Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

A. Eliëns

Semantics for Occam

Computer Science/Department of Software Technology Note CS-N8606 May

Semantics for Occam

Anton Eliëns
Centre for Mathematics and Computer Science
Kruislaan 409, 1098 SJ Amsterdam, The Netherlands

Department of Computer Science, University of Amsterdam
Nieuwe Achtergracht 166, 1018 WV Amsterdam, The Netherlands

A brief description of the language Occam and its relation to the transputer is given.
The problems in specifying a semantics dealing with the real-time instruction *WAIT a period of time* and the possibility of allocating distinct processes to distinct processors are indicated.
A variety of semantics is presented, notably a linear time operational semantics on the basis of a transition-system in the style of (Plo), a branching time denotational semantics in the tradition of (BZ1) and a metric denotational semantics based on the concept of alternation as put forward in (CKS).
One of the aims of developing the latter semantics was to investigate the possibility of an event-structure like semantics as proposed by (Re,Wi1) in a metric denotational framework as developed in (BZ1).
A sketch is given of how to interrelate the semantics.

1980 Mathematics Subject Classification: 68B10, 68C01.
1982 CR Categories: D.1.3., D.3.1., F.1.2., F.3.2., F.3.3. 69D13, 69D41, 69F12, 69F32, 69F33
Key Words & Phrases: Occam, transputer, real time, communication, concurrency, transition-systems, operational semantics, denotational semantics, alternation, event-structures.
Note: This report describes graduate work done under the supervision of Prof.dr. J.W. de Bakker.

Contents

Report CS-N8606
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1. Occam: the language and its transputer

1.1. The transputer

Occam is the language introduced by INMOS to run on their transputer, a VLSI-circuit with a 10 MIPS processor, 4 kbytes 50 nsec static RAM and four links to other transputers. A collection of transputers can be combined to form a network of processes that each can consist of several parallel running processes.
According to the manufacturer, due to the communication-links and the inherent capacity at communication of the transputer, the device solves the problem of interprocessor communication bandwidth and the software overhead necessary to organise processing and communication that arises for instance in bus-oriented processing.

Occam resembles the instruction-set of the transputer, that is developed according to the RISC-principle (reduced instruction set), as to allow efficient compilation.
If it were not for its close link to the transputer Occam as a language (lacking recursion and complex data-types) would not be that interesting.
However its uniform treatment of input and output via communication-channels, the possibility to allocate processes to distinct transputers and the availability of a real-time statement like WAIT *a certain amount of time* make it a likely candidate as an implementation-language for VLSI-applications.

1.2. Processes, communication and synchronisation

Occam's direct (theoretical) ancestor is CSP (Ho). A difference with CSP however is that processes in Occam do not communicate by addressing another program by its name but by sending (or receiving) a value through a channel directly addressed by the channel-name.

Communication between transputers is by point to point connections. When allocating a process to a transputer the inter-transputer links are assigned to channels and communication takes place via the symbolic channel-names.

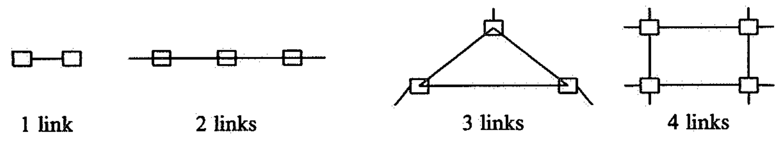
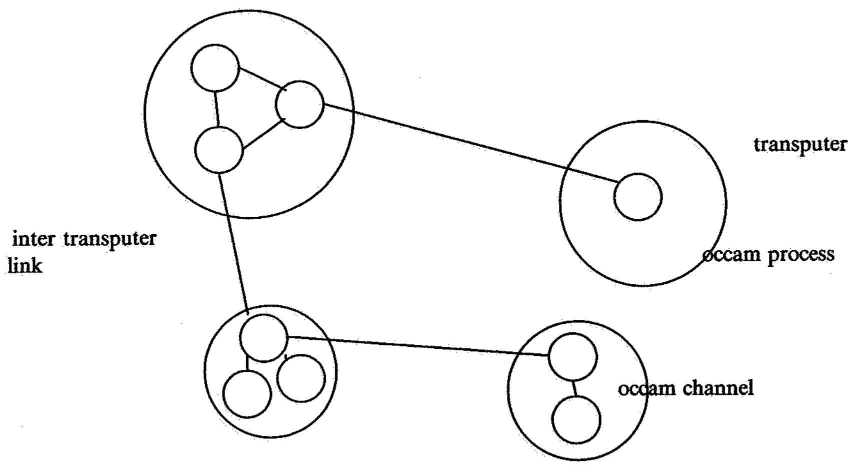


fig. 1.1. Possible networks of transputers.

Communication between transputers is both *rigorous* and *synchronous*, that is: *"Communication proceeds when both the sender is ready to send and the receiver is ready to receive. Whichever process tries to send or receive first and finds the other not ready waits until the other is ready. The synchronisation continues with each byte transferred being handshaken".* (De)

As concerns the relation between the transputer and Occam processes, each transputer implements a process which itself may consist of many processes. (fig 1.2)



Processes cooperate by communicating with each other, either within or between transputers. Although logically there is no difference between an Occam-process and a transputer-process in their communication behavior there is a difference that is reflected in their parallel composition. Distinct transputer processes run truly concurrently, while Occam processes within one transputer run parallel by time-sharing. To adequately capture the real-time behavior of (possibly transputer allocated) Occam-processes it seems necessary to develop a model that reflects this difference.

Process communication and scheduling are interrelated because communication between processes is synchronised. If one process want to communicate with another which is not ready then the first is de-scheduled. For a transputer process this means that the transputer remains idle until a matching communication intention is issued by another transputer. Within a transputer process-scheduling takes place via a process-queue of active processes that contains the respective workspace and instruction pointers.
Communication between processes involves channels. On encountering an input instruction ($c?x$) the channel is tested to see if another process is ready, a word is transferred and the output process (which would have been descheduled and waiting) put on the active process-queue. If a process is not ready then the inputting process is descheduled and the next process on the process-queue is run. Output instructions ($c!e$) are treated similarly.
For processes in seperate transputers exactly the same instructions are used. A process wanting input or output is always descheduled until the data has been transmitted via the hardware channel. After that it is put back onto the process-queue.
In the terminology of (BKT) Occam is a language that supports asynchronous cooperation and synchronous communication in that it allows processes to run independently as long as no communication takes place.

1.3. An informal description of the language

Occam is a simple imperative language.
It contains three distinct classes of commands:

- *Elementary instructions* like assignment, input, output, skip and wait. These instructions give rise to primitive processes.

- *Constructors* to effect respectively sequential processes, parallel processes, alternative processes, repetitive processes and replicated processes.

- *Configuration instructions* that affect the distribution of processes over transputers and within a transputer possibly the priority of components of a parallel or alternative construct.

An informal description, following the INMOS Occam user-manual (In), of the syntax and semantics of the language is given below. A summary of the syntax is given in fig 1.3.

1.3.1. Primitive processes

(1) **Assignment**: $x := e$
An assignment-process transfers the value of its expression e to the named variable x.

(2) **Input**: $c?x$, $c?ANY$, $c[i]?x$, $c[i]?ANY$
An input process transfers a value from a channel to a variable. Each input is seperately synchronised with an output process being executed in parallel. If *ANY* is used instead of a variable then the input value is discarded. This provides a mechanism for receiving synchronisation signals. Only one of the components of a parallel construct may contain input processes for any given channel. An array of channels can be used, indexed by an integer expression.
Output: $c!e$, $c!ANY$, $c[i]!e$, $c[i]!ANY$
Similar to input. Also, only one of the components of a parallel construct may contain output processes for any given channel.

(3) **Wait**: *WAIT e*
A wait is used to delay execution until a period of time has passed. In Occam the wait instruction is more complex than presented here, involving a clock comparison operator AFTER and the expression NOW that delivers the value of the local clock. Occam does not support a global sense of time, not even within one transputer, so that for instance no relationship may be assumed between the values produced by *NOW* in different components of a parallel construct. For a treatment of the semantics of the wait instruction it will be assumed that WAIT e delays execution for $\underline{e}_\sigma$ units of time where $\underline{e}_\sigma$ is the integer value of e.

(4) **Skip**: *SKIP*
Skip terminates with no effect.

Remark: Input and wait processes may be uses as guards in alternative processes. In giving the semantics however also output-processes are allowed as guards. Cf. (Ber)

1.3.2. Constructs

(5) **Sequential processes**: $SEQ(S_1,S_2,...,S_n)$
A sequential process executes its component processes $S_1,S_2,...,S_n$ one after another.

(6) **Parallel Processes**: $PAR(S_1,S_2,...,S_n)$
A parallel process causes its component processes to be executed together. Two component processes of the parallel construct may communicate by sending values using a channel. No other component processes of the parallel construct may use the same channel. Variables are not to be used for communication between the component processes of a parallel construct. However a variable may be used in two or more component processes provided that no component process changes its value by input or assignment.

