$$d : (I_1 \cup I_2) \to \mathbb{N} \cup \{\infty\},$$
$$e : (O_1 \cup O_2) \to \mathbb{N} \cup \{\infty\}$$

Figure 8. Delay profile in a graphical representation.

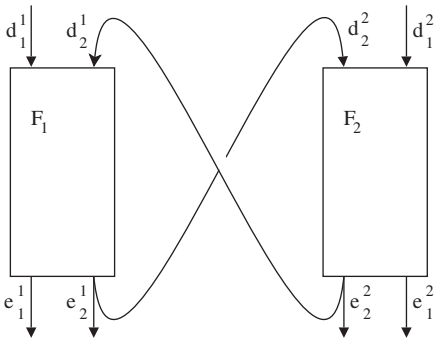

Figure 9. Delay profile for the components in a composition (suppressing channel identifiers).

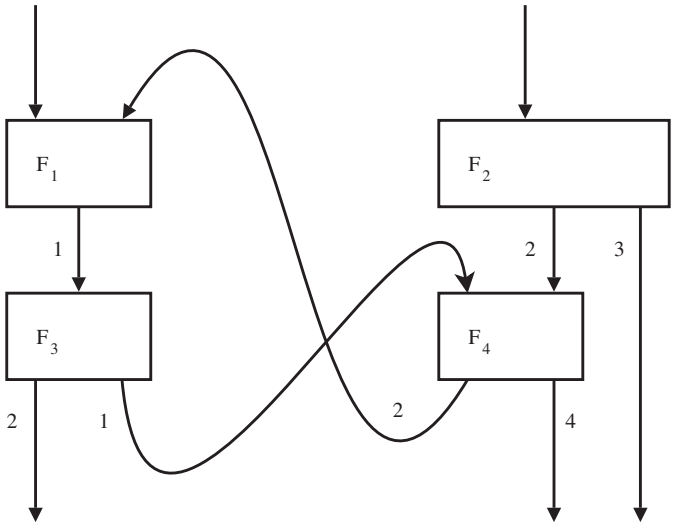

Figure 10. Network of components with given delay for each channel.

we only have to assume for $d^k = d|I_k$ and $e^k = e|O_k$ for $k = 1, 2$:

$$\text{isdelay}(F_1, d^1, e^1)$$

as well as

$$\text{isdelay}(F_2, d^2, e^2)$$

and for the feedback (by $d_2^1 = d^1|(I_1 \cap O_2)$ we denote the vector of delays for the channels in $(I_1 \cap O_2)$, by $d_2^2 = d^2|(I_2 \cap O_1)$ we denote the vector of delays for the channels in $(I_1 \cap O_2)$, by $e_2^2 = e^2|(I_1 \cap O_2)$ we denote the vector of delays for the channels in $(I_1 \cap O_2)$, by $e_2^1 = e^1|(I_2 \cap O_1)$ we denote the vector of delays for the channels in $(I_1 \cap O_2)$)

$$d_2^1 \leqslant e_2^2 \wedge d_2^2 \leqslant e_2^1$$

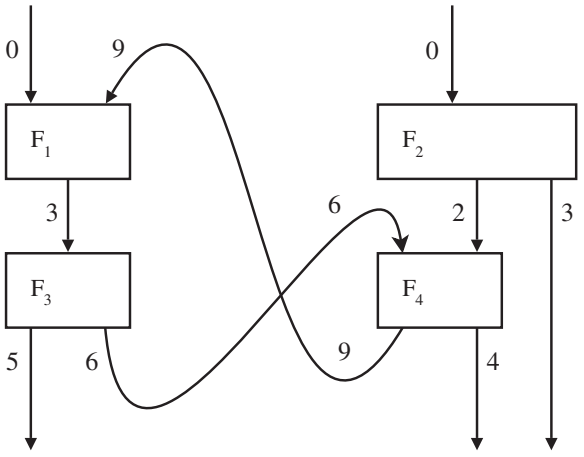

Figure 11. Network of components with accumulated delay for each channel.

to guarantee

$$\text{isdelay}(F1 \otimes F2, d|I, e|O).$$

Note that the guaranteed delay of a feedback channel $c \in (I_1 \cap O_2) \cup (I_2 \cap O_1)$ must be greater at the output than the assumed at the input to guarantee a faithful fixpoint and strong causality. This way of asserting delays of composition can be captured in a simple proof rule.

5.8. Optimal delay profile

In this section we show how to calculate optimal delay profiles for composed systems.

We assume that $\forall\, c \in I_1 \cap I_2$:

$$d^1.c = d^2.c$$

$$\text{isdelay}(F_i, d^i, e^i) \text{ for } i = 1, 2$$

$$\forall c \in I_1 \cap O_2 \colon d^1.c \leqslant e^2.c$$

$$\forall c \in I_2 \cap O_1 \colon d^2.c \leqslant e^1.c$$

to guarantee that

$$\text{isdelay}(F_1 \otimes F_2, d, e)$$

holds, where

$$I = (I_1 \backslash O_2) \cup (I_2 \backslash O_1), O = (O_1 \cup O_2)$$

and

$$d \colon I \to \mathbb{N} \cup \{\infty\}, e \colon O \to \mathbb{N} \cup \{\infty\}$$

are functions defined by the equations:

$$\forall c \in \mathrm{I} : \mathrm{d}.c = \begin{cases} \mathrm{d}^1.c \text{ if } c \in \mathrm{I}_1 \backslash \mathrm{I}_2 \\ \mathrm{d}^2.c \text{ if } c \in \mathrm{I}_2 \backslash \mathrm{I}_1 \\ \min(\mathrm{d}^1.c, d^2.c) \text{ if } c \in \mathrm{I}_1 \cap \mathrm{I}_2 \end{cases}$$

$$\forall c \in \mathrm{O} : \mathrm{e}.c = \begin{cases} \mathrm{e}^1.c \text{ if } c \in \mathrm{O}_1 \backslash \mathrm{O}_2 \\ \mathrm{e}^2.c \text{ if } c \in \mathrm{O}_2 \backslash \mathrm{O}_1 \\ \min(\mathrm{e}^1.c, e^2.c) \text{ if } c \in \mathrm{O}_1 \cap \mathrm{O}_2 \end{cases}$$

These rules allow us to calculate and prove delays in a modular way for composed systems given delays for their components.

