Unité Mixte de Recherche 5104 CNRS - INPG - UJF
Centre Equation
2, avenue de VIGNATE
F-38610 GIERES
tel : +33 456 52 03 40
fax : +33 456 52 03 50
http://www-verimag.imag.fr

Actors without Directors: a Kahnian View of Heterogeneous Systems

A. Benveniste, P. Caspi, R. Lublinerman, S. Tripakis

Verimag Research Report n^o 2008-6

September 9, 2008

Actors without Directors: a Kahnian View of Heterogeneous Systems

A. Benveniste, P. Caspi, R. Lublinerman, S. Tripakis

September 9, 2008

Abstract

This paper intends to clarify recent efforts proposed by the Berkeley school in order to give a formal semantics to the Ptolemy toolbox. We show that the proposed semantics is indeed a generalisation of a Kahn semantics based on tag systems. We make this proposal a precise one by showing for several Ptolemy domains what is the corresponding tag system and how operators can be defined. We first remark that this semantics doesn't need "absent" values. We also discuss some domains which do not obey this semantics. We then discuss some advantages of this semantics, namely that it provides heterogeneity and distribution for free. In particular, we show that while the semantics is naturally expressed in terms of actors, directors are not essential for semantical purposes and are only justified by simulation efficiency purposes. We conclude that in most cases, directors do not play an important part in the definition of a Model of Computation and Communication.

Keywords: Models of Computation and Communication, Tag Systems, Kahn Semantics, Heterogeneity

Reviewers: Florence Maraninchi

Notes: Albert Benveniste is with INRIA/IRISA, Paul Caspi is with Verimag/CNRS, Roberto Lublinerman is with Pennsylvania State University and StavrosTripakis is with Cadence Research Laboratories.

How to cite this report:

@techreport { ,
title = { Actors without Directors: a Kahnian View of Heterogeneous Systems},
authors = { A. Benveniste, P. Caspi, R. Lublinerman, S. Tripakis},
institution = { Verimag Research Report },
number = {2008-6},
year = { 2008},
note = { }
}

Contents

Chapter 1

Introduction

The semantics of heterogeneity The need for heterogeneity in modelling and development tools has been increasing while applications are becoming more and more complex. In view of this state of matters, pioneering frameworks like Ptolemy which have started addressing the issue of heterogeneity a long time ago are becoming always more popular and raising an ever growing interest. Thus, concepts of this framework like models of computation and communication (MoCC), actors, directors and so on have been getting an increasingly larger acceptance.

Among the problems raised by this subject, the semantic question is important. While homogeneous applications are in general well-mastered, problems start at their interfaces when several subsystems are composed to form a larger application. Ambiguities, semantic inconsistencies etc. are likely to produce undesired behaviours which can badly impair the overall functioning of the composed application. To this end, Lee & Sangiovanni have introduced their celebrated tagged signal model [8] which was meant to provide a precise semantics to such frameworks as Ptolemy.

Yet, there was still a large gap between this denotational formalism and the behaviour of Ptolemy which is still largely bound to the operational semantics of the simulation engine. Efforts have been devoted to filling this gap: for instance BIP which is based on operational semantics [1] and 42 [13] which provides building blocks for designing MoCCs in a comprehensive way.

The reason why the gap is so large is that tag systems are very general ones and do not provide any means for executing them. In particular the tag system approach is relational and composition proceeds by intersection of behaviours described in extension (see for instance [3]).

The application of Scott theory to tag systems For this reason a comprehensive step toward closing this gap has recently been taken in [9] so as to make things simpler by getting rid of non determinism, that is to say, by trying to see whether things could get simpler by restricting from relations to functions. After all, determinism is something designers are fond of, most simulators like Ptolemy are deterministic and, when non determinism is needed, in most cases it can be emulated by adding extra inputs to functions, aiming at choosing between several possible

futures.[1]

Yet, this was not sufficient: when composing functions, inputs of one function can become outputs of another one and conversely, creating feedback loops and resulting in the functional aspect being lost: we get systems of equations which can have no solution as well as several solutions.

But this is a well-known issue of denotational semantics and well-known solutions exist. The most widely adopted one is Scott's semantics [15]: if the domain of interest is a complete partial order (CPO) and we restrict ourselves to continuous functions, then we know that every system of equations has a least solution and it is sensible to decide that this is the semantics of the system. Moreover, the least solution is itself a continuous function of its free inputs and thus can in turn be composed at will. The framework is thus closed by composition (and even by lifting to higher orders) and works perfectly well.

But there was another problem. The basic objects of the tagged signal model are signals which in a deterministic point of view can be seen as functions from tags to values. The way Scott turns these signals into CPOs is to turn the value set into a CPO because in this way, the CPO property gets automatically lifted from the image set to the function set. Thus, in this Scott theory applied to tag systems, the tag set does not need to have any order property. But, in tag system theory, tag sets are partially ordered ones and basically have a time flavour: in Ptolemy, computations go from past to future, while in the Scott framework, it does not matter (tags may not have any order and there may be neither past nor future!).

Towards Kahn semantics Thus [11] had to modify Scott's order by requiring a prefix ordering principle in the spirit of the Kahn order [7]: a signal is larger than another one not only if it is more defined but also if both signals are defined on some initial segment of the tag set. In this way, a signal s_1 is larger than another signal s_2 if the initial segment over which s_1 is defined is larger than the initial segment over which s_2 is defined. In this way, computations can only extend the initial segments on which signals are defined and naturally flow from past to future.

There was still a problem due to the fact that some tag sets, for instance associated with the discrete event (DE) domain, are infinite in several dimensions: in this case, initial segments are infinite and thus a signal defined over an initial segment has to have an infinity of values. In some sense, time may not progress in the same way as in the Zeno phenomenon of hybrid systems. But it is not possible to compute an infinite number of values in a simulator. While [11] solved the problem based on a Kahn order, [9] proposed instead to borrow the idea of *absent value* from the French synchronous language school [2]: thus a signal defined on an initial segment may have only a finite number of computed non absent values (while absent values need not be computed).

The paper's goal and content The starting point of this paper was the observation that a theory based on tags does not need to use "absent" values because tags record the time or reaction index on which computations are performed (for a theory of synchronous reactive systems without "absent" values, see [14]). Our purpose in this paper is thus to show how sticking to the

[1] This is the way probability theory works: by adding input spaces about which the only knowledge we can have is their probability measure.

Kahn semantics helps clarifying this point of view and has valuable consequences. Among these consequences there is the ability of Kahn networks to be executed in a chaotic and distributed way. This opens the way for a distributed simulation and execution of heterogeneous systems. In particular, distribution comes for free and this is a very appealing feature of this semantics. But another very appealing feature is that heterogeneity itself comes for free: the (categorical) sum of CPOs is indeed a CPO. This allows freely merging tag sets and, together with the chaotic property of fixpoints, this generalised Kahn semantics does not require several kinds of directors but for efficiency purposes.

The rest of this paper is thus organised as follows: In Section 2 we present the Kahn theory of deterministic tag systems. In Section 3 we provide our application of this theory to several MoCCs and, in Section 4 we discuss the interest of our proposal. Finally we conclude by drawing some perspectives and discussing future work.