(7) **Alternative processes:** $ALT(g_1\ S_1, g_2\ S_2, \ldots, g_n\ S_n)$
An alternative process is used to accept the first message available from a number of channels. An alternative process waits until at least one guarded process of the form b, $b\ C$ or $b\ WAIT\ e$ becomes ready where b is a boolean expression, C an input or output process and e an integer expression.
A guard is called *enabled* if its boolean expression evaluates to *true*. A guard is called *ready* if it is enabled and either the input or the output process is ready (that is a matching output or input process is waiting to communicate through the channel) or the time forced by a wait instruction has elapsed.
If a guarded process containing an input (or output) guard is selected the input (or output) is performed and then the component process is executed.
If a guarded process is itself an alternative construct then it is ready if one or more of the component guarded processes of the alternative is ready.
Occam also allows guards containing multiple inputs. Such a guard is ready if an output process using the same channel as the input is waiting. There is however no way to tell how many outputs the waiting process can provide, so the remaining inputs might as well be placed in front of the component process.
An alternative process fails if it contains no guarded processes ($n=0$), which is a consequence of the fact that execution is delayed until a guard becomes ready.

(8) **Conditional processes:** $IF(b_1\ S_1, b_2\ S_2, \ldots, b_n\ S_n)$
A conditional process executes the first component process S_i for which the boolean expression b_i evaluates to true.
Note that this implies sequential execution unlike the ALT-construct that requires parallel checking of the guards. If no b_i, $1 \leqslant i \leqslant n$, evaluates to *true* or the conditional process is empty ($n=0$) then execution proceeds with the next instruction.

(9) **Repetitive processes:** $WHILE\ b\ S$
A repetitive process executes the component process S each time the expression evaluates to true.

(10) **Replicated processes:**

$$SEQ\ i=[e_1\ FOR\ e_2]\ (S_1, S_2, \ldots, S_n)$$
$$PAR\ i=[e_1\ FOR\ e_2]\ (S_1, S_2, \ldots, S_n)$$
$$ALT\ i=[e_1\ FOR\ e_2]\ (g_1\ S_1, \ldots, g_n\ S_n)$$
$$IF\ i=[e_1\ FOR\ e_2]\ (b_1\ S_1, \ldots, b_n\ S_n)$$

A replicator $i=[e_1\ FOR\ e_2]$ is used with a constructor to replicate the component processes a number of times, substituting succesive integer values for the replicator index i, starting at $\underline{e_{1_\sigma}}$. The substituted value for the replicator index in the last component will be $\underline{(e_{1_\sigma}+e_{2_\sigma})-1}$.
The replicator-index can be used in expressions. If the replicator-count e_2 evaluates to zero or less then an empty construct is generated. This has the effect of termination for sequential, parallel or conditional processes and of never being ready for alternative processes.

(11) **Named processes and substitution:** $PROC$ id$(x_1, x_2, \ldots, x_n) = S$
A name can be given to the text of a process S. The text of S will be substituted for all occurrences of the name in the subsequent process. Channel names $c_1, c_2, \ldots, c_n$ given as actual parameters will replace the formal parameters $x_1, x_2, \ldots, x_n$ when textual substitution takes place.

1.3.3. Configuration

Configuration associates the components of an Occam program with a set of physical resources. Configuration is used to meet speed and response requirements by distributing processes over seperate interconnected transputers and by prioritising processes on single transputers. On any individual transputer a parallel construct may be configured to prioritise its components and an alternative construct may be configured to prioritise its inputs. In the Occam manual it is remarked that the allocation of resources to the concurrent processes in a program does not affect the logical behavior of the program.

(12)**Prioritised alternative processes**: $PRI\ ALT(g_1\ S_1,...,g_n\ S_n)$
If more than one guarded process is ready when a prioritised alternative process is executed the first one in textual sequence is executed.

(13)**Prioritised parallel processes**: $PRI\ PAR(S_1,...,S_n)$
A prioritised parallel construct gives each component process a different priority. The first component has the highest priority and the last component the lowest priority. The progress of a higher priority process is not affected by any lower priority process except by communication on connecting channels.

(14)**Multi-processor execution**: $PLACED\ PAR(S_1,...,S_n)$
A parallel construct which is configured for a network of transputers is called a system. In practice allocations have to be used to give physical resources to processes and channels, so that the full instruction would read:

$$PLACED\ PAR(alloc_1\ S_1,...,alloc_n\ S_n)$$

where $alloc_i$ is to name a processor-number and an allocation of port-numbers (0-3) to channel-names.
A physical connection between two processors connects a port on one processor to a port on the other processor.

$S ::= x := e$
$\mid c?x \mid c?ANY \mid c[i]?x \mid c[i]?ANY$
$\mid c!e \mid c!ANY \mid c[i]!e \mid c[i]!ANY$
$\mid WAIT\ e$
$\mid SKIP$
$\mid SEQ(S_1,S_2,...,S_n)$
$\mid PAR(S_1,S_2,...,S_n)$
$\mid ALT(g_1\ S_1,...,g_n\ S_n)$
$\mid IF(b_1\ S_1,...,b_n\ S_n)$
$\mid WHILE\ b\ S$
$\mid SEQ\ i=[e_1\ FOR\ e_2]\ (S_1,...,S_n)$
$\mid PAR\ i=[e_1\ FOR\ e_2]\ (S_1,...,S_n)$
$\mid ALT\ i=[e_1\ FOR\ e_2]\ (g_1\ S_1,...,g_n\ S_n)$
$\mid IF\ i=[e_1\ FOR\ e_2]\ (b_1\ S_1,...,b_n\ S_n)$
$\mid PROC\ \mathrm{id}(x_1,x_2,...,x_n)=S$
$\mid \mathrm{id}(c_1,c_2,...,c_n)$
$\mid PRI\ ALT(g_1\ S_1,...,g_n\ S_n)$
$\mid PRI\ PAR(S_1,...,S_n)$
$\mid PLACED\ PAR(alloc_1\ S_1,...,alloc_n\ S_n)$

Fig. 1.3. Syntax of Occam (summary)

S denotes a process, x an integer variable, e an integer expression, b a boolean expression, c a channel-name, id a procedure-name and g a guard of the form b, b C with C an input or output instruction, or b *WAIT* e.

2. Operational semantics

2.1. Introduction: observability and time

The purpose of specifying an operational semantics can be two-fold. It can serve:
- an analysis of the behavior of a program, or
- as a specification of what can be observed when the program is executed.

In general these points of view do not coincide. (Cf. Mi2)

Observable behavior can be taken to consist of (Ge):

(1) the final state
(2) the communication-actions on the unshielded channels
(3) the timing of actions

To specify a semantics purely on the level of observable behavior has the disadvantage though that an analysis of how conflicts with respect to the choice of actions and communications (fairness) are resolved on the behavioral level can not be analysed.
Several authors (Mi,Zij) therefore use a two-step approach in which observability is defined by way of abstracting from the behavioral capabilities of a program. (Cf. BR)
In (Zij) an analysis of real-time justice is given by specifying a priority of communication-actions over local actions (involving only one component of a parallel composition) and a priority of local actions over idling while waiting for a communication. The latter priority amounts to a maximality requirement in the sense of (SM). With the help of a Justice-operator a selection among the possible actions is made in such a way that transitions that violate the priority requirements are excluded. What is observed is Just.

As concerns communication, some authors (Mi,Zij) let successful communications disappear as a unit-action, others register communications by *records of communication* (roc's). (Cf. AP,FLP)

With respect to the timing of actions there is even less agreement.
Proposals encountered are:

- sequences of bags of communication intentions and records, with the length of the sequence denoting the time passed (KSRGA)

- attributing actions with a duration (in time-units), and possibly the capability to idle, that is to consume time without acting (Mi2)

- as the latter, but in non-discrete time. (Ca)

An interesting algebraic approach can be found in (Fi).

The stand taken here with respect to these issues is:

- *only observable actions take time*, non-observable actions are considered necessary to adequately describe the behavior of the program,

- *everything that takes time is observable*, that is in principle nothing is unshielded, every channel can be tapped, local actions might be monitored, even idling is considered to be visible as an empty (but time-consuming) action.

A first classification of *elementary actions* is:

α	local assignment
δ	idling while waiting for communication
τ	synchronisation assignment (communication)
ϵ	empty moves (waiting, skip, test and control)

Actually a more refined classification of action-labels is used in which δ and τ are indexed by a channel-name, δ_c as an input-intention is distinguished from $\overline{\delta}_c$, a corresponding output-intention, and ϵ_0 is distinguished from ϵ_l to reflect the distinction between control which (by assumption) takes no time and waiting on purpose or testing.

The purpose of *constructs* is nothing more than to compose elementary actions in some way. Recall from the previous section that the basic constructors were *SEQ, PAR, ALT* and *IF* for respectively sequential, parallel, alternative and conditional composition. For convenience the more usual notations $S_1;S_2$, $S_1 \| S_2$, $g_1\ S_1 + g_2\ S_2$ and $b \supset S_1, S_2$ are sometimes (informally) used.