We construct the maximal guaranteed delay b for the feedback loops in a composition. We assume that for all $i \in \{1, 2\}$

$$\mathrm{isdelay}(\mathrm{F}_i, \mathrm{d}^i, \mathrm{e}^i)$$

holds and that for all channels $c \in \mathrm{I}_1 \cap \mathrm{O}_2$

$$\mathrm{d}^1.c \leqslant \mathrm{b}.c$$

and for all $c \in \mathrm{I}_2 \cap \mathrm{O}_1$

$$\mathrm{d}^2.c \leqslant \mathrm{b}.c$$

where input delay b is defined as

$$\mathrm{b} : \mathrm{I}_1 \cup \mathrm{I}_2 \to \mathbb{N}$$

with

$$\mathrm{b}.c = max\,\{\mathrm{b}^k.c : k \in \mathbb{N}\}$$

and where we inductively calculate the delays

$$\mathrm{b}^k : \mathrm{I}_1 \cup \mathrm{I}_2 \to \mathbb{N}$$

as follows

$$\mathrm{b}^0.c = 0 \text{ if } c \in \mathrm{C}$$

$$\mathrm{b}^0.c = \mathrm{d}^i.c \text{ if } c \in \mathrm{I}_i \backslash \mathrm{C}$$

where C is the set of feedback channels

$$\mathrm{C} = (\mathrm{I}_1 \cup \mathrm{I}_2) \cap (\mathrm{O}_2 \cup \mathrm{O}_1).$$

The delays b^{k+1} are defined inductively by

$$\mathrm{b}^{k+1}.c = \mathrm{d}^i.c \text{ for } c \in \mathrm{I}_i \backslash \mathrm{C}$$

$$\mathrm{b}^{k+1}.c = \mathrm{a}.c \text{ for } c \in \mathrm{C}$$

where the delay a: $\mathrm{O}_1 \cup \mathrm{O}_2 \to \mathbb{N}$ is the maximal numbers such that

$$\mathrm{isdelay}(\mathrm{F}_i, \mathrm{b}^k | \mathrm{I}_i, \mathrm{a} | \mathrm{O}_i)$$

holds.

Note that b is the 'least fixpoint' of the delay function.

The construction uses as its basis the compositionality of the calculus.

5.9. Deducing delay equations

We consider a network N with interface behavior F_N with a number of systems as data flow nodes. Let C be the set of all channels in the net N. We want to calculate a delay profile

$$\mathrm{d}: \mathrm{C} \to \mathbb{N} \cup \{\infty\}$$

With every system behavior F in the network we may associate a function

$$\mathrm{gdel}_F(c): \mathbb{N} \times \ldots \times \mathbb{N} \to \mathbb{N}$$

for every output channel c defined by

$$\mathrm{gdel}_F(c)(\mathrm{d}.c_1, \ldots, \mathrm{d}.c_n) =$$
$$\max\{k \in \mathbb{N} \cup \{\infty\} : \forall x, z : [\forall \mathrm{b} \in \mathrm{I}: (x.\mathrm{b}) \downarrow (t - \mathrm{d}.\mathrm{b}) = (z.\mathrm{b}) \downarrow (t - \mathrm{d}.\mathrm{b})] \Rightarrow$$
$$\{(y.c) \downarrow t + k : y \in \mathrm{F}.x\} = \{(y.\mathrm{c}) \downarrow t + k : y \in \mathrm{F}.z\}$$

This equation calculates the maximal guaranteed delay for channel c given the delays $\mathrm{d}.c_1, \ldots, \mathrm{d}.c_n$ for the input channels of F.

To calculate for every channel c in the network its delay $\mathrm{d}.c$ we associate with every node and every output channel c of one of the nodes F the equation

$$\mathrm{d}.c = \mathrm{gdel}_F(c)(d.x_1, \ldots, \mathrm{d}.x_n)$$

For the input channels c we assume

$$\mathrm{d}.c = 0$$

We get an equation for each channel $c \in \mathrm{C}$. It is easy to show that the functions $\mathrm{gdel}_F(c)$ are all monotonic. Therefore, there exists a least fixpoint d for the equations which is the delay profile.

6. Composition and the choice of the time scale

In this chapter we study the question how time abstraction and composition fit together. A compositional formula for time refinement should read as follows:

$$\mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n) = \mathit{COA}(F_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n)$$

However, this formula does not hold, in general, since making a behavior coarser is an information loss that may result in the loss of strong causality and thus may introduce 'causal loops'. The time abstraction is the origin of the problems with causal loops in approaches advertised under the name 'perfect synchrony' such as Esterel (see Ref. 9). Moreover, the individual timing of the subcomponents may be highly relevant for selecting the behaviors (the output histories).

