

Specification of Communication Protocols: Extending timing diagrams for that purpose.*

© Jörg Fischer
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme.
jfisher@iti.cs.uni-magdeburg.de

June 10, 1999

Abstract

This paper discusses ideas for an approach to specification which was initiated by the analysis of the peculiarities of communication protocols. The primary intuition is given by *timing diagrams* which are used by electronic engineers, for instance, in order to describe the functionality of digital circuits.

This approach is well-suited for description of parallel processes and synchronization between them. An example for their need is the wires of a hardware interface. Another example for synchronized parallel processes is communication partnership which is treated herein.

Timing diagrams are effortlessly understandable. Besides, engineers use them in practice. This paper deals with extension and abstraction of them such that one can use them as a specification tool.

Contents

1	On protocol specification in general.	2
1.1	On its purpose.	2
1.2	On formal semantics.	3
1.3	Verification and prototyping.	3
1.4	Specification of behavior.	3
1.5	Timing diagrams.	4

*Supported by the DFG under project no. NE 592/4 - 1 as well as the FireWork project.

2	The core language.	5
2.1	Templates.	5
2.2	Process types, attributes and conditions.	6
2.3	Methods.	7
2.4	Networks.	8
2.5	Axioms.	13
2.6	Elements.	13
2.7	Semantics.	13
3	Taking elements as templates.	14
4	Conclusion.	14
4.1	Discussion.	14
4.2	Outlook.	15
4.2.1	Object oriented concepts.	15
4.2.2	Verification and prototyping.	15
4.2.3	Real-time issues.	15
4.3	Acknowledgments.	16

List of Figures

1	Character transmission — 5 bit, 1 stop bit.	4
2	Extract from a timing diagram.	4
3	Timing diagram with condition and conclusion.	5
4	Example of a network.	8
5	Module instances.	8
6	A network context and a network.	10
7	Fitting of a network into a network context.	11
8	Two axioms.	13

List of Tables

1	Standard methods.	8
2	Base equations between networks.	12
3	Equations of the real-time process.	15

1 On protocol specification in general.

1.1 On its purpose.

In the practice of computer and electronic engineers one can find many examples where two or more digital devices are connected in order to exchange data. Some of them are

- digital TV,

- digital integrated circuits,
- computer interfaces,
- telecommunication systems.

There is a need for them of a common ‘tongue’, named ‘protocol’, for each partner which interchanges data. The definition of those protocols can be very complex such that

- engineers and people who work on the definition and standardization of protocols need to take much time and effort for keeping track,
- incomplete understanding and therefore misunderstanding of the entirety of a protocol leads to malfunctions of the communication system.

Another problem is to find an unambiguous language for the description of protocols in order to exclude misunderstanding between engineers, developers, and other interested people.

1.2 On formal semantics.

The latter problem forces investigation in semantics of protocol description techniques. The most precise form is to use mathematical means for which one uses the term ‘*formal semantics*’. Besides, a precise mathematical description of a protocol is necessary for performing verification. For practical use formal semantics should

- be effortlessly understandable such that the understanding of the elements of that semantics is not a problem itself,
- be structured such that complex situations can be decomposed.

1.3 Verification and prototyping.

Specifications are the base for verification and prototyping. Verification means checking properties of that thing which was specified. Prototyping is to create an implementation which fulfills the specification in order to test it. Both of them would be fine if they support object paradigm. That means that local properties should imply global ones if one wishes to verify them and the global implementation should use the local ones.

1.4 Specification of behavior.

Specification of protocols is specification of behavior of synchronized concurrent processes. Imagine, a hardware interface has many wires which work in principle independently of each other, but they have to fulfill some synchronization condition. For instance, a signal can stop or trigger a data transmission which is on another wire in that moment or as soon as the current block is transferred.

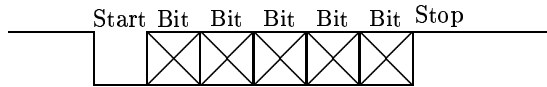


Figure 1: Character transmission — 5 bit, 1 stop bit.