The *operational semantics* is given by means of a transition-system $T=(A,\Gamma,\rightarrow)$ with A ranging over elementary action-labels, $\Gamma \subseteq P \times \Sigma$ where $\sigma \in \Sigma$ is the state-function that delivers a value v from the value-domain V for a variable-identifier x such that $\sigma(x)=v$ and P is the set of syntactically correct Occam-programs. The transition-relation is a relation $\rightarrow\ \subseteq \Gamma \times A \times \mathbb{N} \times \Gamma$ in which the third components stands for the domain of time-values. A possible transition in the system is for instance $<S,\sigma> \rightarrow^{\alpha}_{1} <S',\sigma'>$ indicating that an assignment α changes the configuration $<S,\sigma>$ in unit time into the configuration $<S',\sigma'>$. For this to occur S must be of the form $x:=e;S''$ with e an expression and S'' possibly empty. For a further explanation of transition-systems see (BMOZ1,2,Plo).

Basically there are two ways to treat the *delay* forced by waiting for communication:

(1) Allow the process that indicates an intention to communicate to be *busilly idling*. In (Mil,2) a special delay-operator δ is introduced that satisfies (abbreviating configurations to their corresponding statements)

$$\delta S \rightarrow^{\epsilon}_{1} \delta S \text{ and } (S \rightarrow^{h}_{t} S' \Rightarrow \delta S \overset{h'\cdot h}{\underset{t+t'}{\rightarrow}} S'$$

Here h and h' are the histories of actions, (h' a sequence of t' ϵ's and h the history belonging to $S \rightarrow S'$.
Informally it is stated (M1) that the meaning of a statement which is allowed to idle can be given by

$$\delta S = \mathit{fix} X.(S + \mathit{SKIP};X)$$

This approach would amount to letting a process wait for a communication to keep in pace with other processes by actively idling.

(2) Another approach suggested by (La) is to give each process a clock which it increments with each action. While waiting for a communication the process does nothing so its clock is not incremented. When communication takes place the respective processes inform each other about their clocks and set its value to the maximum plus the time the actual communication takes.

It is evident that while the first approach assumes (apart from knowledge of existing communication-intentions) a global sense of time for each process, in the second approach only a local sense of time and a desire to synchronise is assumed.

The first approach is adopted in giving a synchronous version of (the transition-system for) the operational semantics.

The second approach will be used to specify an extension of the transition-system that includes process-numbering and timing of actions, and later to specify the denotational semantics.

As a remark, the latter approach is more in line with the partial order semantics (Re) and event-structure semantics (Wil) to which (an extension of) the denotational semantics will eventually be related.

2.2. A transition-system

DEFINITION: (*transition-system*)
The transition-system $T=(A,\Gamma,\rightarrow)$ with

$\xi \in A \subseteq \{ \alpha,\epsilon_0,\epsilon_1 \} \quad \cup \{ \delta_c,\overline{\delta}_c,\tau_c\text{: c a channel-name}\}$

$\gamma \in \Gamma \subseteq P\times\Sigma$, the set of configurations

where P is the set of valid Occam-programs, and

$\Sigma = Var \rightarrow V$ the set of state functions

and

$\rightarrow \subseteq \Gamma\times A\times\mathbb{N}\times\Gamma$

specifies a relation $\gamma\rightarrow^{\xi}_{t}\gamma'$ between configurations as defined by the rules and axioms given below.

For arbitrary expressions e the valuation $[e](\sigma)$ in state $\sigma\in\Sigma$ will be denoted by $\underline{e}_\sigma$ with subscript σ possibly omitted. The same goes for boolean expressions b.
A Σ-variant is defined by

$$\sigma[x:=v](y) = \begin{cases} v & \text{if } x=y \\ \sigma(y) & \text{otherwise} \end{cases}$$

To simplify the format of the axioms and the rules an arbitrary statement E (which is not part of Occam) is used.
Unlike as in (BMOZ1,2) the statement E is not defined away by equations but rules are included to remove it. In the following h stands for an arbitrary (non-nil) sequence over A, and $t \in \mathbb{N}$ stands for an arbitrary time-value.

In giving the rules for parallel and alternative composition and their interaction in *selection by communication* the following selection-functions on statements S are used.

first(S)	delivers the set of possible first actions of S that are subject to label the next transition
rest(S)	gives the remaining part of S, that is S with the elementary actions chosen by *first*(S) deleted.

More precisely the multivalued function $<\mathit{first},\mathit{rest}>$ can be defined by

$$<\mathit{first},\mathit{rest}>(A) = \{(A,E)\} \text{ for } A \in \{\ x:=e,\ b,\ c?x,\ c!e,\ \mathit{WAIT}\ e,\ \mathit{SKIP}\ \}$$

where b is a boolean test that might appear in front of a guard, and

$$(A,S) \in <\mathit{first},\mathit{rest}>(\mathit{SEQ}(S_1,...,S_n))$$
$$\text{if } (A,S') \in <\mathit{first},\mathit{rest}>(S_1)\ \&\ S = \mathit{SEQ}(S',...,S_n)$$
$$(A,S) \in <\mathit{first},\mathit{rest}>(\mathit{PAR}(S_1,...,S_n))$$
$$\text{if } (A,S_i') \in <\mathit{first},\mathit{rest}>(S_i)\ \&\ S = \mathit{PAR}(...,S_i',...)$$
$$(A,S) \in <\mathit{first},\mathit{rest}>(\mathit{ALT}(g_1\ S_1,...,g_n\ S_n))$$
$$\text{if } (A,g_i'\ S_i) \in <\mathit{first},\mathit{rest}>(g_i\ S_i)\ \&\ S = \mathit{ALT}(...,g_i'\ S_i,...)$$

For all other statement forms: $<\mathit{first},\mathit{rest}>(S) = \varnothing$.
Informally $\mathit{first}(S)=A$ and $\mathit{rest}(S)=S'$ will be used meaning $(A,S') \in <\mathit{first},\mathit{rest}>(S)$.

A more direct treatment of the alternative construct would give

$(A,S) \in <\mathit{first},\mathit{rest}>(\mathit{ALT}(g_1\ S_1,...,g_n\ S_n))$

(i) if $g_1 = b$ then $(A,S) = (b,S_i)$
(ii) if $g_i = b_i\ C_i$ for C_i a communication or wait-instruction then $(A,S) = (b_i,\mathit{ALT}(...,C_i\ S_i,...))$
(iii) if $g_i = c?x$ or $c!e$ then $(A,S) = (g_i,S_i)$
(iv) if $g_i = \mathit{WAIT}\ e$ then $(A,S) = (\mathit{WAIT}\ e,S_i)$

Given an axiom of the form $<\mathit{ALT}(...,E\ S,...),\sigma> \rightarrow_0^{\xi} <S,\sigma>$ these cases give the same result as the more general formulation above. In the actual transition-system only (iii) will be used.
Note that the function $<\mathit{first},\mathit{rest}>$ is *purely* syntactic.

The semantics to be specified should respect the following identities:

$$\mathit{SEQ}(S) = \mathit{PAR}(S) = S$$
$$\mathit{SEQ}(...,\mathit{SEQ}(S_1,...,S_n),...) = \mathit{SEQ}(...,S_1,...,S_n,...)$$
$$\mathit{PAR}(...,\mathit{PAR}(S_1,...,S_n),...) = \mathit{PAR}(...,S_1,...,S_n,...)$$
$$\mathit{ALT}(...,\mathit{ALT}(g_1\ S_1,...,g_n\ S_n),...) = \mathit{ALT}(...,g_1\ S_1,...,g_n\ S_n,...)$$
$$\mathit{IF}(...,\mathit{IF}(b_1\ S_1,...,b_n\ S_n),...) = \mathit{IF}(...,b_1\ S_1,...,b_n\ S_n,...)$$

and for PAR and ALT moreover:

$$\mathit{PAR}(S_1,...,S_n) = \mathit{PAR}(S_{\theta 1},...,S_{\theta n})$$
$$\mathit{ALT}(g_1\ S_1,...,g_n\ S_n) = \mathit{ALT}(g_{\theta 1}\ S_{\theta 1},...,g_{\theta 1} S_{\theta n})$$

for an arbitrary permutation θ over $\{1,...,n\}$.

The version of the transition-system to follow is merely meant to be preparatory to the extensions that handle timing (and process-numbering).
Timing of actions is included but can not be sensibly dealt with until a choice is made how to

combine the timings of components in a parallel composition.

2.2.1. . Transition-rules for Occam

(1) **Assignment:**

$$<x:=e,\sigma> \rightarrow_1^{\alpha} <E,\sigma[x:=\underline{e}_\sigma]>$$

Assignment takes unit time and leaves an empty process E. The state is modified by setting the value for x to the value of e in state σ.