Chapter 2

A Kahn Theory of Deterministic Tag Systems

2.1 Deterministic Tag Systems

Signals The basic idea of the Tagged Signal Model [8] is to consider a signal $s \in S$ as a subset of pairs "tag, value" which can be formalised as:

$$S = \{s \mid s \subseteq \mathbb{T} \times V\}$$

where

- V is a set of values,
- $(\mathbb{T}, \leq)$ is a partially ordered set of tags

Deterministic Signals These signals are non deterministic ones: several values can be associated with the same tag. As we aim at considering deterministic tag systems, we first need to consider deterministic signals. This amounts to saying that we only consider signals that are partial functions from tags to values which we denote as:

$$S = \mathbb{T} \hookrightarrow V$$

Processes In the deterministic setting, processes (or actors, following the Ptolemy terminology) are just functions transforming input signals into output signals. For the sake of simplicity, we do not consider the types of signal values and assume an "universal" type V. Thus, the set of processes, P, is just the set of total functions from S^m to S^n:

$$P = S^m \mapsto S^n$$

where m, n are the input and output arities.

Functional Composition In this deterministic setting, things are very simple. Processes are composed by functional composition and a composed process is just a system of equations, *e.g.,*

$$\begin{aligned} s_3 &= p_1(s_1, s_2) \\ s_4 &= p_2(s_1, s_3) \end{aligned}$$

which can define another process p_3 such that

$$(s_3, s_4) = p_3(s_1, s_2)$$

Feedback Loops However, this raises the question of feedback loops: for instance consider the composed system:

$$\begin{aligned} s_3 &= p_1(s_1, s_2) \\ s_2 &= p_2(s_1, s_3) \end{aligned}$$

What does it compute? This system of equations may have no solution or it may have several solutions. Then determinism can be lost.

2.2 Scott Semantics

Scott semantics [15] provides a well-known solution to this issue. It consists of the following changes to the previous framework:

1 Add to V an undefined element $\perp$ and a partial order relation $\leq$ such that:

- $V^\perp = V \cup \{\perp\}$
- $\leq$ is the least order relation over $V^\perp$ generated by:

$$\forall v \in V, \perp \leq v$$

This makes $(V^\perp, \leq, \perp)$ a (flat) CPO. $\perp$ is clearly the least element of $V^\perp$ and any sequence of ordered elements (a chain) has a least upper bound ($\bigvee$) which is $\perp$ if the chain contains only $\perp$'s, or some v_1 if the chain contains this v_1: note that in the latter case the chain cannot contain another v_2 distinct from v_1 as the two are incomparable.

2 Redefine S as the set of *total* functions from $\mathbb{T}$ to $V^\perp$:[1]

$$S = \mathbb{T} \mapsto V^\perp$$

Then S inherits the CPO property of $V^\perp$ by defining:

[1] Obviously, a partial function can be made total by giving it the value $\perp$ whenever it is not defined.

- $s \leq s'$ if for any $t \in \mathbb{T}$, $s(t) \leq s'(t)$ which amounts to saying that s is smaller than s' if it is less defined,
- the bottom element of S, also denoted $\bot$, as the signal which is undefined everywhere: $\bot(t) = \bot$.

Given a chain of signals $s_0, \ldots s_n, \ldots$, and given any tag t, $s_0(t), \ldots s_n(t), \ldots$ is chain of values and

$$\bigvee\{s_0, \ldots s_n, \ldots\}(t) = \bigvee\{s_0(t), \ldots s_n(t), \ldots\}$$

3 Restrict processes to *continuous* functions from input to output signals, which means that, given a chain of inputs, that is to say a sequence of more and more defined signals, the outputs should form a chain and

$$\bigvee\{p(s_0), \ldots p(s_n), \ldots\} = p(\bigvee\{s_0, \ldots s_n, \ldots\})$$

Note that continuity implies *order preservation*:

$$s \leq s' \Rightarrow p(s) \leq p(s')$$

and note that this definition for single-input/single-output processes can be extended naturally to processes of different arities because products of CPOs inherit the CPO structure of their components. In particular, the order on the product is the component-wise order.

4 Then the Kleene theorem says that any system of equations has a (unique) least solution, which is in turn a continuous function of its free input signals. Thus composition preserves determinism.

2.3 The Causality Problem

But this solution is still unsatisfactory because it does not take advantage of the ordering over tags which have a flavour of time. In particular, a process may as well compute from future to past. For instance, we can easily design a process which is continuous in the Scott sense but not causal:

Take $\mathbb{T} = \mathbb{N}$ and the usual order over integers. Then define:

- $\text{next}(x)(n) = x(n+1)$
- $\text{if}_9(x,y)(n) = \begin{cases} x(n) & \text{if } n \leq 9 \\ y(n) & \text{otherwise} \end{cases}$
- $(x+1)(n) = \begin{cases} x(n)+1 & \text{if } x(n) \neq \bot \\ \bot & \text{otherwise} \end{cases}$

It is easy to see that these processes are continuous and the same holds for the composite process defined by the system of equations:

$$x = \text{if}_9(\text{next}(x) + 1, y)$$

Indeed, there is a unique solution to this system:

$$x(n) = \begin{cases} \bot & \text{if } n < 10 \text{ and } y(10) = \bot \\ y(10) + 10 - n & \text{if } n < 10 \text{ and } y(10) \neq \bot \\ y(n) & \text{otherwise} \end{cases}$$

But it is also easy to see that this solution needs sometimes to be computed "backwards", from future to past. Assume y is everywhere defined. Then the first item of x that can be computed is $x(10) = y(10)$. Then we can compute $x(9) = x(10) + 1$ and then proceed to $x(8)$ and so on. Let us intuitively call *non causal processes* those processes which may need such backward computations.

2.4 Kahn Theory

Kahn's world is a special case of Scott's world. In Kahn's world, the tag domain is $\mathbb{N}$, which is a totally-ordered and enumerable set. Signals are partial functions from $\mathbb{N}$ to a set of values V. In addition, all signals are assumed to be *prefix-closed*, meaning that if they are undefined at some time n then they remain undefined for all $n' > n$.

Note that in Kahn's original paper [7] signals are elements of $V^\infty = V^* \cup V^\omega$ where V^* is the set of all finite sequences over V and V^ω is the set of all infinite sequences over V. The set of all prefix-closed signals from $\mathbb{N}$ to V is isomorphic to V^∞: partially-defined signals correspond to finite sequences and totally-defined signals to infinite sequences.

Looking at signals x and y as sequences, $x \leq y$ means that x is a prefix of y (there exists a sequence x' such that $y = x \cdot x'$, where $x \cdot x'$ denotes the concatenation of x and x'). With this order, the set of all signals becomes a poset. It is in fact a CPO: (1) $\bot$ is the empty sequence ϵ; (2) the least upper bound of a chain of increasingly defined signals is either: (2.1) the most defined one if the chain is finite or: (2.2) the infinite sequence defined by the chain, because an infinite sequence is a maximal element of the CPO (concatenating a sequence to an infinite sequence does not change this sequence).

Processes are assumed to be continuous functions from S^m to S^n. Again we can define the semantics as the least solution of systems of equations. Thus the Kahn theory solves the feedback loop problem. But it also solves the causality problem: a continuous process is order preserving and, in terms of Kahn order, this means that if x is a prefix of y, it is also the case that $f(x)$ is a prefix of $f(y)$. This means that the future of x cannot influence the present of x. Computations are guaranteed to flow from past to future.

2.5 Kahn Generalisation

Kahn signals can be seen as tagged signals over the particular tag set $\mathbb{N}$. In order to get rid of non-causal processes that can arise from Scott's theory, we would like to generalise Kahn's approach to other tag sets. The technical difficulty here is that in general, tag signals cannot be prefix closed because prefixes can be infinite. While in [9] this problem is solved by introducing a special "absent" value, [11] proposes an alternative, more general approach which does not require introducing absent values while still being compatible with infinite prefixes.

Let S be the set of all functions $\mathbb{T} \mapsto V^{\perp}$, where $\mathbb{T}$ is a poset. Following Kahn, let us endow S with a partial order, defined as:

Definition 2.5.1 (Prefix order over signals)**.** A signal x is a prefix of a signal y, if for all t, $y(t) \neq x(t)$ implies for all $t' \geq t$, $x(t') = \perp$.