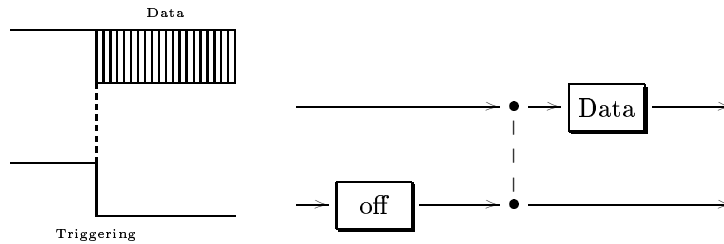


Figure 2: Extract from a timing diagram.

Actually, the time correspondence of processes like those of the wires in this example plays an important role. That is why we focus on this aspect in our approach.

1.5 Timing diagrams.

Pictures as like those in Figure 1 are used by electronic engineers. This picture could describe the usual protocol of a serial transmission of a 5-bit-character. If no character is being sent the wire is on level 'high'. As soon as the level toggles to 'low' a character is coming.

But, not only the behavior on one wire has to be expressed. There is need for synchronization between many wires as in the left hand side of Figure 2. That says that first the wire which is at the bottom in that picture toggles the signal level from 'high' to 'low' and then a thing called 'Data' comes on the wire which is on the top in that picture. Please note that dashed lines depict simultaneity. Without that the behavior of wires has no temporal relationships to each other. That kind of proposition we can also express in that diagrams.

But, there are still two problems with these diagrams so that we extend them by frames as in the right hand side of Figure 3.

- Because mostly these timing diagrams are infinite, we have to investigate how to present timing diagrams finitely.

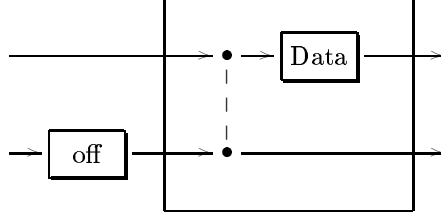


Figure 3: Timing diagram with condition and conclusion.

- One is not able to express *causal* relationship. For instance, the picture on the right hand side of Figure 2 says that ‘off’ is performed and then ‘Data’ is performed.
However Figure 3 want to say that occurrence of ‘Data’ is *logical premise* that ‘off’ happened. One can interpret that as follows: If one observed ‘Data’ then one know that ‘off’ had preceded because this is a *logical consequence* of the occurrence of ‘Data’.
The framed parts contains the conditions (‘if ...’!) and the part outside of that frame is the conclusion (‘then ...’!) of such diagrams.

2 The core language.

2.1 Templates.

Templates play the role of unstructured process specifications in our framework. A template T is a tuple

$$(P_T, A_T, M_T, R_T) \quad (1)$$

whereby:

- P_T is the set of all *process types* which are in use.
- $A_{T,p,d}$ is the set of *attributes* for each process type p and *data type* d . There are a flock of *data types* which are predefined for the sake of simplicity and have predefined *instances*. The purpose of attributes is, roughly spoken, to represent *states*.
- $M_{T,A,B}$ is the set of *methods* for every two $A, B \in P^*$. P^* is the free monoid over P . Intuitively, the methods represent *changes of state* in our approach.
- R_T is the set of *axioms*.

A *signature* Σ is a tuple like that

$$\Sigma = (P_\Sigma, A_\Sigma, M_\Sigma) \quad (2)$$

where the symbols in it has respective meaning as above. Each template T has a signature Σ_T .

Where does the name of the term ‘process type’ come from? In order to avoid misunderstandings in the following explanations I have to give some remarks on the intuition behind the term ‘process’.

The first one says that processes are *persistent* objects that means objects whose identities remain during their life-time despite all changes, state transitions, transformations and whatsoever. For instance, specification language TROLL ([JSHS96]) shared to that view.