(2) **Communication:**

$$<c?x,\sigma> \rightarrow_1^{\delta_c} <c?x,\sigma>$$

$$<c!e,\sigma> \rightarrow_1^{\overline{\delta_c}} <c!e,\sigma>$$

If an intention to communicate is expressed the process expressing it is allowed to idle until a matching communication-intention is available. These axioms are however not to be used unless no other axiom or rule applies.

(3) **Wait:**

$$<WAIT\ 0,\sigma> \rightarrow_0^{\epsilon} <E,\sigma>$$

$$<WAIT\ e,\sigma> \rightarrow_1^{\epsilon} <WAIT\ \underline{e}_\sigma-1,\sigma>$$

A wait instruction forces the process that is appended to it to idle for $\underline{e}_\sigma$ units of time.

(4) **Skip:**

$$<SKIP,\sigma> \rightarrow_1^{\epsilon} <E,\sigma>$$

A skip doesn't affect anything, takes nevertheless one unit of time.

(5) **Sequential composition:**

$$<SEQ(E),\sigma> \rightarrow_0^{\epsilon} <E,\sigma>$$

$$<SEQ(E,S_2,...,S_n),\sigma> \rightarrow_0^{\epsilon} <SEQ(S_2,...,S_n),\sigma>$$

$$\frac{<S,\sigma> \rightarrow_t^{h} <S',\sigma'>}{<SEQ(S,...,S_n),\sigma> \rightarrow_t^{h} <SEQ(S',...,S_n),\sigma'>}$$

The removal of empty processes in a sequential composition is evidently sequential.
The last rule says that whenever the first component of a sequential composition makes any progression then so does the sequential process itself. Moreover the conclusion respects history and time.

(6) **Parallel composition:**
This case is more intricate because of the fact that communication between components of the parallel construct may take place. The situation that an actual communication (possibly

involving some substatements of the components of the parallel construct) occurs can be expressed by the rule:

$$\frac{\mathit{first}(S_i) = c?x,\ \mathit{first}(S_j) = c'!e,\ c = c',\ i \neq j}{<PAR(\dots,S_i,\dots,S_j,\dots),\sigma> \rightarrow^{\tau}_{1} <PAR(\dots,\mathit{rest}(S_i),\dots,\mathit{rest}(S_j),\dots),\sigma[x:=\underline{e}_{\sigma}]>}$$

and a similar rule for $\mathit{first}(S_j)=c?x$ & $\mathit{first}(S_i)=c'!e$, where $<\mathit{first},\mathit{rest}>(S)$ gives the first atomic instruction and the remainder of the statement S as explained. The condition $c=c'$ is included to allow the matching of indexed channels.
In case there is no communication execution of a parallel construct progresses as the execution of one of its components progresses, which is covered by the rule;

$$\frac{<S_i,\sigma> \rightarrow^{h}_{t} <S_i{}',\sigma'>}{PAR(\dots,S_i,\dots),\sigma> \rightarrow^{h}_{t} <PAR(\dots,S_i{}',\dots),\sigma'>}$$

A parallel construct terminates if all of its components are empty.

$$<PAR(E,\dots,E),\sigma> \rightarrow^{\epsilon}_{0} <E,\sigma>$$

Note that it makes not much sense to speak about the time a parallel construct takes unless either a record is kept of the time each component consumes over which the maximum can be taken as a lower bound or to add the times of all steps and take the worst case as an upper-bound.
However since in this transition-system a communication-intention may idle indefinitely long this upperbound seems to be trivially infinite. To resolve this situation the transition-system should state some priority of actual communications over idling because of a communication-intention.
To include the priority of local transitions over idling would lead to maximality or real-time justice as explained in (Zij).

(7) **Alternative composition:**
The treatment of alternative composition is complicated by the fact that an *ALT*-construct may contain guarded processes without a communication or wait-guard but only a boolean expression as a guard.
The most general solution is to redefine the *ALT*-construct as to allow guards of the form E, b, $c?x$, $c!e$, $WAIT\ e$, $b\ c?x$, $b\ c!e$, $b\ \ WAIT\ e$, respectively an empty, boolean, communication and wait-guard, and a combination of the last three with a boolean.
Let $\mathit{trivial}(g)$ be the predicate that indicates that g consists only of a boolean expression, $\mathit{enabled}(g,\sigma)$ the predicate that indicates whether the boolean part of a guard is *true* or *false* with $\mathit{enabled}(g,\sigma)$ undefined if g does not contain a boolean part, then

$$<ALT(\dots,g\ S,\dots),\sigma> \rightarrow^{\epsilon}_{1} <S,\sigma> \text{ if } \mathit{trivial}(g)\ \&\ \mathit{enabled}(g,\sigma)$$

For g_i, $1 \leqslant i \leqslant n$, not trivial let $\overline{g_i}$ denote the guard g_i with the boolean part removed, i.e. $\overline{b\ WAIT\ e} = WAIT\ e$ and $\overline{WAIT\ e} = WAIT\ e$.
Then to put the communication or wait-part of the guard in front apply

$$<ALT(\dots,g\ S,\dots),\sigma> \rightarrow^{\epsilon}_{1} <ALT(\dots,\overline{g}\ S,\dots),\sigma> \text{ if } \mathit{enabled}(g,\sigma).$$

Application of this axiom leaves guards of the form $c?x$, $c!e$ *or* $WAIT\ e$.
For a wait-guard the rules are

$$<ALT(\dots,WAIT\ 0,\dots),\sigma> \rightarrow^{\epsilon}_{0} <S,\sigma>$$

$$<ALT(\dots,WAIT\ e,\dots),\sigma> \rightarrow^{\epsilon}_{1} <ALT(\dots,WAIT\ \underline{e}_{\sigma}-1\ S,\dots,\sigma>$$

To cope with an input or output-guard, due to the definition of $<\mathit{first},\mathit{rest}>$ case *ALT*(iii), the

rules for parallel composition suffice.
Note that in order to pick out branches with trivial guards first a priority of the first axiom over the others is assumed.

A more general formulation of the above is given by

$$<ALT(...,E\ S,...),\sigma> \rightarrow_0^\epsilon <S,\sigma>$$

$$\frac{<g,\sigma> \rightarrow_i^\epsilon <g',\sigma>}{<ALT(...,g\ S,...),\sigma> \rightarrow_i^\epsilon <ALT(...,g'\ S,...),\sigma>}$$

with the understanding that the first axiom has priority over the others, Composed guards are taken to be the sequential composition of their components, and the rule is only to be applied on non-trivial guards if the construct contains no trivial guards.

(8) **Conditional composition:**

$$<IF(E),\sigma> \rightarrow_0^\epsilon <E,\sigma>$$

$$<IF(E,b_2\ S_2,...,b_n\ S_n),\sigma> \rightarrow_0^\epsilon <IF(b_2\ S_2,...,b_n\ S_n),\sigma>$$

$$<IF(b_1\ S_1,...,b_n\ S_n),\sigma> \rightarrow_1^\epsilon <S_1,\sigma> \text{ if } \underline{b}_\sigma = tt$$

$$<IF(b_1\ S_1,...,b_n\ S_n),\sigma> \rightarrow_1^\epsilon <IF(E,...,b_n\ S_n),\sigma> \text{ if } \underline{b}_\sigma = ff$$

As already informally described the treatment of the conditional construct amounts to sequentially testing the truth of the conditions.
Note that a rule of the form

$$\frac{<S,\sigma> \rightarrow_i^h <S',\sigma'>\ ,\ \underline{b}_\sigma = tt}{<IF(b\ S,...),\sigma> \rightarrow_i^h <IF(b\ S',...),\sigma'>}$$

is not valid in general since b might acquire a different value during the execution of S in σ that leads to $<S',\sigma'>$.

(9)**Repetitive composition:**

$$<WHILE\ b\ S,\sigma> \rightarrow_1^\epsilon <E,\sigma> \text{ if } \underline{b}_\sigma = ff$$

$$<WHILE\ b\ S,\sigma> \rightarrow_1^\epsilon <SEQ(S,WHILE\ b\ S),\sigma> \text{ if } \underline{b}_\sigma = tt$$

These are the usual rules for iteration.

(10)**Composition by replication:**
Note that the use of a replicator amounts to a purely syntactic operation on the to be replicated processes.
Thus when $S[v\ /x]$ denotes syntactic substitution of the value v for the (replicator) variable x then the effect of the use of a replicator can be expressed in the collection of axioms:

$$<OP\ i=[e_1\ FOR\ e_2](\overline{X}),\sigma> \rightarrow_0^\epsilon <OP(\overline{X}[\underline{e_1}_\sigma\ /\ i],...,\overline{X}[\underline{e_1}_\sigma + \underline{e_2}_\sigma - 1\ /\ i]),\sigma>$$

where $OP \in \{SEQ, PAR, ALT, IF\}$, $\overline{X} \equiv S_1,...,S_n$ in the case of SEQ and PAR, $\overline{X} \equiv g_1\ S_1,...,g_n\ S_n$ when $OP = ALT$, and $\overline{X} \equiv b_1\ S_1,...,b_n\ S_n$ when $OP = IF$.
Note that the above is equivalent (ignoring ϵ_o-transitions) to

$$<OP\ i=[e1FORe2](\overline{X}),\sigma> \rightarrow_0^{\epsilon} <OP(OP(\overline{X})[\underline{e_1}_\sigma / i],...,OP(\overline{X})[\underline{e_1}_\sigma+\underline{e_2}_\sigma-1 / i]),\sigma>$$

for *SEQ*, *PAR* and *ALT* and for *IF* if the semantics of *IF* are changed in such a way that

$$IF(IF(b_1\ S_1,...,b_n\ S_n),IF(b_1'\ S_1',...,b_m',\ S_m')) = IF(b_1\ S_1,...,b_m'\ S_m')$$

which is a minor change after all that moreover turns out to be convenient in specifying the denotational semantics.