It is easy to see that this is indeed an order relation. Please note also that this definition allows "holes" in the defined values; for instance we could have:

t	1	2	3	4	5	$6\ldots$
x	1	$\perp$	2	$\perp$	$\perp$	$\perp\ldots$
y	1	$\perp$	2	$\perp$	3	$\perp\ldots$

In the above example we have indeed $x \leq y$.

Definition 2.5.2 (Order-Preserving Process)**.** A process p is order-preserving if, for any two signals, x, y if x is a prefix of y, then $p(x)$ is a prefix of $p(y)$.

This means that only the past can influence the present value of a process.

We can understand now why the example of Section 2.3 is not order preserving: this is due to the fact that if_9 is not order preserving. Take, for instance:

$$\begin{aligned} y(n) &= n \\ x_1(n) &= \perp \\ x_2(n) &= \begin{cases} 0 & \text{if } n = 0 \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

Thus, x_1 is a prefix of x_2. However, by letting

$$\begin{aligned} z_1(n) = \text{if}_9(x_1, y)(n) &= \perp \text{if } n < 9, n \text{ otherwise} \\ z_2(n) = \text{if}_9(x_2, y)(n) &= \perp \text{if } 0 < n < 9, n \text{ otherwise} \end{aligned}$$

we can see that z_1 is not a prefix of z_2, as $z_2(0) \neq z_1(0)$ and $z_1(10) \neq \perp$.

Proposition 2.5.1 (CPO)**.** *S endowed with the prefix order is a CPO.*

Proof. First, take $\perp(t) = \perp$. Then we notice that given $x \leq y$ in S, for any t, $x(t) \leq y(t)$ according to the CPO $V^{\perp}$. Thus, if $\{x_n\}$ is a chain, then, for any t, $\{x_n(t)\}$ is a chain and we can take:

$$\bigvee\{x_n\}(t) = \bigvee\{x_n(t)\}$$

□

Then, restricting to continuous processes allows us to preserve both *determinism and causality*.

2.6 From Generalised to Ordinary Kahn Theory

In this generalised Kahn theory, signals are partial functions from a partially ordered tag set $\mathbb{T}$ to a set of values V. These signals can thus be seen as "labelled partial orders" and are thus fairly general. Yet, in many cases, for instance those which correspond to what is considered in [11] and which are addressed at chapter 3, this generality is not needed and the approach can be simplified. This simplification is based on two assumptions and when these assumptions are in force, signals can be seen as streams and the theory reduces to an ordinary Kahn theory.

2.6.1 Simplifying assumptions

Assumption 1: *The tag set is a total order.*

Assumption 2: *The defined values of a signal can be indexed in non decreasing order.*
This means that there is an order preserving isomorphism from the domain of a signal ($dom(x) = \{t \mid x(t) \neq \perp\}$) to (an initial segment of) $\mathbb{N}$. We call this a *discrete signal*.

Combining these two assumptions yields signals whose defined tags are order isomorphic to (an initial segment of) $\mathbb{N}$.[2] In this case we speak of a *discrete total order*.

An important question arising from the following assumptions is whether the restriction of signals based on these assumptions still keep the CPO structure defined in 2.5.1. The following proposition provides a positive answer.

Proposition 2.6.1. *The set of discrete signals over a totally ordered tag set (DTOS) endowed with the prefix order is indeed a CPO.*

The proof requires an additional lemma:

Lemma 2.6.2. *Infinite DTOS signals (that is to say signals with an infinite number of defined values) are maximal elements among DTOS signals.*

[2]Note here the importance of requiring an order preserving isomorphism in assumption 2. Rationals are both totally ordered and countable but not order isomorphic to $\mathbb{N}$.

Proof. (lemma 2.6.2) Let x be a DTOS signal with an infinite number of defined values and assume by contradiction a DTOS signal x' such that $x < x'$. Then since $x \neq x'$, there exists a tag t such that $x(t) \neq x'(t)$ and since $x \leq x'$, for all $t' \geq t$, $x(t') = \bot$

Thus, $x'(t)$ is defined and, x' being DTOS, t has some finite index i in x'. Since $x \leq x'$, x has less defined values than x' before t and has no defined value after t. Since the tag set is a total order, it comes that x has less than i defined values and this contradicts the assumption according to which x has an infinity of defined values. □

Proof. (proposition 2.6.1) As previously we take $\bot(t) = \epsilon$ which obviously belongs to DTOS.

Then we consider a chain $\{x_0 \leq \dots x_n \leq \dots\}$ of DTOS signals. There are two cases:

1. there is some infinite signal x_n in the chain. Then according to the previous lemma, this is a maximal element and no other signal can be larger in this chain. Thus the chain has only a finite number of distinct signals and x_n is the largest one. The least upper bound of the chain is thus x_n.

2. every signal in the chain is finite. Each of them is indexed over an initial segment of $\mathbb{N}$ thus included into $\mathbb{N}$. The indexing sets form an increasing sequence of initial segments of $\mathbb{N}$, upper bounded by $\mathbb{N}$. The least upper bound of the chain can thus be indexed on $\mathbb{N}$and is therefore a DTOS signal.

□

2.6.2 DTOS signals as streams

When dealing with DTOS signals, tags associated to defined values can be indexed in increasing order and signals can be seen as streams of couples (value, tag) that is to say to a DTOS signal x we can associate its stream $sx : (V \times \mathbb{T})^\infty$ where there is no more need to consider an undefined value.

The tag ordering constraint is then for any $s \in DTOS$ such that $s = (v_1, t_1).(v_2, t_2).s'$,

1. $t_1 < t_2$
2. $s' \in DTOS$

It is then clear that the stream view of DTOS signals enjoys the same properties as the functional one. Formally the stream transformation is as follows:

$$\begin{aligned} St(\bot) &= \epsilon \\ St(x) &= (x(t_1), t_1).St(x[t_1 \rightarrow \bot]) \end{aligned}$$

where:

- t_1 is the least tag yielding a defined value in x
- $x[t_1 \rightarrow \bot]$ means the function x where the value at t_1 has been changed to $\bot$.

We can then formally state the property:

Proposition 2.6.3. *St is a CPO isomorphism between DTOS signals and streams.*

This view is inspired by some previous work due to some of the authors of the present report [3].

Chapter 3

Examples

3.1 Kahn Process Networks

Kahn Process Networks (KPN) naturally fit into that landscape. $\mathbb{N}$ is the tag set and there are no "holes" between defined values. Thus signals are just streams of defined values. An operator like the sum operator over numbers can be lifted to streams according to the following Haskell-like definition:

$$\begin{array}{lcl} \text{sum}_K\ \epsilon\ y & = & \epsilon \\ \text{sum}_K\ x\ \epsilon & = & \epsilon \\ \text{sum}_K\ v.x\ v'.y & = & (v+v').\text{sum}_K\ x\ y \end{array}$$

where ϵ denotes the empty stream and "." denotes concatenation. Basically, the Kahn actor sum_K waits until both its input queues are non-empty. Then, it removes their heads, adds them, puts the result into its output queue and starts again. Note that waiting until both queues are non-empty can be implemented using Kahn's *blocking read* operator, without having to test both queues *simultaneously*: sum_K simply blocks on one queue, then on the other. The order in which queues are read can be arbitrary.

3.2 Discrete Events

3.2.1 A Tagged View of Discrete Event Signals

Discrete event (DE) signals are discrete signals (according to assumption 1) with real-time stamps.