6.1. Strong causality and compositionality of coarsening

In this section we study cases that guarantee the validity of the equation

$$\mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n) = \mathit{COA}(F_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n)$$

As we will explain in Section 9.3, coarsening is fundamental for a flexible timing of large systems architectures.

That this equation does not hold, in general, can immediately be concluded from our discussion of causal and not causal fixpoints.

Theorem: Let F_1 and F_2 be behaviors that can be composed (according to consistent channel naming and typing); then the following equation holds

$$\mathrm{COA}(F1 \otimes F2, n) \subseteq \mathrm{COA}(F1, n) \otimes \mathrm{COA}(F2, n)$$

Proof: By definition, the statement (let all definitions be as in the definition of $\otimes$)

$$y \in (F_1 \otimes F_2).x$$

stands for: there exists a history z for all channels in I_1, I_2, O_1, and O_2 such that $y = z|O$ and

$$\begin{aligned} & z|I = x|I \\ \wedge\ & z|O_1 \in F_1(z|I_1) \\ \wedge\ & z|O_2 \in F_2(z|I_2) \end{aligned}$$

Now let us assume $y' \in \mathrm{COA}(F_1 \otimes F_2, n).x'$; then there exists $y \in (F_1 \otimes F_2).x$ such that $x' = \mathrm{COA}(n).x$ and $y' = \mathrm{COA}(n).y$ and there exists z such that $y = z|O$ and

$$\begin{aligned} & z|I = x|I \\ \wedge\ & z|O_1 \in F_1(z|I_1) \\ \wedge\ & z|O_2 \in F_2(z|I_2) \end{aligned}$$

From this, we get with $z' = \mathrm{COA}(n).z$:

$$\begin{aligned} & z'|I = x'|I \\ \wedge\ & z'|O_1 \in \mathrm{COA}(n).F_1(z|I_1) \\ \wedge\ & z'|O_2 \in \mathrm{COA}(n).F_2(z|I_2) \end{aligned}$$

Since by definition of $\mathrm{COA}(n)$ we get

$$\mathrm{COA}(n).(F.x) \subseteq \mathrm{COA}(F, n).(\mathrm{COA}(n).x)$$

we get

$$\begin{aligned} & z'|I = x'|I \\ \wedge\ & z'|O_1 \in \mathrm{COA}(F_1, n).(\mathrm{COA}(n).(z|I_1)) \\ \wedge\ & z'|O_2 \in \mathrm{COA}(F_2, n).(\mathrm{COA}(n).(z|I_2)) \end{aligned}$$

and finally

$$\begin{aligned} & z'|I = x'|I \\ \wedge\ & z'|O_1 \in \mathrm{COA}(F_1, n).(z'|I_1) \\ \wedge\ & z'|O_2 \in \mathrm{COA}(F_2, n).(z'|I_2) \end{aligned}$$

which proves that

$$y' \in (\mathrm{COA}(F_1, n) \otimes \mathrm{COA}(F_2, n)).x'$$

and thus that

$$\mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n) \subseteq \mathrm{COA}(\mathrm{F}_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n)$$

This completes the proof.

The theorem shows that

$$\mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n)$$

is a refinement of

$$\mathrm{COA}(\mathrm{F}_1, \mathrm{n}) \otimes \mathrm{COA}(\mathrm{F}_2, n)$$

The converse statement is not true, in general. It holds only if both F_1 and F_2 are not sensitive to the finer timing.

Theorem: Let $n > 1$ hold. Assume for $i = 1, 2$ that we have (for all $x, x' \in \vec{\mathrm{I}}$)

$$\mathrm{COA}(n).x = \mathrm{COA}(n).x' \Rightarrow \mathrm{F}_i.x = \mathrm{F}_i.x'$$

and that delay(F_i, n) holds, then we get:

$$\mathrm{COA}(\mathrm{F}_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n) \subseteq \mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n)$$

Proof: The proof that under the given assumptions the formula holds uses the stepwise construction of the history z such that

$$z|\mathrm{I} = x|\mathrm{I} \wedge z|\mathrm{O}_1 \in \mathrm{F}_1(z|\mathrm{I}_1) \wedge z|\mathrm{O}_2 \in \mathrm{F}_2(z|\mathrm{I}_2)$$

for any given z' with

$$\begin{aligned} & z'|\mathrm{I} = x'|\mathrm{I} \\ \wedge\ & z'|\mathrm{O}_1 \in \mathrm{COA}(\mathrm{F}_1, n).(z'|\mathrm{I}_1) \\ \wedge\ & z'|\mathrm{O}_2 \in \mathrm{COA}(\mathrm{F}_2, n).(z'|\mathrm{I}_2) \end{aligned}$$

with $z' = \mathrm{COA}(n).z$. Note that $y' \in (\mathrm{COA}(\mathrm{F}_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n)).x'$ iff such a history z' exists with $y' = z'|\mathrm{O}$. From the existence of z we can conclude $y' = \mathrm{COA}(n).z$ and $y' \in \mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n)$.

This construction is inductive and uses the fact that the behaviors are strongly causal. Since delay(F_i, n) holds for $i = 1, 2$, we can conclude that COA(F_i, n) is strongly causal for $i = 1, 2$.