(11)**Procedure-invocation**

$$\frac{<S[c_i / x_i]_{i=1}{}^n,\sigma> \rightarrow_t^h <S',\sigma'>}{p(c_1,c_2,...,c_n),\sigma> \rightarrow_t^h <S',\sigma'>}$$

Here $S[c / x]$ denotes syntactic substitution of channel-names for channel-variables, which are names as well. This rule implies that any transition allowed by a statement that contains the body of a procedure with channel-names substituted for the formal parameters is allowed by a statement that contains the name of the procedure with the channel-names as actual parameters.

Prioritised composition

Define $PRI-<first,rest>(S)$ for S a PAR or ALT-construct to be the partially ordered collection of pairs $(A',S'), (A'',S''),... \in <first,rest>(S)$, $\underline{S}' = PRI\ S'$, $\underline{S}'' = PRI\ S''$,... and $(A',\underline{S}') < (A'',\underline{S}'')$

(i) for $(A',S'), (A'',S'') \in <first,rest>PAR(S_1,...,S_n))$
if $(A',S_i') \in <first,rest>(S_i)\ \&\ S' = PAR(...,S_i',...)$ and
$(A'',S_j') \in <first,rest>(S_j)\ \&\ S'' = PAR(...,S_j',...)$ and $i<j$

(ii) for $(A',S'), (A'',S'') \in <first,rest>ALT(g_1\ S_1,...,g_n\ S_n))$
if $(A',g_i'\ S_i) \in <first,rest>(g_i\ S_i)\ \&\ S' = ALT(...,g_i'\ S_i,...)$ and
$(A'',g_j'\ S_j) \in <first,rest>(S_j)\ \&\ S'' = ALT(...,g_j'\ S_j,...)$ and $i<j$

and for all other cases $(A',\underline{S}') \not< (A'',\underline{S}'')$ and $(A'',\underline{S}'') \not< (A',\underline{S}')$
Now augment the definition of $<first,rest>$ as given earlier by extending it to prioritised statements with

$$<first,rest>(PRI\ S) = PRI-<first,rest>(S)$$

and by taking the result also for the other statements to be ordered. Notice that the only order introduced is when encountering a prioritised statement, and so in the absence of ordering a minimal element can be arbitrarily chosen among the elements.
Now let $first(S)=A$ and $rest(S)=S'$ informally denote

$$(A,S') \text{ is minimal in } <first,rest>(S).$$

(12) **Prioritised alternative composition**
Note that also in this construct the alternatives with a trivial guard have priority over alternatives with non-trivial guards. The definition of rules similar as for (plain) alternative composition modified so that the left-most alternative will be chosen ensure that these cases will be picked out first. So assume that all components have non-trivial guards waiting for input. Then since communication involves some other component in parallel composition the communication-rule for

parallel composition can be formulated as

$$\frac{<PAR(S,g_i),\sigma> \rightarrow_1^\tau <PAR(S',\sigma'>}{\begin{array}{c}<PAR(\ldots,S,\ldots,PRI\ ALT(\ldots,g_i\ S_i,\ldots),\ldots),\sigma> \rightarrow_1^\tau <PAR(\ldots,S',\ldots,S_i,\ldots),\sigma'> \\ <PAR(\ldots,PRI\ ALT(\ldots,g_i\ S_i,\ldots),\ldots,S,\ldots),\sigma> \rightarrow_1^\tau <PAR(\ldots,S_i,\ldots,S',\ldots),\sigma'>\end{array}}$$

provided that there is no branch $j<i$ with a matching communication in S or another component of the *PAR*-construct.

A more general approach is to define a function $matching(S_1,S_2)$ that selects the least matching pair of communication intentions as follows

(i) if both $<first,rest>(S_1)$ and $<first,rest>(S_2)$ are unordered then arbitrarily choose a matching pair (C,S_1'), $(\bar{C},S_2')$ with C and $\bar{C}$ matching communication intentions.

(ii) if only $<first,rest>(S_i)$, $i=1$ or 2, is ordered then take the least (C,S_i') for which there is a matching pair $(\bar{C},S_j')$ in $<first,rest>(S_j)$, $j \neq i$.

(iii) if both $<first,rest>(S_1)$ and $<first,rest>(S_2)$ are ordered then take the lexicographically least pair where the lexicographic order on pairs of (A,S)-pairs is defined by

$((A_m,S_m),(A_n,S_n)) < ((A_i,S_i),(A_j,S_j))$ if $(A_m,S_m) < (A_i,S_i)$ *or*

if $(A_m,S_m) \not< (A_i,S_i)$ & $(A_i,S_i) \not< (A_m,S_m)$ then if $(A_n,S_n) < (A_j,S_j)$

Now the communication rule for parallel composition can be formulated in its most general form as

$$\frac{((c?x,S_i'),(c!e,S_j'))=matching(S_i,S_j) \text{ or } ((c!e,S_i'),(c?x,S_j'))=matching(S_i,Sj)}{<PAR(\ldots,S_i,\ldots,S_j,\ldots),\sigma> \rightarrow_1^\tau <PAR(\ldots,S_i',\ldots,S_j',\ldots),\sigma[x:=\underline{e}_\sigma]>}$$

which treats prioritised alternative composition as well.

(13) **Prioritised parallel composition**:
For a treatment of the prioritised parallel construct extend the predicate $enabled(S)$ to be *true* if the first action is an assignment, a test, a wait or a skip instruction.
Now to get hold of the *least action* it suffices to compare the least matching pair with the left-most enabled statement.

if $\neg\exists k<i.enabled(S_k)$ & $\neg\exists(m,n)<(i,j).matching(S_m,S_n)$ then

$$\frac{<PAR(\ldots,S_i,\ldots,S_j,\ldots),\sigma> \rightarrow_1^\tau <PAR(\ldots,S_i',\ldots,S_j',\ldots),\sigma'>}{<PRI\ PAR(\ldots,S_i,\ldots,S_j,\ldots),\sigma> \rightarrow_1^\tau <PRI\ PAR(\ldots,S_i',\ldots,S_j',\ldots.),\sigma'>}$$

where $matching(S_m,S_n)$ is true if not empty,
otherwise

if $\neg\exists k<i.enabled(S_k)$ & $\neg\exists(m,n).m<i$ & $matching(S_m,S_n)$ then

$$\frac{<PAR(\ldots,S_i,\ldots),\sigma> \rightarrow_t^h <PAR(\ldots,S_i',\ldots),\sigma'>}{<PRI\ PAR(\ldots,S_i,\ldots),\sigma> \rightarrow_t^h <PRI\ PAR(\ldots,S_i',\ldots),\sigma'>}$$

(14) **Multi-processor execution**:

The Occam-manufacturers claim that there is no distinction in the logical behavior of a configurated parallel construct and an unconfigurated (multi-programmed) construct. So the rule

$$\frac{<PAR(S_1,\ldots,S_n),\sigma> \rightarrow_t^h <S',\sigma'>}{PLACED\ PAR(alloc_1\ S_1,\ldots,alloc_n\ S_n),\sigma> \rightarrow_t^h <S',\sigma'>}$$

should suffice.