Tag set: A tag set for DE is:

$$\mathbb{T} = \mathbb{R}_+ \times \mathbb{N}$$

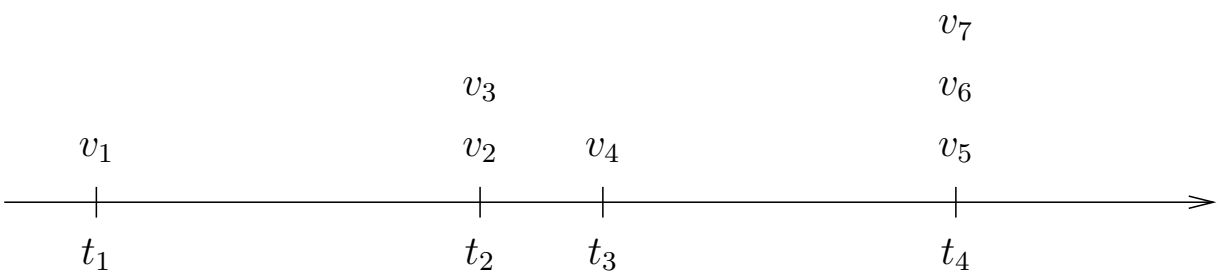

Figure 3.1: A discrete-event signal.

where if $\tau = (t, n)$, t denotes a time stamp and n the index of events sharing the same time stamp.[1] This tag set is ordered with the lexicographic order:

$$(t, n) \leq (t', n') \text{ iff either } t < t' \text{ or } t = t' \text{ and } n \leq n'$$

This has been called *super-dense time* by some authors [12, 11]. It is interesting to notice that this is a total order.

Then a discrete event signal x is a total function $x : \mathbb{T} \mapsto V^{\perp}$ such that the following constraint holds: for any two tags $\tau, \tau' \in \mathbb{T}$, with $\tau = (t, n)$ and $\tau' = (t, n')$ and $n \leq n'$, if $x(\tau') \neq \perp$ then $x(\tau) \neq \perp$.

Figure 3.1 shows an example of such a signal. In this example the following table provides the correspondence between tags and values:

tag	$t_1, 1$	$t_2, 1$	$t_2, 2$	$t_3, 1$	$t_4, 1$	$t_4, 2$	$t_4, 3$
value	v_1	v_2	v_3	v_4	v_5	v_6	v_7

We can remark that in this definition, there are many undefined (or absent) values namely between two consecutive time stamps holding defined values and after the last defined value sharing a given time stamp.

3.2.2 A Streamed View of Discrete Events

Since discrete event signals are discrete signals over a totally ordered tag set, they are DTOS signals and we can apply the results of section 2.6.1, thus providing a streamed view of them:

$$sx : (V \times (\mathbb{R}_+ \times \mathbb{N}))^{\infty}$$

where $\mathbb{R}_+$ is the set of non-negative reals modeling the physical (or *real*) time.

Furthermore we can remark that in this definition, the second component of the tag set is not necessary because we can always rebuild it: it suffices to add increasing indices to consecutive

[1] This approach could be easily extended to tag sets $\mathbb{T} = \mathbb{R}_+ \times \mathbb{N}^{\mathbb{N}}$ which could account for so-called "nested oversamplings". However, for the sake of simplicity, we do not address here such an extension.

events sharing the same timestamp in the stream. Formally, we can define an index rebuilding mapping Ir : $(V \times \mathbb{R}_+)^\infty \to (V \times (\mathbb{R}_+ \times \mathbb{N}))^\infty$ by:

$$
\begin{array}{lcl}
\text{Ir}_1(t', n, \epsilon) & = & \epsilon \\
\text{Ir}_1(t', n, (v,t).sx) & = & \text{if } t == t' \\
 & & \text{then } (v,(t,n+1)).\text{Ir}_1(t, n+1, sx) \\
 & & \text{else } (v,(t,1)).\text{Ir}_1(t, 1, sx) \\
 & & \\
\text{Ir}(\epsilon) & = & \epsilon \\
\text{Ir}((v,t).sx) & = & (v,(t,1)).\text{Ir}_1(t, 1, sx)
\end{array}
$$

3.2.3 An Actor Example

It is interesting to see how to define some primitive actors in DE. Let us start by trying to define the sum of two signals.

Sum: First Definition

We could define the sum as follows, by considering that DE signals are streams of pairs in $V \times \mathbb{R}_+$:

$$
\begin{array}{lcl}
\text{sum}_{DE1}\ \epsilon\ y & = & \epsilon \\
\text{sum}_{DE1}\ x\ \epsilon & = & \epsilon \\
\text{sum}_{DE1}\ (v,\tau).x\ (v',\tau').y & = & \\
 & & \text{if } \tau < \tau' \\
 & & \text{then sum}_{DE1}\ x\ (v',\tau').y \\
 & & \text{else if } \tau = \tau' \\
 & & \quad \text{then } (v+v',\tau).\text{sum}_{DE1}\ x\ y \\
 & & \quad \text{else sum}_{DE1}\ (v,\tau).x\ y
\end{array}
$$

But this definition is a bit unsatisfactory as, since events are countable, they may appear only rarely at the same time and the sum will be mostly undefined.

Sum: Second Definition

To correct the above problem, we attempt the following idea: output any signal which is defined at a given tag and output their sum when they are both defined:

$$
\begin{array}{lll}
\text{sum}_{DE2}\ x\ \epsilon & = & x \\
\text{sum}_{DE2}\ \epsilon\ y & = & y \\
\text{sum}_{DE2}\ (v,\tau).x\ (v',\tau').y & = & \\
\qquad \text{if } \tau < \tau' & & \\
\qquad \text{then } (v,\tau).(v',\tau').\text{sum}_{DE2}\ x\ y & & \\
\qquad \text{else if } \tau = \tau' & & \\
\qquad\qquad \text{then } (v+v',\tau).\text{sum}_{DE2}\ x\ y & & \\
\qquad\qquad \text{else } (v',\tau')'.(v,\tau).\text{sum}_{DE2}\ x\ y & &
\end{array}
$$

But this definition is not correct because it is not order-preserving. Consider, for example, $x = (1,1)$, $y = (1,1)(2,2)$ and $z = (1,2)(2,3)$. We have $x \leq y$, so if sum_{DE2} were order preserving we should have $\text{sum}_{DE2}\ x\ z \leq \text{sum}_{DE2}\ y\ z$. But $\text{sum}_{DE2}\ x\ z = (1,3)(2,3)$ and $\text{sum}_{DE2}\ y\ z = (1,3)(2,5)$ which are not comparable.

A way to correct this issue is to wait before outputting some item to be sure that it will not be outdated by a previous one appearing on the other input queue.

$$
\begin{array}{lll}
\text{sum}_{DE2}\ x\ \epsilon & = & \epsilon \\
\text{sum}_{DE2}\ \epsilon\ y & = & \epsilon \\
\text{sum}_{DE2}\ (v,\tau).x\ (v',\tau').y & = & \\
\qquad \text{if } \tau < \tau' & & \\
\qquad \text{then } (v,\tau).\text{sum}_{DE2}\ x\ (v',\tau').y & & \\
\qquad \text{else if } \tau = \tau' & & \\
\qquad\qquad \text{then } (v+v',\tau).\text{sum}_{DE2}\ x\ y & & \\
\qquad\qquad \text{else } (v',\tau')'.\text{sum}_{DE2}\ (v,\tau).x\ y & &
\end{array}
$$

Figure 3.2 illustrates this last definition which can be proved to be continuous though it uses, unlike in KPN, the infamous[2] operation that tests values in input queues without removing (consuming) them.

Sum: Third Definition

Another solution is to store values of events in registers (i.e., state variables). These registers, denoted v and v' below, should hold default values.

[2]It is called infamous because its undisciplined usage may result in non-continuous processes.