By the assumption we get for any history d and b such that

$$b \in \mathrm{COA}(\mathrm{F}_i, n).(\mathrm{COA}(n).\mathrm{d})$$

with the history $a \in \mathrm{O}_i \rightarrow (\mathrm{M}^*)^{n^*k}$ specified by

$$a = (\mathrm{COA}(n).b) \downarrow (n^*k)$$

and

$$a \in (\mathrm{F}_i.\mathrm{d}) \downarrow (n^* k)$$

that there exists a history $a' \in \mathrm{O_i} \rightarrow (\mathrm{M}^*)^{n^*(k+1)}$ such that

$$a' = (\mathrm{COA}(n).b) \downarrow (n^*(k+1))$$

and

$$a' \in (\mathrm{F}_i.d) \downarrow (n^*(k+1))$$

Now let

$$y' \in (\mathrm{COA}(\mathrm{F}_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n)).(\mathrm{COA}(n).x)$$

hold; then there exists a history z' such that $y' = z'|\mathrm{O}$ and

$$z'|\mathrm{I} = x'|\mathrm{I}$$
$$\wedge\, z'|\mathrm{O}_1 \in \mathrm{COA}(\mathrm{F}_1, n).(z'|\mathrm{I}_1)$$
$$\wedge\, z'|\mathrm{O}_2 \in \mathrm{COA}(\mathrm{F}_2, n).(z'|\mathrm{I}_2)$$

We construct a history z such that $z' = \mathrm{COA}(n).z$ holds and

$$z|\mathrm{I} = x|\mathrm{I} \wedge z|\mathrm{O}_1 \in \mathrm{F}_1(z|\mathrm{I}_1) \wedge z|\mathrm{O}_2 \in \mathrm{F}_2(z|\mathrm{I}_2)$$

inductively as follows: since delay(F_i, n) holds there exists a history

$$z^{(0)} \in \mathrm{C} \to (\mathrm{M}^*)^n$$

such that $z' \downarrow 1 = \mathrm{COA}(n).z^{(0)}$ and for every z'' we have (for $i = 1, 2$)

$$z^{(0)}|\mathrm{I} = (x \downarrow n)|\mathrm{I} \wedge z^{(0)}|\mathrm{O}_i \in \mathrm{F}_i.(z^{(0)\wedge}z''|\mathrm{I}_i) \downarrow n$$

This follows from the fact that $\mathrm{F}_i.a\downarrow n = \mathrm{F}_i.b\downarrow n$ for all a, b since delay(F_i, n) holds and therefore

$$\mathrm{COA}(n).(\mathrm{F}_i.(z^{(0)\wedge}z''|\mathrm{I}_i) \downarrow n) = (\mathrm{COA}(\mathrm{F}_i, n).(z'|\mathrm{I}_2)) \downarrow 1$$

Given $z^{(k)} \in \mathrm{C} \to (\mathrm{M}^*)^{n^*(k+1)}$ where

$$z' \downarrow (k{+}1) = \mathrm{COA}(n).z^{(k)}$$

and

$$z^{(k)}|\mathrm{I} = (x \downarrow (n^*(k{+}1)))|\mathrm{I}$$
$$\wedge\, z^{(k)}|\mathrm{O}_i \in \mathrm{F}_i.(z^{(k)\wedge}z''|\mathrm{I}_i) \downarrow (n^*(k{+}1))$$

there exists $z^{(k+1)} \in \mathrm{C} \to (\mathrm{M}^*)^{n^*(k+2)}$ such that

$$z^{(k+1)}|\mathrm{I} = (x \downarrow (n^*(k{+}2)))|\mathrm{I}$$
$$\wedge\, z^{(k+1)}|\mathrm{O}_i \in \mathrm{F}_i.(z^{(k+1)\wedge}z''|\mathrm{I}_i) \downarrow (n^*(k{+}2))$$

This follows from the fact that

$$\mathrm{F}_i(z^{(k)}|\mathrm{I}_i^{\wedge}a) \downarrow n = \mathrm{F}_i(z^{(k)}|\mathrm{I}_i^{\wedge}b) \downarrow (n^*k)$$

For all a, b since delay(F_i, n) holds and therefore

$$\mathrm{COA}(n).(\mathrm{F}_i.(z^{(k)\wedge}z''|\mathrm{I}_i) \downarrow (n^* k)) = (\mathrm{COA}(\mathrm{F}_i, n).(z'|\mathrm{I}_2)) \downarrow k$$

By induction we get for all k:

$$z' \downarrow (k{+}1) = \mathrm{COA}(n).z^{(k)}$$

and

$$z^{(k)}|\mathrm{I} = (x \downarrow (n^*(k{+}1)))|\mathrm{I}$$
$$\wedge\, z^{(k)}|\mathrm{O}_i \in \mathrm{F}_i.(z^{(k)\wedge}z''|\mathrm{I}_i) \downarrow (n^*(k{+}1))$$

and thus for z with $z\downarrow(n^*(k{+}1)) = z^{(k)}$ for all k

$$z' = \mathrm{COA}(n).z^{(k)}$$

and

$$z|\mathrm{I} = x|\mathrm{I}$$
$$\wedge\, z|\mathrm{O}_i \in \mathrm{F}_i.(z|\mathrm{I}_i)$$

This proves that

$$\mathrm{COA}(\mathrm{F}_1, n) \otimes \mathrm{COA}(\mathrm{F}_2, n) \subseteq \mathrm{COA}(\mathrm{F}_1 \otimes \mathrm{F}_2, n)$$

and thus concludes the proof.

Moreover, if a component is time independent, then we have the validity of the following equation:

$$\mathrm{COA}(\mathrm{F}, n).\mathrm{COA}(n).x = \mathrm{COA}(n).(\mathrm{F}.x)$$

However, in contrast to problems when coarsening behaviors, the equation

$$\mathrm{FINE}(\mathrm{F}_1 \otimes \mathrm{F}_2, n) = \mathrm{FINE}(\mathrm{F}_1, n) \otimes \mathrm{FINE}(\mathrm{F}_2, n)$$

does always hold, as long as F_1 and F_2 are strongly causal.