However, someone has another imagination when he use the term ‘process’. They think that processes are the same thing as states because they argue that a process after transforming, changing or whatsoever is another process. The CCS-calculus ([Mil80]) and its forthcoming calculi as well as the approach presented in this paper are examples for this understanding.

Now, we are going to describe these parts, in particular, their structure.

2.2 Process types, attributes and conditions.

Given a set of *process types* P and a set of *data types* D . Then we have to fix a family of attributes:

$$A := (A_{p,d} \mid p \in P, d \in D) \quad (3)$$

Syntactically, we do this by the following construction:

```

att  p
      a1 : d1
      ⋮
      am : dm
end

```

wherein $m \geq 0$. Its meaning is

$$p \in P \quad (4)$$

$$d_i \in D \text{ for each } i \in \{1, \dots, m\}. \quad (5)$$

$$a_i \in A_{p,d_i} \text{ for each } i \in \{1, \dots, m\}. \quad (6)$$

As mentioned before we use only predefined data types herein. The only data type which is considered in this paper is the enumeration type:

$$\{i_1, \dots, i_n\} \quad (7)$$

with $n \geq 1$ where i_n are the instances.

Example 1 (Wire.)

```

att  Wire
      Lev : {◦, •}
end

```

Now, we can build *conditions* from that. At least we need a family

$$C := (C_p \mid p \in P) \quad (8)$$

which is closed under

$$\frac{\mathcal{C} \subseteq C_p \quad \mathcal{C} \text{ finite}}{\bigwedge \mathcal{C} \in C_p} \quad (9)$$

and a *consequence relation* \vdash

$$\vdash := (\vdash_p \subseteq C_p \times C_p \mid p \in P) \quad (10)$$

where

$$\frac{}{a \vdash a} \quad \frac{a \vdash b \quad b \vdash c}{a \vdash c} \\ \frac{a \in A}{\bigwedge A \vdash a} \quad \frac{\forall a \in A. (x \vdash a)}{x \vdash \bigwedge A}$$

We will denote:

$$\mathbf{tt} := \bigwedge \emptyset \quad (11)$$

$$a \wedge b := \bigwedge \{a, b\} \quad (12)$$

For the purposes of this paper we define C as simple as possible. It can easily be seen that one could use more sophisticated concepts for the definition of term ‘condition’: There should be the following form of base conditions:

$$\frac{p \in P \quad d \in D \quad a \in A_{p,d} \quad x \in d}{a = x \in C_p} \quad (13)$$

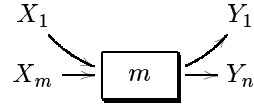
wherein $x \in d$ means that x is an instance of data type d .

2.3 Methods.

The next step is to define a set family of *methods*:

$$M := (M_{A,B} \mid A, B \in P^*) \quad (14)$$

wherein P^* is the free monoid over P . We use a graphical representation for a method $m \in M_{X_1 \dots X_m, Y_1 \dots Y_n}$:



The sets of methods have to contain the *standard methods* (Table 1).

Example 2 (Wire.) (*cf. example 1.*) *The methods of a wire are:*



We omit the sorts in these pictures if they are obvious.

<p>Synchronization</p> $\begin{array}{c} x \rightarrow \bullet \rightarrow x \\ \\ y \rightarrow \bullet \rightarrow y \end{array} \quad x, y \in P$	<p>Condition</p> $\begin{array}{c} c \\ \downarrow \\ x \rightarrow x \end{array} \quad x \in P, c \in C_x$
<p>Birth</p> $* \rightarrow x \quad x \in P$	<p>Death</p> $x \rightarrow \dagger \quad x \in P$

Table 1: Standard methods.

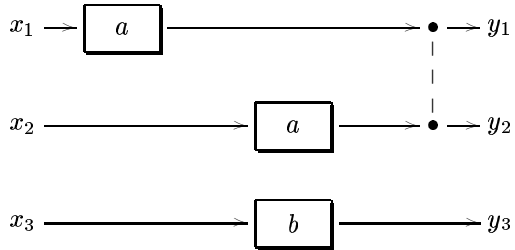


Figure 4: Example of a network.