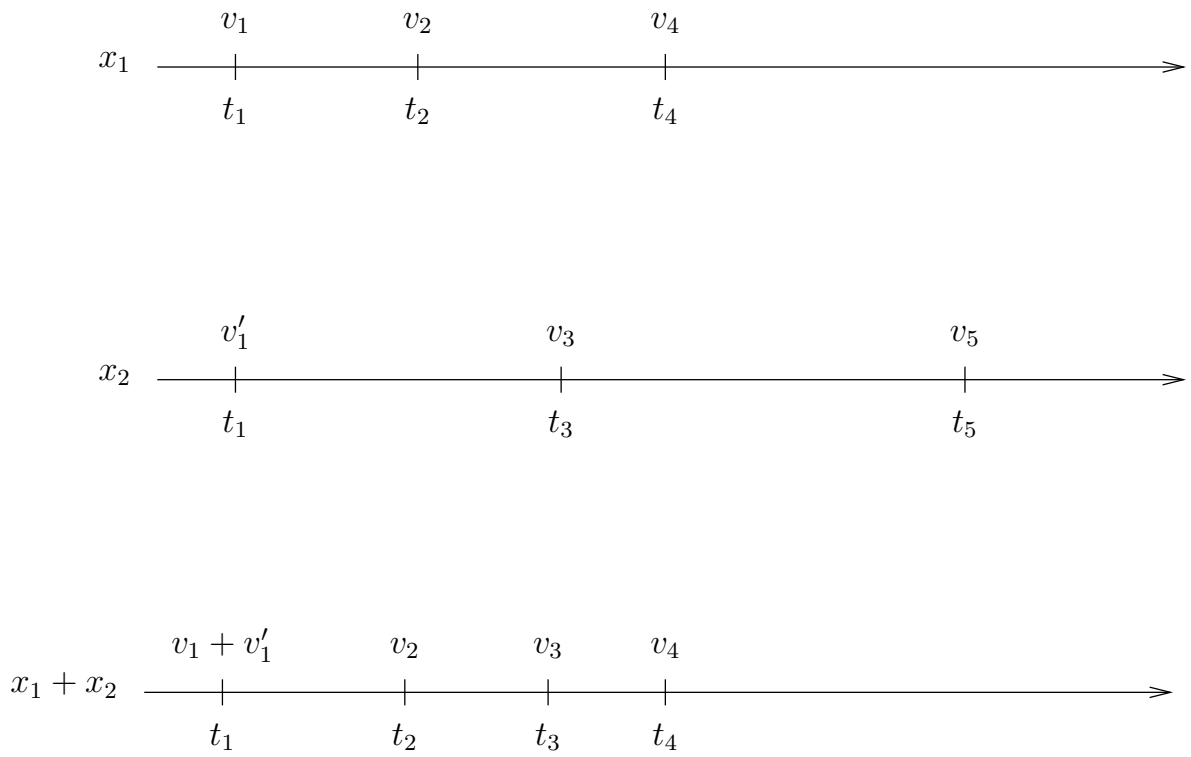

Figure 3.2: Sum (DE2) of two discrete-event signals.

$$
\begin{array}{lll}
\text{sum}_{DE3}\ (v_0, v_0')\ x\ \epsilon & = & \epsilon \\
\text{sum}_{DE3}\ (v_0, v_0')\ \epsilon\ y & = & \epsilon \\
\text{sum}_{DE3}\ (v_0, v_0')\ (v, \tau).x\ (v', \tau').y & = & \\
\quad \text{if } \tau < \tau' & & \\
\quad \text{then } (v + v_0', \tau).\text{sum}_{DE3}\ (v, v_0')\ x\ (v', \tau').y & & \\
\quad \text{else if } \tau = \tau' & & \\
\quad\quad \text{then } (v + v', \tau).\text{sum}_{DE3}\ (v, v')\ x\ y & & \\
\quad\quad \text{else } (v_0 + v', \tau').\text{sum}_{DE3}\ (v_0, v')\ (v, \tau).x\ y &
\end{array}
$$

Basically, each input event is bufferised and buffers are overwritten by a new occurrence of the same input. When an input event occurs, and we are sure that no event with the same tag is present in the other input channel, its value is added to the bufferised value of the other input and the result is output. When both input events are present with the same tag, their values are added and the result is output.

3.3 Synchronous Reactive

Synchronous reactive (SR) systems are very similar to discrete event ones, but real-time is replaced with a logical integer time. Thus the tag set is[3] $\mathbb{N} \times \mathbb{N}$ where the first component gives the reaction logical time and the second one the multiplicity index in that reaction. The reason we need both components is to model easily *multi-clock* systems: in such systems, a signal may be "absent" in some reactions As with DE signals, we can have a stream view of SR signals as follows:

$$sx : (V \times \mathbb{N})^{\infty}$$

In this interpretation $x(n) = (v, r)$ means the n-th occurrence of x has value v and takes place within the r-th reaction.

Thus, ordinary SR actors are very similar to those of DE. Yet in the case of the sum, it is likely that the first version will be chosen. Indeed, if it is unlikely that two independent events take place at the same time, it is not the case of synchronous events which can share the same reaction index.

A special treatment can be used for clock and boolean actors which are likely to follow the so-called constructive semantics of Berry [4]. For instance, a "logical-or" actor which receives a "true" value from one input does not need to wait for the other input. Indeed, be it present or not and whatever be its value, the constructive semantics allows outputting a "true" value at the corresponding reaction index.

[3] The same remark on a possible extension to nested oversamplings as stated at section 3.2.1 applies here.

$$
\begin{array}{lll}
\epsilon \text{ or } y & = & \epsilon \\
x \text{ or } \epsilon & = & \epsilon \\
(v_x, n_x).x \text{ or } (v_y, n_y).y & = & \\
\quad \text{if } n_x < n_y & & \\
\quad \text{then if } v_x & & \\
\quad\quad \text{then } (v_x, n_x).(x \text{ or } (v_y, n_y).y) & & \\
\quad\quad \text{else } x \text{ or } (v_y, n_y).y & & \\
\quad \text{else if } n_x > n_y & & \\
\quad\quad \text{then if } v_y & & \\
\quad\quad\quad \text{then } (v_y, n_y).(x \text{ or } y) & & \\
\quad\quad\quad \text{else } (v_x, n_x).x \text{ or } y & & \\
\quad\quad \text{else } ((v_x \vee v_y), n_x).(x \text{ or } y) &&
\end{array}
$$

3.4 Continuous Time

The case of continuous time is more involved. We can consider several issues here:

Exact Semantics

The issue of exact semantics is linked to the theory of ordinary differential equations and it has been addressed in terms of Cauchy-Lipschitz theorems on the existence of solutions [10] and even in terms of non-standard analysis [5]. Yet it seems to us that exact CT does not fit into the Kahn landscape: in order to exactly solve a differential equation, this equation has to be considered globally as a whole and it cannot be decomposed into its components. For instance we cannot define the exact behaviour of an integrator, independently from the network of operators which feeds it.[4]

Approximate Semantics

This is the realm of numerical solvers. There are many solutions and it is not clear which one fits into the Kahn landscape. Yet, we can claim here that, at least, Euler first order integration method fits into this landscape by providing the corresponding definition:

Given a differential equation $y' = f(y), y(t_0) = y_0$ which we rewrite as:

$$
\begin{array}{lll}
y(t) & = & y_0 + \int_{t_0}^{t} x(u)du \\
x & = & f(y)
\end{array}
$$

the Euler method proposes the iterative solution

$$y(t') = y(t) + x(t)(t' - t)$$

with the accuracy $a = |x'(t)|(t' - t)/2$.