7. Pros and cons for strong causality

As formulated in the hypothesis above, we may assume that for each model of a physical system behavior, there is a time scale that is fine enough to capture all essential time differences, especially for the delay between input and output, to guarantee the property of strong causality. Modeled in an appropriate time scale the behavior is always strongly causal according to the principle of strong causality.

Strong causality has a number of significant advantages since it makes the reasoning about systems more concrete and simpler since reasoning about feedback loops is reduced to induction. In particular, it is easy to treat feedback loops by fixpoints for strongly causal behaviors since strong causality guarantees, in particular, the existence of unique fixpoints for deterministic functions. In other words, for strongly causal, fully realizable system behaviors all fixpoints are causal and thus computationally appropriate in the sense that they faithfully reflect computations.

The disadvantage of strong causality is its limited abstractness illustrated for instance by the fact that in sequential composition delays accumulate. This well known effect observed for the composition of strongly causal components can be nicely demonstrated for pipelining.

Let the behaviors

$$\mathrm{F}_1\colon \mathrm{I}_1 \to \wp(\mathrm{O}_1), \mathrm{F}_2\colon \mathrm{O}_1 \to \wp(\mathrm{O}_2)$$

be given. We obtain

$$\mathrm{delay}(\mathrm{F}_1, m) \wedge \mathrm{delay}(\mathrm{F}_2, n) \Rightarrow \mathrm{delay}(\mathrm{F}_1 \circ \mathrm{F}_2, m + n)$$

In the case of two strongly causal functions F1 and F2 we get (at least)

$$\mathrm{delay}(\mathrm{F}_1 \circ \mathrm{F}_2, 2)$$

On one hand this observation is very satisfactory since it leads to a useful delay calculus (see above). On the other hand it shows an unfortunate inflexibility of the design calculus for timed systems. If we want to represent a function by an

architecture with two functions composed by pipelining we always have to accept a delay by at least 2 if the functions are strongly causal. In fact, if we insist on a delay less than 2 then a component cannot be implemented by a system consisting of two components composed sequentially. This seems unacceptable, since it makes the design very inflexible, and seems to be a good reason to reject our approach based on a global discrete time altogether. In the remainder of this paper we deal with this issue and show how to gain the necessary flexibility.

The choice of the time granularity is crucial for the behavior. As a result of strong causality composition is not as abstract as wanted and needed. A sequentially composed system with strongly causal components always shows delays larger than one. This difficulty can be avoided by lowering the constraint of strong to weak causality. But then the characterization of fixpoints being causally sound and computationally realistic is no longer guaranteed and the reasoning about fixpoints and the characterization of causally correct fixpoints gets much more involved or even impossible.

Fixpoints are of major interest for systems since they are the classical technique to give meaning to feedback loops, mutual interaction, and recursive definitions of behaviors. A feedback loop introduced for a behavior F with input channel e and output channel c is specified by a fixpoint characterization

$$x.e \in \{y.c : y \in \mathrm{F}.x\}$$

Let us consider a function $\mathrm{F}: \vec{C} \to \wp(\vec{C})$. Feeding back all its channels corresponds to the fixpoint property $x \in \mathrm{F}.x$. This property characterizes the computations in feedback loops correctly if F is strongly causal. Otherwise there may occur fixpoints that are computationally infeasible since they are the result of so-called causal loops. A causal loop yields a fixpoint where certain output is produced under the assumption that this output is fed back within the same time interval as the corresponding input ('self fulfilling prophecy'). There exist fixpoints that are not causal in the sense that they do not reflect proper computations. We demonstrate this observation by some simple examples.

For a feedback loop a fixpoint is causal, if all produced output is actually caused by some input or – more precisely – is enforced by some input that is available before the output is generated. This corresponds to the well-known concept of least fixpoints and inductive definitions that reflect in an abstract manner a concept of computation. A fixpoint is not causal if there is output that is produced before the corresponding input arrives, which then in turn is used as the corresponding input in a feedback loop to 'logically justify' the output.

Example: Causal loops for the identity function

Let us consider as a simple example the weakly causal time independent identity function Id characterized by the equation

$$\overline{\mathrm{Id}.x} = \bar{x}$$

In fact there are many time independent identity functions (functions that fulfill the equation above) that are only distinct with respect to their time delay and timing. In particular, there is a weakly causal identity function that fulfills the following specification (recall that $\#x$ denotes the length of a sequence)

$$\mathrm{Id}.x = \{x' : \mathrm{x}' = \bar{\mathrm{x}} \wedge \forall t \in \mathbb{N} : \#\overline{x' \downarrow t} \leqslant \#\overline{x \downarrow t}\}$$

A strongly causal version of the identity shows in addition the property

$$\overline{\mathrm{Id}.x \downarrow k+1} \sqsubseteq \overline{x \downarrow k} \quad (*)$$

If we are looking for a fixpoint

$$x \in \mathrm{Id}.x$$

we easily prove for a strongly causal version by induction from the equation (*) the expected property $\bar{x} = \langle\,\rangle$. For the most general, only weakly causal identity $\mathrm{Id}.x = x\uparrow$, however, each history is a fixpoint.