2.4 Networks.

Networks. Before considering axioms we have to deal with their components — the *terms* or *networks*. A network will be a class of so-called *network presentations* factorized by the α -equivalence.

But before the precise definition will be given some terms which play an important role are illustrated in order to support the reader's imagination.

Let us have a look at Figure 4! We want to define that kind of picture. What do we need for that purpose? Firstly the boxes, cf. Figure 5, which are referred to as *method instances*. Each method instance belongs to a method which is

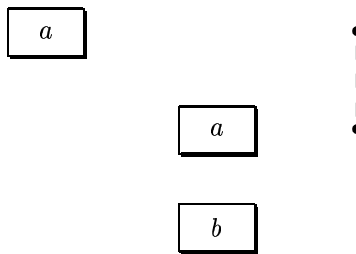


Figure 5: Module instances.

inside of the box in the graphical representation.

The arrows in Figure 4 are the *nodes* of a network.

Because the arrows represent the chronological order we need a notion of *acyclicity*.

Now, that is the summarizing definition:

Definition 1 (Network presentation.) *A network presentation N with respect to a signature Σ between $(\phi_1, \dots, \phi_m) \in \nu_{N,X}$ and $(\psi_1, \dots, \psi_n) \in \nu_{N,Y}$ with $m, n \geq 0$ is a tuple*

$$(\mu_N, \nu_N) \quad (15)$$

wherein

- ν_N is the family of pairwise disjoint set of nodes

$$\nu_N := (\nu_{N,p} \mid p \in P_\Sigma) \quad (16)$$

hereafter we define

$$\nu_{N,p_1 \dots p_k} := \nu_{N,p_1} \times \dots \times \nu_{N,p_k} \quad (17)$$

- μ_N is a set family of method instances

$$\mu_N := (\mu_{N,A,m,B} \mid m \in M_{U,V}, U, V \in P_\Sigma^*, A \in \nu_{N,U}, B \in \nu_{N,V}) \quad (18)$$

wherein with $r, s \geq 0$ for each $1 \leq i \leq r$ and $1 \leq j \leq s$

$$\frac{u \in \mu_{N,(\alpha_1, \dots, \alpha_r), m, C} \quad v \in \mu_{N,(\beta_1, \dots, \beta_s), n, C} \quad \alpha_i = \beta_j}{u = v}$$

$$\frac{u \in \mu_{N,A,m,(\gamma_1, \dots, \gamma_r)} \quad v \in \mu_{N,A,n,(\delta_1, \dots, \delta_s)} \quad \gamma_i = \delta_j}{u = v}$$

- acyclicity holds. That means that the smallest relation on ν_N , the temporal order \leq_N , which fulfills the following rules

$$\frac{\alpha \in \nu_{N,p}}{\alpha \leq_N \alpha}$$

$$\frac{\alpha \leq_N \beta \quad \beta \leq_N \gamma}{\alpha \leq_N \gamma}$$

$$\frac{u \in \mu_{N,(\alpha_1, \dots, \alpha_r), m, (\beta_1, \dots, \beta_s)}}{\alpha_i \leq_N \beta_j}$$

is a partial ordering. This holds, if the following rule is valid:

$$\frac{\alpha \leq_N \beta \quad \beta \leq_N \alpha}{\alpha = \beta}$$

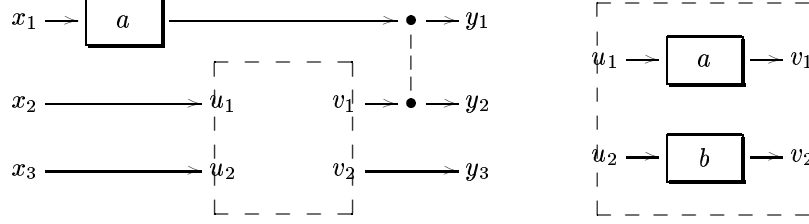


Figure 6: A network context and a network.