(1) $<x:=e,\sigma> \rightarrow_1^\alpha <E,\sigma[x:=\underline{e}_\sigma]>$

(2) $<c?x,\sigma> \rightarrow_1^{\delta_c} <c?x,\sigma>$

$<c!e,\sigma> \rightarrow_1^{\bar{\delta}_c} <c!e,\sigma>$

(3) $<WAIT\ 0,\sigma> \rightarrow_0^\epsilon <E,\sigma>$

$<WAIT\ e,\sigma> \rightarrow_1^\epsilon <WAIT\ \underline{e}_\sigma-1,\sigma>$

(4) $<SKIP,\sigma> \rightarrow_1^\epsilon <E,\sigma>$

(5) $<SEQ(E),\sigma> \rightarrow_0^\epsilon <E,\sigma>$

$<SEQ(E,S_2,\ldots,S_n),\sigma> \rightarrow_0^\epsilon <SEQ(S_2,\ldots,S_n),\sigma>$

$$\frac{<S,\sigma> \rightarrow_t^h <S',\sigma'>}{<SEQ(S,\ldots,S_n),\sigma> \rightarrow_t^h <SEQ(S',\ldots,S_n),\sigma'>}$$

(6) $<PAR(E,\ldots,E),\sigma> \rightarrow_0^\epsilon <E,\sigma>$

$$\frac{<S_i,\sigma> \rightarrow_t^h <S_i',\sigma'>}{PAR(\ldots,S_i,\ldots),\sigma> \rightarrow_t^h <PAR(\ldots,S_i',\ldots),\sigma'>}$$

$$\frac{first(S_i) = c?x,\ first(S_j) = c'!e,\ c = c', i \neq j}{<PAR(\ldots,S_i,\ldots,S_j,\ldots),\sigma> \rightarrow_1^\tau <PAR(\ldots,rest(S_i),\ldots,rest(S_j),\ldots),\sigma[x:=\underline{e}_\sigma]>}$$

(7) $<ALT(\ldots,E\ S,\ldots),\sigma> \rightarrow_0^\epsilon <S,\sigma>$

$$\frac{<g,\sigma> \rightarrow_t^\epsilon <g',\sigma>}{<ALT(\ldots,g\ S,\ldots),\sigma> \rightarrow_t^\epsilon <ALT(\ldots,g'\ S,\ldots),\sigma>}$$

(8) $<IF(E),\sigma> \rightarrow_0^\epsilon <E,\sigma>$

$<IF(E,b_2\ S_2,\ \ldots\ ,b_n\ S_n),\sigma> \rightarrow_0^\epsilon <IF(b_2\ S_2,\ldots,b_n\ S_n),\sigma>$

$<IF(b_1\ S_1,\ldots,\ bn\ S_n),\sigma> \rightarrow_1^\epsilon <S_1,\sigma>$ if $\underline{b}_\sigma = ff$

$<IF(b_1\ S_1,\ldots,\ bn\ S_n),\sigma> \rightarrow_1^\epsilon <IF(E,\ldots,b_n\ S_n),\sigma>$ if $\underline{b}_\sigma = tt$

(9) $<WHILE\ b\ S,\sigma> \rightarrow_1^\epsilon <E,\sigma>$ if $\underline{b}_\sigma = ff$

$<WHILE\ b\ S,\sigma> \rightarrow_1^\epsilon <SEQ(S,WHILE\ b\ S),\sigma>$ if $\underline{b}_\sigma = tt$

(10) $<OP\ i = [e_1\ FOR\ e_2](X),\sigma> \rightarrow_0^\epsilon <OP(X[\underline{e_1}_\sigma / i],\ldots,X[\underline{e_1}_\sigma + \underline{e_2}_\sigma - 1 / i]),\sigma>$

fig 2.1 Transition-system for Occam
OP ranges over $\{SEQ,PAR,ALT,IF\}$

DEFINITION: (*computations*)
A T-comptation w of a statement S in P from a state σ (notation $T\vdash w$) can take one of the following forms (with $<S,\sigma> \; = \; <S_0,\sigma_0>$):

(1)succesful:

$$T\vdash <S_0,\sigma_0> \xrightarrow[t_1]{\xi_1} \cdots \xrightarrow[t_n]{\xi_n} <S_n,\sigma_n> \text{ with } S_n=E \text{ (the auxiliary statement)}$$

(2)blocked:

$$T\vdash <S_0,\sigma_0> \xrightarrow[t_1]{\xi_1} \cdots \xrightarrow[t_n]{\xi_n} <S_n,\sigma_n> \text{ with no configuration derivable from } <S_n,\sigma_n>$$

(3)infinite:

$$T\vdash <S_0,\sigma_0> \xrightarrow[t_1]{\xi_1} \cdots \xrightarrow[t_n]{\xi_n} <S_n,\sigma_n> \rightarrow \cdots$$

DEFINITION: (*event-sequences*)
Let $E_0 \subseteq A \times \mathbb{N} \times \Sigma \times P$ be the set of possible events consisting of an action-label, a time-stamp, a state and an Occam-program (fragment) or E, and let $\mathbb{E}=E_0{}^* \cup E_0{}^\omega \cup E_0{}^* \cdot \{FAIL\}$ be the set of possibly infinite sequences of events $e \in E_0$
Define the projections π_i ,$i=0,..,3$ such that for a given event e

$$\pi_0(e)\in A \quad \pi_1(e)\in\mathbb{N} \quad \pi_2(e)\in\Sigma \quad \pi_3(e)\in P \quad \text{and} \quad \pi_i(FAIL)=FAIL \text{ , } i=0,...,3$$

and for a sequence of events $\underline{w}$ $\pi_i(\underline{w})$ delivers the projected sequence.

DEFINITION: (*mapping computations to event-sequences*)
An Occam-program S in state σ gives rise to an event-sequence $\underline{w}$ (notation $T\Vdash\underline{w}$), with $(S_0,\sigma_0) = (S,\sigma)$

$$\textit{(1) } T\Vdash <\xi_0,\underline{t}_0,\sigma_0,S_0>\rightarrow<\xi_n,\underline{t}_n,\sigma_n,S_n> \text{ with } S_n=E$$

$$\textit{(2) } T\Vdash <\xi_0,\underline{t}_0,\sigma_0,S_0>\rightarrow<\xi_n,\underline{t}_n,\sigma_n,S_n>\rightarrow FAIL$$

$$\textit{(3) } T\Vdash <\xi_0,\underline{t}_0,\sigma_0,S_0>\rightarrow<\xi_n,\underline{t}_n,\sigma_n,S_n>\rightarrow \cdots$$

iff
there exists a T-computation corresponding to the respective cases such that
(i) $\underline{t}_0=0$
(ii) $\underline{t_{i+1}} = \underline{t}_i + t_{i+1}$

LEMMA: The mapping $\_ : T-comp\rightarrow\mathbb{E}$ is well-defined
Note that the mapping is only defined for complete computations (finite or infinite). Only in the finite (blocked) case is a special failure-event appended.
There are two cases:

(i) $<S_0,\sigma_0> \mapsto <\epsilon_0,t_0,\sigma_0,S_0>$, and

(ii) $(\rightarrow_{t_{i+1}}{}^{\xi_{i+1}} <S_{i+1},\sigma_{i+1}> \mapsto (\rightarrow<\xi_{i+1},\underline{t_{i+1}},\sigma_{i+1},S_{i+1}>)$

Case (i) implies that an empty control-action is prefixed which adds nothing to the computation. For case (ii) note that the correct value of $\underline{t_{i+1}}$ follows from the induction-hypothesis for $\underline{t_i}$.

DEFINITION: (*restriction, elimination of control-actions)*)
Both on T-computations and $\mathbb{E}$-sequences a restriction-operator can be defined that eliminates the empty moves (events, with exception of the first) from the computation (event-sequence).
Notation $\setminus \epsilon_0$.

LEMMA: *No infinite computation (event-sequence) becomes finite by ϵ_0-restriction.*
Proof: This is an immediate consequence of the finiteness of the program.

DEFINITION: (*Σ-sequence, operational semantics $\mathcal{O}$*)
For a statement S in Occam the Σ-sequence $w_\Sigma \in \mathcal{O}(S)(\sigma)$ iff $\exists \underline{w}.T$, $<S,\sigma> \Vdash \underline{w}$ and $w_\Sigma = \pi_2(\underline{w} \setminus \epsilon_0)$.

Remarks:

- This definition of the operational semantics although differently phrased coincides with the definition in (BKMOZ). This rather long winded version is solely meant to facilitate future proofs. Also the more general formulation allows to vary the operational semantics on the basis of what is considered observable.
- The system of transition-rules is not yet satisfying with respect to the timing of actions (and states). Note that every observable action in whatever process it takes place increments the global clock, which is clearly not intended. In the following modifications of the transition-system will be discussed that treat timing more nicely.
- The set of computations form a synchronisation-tree as described in (Wi2). In this definition the operational semantics can be taken to determine the equivalence-classes under weak bisimulation. (Cf BR) Such a class gives all interleavings of elementary actions that can be considered equivalent.
- In the system as presented deadlocks lead to an infinite computation, since the δ-actioms are not to be applied unless there is no alternative. A meaningful application of the δ-axioms is defined however for the maximally synchronous version to be presented shortly. The only case in which failure can occur is in a totally unabled alternative construct.

2.4. Partial orders and event-structures

The operational semantics presented in the previous section is not satisfying with respect to the timing of actions and communications. To remedy this defect there are several solutions at hand (Bes,Mi,Re,SM,Wi) some of which are briefly discussed.