For a weakly causal version of the identity function Id the proof that $x \in \mathrm{Id}.x \Rightarrow \bar{x} = \langle\,\rangle$ as shown for strong causality does not work. There are fixpoints $x \in \mathrm{Id}.x$ that are not the empty history (i.e. $\bar{x} \neq \langle\,\rangle$) and do not represent causal computations, of course. An example is the history x with $x.t = \langle 1 \rangle$ for all $t \in \mathbb{N}$. Such fixpoints do not reflect feasible 'causally proper' computations. They are not causal and instances of a causal loop.

Weak causality is the result of selecting too coarse time scales. It leads to models that are too abstract. As a result, in these models composition may not be operationally faithfully definable since fixpoints may correspond to causal loops and – even worse – we cannot distinguish between causal and non-causal fixpoints, in general. Strong causality allows us to avoid causal loops at the cost of less abstract models. Strong causality provides a computationally faithful, more down to earth model of composition and interaction.

8. A compromise

We have extensively discussed so far the advantages and disadvantages of strong in contrast to weak causality. Weak causality is transparent with respect to composition. A weakly causal component can always be decomposed into a system of weakly causal components with feedback loops and sequential composition, which does not hold for strongly causal components. Strong causality is not abstract enough, while weak causality is too abstract in cases of feedback loops and fixpoints. The challenge is to combine the advantages of weak and strong causality. In this section we look for a methodological compromise that incorporates, as much as possible, both their advantages and avoids most of their disadvantages by supporting a refinement of time.

8.1. Strongly and weakly causal behaviors

In this section we explore possibilities to identify causal fixpoints for weakly causal behaviors. In fact, there is a natural implicit notion of strong causality that applies even for weakly causal systems.

Definition: *Causality*
An I/O-behavior (that is weakly causal)

$$F : \vec{I} \to \wp(\vec{O})$$

is called *causal* (*properly timeable*), if there exists a number $n \in \mathbb{N}$ and a strongly causal function

$$F' : \vec{I} \to \wp(\vec{O})$$

such that $F = COA(F', n)$.

In other words, in the definition above, F is a causal function that is weakly causal and can be understood as an abstraction (by time coarsening) of some strongly causal function F′. The critical question, however, is whether such a strongly causal function is unique and thus leads to a canonical construction of causal fixpoints.

Every fixpoint of the strongly causal function F′ is a fixpoint of the weakly causal function F since

$$\begin{aligned} & x \in F'(x) \\ \Rightarrow\ & COA(n).x \in COA(n).F'(x) \\ \Rightarrow\ & COA(n).x \in COA(F, n).COA(n).x \end{aligned}$$

Given a weakly causal function F the existence of a strongly causal function F′ such that F is an abstraction of F′ does not really help, however, to separate faithful fixpoints from causal loops. There are cases where there are several choices of strongly causal behaviors F′, say F_1' and F_2', such that some fixpoints of F are causal for F_1' but not for F_2'. This leads to the following fact: time abstractions may lose the notion of strong causality leading to models where we cannot distinguish causal from non-causal fixpoints. The information loss by time abstraction is therefore critical, in general, and does not allow any longer distinguishing causal from non-causal fixpoints.

We illustrate this problem by an example. First we show an example where weak causality works.

Example. Identity
Consider again some identity function F with $\overline{F(x)} = \bar{x}$. We can prove that fix F is the empty stream. The proof is essentially the same as the one given above. We assume that there is a strongly causal function and that every causal fixpoint of F corresponds to a fixpoint of the strongly causal function. This way we easily deduce $\bar{x} = \langle\,\rangle$ for every fixpoint.

This shows that the assumption of strong causality leads to proof of principles for systems with feedback loops. But the assumption does not allow us to rule out non-causal fixpoints in a canonical way. Only strong causality guarantees that all fixpoints are causal and that we do not have fixpoints for which we do not know whether they are causal or not.

Finally, to demonstrate the significance of strong causality let us look at a slightly trickier example. We consider a function F with the following characteristic behavior:

$\bar{x}$	$\overline{\mathrm{F}.x}$
$\langle\,\rangle$	$\{\langle\,\rangle, \langle 1\rangle\}$
$\langle 1\rangle$	$\{\langle 1\rangle, \langle 1\ 1\rangle\}$
$\langle 1\ 1\rangle$	$\{\langle 1\ 1\rangle\}$

Depending on the structure of its timing the history x with $\bar{x} = \langle 1\rangle$ can be a causal fixpoint or not (here $\langle\langle\,\rangle\rangle^{\infty}$ denotes the infinite stream with empty sequences of messages in each time interval) depending on the actual definition of F. The following table shows a particular version of F:

x	F.x
$\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$
$\langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$
$\langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$

In this case F is (more precisely, can be extended to) a strongly causal function and the stream $\langle 1\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$ is a causal fixpoint.

Now consider the strongly causal function F′ with the following characteristic behavior

x	F′.x
$\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$
$\langle\langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle\,\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$
$\langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$	$\{\langle\langle\,\rangle\rangle^{\infty}, \langle\langle\,\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}, \langle\langle 1\rangle\ \langle 1\rangle\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}\}$

In this case $\langle 1\rangle\hat{}\langle\langle\,\rangle\rangle^{\infty}$ is not a causally correct fixpoint of F′, although

$$\overline{F.x} \text{ and } \overline{F'.x}$$

do coincide. In such a case, the timing of the output is essential for determining the causality of fixpoints and therefore for determining which of the fixpoints are causal.

9. Causality, time, and composition

As we have shown, causality is essential for an inductive reasoning about composed systems with feedback loops.