α -conversion. An α -conversion of a network presentation N with respect to a signature Σ to a network presentation N' with respect to the same signature is a bijection

$$\iota = (\iota_p \in \nu_{N,p} \cong \nu_{N',p} \mid p \in P_\Sigma) \quad (19)$$

$$\iota_{p_1 \dots p_k} := \iota_{p_1} \times \dots \times \iota_{p_k} \quad (20)$$

together with the following family

$$(\iota_m \in \mu_{N,G,m,H} \cong \mu_{N',G',m,H'} \mid m \in M_{U,V}, (G, G') \in \iota_U, (H, H') \in \iota_V) \quad (21)$$

It can easily be seen that the set of network presentations reachable by α -conversions is an equivalence class which we call *network*. The family of networks should be denoted as follows:

$$N := (N_{X,Y} \mid X, Y \in P_\Sigma^*) \quad (22)$$

$N_{X,Y}$ contains all networks between X and Y .

Network contexts. Again, some pictures (see Figure 6 and Figure 7) will illustrate the following rather complex definitions.

Definition 2 (Network context.) A network context C with respect to a signature Σ from $X = X_1 \dots X_m \in P_\Sigma^*$ and $V = V_1 \dots V_b \in P_\Sigma^*$ to $U = U_1 \dots U_a \in P_\Sigma^*$ and $Y = Y_1 \dots Y_n \in P_\Sigma^*$ is a network between $(x_1, \dots, x_m, v_1, \dots, v_b) \in \nu_{N, X, V}$ and $(y_1, \dots, y_n, u_1, \dots, u_a) \in \nu_{Y, U}$ wherein there is no sequence from v_j to u_i for each $1 \leq i \leq a, 1 \leq j \leq b$. That is $v_j \not\prec_C u_i$.

As it can readily be seen, networks can be regarded as network contexts wherein $a = b = 0$.

Definition 3 (Network fitting.) A network fitting $C[N]$ of a network N between $U = U_1 \dots U_m$ and $V = V_1 \dots V_n$ and a network context C from X and V to U and Y is that network between X and Y having an α -representant¹ which is defined as follows:

¹That is a network presentation!

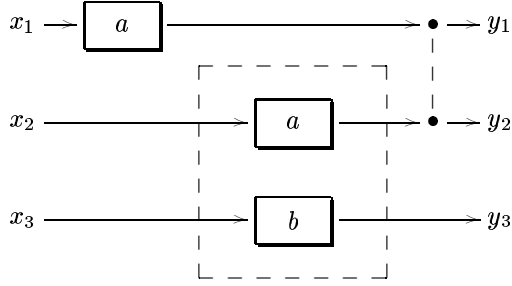


Figure 7: Fitting of a network into a network context.

- $\nu_{C[N]}$ is defined as follows:

$$\nu_{C[N]} := \nu_C \cup \nu_N \quad (23)$$

wherein one has to choose an α -representant of C from G and R to H and S and an α -representant of N between R and S with $R =: (\phi_1, \dots, \phi_m)$ and $S =: (\psi_1, \dots, \psi_n)$ such that

$$\nu_C \cap \nu_N = \{\phi_1, \dots, \phi_m\} \cup \{\psi_1, \dots, \psi_n\} \quad (24)$$

- $\mu_{C[N]}$ is defined as follows:

$$\mu_{C[N]} := \mu_C \cup \mu_N \quad (25)$$

wherein one has to choose α -representants of C and N such that

$$\mu_C \cap \mu_N = \emptyset \quad (26)$$

Finite static implication. The network fitting is the base of the semantics of our approach. Beside this, one has to pay attention that this approach needs some equations, see Table 2.