(1) **maximal synchronisity**: (Mi,SM)
The requirement of maximal synchronisity imposes a restriction on the derivation of a computation in that when encountering a parallel construct every component of it has to be developed one time-unit step for the parallel compound to proceed one time unit.
Formally (disregarding communication) this could be stated in a rule like

$$\frac{<S_i,\sigma_i> \rightarrow_1^{\xi_i} <S_i',\sigma_i'> \quad , \ 1 \leqslant i \leqslant n}{<PAR(S_1,\ldots,S_n),\sigma> \rightarrow_1^{H} <PAR(S_1',\ldots,S_n'),\sigma'>}$$

with $\sigma = \cup \sigma_i$ and $\sigma' = \cup \sigma_i'$.
Such a solution is allowed since variable-disjointness can be assumed for the respective

components. A difficulty however with this solution is that when components differ in the number of steps they need the empty statement E has to be allowed to actively idle. This would amount to two delayable types of statements, communication-intentions and end-instructions.
Another problem that adheres to this solution is the complex form of the history H. (Mi) uses a product-notation for a synchronous parallel construct such that $(S_1 \xrightarrow{a} S_1', S_2 \xrightarrow{b} S_2' \Rightarrow S_1 \times S_2 \xrightarrow{ab} S_1' \times S_2')$. Following a suggestion of (Ma) then the interleaving of the time-stamped components of the combined history could be interleaved at the observational rather than the behavioral level.

(2) **partial orders**: (Bes,Re,Wi)
The previous solution has the disadvantage that when the assumption of unit-time cost for the execution of an atomic statement is abandoned the product-construction will not work, let alone the interleaving of their projections.
More general is the solution proposed by (Re). For distinct components of a parallel construct computations are developed independently unless they require synchronised interaction with some other component-process.
For example the computation of $(a_1;c;a_2) \| (\bar{c};a_3)$ where c and $\bar{c}$ are to be synchronised and a_i ,$i=1,...,3$ are atomic instructions can be pictured as (with τ the action of synchronisation):

$$\begin{tikzcd}
(a_1;c;a_2) \arrow[r, "\alpha"] & (c;a_2) \arrow[dr] & & (a_2) \arrow[r, "\alpha"] & (E) & (1) \\
& & \tau \arrow[ur] \arrow[dr] & & & \\
(\bar{c};a_3) \arrow[urr] & & & (a_3) \arrow[r, "\alpha"] & (E) & (2)
\end{tikzcd}$$

Timing (although this is not the issue (Re) is interested in) can be modelled by a mutual check on the clock-values of the components. His solution is hampered by the the restriction that no nested parallelism is allowed. It seems however that if you are willing to let the parallel construct be surrounded by an empty begin and end statement the approach also works for the more general case of nested parallelism.

The appeal of the latter approach is that parallel composition and alternative composition are treated orthogonally in that (in net-theoretic fashion)

- parallel composition branches on transitions
- alternative composition branches on states

Graphically:

$$\begin{tikzcd}
 & \mathit{state}_{01} \arrow[d] & \\
 & \text{begin} \arrow[dl] \arrow[dr] & \\
\mathit{state}_{12} \arrow[d, phantom, "\vdots"] & & \mathit{state}_{22} \arrow[d, phantom, "\vdots"] \\
\mathit{state}_{12} \arrow[dr] & & \mathit{state}_{22} \arrow[dl] \\
 & \text{end} \arrow[d] & \\
 & \mathit{state}_{0,max(n,m)} &
\end{tikzcd}$$

parallel composition

$$\begin{tikzcd}
 & \mathit{action}_0 \arrow[d] & \\
 & \mathit{state}_0 \arrow[dl] \arrow[dr] & \\
\mathit{action}_1 \arrow[d, phantom, "\vdots"] & & \mathit{action}_1 \arrow[d, phantom, "\vdots"] \\
\mathit{action}_m \arrow[dr] & & \mathit{action}_{m'} \arrow[dl] \\
 & \mathit{state}_k \arrow[d] & \\
 & \mathit{action}_{k+1} &
\end{tikzcd}$$

alternative composition

$$\begin{tikzcd}
\mathit{state}_{1n} \arrow[dr] & & \mathit{state}_{2m} \arrow[dl] \\
 & \mathit{comm}_k \arrow[dl] \arrow[dr] & \\
\mathit{state}_{1k} & & \mathit{state}_{2k}
\end{tikzcd}$$

with $k = \max(n,m)+1$

communication

Note that in the case of alternative composition k is either m or m' dependent on which branch is chosen, so this case could be modelled as well by the usual tree.
For the case of parallel composition it is evident that the auxiliary begin and end transition are obligatory. The state immediately after the end can be taken to be the union of the previous states ($1n$ and $2m$) and its time conveniently set to the maximum of the clock-values of state$_{1n}$ and state$_{2m}$.

Composition of nets of this form is described in (GM,Bes) and will not be pursued here.

2.5. The revised semantics

To recapitulate, in order to model the timing of actions either a requirement of maximal synchronisity can be imposed on the derivation, or the weaker requirement of respecting the time-dependencies as concerns synchronisation and the termination of parallel constructs. Both solutions involve some kind of process-numbering to keep account of which process has taken how many steps, when.

Process-numbering can be done in two ways, statically or dynamically.

DEFINITION: (*static process-numbering*)

Let for a tuple $(S,m) \in P \times \mathbb{N}$ (Occam-programs and natural numbers) $(S,m)_0 = S$ and $(S,m)_1 = m$. The mapping $\mathcal{N}: P \rightarrow (\mathbb{N} \rightarrow (P \times \mathbb{N}))$ is defined by induction on the structure of S. Let $m_0 = m$ and $m_{i+1} = (\mathcal{N}(S_{i+1})(m_i))_1$, then

$$\mathcal{N}(A) = \lambda m.(<m,A>,m) \text{ for } A \in \{\ x := e,\ c?x,\ c!e,\ WAIT\ e,\ SKIP,\ b\ \}$$

$$\mathcal{N}(b\ S) = \lambda m.((\mathcal{N}(b)(m))_0(\mathcal{N}(S)(\mathcal{N}(S)(\mathcal{N}(b)(m))_1)_0,(\mathcal{N}(S)(\mathcal{N}(S)(\mathcal{N}(b)(m))_1)_1)$$

$$\mathcal{M}(SEQ(S_1,...,S_n)) = \lambda m.\,(SEQ((\mathcal{M}(S_1)(m_0))_0,...,(\mathcal{M}(S_n)(m_{n-1}))_0),m_n)$$
$$\mathcal{M}(PAR(S_1,...,S_n)) = \lambda m.\,(PAR((\mathcal{M}(S_1)(m_0+1))_0,...,(\mathcal{M}(S_n)(m_{n-1}+1))_0),m_n)$$
$$\mathcal{M}(ALT(g_1\ S_1,...,g_n\ S_n)) = \lambda m.\,(ALT((\mathcal{M}(g_1\ S_1)(m_0))_0,...,(\mathcal{M}(g_n\ S_n)(m_{n-1}))_0),m_n)$$
$$\mathcal{M}(IF(b_1\ S_1,...,b_n\ S_n)) = \lambda m.\,(IF((\mathcal{M}(b_1\ S_1)(m_0))_0,...,(\mathcal{M}(b_n\ S_n)(m_{n-1}))_0),m_n)$$
$$\mathcal{M}(WHILE\ b\ S) = \lambda m.\,(WHILE\ (\mathcal{M}(b\ S)(m))_0,(\mathcal{M}(b\ S)(m))_1)$$
$$\mathcal{M}(OP\ i{=}[e_1\ FOR\ e_2](X) = \lambda m.\,(OP\ i{=}[e_1\ FOR\ e_2](\mathcal{M}(X)(m))_0,(\mathcal{M}(X)(m))_1)$$

LEMMA: *Static process numbering suffices only if the use of PAR-constructs in repetitive and replicated constructs is prohibited.*
Each action is given the number of the process it belongs to where a process consists of a possibly iterated sequence of actions or a choice between sequences of actions.
Process-numbers are generated in increasing order, and the only increment is when encountering a parallel component. A counter-example for the while construct is immediate. However it could be defended that iterated processes get the same process-number independent of their degree of iteration, that could be taken into account dynamically. Note that there is no recursion of the form $\mu x[a \,\|\, x]$.

More serious is the problem with replicated constructs.
As an example consider the definition of a buffer of size n:

$$PAR\ i{=}[0\ FOR\ n]\ WHILE\ true\ SEQ(c[i]?x,c[i+i]!x)$$

which amounts to

c[0]
x
1
c[1]
· · ·
c[n − 1]
x
n
c[n]

setting up n processes connected by $n+1$ channels buffering a maximum of n values. As the value of n is determined dynamically there is no way to know at forehand how many process-numbers to reserve.
Of course it could be demanded that n is fixed, but this is not required in Occam, nor seems very sensible from a programming point of view. So the only way out seems to be a dynamic numbering scheme.

Rather than start duplicating the entire transition-system by burdening it with notation to keep track of process-numbers and timing it will first be indicated how to adapt the system T given before to model the timing of actions.