9.1. Causality and composition

Given a set of specified components we get for each component

$$F : \vec{I} \to \wp(\vec{O})$$

with the relational specification ('specifying assertion') for $y \in F.x$ in the form of a predicate P:

$$P(x, y)$$

by imposing causality the logical weakest specification Pc that fulfills the following equation

$$\begin{aligned} Pc(x, y) \equiv P(x, y) \wedge \forall t \in \mathbb{N} : \exists x' : x' \downarrow t = x \downarrow t \Rightarrow \\ \exists y' : Pc(x', y') \wedge y' \downarrow t{+}1 = y \downarrow t{+}1 \end{aligned}$$

This way we can add the requirement of strong causality to every specification to get a strongly causal specification. Note, that the specification may become inconsistent if the original specification contradicts the notion of strong causality. As is well-known (and straightforward to prove), if we compose strongly causal systems we get again strongly causal systems.

9.2. Composition and coarsening time

If we compose strongly causal systems, then their time delays accumulate. For instance, if each component has a time delay of at most one, then computing the time delay of a composed system is the result of calculating the minimal number of delays of components on the shortest path from an input to an output channel. This allows us to calculate delays in architectures of composed systems. But sometimes we may be interested in a more abstract view on the function computed by an architecture.

If we work with a component F with delay k, we easily construct a strongly causal component of delay one by the component

$$COA(F, k)$$

This construction can be understood as a way to abstract from the finer time scale used inside for the sub-systems of a system to a coarser time scale outside of the system. We may speak of a macro/micro time scale. As long as we are not interested in the fine-grained notion of time, such an abstraction is certainly appropriate. It is the best abstraction maintaining crucial notions such as causal fixpoints and ways to reason about them.

9.3. Local timing

The flexibility of our timing calculus can be seen in the following by demonstrating how we can work with local timing and local clocks.

For each subsystem (subcomponent) of a network we can introduce a local time and a local time granularity. This idea is demonstrated by a simple example.

Example: Architecture with local Time Granularities

The following expression

$$\mathrm{COA}(\mathrm{COA}(\mathrm{F} \otimes \mathrm{G}, 10) \otimes \mathrm{COA}(\mathrm{H} \otimes \mathrm{K}, 20) \otimes \mathrm{L}, 2)$$

denotes a system with a subsystem

$\mathrm{F} \otimes \mathrm{G}$ which works in a 10 times faster mode

$\mathrm{H} \otimes \mathrm{K}$ which works in a 20 times faster mode

This way we get a system with different local time scales where each time scale can be chosen fine enough for each of the components locally to guarantee strong causality of each of its subcomponents such that all local computations are captured on the right level of time granularity. The components operate locally on a higher frequency. Since delays accumulate in a system the larger components composed of many subcomponents show typically larger delays such that we can choose coarser time granularity for them without losing strong causality.

Figure 12 shows an informal example of a complex system composed of three subsystems, each of which is composed of four components. For each of the subsystems we may select local proprietary time granularity.

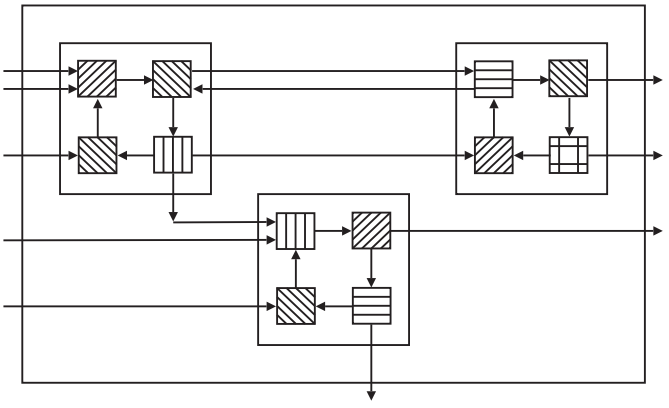

Figure 12. Hierarchical architectures of components with different time scales.

9.4. Multiplexing of the components of a network on one CPU

In general, we cannot expect that each dedicated node in a data flow net always runs on its own CPU. We rather expect that a complete net or sub-net of components is executed on one CPU. Then the multiplexing of the components, which may be running at different speeds, have to be mapped by a scheduling onto the CPU cycles.

9.5. Avoiding causal loops

In the following, we finally discuss the problem of causal loops. We show how we can add strong causality as a property to specifications to reason about feedback loops.

Example: Interface specification
Consider the following specification of a component that copies its input on channel x on both of its output channels y and r.

Copy

in x: T **out** y, r: T
$\bar{x} = \bar{y} \wedge \bar{x} = \bar{r}$

Strong causality as an assumption on the specification Copy allows us to conclude in addition to the specifying assertion the following property:

$$\forall t \in \mathbb{N} : \forall m \in \mathrm{T} :$$
$$\{m\}\#\overline{r \downarrow t+1} \leqslant \{m\}\#\overline{x \downarrow t}$$
$$\wedge\ \{m\}\#\overline{y \downarrow t+1} \leqslant \{m\}\#\overline{x \downarrow t}$$

where $M\#x$ denotes the number of copies of messages in the set that occur in x. Another example is the specification of component Repeater, which repeats each of its inputs infinitely often. Its specification reads as follows:

Repeater

in r : T **out** x : T
$\forall m \in \mathrm{T} : \{m\}\#\bar{r} > 0 \Rightarrow \{m\}\#\bar{x} = \infty$ $\wedge\ \{m\}\#\bar{r} = 0 \Rightarrow \{m\}\#\bar{x} = 0$

Strong causality allows us to conclude in addition to the specifying assertion the following property

$$\forall t \in \mathbb{N} : \forall m \in \mathrm{T} :$$
$$\{m\}\#\overline{r \downarrow t} = 0 \Rightarrow \{m\}\#\overline{x \downarrow t+1} = 0$$

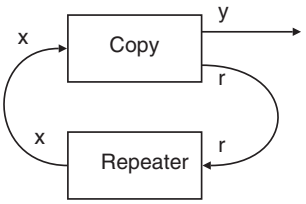

Figure 13. Composition repeater $\otimes$ copy.