The *finite static implication* relation $\sqsubseteq_{\mathbf{fn}} := (\sqsubseteq_{\mathbf{fn}, X, Y} \subseteq N_{X, Y} \times N_{X, Y} \mid X, Y \in P_{\Sigma}^*)$ is the smallest partial order which contains the pairs presented in Table 2 and obeys the condition of *congruence*

$$\frac{M \sqsubseteq_{\mathbf{fn}} N \quad Q \sqsubseteq_{\mathbf{fn}} R}{M [Q] \sqsubseteq_{\mathbf{fn}} N [R]}$$

whereby $M \sqsubseteq_{\mathbf{fn}} N$ means the relation on the underlying networks of the contexts M and N .

Finite networks. In the sequel we use *finite* network presentations, networks and network contexts which consist of finitely many nodes and finitely many module instances.

	=			
	=			
	=		=:	
a 	=	b 	=:	$a \wedge b$
c 	=	c 	=:	c
tt 	=			
b 	\sqsubseteq_{fn}	a 	iff	$a \vdash b$
	\sqsubseteq_{fn}			
$\rightarrow \dagger$	*	\rightarrow		
\vdots	\vdots	\vdots	\sqsubseteq_{fn}	ϕ
$\rightarrow \dagger$	*	\rightarrow		

Table 2: Base equations between networks.

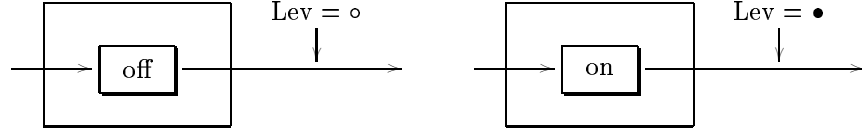


Figure 8: Two axioms.

2.5 Axioms.

A *proposition* with respect to a signature Σ is a pair

$$(Q, C [Q]) \quad (27)$$

with a finite network Q and a finite network context C such that one can build $C [Q]$. We present them pictorially by framing Q .

Example 3 (Wire.) (*continuation of example 1 and example 2:*) In Figure 8 one can see examples for axioms. Their intuitive meaning is that application of method ‘on’ forces the attribute *Lev* to have value \bullet and application of method ‘off’ forces this attribute to have the value \circ . The framed part of an axiom represent its condition. The part outside of the frame represents its effect.

For each template T there is a set R_T of propositions whose elements are referred to as *axioms*.

2.6 Elements.

An *element* x of a template T is an element of one of its parts: either a process type or an attribute or a condition or a method or an axiom of that template. We express that x is an element of a template T by:

$$x \in T \quad (28)$$

Subtemplates S of a template T are templates wherein for each element $x \in S$ obeys $x \in T$. We will denote this relation as follows:

$$S \subseteq T \quad (29)$$

2.7 Semantics.

Now, we deal with the semantics of a template which is determined by the *model* relation $_ \models_T _$: It is the smallest relation where

$$\frac{\frac{Q \sqsubseteq_{\mathbf{an}} R \quad Q \models_T a}{R \models_T a} \quad \forall X. (X [Q] = R \rightarrow \exists Y. X = Y [C])}{R \models_T (Q, C [Q])} \quad (30)$$

A *network* R of a template T is a *model* of a template S wherein $S \subseteq T$ noted as follows:

$$R \models S \tag{31}$$

iff for each axiom $a \in A_S$ it holds $R \models_T a$.

3 Taking elements as templates.

Because we must not tract too many things if we regard concepts of modularization now we mention that each process type, each attribute and each method should be regarded as a template:

Process type. The template for a process type p contains only

```
att  p
end
```

Attribute. The template for an attribute a of the process type p and the data type d contains only

```
att  p
      a : d
end
```

Method. The template for a method contain its picture and the required definitions of attribute and process types.

4 Conclusion.

4.1 Discussion.

- Some specification systems like SDL, ESTELLE and LOTOS ([Tur93]) are already in use and are standardized. These systems are the result of a long way. The developers of them had spent much effort to provide comfortable expression means for many levels of specification and engineering. Therefore, these systems are very complex. For many reasons, for instance for investigation in formal semantics and verification, one want to compose complex systems as SDL from a small set of base principles. The proposed kind of axiom is the only in this approach and I believe this is enough for the special purpose for which our approach was created.
- This paper had extended representation with *timing diagrams* such that one can use it as a specification tool. Timing diagrams are effortlessly understandable. Besides, engineers are familiar with them because they are yet in use for the definition of behavior of digital circuits.