[4] Indeed in theory it would need to check whether this network computes a Lipschitz function or not.

We can propose the following Haskell-like implementation. In this definition, the clock cl stands for the stream of requests at which the result is asked for, in general but not always a periodic sampling. The operator sum_{aux} is the running function. It stores the last value of the integral, the past derivative and the time at which it has taken place. When the new derivative arrives, it computes the second derivative x' and compares the requested step with the required accuracy a. If the accuracy is not met, the step is shortened so as to meet it. In this definition, we assume a close loop differential equation in such a way that the derivative is a function of the integral. In this case we know that the derivative cannot run faster than the past value of the integral.

$$
\begin{array}{lcl}
\text{sum}_{CT}\ (y_0)\ x\ \epsilon & = & \epsilon \\
\text{sum}_{CT}\ (y_0)\ \epsilon\ cl & = & \epsilon \\
\text{sum}_{CT}\ (y_0)\ (x,t).xs\ t'.cl & = & \\
\quad \text{if } t' \leq t & & \\
\quad \text{then } (y_0,t').\text{sum}_{CT}\ (y_0)\ (x,t).xs\ cl & & \\
\quad \text{else let } y = y_0 + x(t'-t) & & \\
\quad\quad \text{in } (y,t').\text{sum}_{aux}(y,t,x)\ xs\ cl &&
\end{array}
$$

where

$$
\begin{array}{lcl}
\text{sum}_{aux}\ (y_1,t_0,x_0)\ x\ \epsilon & = & \epsilon \\
\text{sum}_{aux}\ (y_1,t_0,x_0)\ \epsilon\ cl & = & \epsilon \\
\text{sum}_{aux}\ (y_1,t_0,x_0)\ (x,t).xs\ t'.cl & = & \\
\quad \text{let } x' = |x-x_0|/(t-t_0) & & \\
\quad \text{in if } x'(t'-t)^2 < 2a & & \\
\quad\quad \text{then let } y = y_1 + x(t'-t) & & \\
\quad\quad\quad \text{in } (y,t').\text{sum}_{aux}(y,t,x)\ xs\ cl & & \\
\quad\quad \text{else let } t'' = t + \sqrt{2a/x'} & & \\
\quad\quad\quad\quad y = y_0 + x(t''-t) & & \\
\quad\quad\quad \text{in } (y,t'').\text{sum}_{aux}(y,t,x)\ xs\ t'.cl &&
\end{array}
$$

We note that CT streams have exactly the same tag set as the DE domain. This is not surprising as, indeed, we implicitly work with sampled systems.

Here also we can show that this actor is continuous.

Threshold-Crossing

Zero and threshold crossing raises the problem of adapting the step of computation so as not to miss, exactly in ideal semantics, or with some precision in approximate semantics, the point in time where a given signal hits a threshold. Note that step adaptation is what we have performed in the Euler integrator. But in this previous case we had some accuracy computation method which allowed us to choose in advance what could be the next step. In the zero-crossing problem, we hardly have such a way of predicting the step and we in general know that the step was too large when the threshold has been missed. Then the solution is to recompute the signal with a shorter step but this recomputation goes backward in time and is not Kahn order preserving.

Perhaps surprisingly, the problem here is the approximate semantics,as acyclic models (without feedback loops) could be considered Kahnian in exact continuous time semantics. Indeed, threshold crossing does not seem to raise any difficulty in the ideal semantics because there is no feedback problem. It suffices that locally, the solution exists and if we move with an infinitely small step, we are granted to "hit" the threshold.

Chapter 4

Actors without Directors

4.1 Heterogeneity

Up to now, this Kahn theory applies to actors and signals sharing the same tag set. The question is then to deal with actors and signals defined on different tag sets. Notice that there is generally no "golden rule" saying what the meaning of composing actors with different tag sets should be. Therefore, a first step is to define *tag-conversion* actors, *i.e.,* heterogeneous actors operating on different tag sets. Then the problem is to extend the previous Kahn theory to encompass these heterogeneous actors. But this in our framework comes for free: the sum of CPOs is indeed a CPO and we will show how to extend each heterogeneous actor to operate on this unique sum CPO, retrieving in this way the previous theory. Let us first propose some tag-conversion actors.

Tag Conversion Actors

We give now some standard conversion actors to allow the interconnection of signals of different tags. These are only a few examples and other tag-conversion actors can obviously be defined.

From DE and SR to KPN Going from DE and SR to KPN can be done by "forgetting" the tag:

$$\begin{array}{lcl} \text{forget } \epsilon & = & \epsilon \\ \text{forget } (x,t).xs & = & x.\text{forget } xs \end{array}$$

From KPN to DE and SR In the opposite direction, a "timestamping" actor can be used. This actor uses a clock that specifies the timestamps:

$$\begin{array}{lcl} \text{timestamp } \epsilon\ cl & = & \epsilon \\ \text{timestamp } xs\ \epsilon & = & \epsilon \\ \text{timestamp } (x.xs)\ (t.ts) & = & (x,t).(\text{timestamp } xs\ ts) \end{array}$$

From DE to CT This can be done by "freezing" or "holding" the DE signal value between the times it changes. To this end we can define a "latch" actor that uses a sampling clock so as to know where to hold the DE signal value:

$$\begin{array}{lll} \text{latch } x_0 \in cl & = & \epsilon \\ \text{latch } x_0\ xs\ \epsilon & = & \epsilon \\ \text{latch } x_0\ ((x,t_x).xs)\ (t.ts) & = & \\ \qquad \text{if } t < t_x & & \\ \qquad \text{then } (x_0,t).(\text{latch } x_0\ ((x,t_x).xs)\ ts & & \\ \qquad \text{else latch } x\ xs\ (t.ts) & & \end{array}$$

It is worth noticing here that this actor provides a signal which is tagged on the tags of the clock. If the clock does not have multiple events sharing the same tag, this will be also the case in the resulting signal.

Let us see now how these actors can be seen as operating on the sum CPO.

Sum of CPOs

The *sum* of two CPOs S_1 and S_2 is a CPO $S_1 + S_2$, defined as

$$S_1 + S_2 = (S_1 - \{\epsilon_1\}) \cup (S_2 - \{\epsilon_2\}) \cup \{\epsilon\}$$

where ϵ is a "fresh" bottom element. The order on each of S_1, S_2 is maintained in the sum, but two elements from the two sets are not comparable. Therefore, chains can only be formed of elements of a single set and the $\bigvee$s are preserved.

Take for instance an actor $f : S_1 \to S_2$. It can be extended to an actor $f' : (S_1 + S_2) \to (S_1 + S_2)$ by:

$$\begin{array}{lll} f'\epsilon & = & \epsilon \\ f'(in_1\ x) & = & fx \\ f'(in_2\ x) & = & \epsilon \end{array}$$

where in_1, in_2 are the canonical injection of each CPO into the sum.

Thus we can see that, provided these very simple generalisations, the Kahn theory of heterogeneous deterministic tag systems works exactly the same. In particular, we can define tag changing actors allowing one to move from a tag system to another, that is to say from one MoCC to another. As any other actors, these actors should be continuous and as such preserve the order structure from the input tag set to the output tag set.

4.2 Distribution

It is well known that Kahn actors are intended to model deterministic distributed systems. In such distributed systems, each actor runs at its own pace in an unsupervised way. If it does not have data inputs, it waits. When it has data, it can consume them and can produce data outputs (Kahn networks). How can it be then that the behaviour is deterministic? This is due to

1. the communication medium between actors which keeps the production order of data (FIFOs),

2. fixpoint iterations which can be executed in a chaotic manner.

We claim here that our Kahn generalisation shares the same property: generalised Kahn actors can be linked together via FIFOs so as to form generalised Kahn networks and then can be executed in a chaotic manner, while being granted to produce the same results.