Now let us compose the two components into the composed system characterized by the term

$$\text{Repeater} \otimes \text{Copy}$$

as shown in Figure 13. We get a system that forms a feedback loop. If we want to reason about the composed system, we have to refer to the assumption of strong causality to reason about the system in a way where we avoid causal loops. Without strong causality arguments we get the following formula:

$$\begin{aligned}
&\bar{x} = \bar{y} \wedge \bar{x} = \bar{r} \\
&\wedge \\
&\forall m \in \mathrm{T} : \{m\}\#\bar{r} > 0 \Rightarrow \{m\}\#\bar{x} = \infty \\
&\qquad \wedge \{m\}\#\bar{r} = 0 \Rightarrow \{m\}\#\bar{x} = 0
\end{aligned}$$

which simplifies to

$$\bar{x} = \bar{y} \wedge \bar{x} = \bar{r} \wedge \forall m \in \mathrm{T} : \{m\}\#\bar{r} > 0 \Rightarrow \{m\}\#\bar{r} = \infty$$

which indicates for the output y of the network only

$$\forall m \in \mathrm{T} : \{m\}\#y > 0 \Rightarrow \{m\}\#y = \infty$$

With strong causality arguments we get in addition the assertion

$$\begin{aligned}
&\forall t \in \mathbb{N} : \forall m \in \mathrm{T} : \\
&\{m\}\#\overline{r \downarrow t+1} \leqslant \{m\}\#\overline{x \downarrow t} \wedge \{m\}\#\overline{r \downarrow t} \leqslant \{m\}\#\overline{x \downarrow t} \\
&\wedge \\
&\forall t \in \mathbb{N} : \forall m \in \mathrm{T} : \\
&\{m\}\#\overline{r \downarrow t} = 0 \Rightarrow \{m\}\#\overline{x \downarrow t+1} = 0
\end{aligned}$$

that allows us to conclude (by a simple inductive argument) the validity of the following formula

$$\bar{y} = \langle\,\rangle$$

which cannot be concluded without strong causality.

10. Conclusion: robust flexible timing

The goal of this paper was to show the relationship between causality and the choice of the time scale and granularity as well as its influence onto compositionality. The main issues here were the dependencies between the time scale and composition with feedback.

If we choose the time scale fine enough the definition of faithful composition is quite straightforward. Our approach supports flexibly chosen time scales. We worked out the following idea of flexible timing:

- The leaves of the component hierarchy are state machines.
- Each state machine runs in its own local time scale, which defines the time duration of each of its steps.
- Each processor (CPU, controller) contains a family of state machines, which run with different speed (time granularity). This defines the finest time grain steps and the set of steps (state transitions) that have to be executed (scheduled) in a time slot. This determines the workload.

Hence, we can design a static schedule at the abstract level of behavior without being forced to address low-level technical issues such as schedulers or operating systems. Moreover, by strong causality we get a very robust inductive technique for reasoning by feedback.

We obtain a flexible and modular theory of timing of systems and sub-systems this way, which provides the following helpful properties

- high level abstract and flexible modeling
- multiplexing and scheduling on abstract level
- application oriented time model close to hardware time model
- causality for inductive reasoning
- avoiding causal loops
- rich algebraic properties
- discrete model of time that provides the same flexibility as analog models of time (real time)
- time abstraction by coarsening the time granularity to get rid of unwanted delays.

In the end we can generate from such a time model a static scheduling for the system.

References

1. M. Broy (1997) Refinement of time. In: M. Bertran and Th. Rus (eds) *Transformation-Based Reactive System Development. ARTS'97*, Mallorca. *Lecture Notes in Computer Science*, **1231**, pp. 44–63.
2. C. A. R. Hoare (1985) *Communicating Sequential Processes* (Englewood Cliffs: Prentice Hall).

3. R. Milner (1980) *A Calculus of Communicating Systems. Lecture Notes in Computer Science*, **92**.
4. M. Broy and K. Stølen (2001) *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement* (Berlin, Heidelberg, New York: Springer).
5. F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad and M. Stal (1996) *Pattern-Oriented Software Architecture: A System of Patterns* (New York: Wiley).
6. J. D. Brock and W. B. Ackermann (1981) Scenarios: a model of nondeterminate computation. In: J. Diaz and I. Ramos (eds) *Lecture Notes in Computer Science*, **107**, pp. 225–259.
7. P. Andrews (1986) *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Computer Science and Applied Mathematics (Oxford, New York: Academic Press).
8. J. M. Spivey (1988) *Understanding Z-A Specification Language and Its Formal Semantics. Cambridge Tracts in Theoretical Computer Science 3* (Cambridge University Press).
9. G. Berry and G. Gonthier (1988) The ESTEREL synchronous programming language: design, semantics, implementation. *INRIA, Research Report*, **842**.
10. G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language for Object-Oriented Development*. Version 1.0. RATIONAL Software Cooperation.

About the Author

Manfred Broy is a full professor for computer science at the Technische Universität München. His research interests are software and systems engineering, comprising both theoretical and practical aspects. This includes system models, specification and refinement of systems, development methods and verification techniques.