$\mathbf{time} \rightarrow \boxed{x} \rightarrow \boxed{y} \rightarrow \mathbf{time}$	=	$\mathbf{time} \rightarrow \boxed{x+y} \rightarrow \mathbf{time}$
$\mathbf{time} \rightarrow \boxed{0} \rightarrow \mathbf{time}$	=	$\mathbf{time} \rightarrow \mathbf{time}$

Table 3: Equations of the real-time process.

- Some calculi like CCS ([Mil80]), CSP ([Hoa85]) or PETRI nets ([Rei85]) compose devices using special construction principles. For these devices an *operational semantics* is defined. Our approach has *declarative flavor* like modal and temporal logics ([Sti96]). That means that axioms say which behavior is possible and which one is not.

4.2 Outlook.

4.2.1 Object oriented concepts.

Means for structuring specifications we have not presented. This is the subject of a forthcoming paper.

4.2.2 Verification and prototyping.

Verification as well as prototyping techniques are still missing. This sticks a task for the future. I think it is useful for relating our approach to another mathematical models for process behavior where there are known results like coalgebraic specifications ([JR97]), *flownomials* ([S94]), *upas* ([FS90]) or CHU spaces ([Gup94], [Pra95]).

4.2.3 Real-time issues.

In order to express real-time information one could introduce the standard process type **time**. Each specification of a length of a time interval is a method in $M_{\mathbf{time}, \mathbf{time}}$. An example for such a method is

$$\mathbf{time} \rightarrow \boxed{20 \text{ msec}} \rightarrow \mathbf{time}$$

Whenever

$$\begin{array}{c}
 \mathbf{time} \rightarrow \bullet \rightarrow \boxed{x} \rightarrow \bullet \rightarrow \mathbf{time} \\
 \vdots \\
 \mathbf{time} \rightarrow \bullet \rightarrow \boxed{y} \rightarrow \bullet \rightarrow \mathbf{time}
 \end{array}$$

is a part of a network $x = y$ holds. Additionally, the equations of Table 3 hold.

4.3 Acknowledgments.

I am grateful to Stefan CONRAD as well as Maritta HEISEL for the fruitful discussion on this paper.

References

- [S94] Gheorghe Ștefănescu. Algebra of flownomials. part i: Binary flownomials, basic theory. Report I9437, Department of Computer Science, Technical University Munich, November 1994. SFB-Bericht Nr. 342/26/94 A.
- [FS90] P. J. Freyd and A. Scedrov. *Categories, Allegories*, volume 29 of *North-Holland Mathematical Library*. North-Holland, Amsterdam, 1990.
- [Gup94] Vineet Gupta. *Chu Spaces: A Model of Concurrency*. PhD thesis, Stanford University, September 1994.
- [Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, Englewood Cliffs, 1985.
- [JR97] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222 – 259, 1997.
- [JSHS96] Ralf Jungclaus, Gunter Saake, Thorsten Hartmann, and Cristina Sernadas. TROLL — A language for object-oriented specification of information systems. *ACM Transactions on Information Systems*, 14(2):175 – 211, April 1996.
- [Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag, Berlin, 1980.
- [Pra95] V. Pratt. Chu spaces and their interpretation as concurrent objects. *Lecture Notes in Computer Science*, 1000:392–??, 1995.
- [Rei85] W. Reisig. *Petri Nets. An Introduction*. Springer-Verlag, Berlin, 1985.
- [Sti96] Colin Stirling. Modal and temporal logics for processes. *Lecture Notes in Computer Science*, 1043:149–??, 1996.
- [Tur93] K. J. Turner. *Using Formal Description Techniques*. John Wiley & sons, Chichester, New York, Brisbane, Toronto, Singapur, 1993.