Take for instance the first definition of the discrete event sum. The actor has two input FIFO queues `x` and `y` and an output FIFO queue `z`. The queues contain pairs $(v : V, t : \mathbb{R}_+)$. Indeed the $\mathbb{N}$ component of the DE tag set is useless because the FIFO queues preserve the order of production. Thus an operational version of the sum actor can be defined in C-like syntax as:

```
void sum(input queue x, input queue y, output queue z) {
    if (x.empty() OR y.empty()) return;
    if (x.head().tag() < y.head().tag()) {
       x.erase_head();
       return;
    }
    if (x.head().tag() == y.head().tag()) {
       z.append(x.head().tag(), x.head().val() + y.head().val());
       x.erase_head();
       y.erase_head();
       return;
    }
    // it must be that: x.head().tag() > y.head().tag()
    y.erase_head();
    return;
}
```

Basically, the sum process needs that its two input files be non empty to execute. Otherwise it waits.

If it can execute, it takes the two tagged heads and compares their tags. If they are equal, it sums up the two values, tags the result with the common tag, puts it in the output queue and erases the two heads from the input queues.

If the two tags are different, the earlier tagged value is erased from its input queue (as a matter of fact we know that, since the input queue values are produced in an orderly manner, it will not be possible that the other queue will later contain an item matching this earlier tag) and the other queue is left unchanged. Nothing is produced and the process waits.

Indeed, we could say, adopting the Ptolemy terminology that such networks do not need directors, *i.e.,* some *deus ex machina* able to schedule the executions of each actor. In a simulation engine, the only need is that execution is fairly distributed between actors in such a way that no actor is infinitely excluded from execution. Please note also that there is no "event queue" like what is found in most simulation engines like Ptolemy and this feature avoids the burden of building a distributed event queue. Distributed actors are truly autonomous, they only know of the heads of their input queues.

4.3 Hierarchy

It has been advocated that the use of directors enforces a clean separation between several MoCCs in a hierarchical way: in order to get a communication between two different MoCCs these have to be encapsulated within a "larger" MoCC which encompasses the former ones. It is true that this is a good design practice but the "flat" directorless approach we present here is fully compatible with hierarchy: Kahn actors can be gathered so as to form compound actors and this hierarchical composition can be extended at will.

Chapter 5

Conclusion

This paper has intended to clarify some recent efforts proposed by the Berkeley school in order to give a formal semantics to the Ptolemy toolbox. We have shown that the proposed semantics is indeed a generalisation of a Kahn semantics based on tag systems and made this proposal a precise one by showing for several Ptolemy domains what is the corresponding tag system and how operators can be defined. In particular we have shown that so-called " absent" values do not seem to be necessary in the presence of tags. We also discussed some domains which do not appear to obey this semantics. We emphasized some of the appealing properties of this semantics, namely that it provides heterogeneity and distribution for free. In particular, we showed that while the semantics is naturally expressed in terms of actors, directors are not essential for semantical purposes and are only justified by simulation efficiency purposes. We thus can claim that, at least semantically, directors do not play an important part in the definition of a Model of Computation and Communication.

An important issue not addressed in the paper is the issue of liveness (also called productivity in the co-algebraic framework). This issue has already been partially addressed in [11] and its adaptation to our stream approach will be a subject for future work.

Another semantic theory for stream-based systems, alternative to Kahn, is the co-algebraic theory of streams (see for instance [6]). Basically, moving to co-algebraic streams would consist, in the stream programs shown in the paper, to remove the ϵ cases. Indeed, this is what has been done in the Haskell prototype we have implemented of our framework. Examining the consequences of such an alternative choice is also a subject for future work.

Bibliography

[1] A. Basu, M. Bozga, and J. Sifakis. Modeling Heterogeneous Real-time Components in BIP. In *SEFM '06: Proceedings of the Fourth IEEE International Conference on Software Engineering and Formal Methods*, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society. 1

[2] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. *IEEE Proceedings*, 79:1270–1282, September 1991. 1

[3] A. Benveniste, B. Caillaud, L. P. Carloni, P. Caspi, and A. L. Sangiovanni-Vincentelli. Composing heterogeneous reactive systems. *ACM Trans. Embedded Comput. Syst.*, 7(4), 2008. 1, 2.6.2

[4] G. Berry. The foundations of Esterel. In *Proof, language, and interaction: essays in honour of Robin Milner*, pages 425–454. MIT Press, 2000. 3.3

[5] S. Bliudze and D. Krob. Towards a functional formalism for modelling complex industrial systems. *ComPlexUs, Special Issue: Complex Systems - European Conference - Part 1, (ECCS'05)*, 2(3-4):163–176, November 2005. 3.4

[6] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of EATCS*, 62:229–259, 1997. 5

[7] G. Kahn. The semantics of a simple language for parallel programming. In *Information Processing 74, Proceedings of IFIP Congress 74*, Stockholm, Sweden, 1974. North-Holland. 1, 2.4

[8] E. A. Lee and A. Sangiovanni-Vincentelli. A unified framework for comparing models of computation. *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, 17(12):1217–1229, December 1998. 1, 2.1

[9] E.A. Lee and H. Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In *ACM Conf. Embedded Software (EMSOFT'07)*, 2007. 1, 1, 2.5

[10] J. Liu and E.A. Lee. On the causality of mixed-signal and hybrid models. In *HSCC*, pages 328–342, 2003. 3.4

[11] X. Liu and E.A. Lee. CPO Semantics of Timed Interactive Actor Networks. Technical Report UCB/EECS-2006-67, EECS Department, University of California, Berkeley, May 2006. 1, 2.5, 2.6, 3.2.1, 5

[12] Z. Manna and A. Pnueli. Verifying hybrid systems. In *Hybrid Systems*, pages 4–35, 1992. 3.2.1

[13] F. Maraninchi and T. Bouhadiba. 42: Programmable models of computation for a component-based approach to heterogeneous embedded systems. In *Sixth ACM International Conference on Generative Programming and Component Engineering (GPCE'07)*. ACM, 2007. 1

[14] D. Nowak. Synchronous structures. *Information and Computation*, 204(8):1295–1324, 2006. 1

[15] D. Scott. Data types as lattices. *SIAM Journal on Computing*, 10(3):522–587, 1976. 1, 2.2

Appendix A

Proofs

A.1 Continuity of the sum$_{DE2}$ actor

Consider two chains of inputs,

$$\begin{aligned} X &= \{x_1 \leq x_2 \leq \dots x_n \dots\} \\ Y &= \{y_1 \leq y_2 \leq \dots y_n \dots\} \end{aligned}$$

and the sequence of outputs

$$Z = \{z_1, z_2, \dots z_n \dots\}$$

where $z_n = \text{sum}_{DE2}\ (\bigvee\ X)\ y_n$

We wish to show that:

1. Z is indeed a chain: $\{z_1 \leq z_2 \leq \dots z_n \dots\}$
2. $\bigvee Z = \text{sum}_{DE2}\ (\bigvee\ X)\ (\bigvee\ Y)$

1: Let us take $y \leq y'$. In our stream view, there exists y'' such that $y' = y.y''$. We wish to show that $\text{sum}_{DE2}\ (\bigvee\ X)\ y \leq \text{sum}_{DE2}\ (\bigvee\ X)\ y'$. There are two cases:

Either y is infinite. Then $y' = y$ because concatenating a stream to an infinite stream cannot change it. Thus, $\text{sum}_{DE2}\ (\bigvee\ X)\ y = \text{sum}_{DE2}\ (\bigvee\ X)\ y'$.

Or y is finite. The proof is by induction:

Assume $y = \epsilon$. Then $\text{sum}_{DE2}\ (\bigvee\ X)\ y = \epsilon \leq \text{sum}_{DE2}\ (\bigvee\ X)\ y'$. y

Assume that the property holds for any $y_1 \leq y'1$ and consider $y = (v, \tau).y_1$. Then $y \leq y'$ implies that $y' = (v, \tau).y'_1$ for some y'_1 with $y_1 \leq y'_1$. There are then three cases:

- $\bigvee\ X = \epsilon$. Then we are done
- The maximum tag in $\bigvee\ X$ is smaller than τ. Then we never consume τ and the two outputs stay equal for ever.

- There is a tag in $\bigvee\ X$ equal or larger than τ. Then τ is consumed and the induction hypothesis can be applied.

2: Let $|x|$ denote the length (possibly infinite) of a stream x. It comes from the definition of sum_{DE2} that $|z_n| = |\bigvee\ X| + |y_n| - |(\bigvee\ X) \cap y_n|$, where $|x \cap y|$ denotes the number of tags shared in streams x and y.

It comes then that

$$|\bigvee\ Z| = \lim_{n\to\infty} |z_n| = |\bigvee\ X| + \lim_{n\to\infty} |y_n| - |(\bigvee\ X) \cap y_n| = |\bigvee\ X| + |\bigvee Y| - |(\bigvee\ X) \cap \bigvee\ Y|$$

Thus $\bigvee Z$ and $\text{sum}_{DE2}\ (\bigvee\ X)\ (\bigvee\ Y)$ have equal (possibly infinite) lengths and cannot be but equal.

Appendix B

A Haskell Implementation

B.1 The Untimed Library

This corresponds to untagged streams that is to say to Kahn Process Network.

```
module Untimed
where

-- constant

const_u x = x : (const_u x)

-- unary nodes

unary_u f (x:xs) = (f x) : (unary_u f xs)

-- the Untimed delay operator

delay_u x xs = x : xs

-- binary nodes

binary_u f (x:xs) (y:ys) = (f x y):(binary_u f xs ys)

-- Untimed when

when_u (True:cs)  (x:xs) = x:(when_u cs xs)
when_u (False:cs) (x:xs) = (when_u cs xs)

-- Untimed merge

merge_u (True:cs)  (x:xs)  ys    = x:(merge_u cs xs ys)
merge_u (False:cs)   xs  (y:ys)  = y:(merge_u cs xs ys)
```

```
-- display the first n elements

display_u n xs  = let disp p (x:xs) =
                          if p<=0 then []
                          else ((n-p),x):(disp (p-1) xs)
                  in (disp n xs)
```

B.2 The Timed Library

It corresponds to both DE and SR domains, due to the polymorphic properties of Haskell which allows us to not immediately distinguish between real and integer tags. Note that we use here a different convention by reversing values and tags.

```
module Timed -- specialisation to Synchronous Reactive and Discrete Events
             -- will follow
where

-- constant

const_t x  (t:ts) = (t,x) : (const_t x ts)

-- unary nodes

unary_t f ((t,x):xs) = (t,(f x)) : (unary_t f xs)

-- the Timed latch operator

latch_t xo (t:ts)  ((tx,x):xs)  = if    t<tx
                                  then (t,xo):(latch_t xo ts ((tx,x):xs))
                                  else (latch_t x (t:ts) xs)
-- binary nodes (synchronous)

binary_t f ((tx,x):xs)
           ((ty,y):ys) = if tx < ty
                         then binary_t f xs ((ty,y):ys)
                         else if tx == ty
                              then (tx,(f x y)):binary_t f xs ys
                              else binary_t f ((tx,x):xs) ys

-- clock extraction

clock_t ((t,x):xs) = t:(clock_t xs)
```

```
-- latched binary operators

binary_latched f (xo, yo) (xs, ys) =
    let tx = clock_t xs
        ty = clock_t ys
    in binary_t f (latch_t xo ty xs) (latch_t yo tx ys)

-- Interfaces with Untimed

-- From Untimed to Timed

timestamp (t:ts) (x:xs) = (t,x):(timestamp ts xs)

-- From Timed to Untimed

forget ((t,x):xs) = x:(forget xs)

-- display the first n elements

display_t n ((t,x):xs) = if n<=0 then []
                         else (t,x):(display_t (n-1) xs)
```

B.3 A Continuous Time Integrator

We provide here an improved version of an Euler integrator. Improvements come from the fact that it can work in a fully distributed manner.

```
module Continuous
where

integr1 (to,yo,t1,x1,er) ((tx,x):xs) (t:ts) =
    if tx < to
    then integral(to,yo,tx,x,er) xs (t:ts)
    else let xi = x1 + (x - x1)*(to-t1)/(tx-t1)
             xp = abs(xi-x1)/(to - t1)
         in  if xp*(t-to)**2 < 2*er
             then let y = yo + (t-to)*x
                  in (t,y):integral (t,y,to,xi,er) ((tx,x):xs)  ts
             else let tn = to + sqrt(er*2/xp)
                      y = yo + (tn - to)*x
                  in (tn,y):integral (tn,y,to,xi,er) ((tx,x):xs) (t:ts)
```

```
integral (to,yo,t1,x1,er) xs (t:ts) =
    if (to < t)
    then integr1(to,yo,t1,x1,er) xs (t:ts)
    else if to==t
         then (to,yo):integral (to,yo,t1,x1,er) xs ts
         else integral (to,yo,t1,x1,er) xs ts

-- a filter that eliminates intermediate values

filter_t xo ((tx,x):xs) (t:ts) =
    if   t<tx
    then (t,xo):filter_t xo ((tx,x):xs) ts
    else if tx == t
         then (tx,x):filter_t x xs ts
         else filter_t x xs (t:ts)

-- A periodic clock generator

incr p (x:xs) = (x + p):incr p xs

p_clock (i,p) = let cl = i:incr p cl
                in cl
```

B.4 Some Test Code

Here is a module in order to test the integrator:

```
   module Test_continuous
where

import Untimed
import Timed
import Continuous


-- a periodic clock for sampling results

p_cl = p_clock(0.0,1.0)

-- a differential equation xdot = -x, x(0) = 1

minus  = unary_t (\x->(-x))
```

```
set1 = (0.0, 1.0, -1.0, 0.0, 0.0001) -- settings for the integral

-- simple differential equation

test1 =
    let x = integral set1 (minus x) p_cl
    in (display_t 10 x)

test3 =
    let x = integral set1 (minus x) p_cl
    in (display_t 10 (filter_t 0.0 x p_cl))

-- integrating a constant. Note the need for feedback

set2 = (0.0, 0.0, -1.0, 1.0, 0.0001)

test2 =
    let y = integral set2 (const_t 1.0 (0.0:clock_t y)) p_cl
    in (display_t 10 y)

test4 =
    let y = integral set2 (const_t 1.0 (0.0:clock_t y)) p_cl
    in (display_t 10 (filter_t 0.0 y p_cl))


-- double integrator

setsin = (0.0, 0.0, -0.1, 1.0, 0.0001)
setcos = (0.0, 1.0, -0.1, 0.1, 0.0001)
p_cl2 = p_clock(0.0,0.1)

test5  = let sin = integral setsin  cos p_cl2
             cos = integral setcos  (minus sin) p_cl2
         in display_t 30 (filter_t 0.0 sin p_cl2)

test6  = let sin = integral setsin  cos p_cl2
             cos = integral setcos  (minus sin) p_cl2
         in display_t 30 (filter_t 0.0 cos p_cl2)

test7  = let sin = integral setsin  cos p_cl2
             cos = integral setcos  (minus sin) p_cl2
         in display_t 20 cos
```