DEFINITION: (*timing*)

Let T' be the intended system, then T' is constructed from T by:

(0) If $T \vdash <S,\sigma> \rightarrow_1^{\delta} <S,\sigma>$ then $T' \vdash <(S,t),\sigma> \rightarrow^{\delta} <(S,t),\sigma>$

(1) If $T \vdash <S,\sigma> \rightarrow_t^{h} <S',\sigma'>$ then $T' \vdash <(S,n),\sigma> \rightarrow^{h} <(S',n+t),\sigma'>$

(2) If $T \vdash <PAR(...,S_i,...),\sigma> \rightarrow_t^{h} <PAR(...,S_i',...),\sigma'>$ then

$$T' \vdash <(PAR(...,S_i,...),(...,t_i,...)),\sigma> \rightarrow^{h}\ <(PAR(...,S_i',...),(...,t_i+t,...)),\sigma'>$$

(3) If $T \vdash <PAR(...,S_i,...,S_j,...),\sigma> \rightarrow_1^{\tau} <PAR(...,S_i',...,S_j',...),\sigma'>$ then

$$T' \vdash <(PAR(\ldots,S_i,\ldots,S_j,\ldots),(\ldots,t_i,\ldots,t_j,\ldots)),\sigma> \rightarrow^{\tau}$$
$$<(PAR(\ldots,S_i',\ldots,S_j',\ldots),(\ldots,t,\ldots t,\ldots)),\sigma'> \text{ with } t=\max(t_i,t_j)+1$$

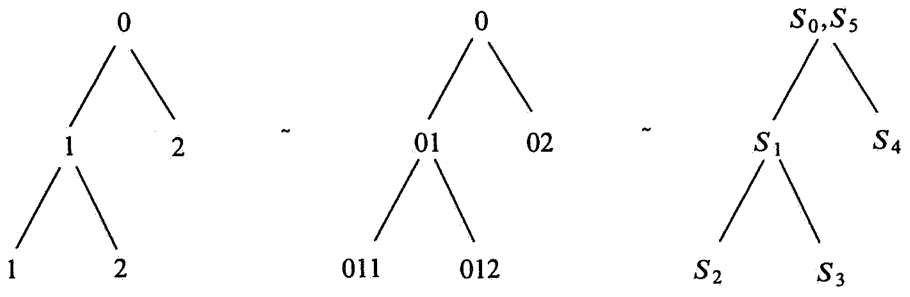
(4) If $T \vdash <PAR(E,\ldots,E),\sigma> \rightarrow_0^{\epsilon} <E,\sigma>$ then

$$T' \vdash <(PAR(E,\ldots,E),(t_1,\ldots,t_n)),\sigma> \rightarrow^{\epsilon_0} <(E,t_i),\sigma>$$

Note that (4) implies that when a statement takes over from a parallel construct in sequential composition it arbitrarily selects a clock-value from one of the components.
This is in agreement with an arbitrary interleaving approach in which all possible take-overs can be considered.
More appropriate however would be to take the maximum, but this turns out harder to deal with denotationally.
Also note that in this revision idling while waiting for communication no longer takes time and, by definition, no longer is observable. It might be worthwhile to accomodate the definition of observability in this respect.
Unfortunately a similar modification seems not fit to cope with process-numbering.

DEFINITION: (*dynamic process-numbering*)
Define $\hat{}:\mathbb{N}^+ \times \mathbb{N}_{>0} \rightarrow \mathbb{N}^+$ to be the function that appends an integer (>0) to a sequence of integers. For example $<012>\hat{}1=<0121>$. Let 0 denote the sequence $<0>$. The function is used to dynamically create process-numbers such that no two distinct processes are active that have the same number.

EXAMPLE: $S_0;((S_1;(S_2 \| S_3)) \| S_4);S_5$ gives rise to the process-numbering:



Note that if applied in a binary fashion process-numbers are not unique, namely $S_1 \| (S_2 \| S_3)$ gives a different numbering than $(S_1 \| S_2) \| S_3)$.

In the following a revision of the transition-system is proposed that models process-numbering and timing.
There are now four types of axioms:

- process-number introduction
- time-introduction
- E-removal
- atomic actions

The necessity of the first type will be evident when presenting the axioms. The other types are already familiar. The action-labels are extended to include the process-number and the time the action took place. An action-label of the form $(\tau_c{:}v, (m_i,m_o),t)$ records the input and output process participating in a communication, the value transferred, the channel and the time the communication took place. By convention unlabelled transitions are to be labelled with ϵ_0 and whatever process-number and t appropriate.

Let m stand for a process-number.

(*) process-number introduction

M1 $<(SEQ(S_1,...,S_n),m),\sigma> \rightarrow <SEQ((S_1,m),...,(S_n,m)),\sigma>$

M2 $<(PAR(S_1,...,S_n),M),\sigma> \rightarrow <PAR((S_1,m\hat{}1),...,(S_n,m\hat{}n)),\sigma>$

M3 $<(ALT(g_1\ S_1,...,g_n\ S_n),m),\sigma> \rightarrow <ALT((g_1\ S_1,m),...,(g_n\ S_n,m)),\sigma>$

M4 $<(IF(b_1\ S_1,...,b_n\ S_n),m),\sigma> \rightarrow <IF((b_1\ S_1,m),...,(b_n\ S_n,m)),\sigma>$

M5 $<(WHILE\ b\ S,m),\sigma> \rightarrow <WHILE\ (b\ S,m),\sigma>$

M6 $<(OP\ i=[e_1\ FOR\ e_2](X),m),\sigma> \rightarrow <(OP(X[\underline{e_1}_\sigma / i],...,X[\underline{e_1}_\sigma + \underline{e_2}_\sigma - 1 / i]),m),\sigma>$

(*) time introduction

T1 $<(SEQ((S_1,m),...,(S_n,m)),t),\sigma> \rightarrow <SEQ(S_1,m,t),...,(S_n,m)),\sigma>$

T2 $<(PAR((S_1,m_1),...,(S_n,m_n)),t),\sigma> \rightarrow <PAR((S_1,m_1,t),...,(S_n,m_n,t)),\sigma>$

T3 $<(ALT((g_1\ S_1,m),...,(g_n\ S_n,m)),t),\sigma> \rightarrow <ALT((g_1\ S_1,m,t),...,(g_n\ S_n,m,t)),\sigma>$

T4 $<(IF(b_1\ S_1,m),...,(b_n\ S_n,m)),t),\sigma> \rightarrow <IF((b_1\ S_1,m,t),...,(b_n\ S_n,m)),\sigma>$

T5 $<(WHILE\ (b\ S,m),t),\sigma> \rightarrow <WHILE\ (b\ S,m,t),\sigma>$

(*) E-removal

E1 $<SEQ((E,m,t)),\sigma> \rightarrow <(E,m,t),\sigma>$

$<SEQ((E,m,t),(S_2,m'),...),\sigma> \rightarrow <SEQ((S_2,m',t),...,\sigma>$

E2 $<PAR((E,m_1,t_1),...,(E,m_n,t_n)),\sigma> \rightarrow <(E,m_i,t_i),\sigma>\quad 1 \leqslant i \leqslant n$

E3 $<ALT(...,(E\ S,m,t),...),\sigma> \rightarrow <(S,m,t),\sigma>$

E4 $<IF((E,m,t)),\sigma> \rightarrow \rightarrow <(E,m,t),\sigma>$

$<IF((E,m,t),(b_2\ S_2,m),...),\sigma> \rightarrow <IF((b_2\ S_2,m,t),...),\sigma>$

(*) atomic actions

A1 $<(x:=e,m,t),\sigma> \xrightarrow{(\alpha,m,t+1)} <(E,m,t+1),\sigma>$

A2 $<(c?x,m,t),\sigma> \xrightarrow{(\delta_c,m,t)} <(c?x,m,t),\sigma>$

$<(c!e,m,t),\sigma> \xrightarrow{(\overline{\delta}_c,m,t)} <(c!e,m,t),\sigma>$

A3 $<(WAIT\ 0,m,t),\sigma> \rightarrow <(E,m,t),\sigma>$

$<(WAIT\ e,m,t),\sigma> \xrightarrow{(\epsilon_1,m,t+1)} <(WAIT\ \underline{e}_\sigma - 1,m,t+1),\sigma>$

A4 $<(SKIP,m,t),\sigma> \;-\!(\epsilon_1,m,t+1)\!\rightarrow\; <(E,m,t+1),\sigma>$

(*) other rules (and axioms)

R1 $$\frac{<(S,m,t),\sigma> \rightarrow^{\xi} <(S',m',t'),\sigma'>}{<SEQ((S,m,t),...),\sigma> \rightarrow^{\xi} <SEQ((S',m',t'),...),\sigma'>}$$

R2 $$\frac{<(S,m,t),\sigma> \rightarrow^{\xi} <(S',m',t'),\sigma'>}{<PAR(...,(S,m,t),...),\sigma> \rightarrow^{\xi} <PAR(...,(S',m',t'),...),\sigma'>}$$

$$\frac{first(S_i)=c?x \;\&\; first(S_j)=c!e \;\; or \;\; first(S_i)=c!e \;\&\; first(S_j)=c?x}{<PAR(...,(S_i,m_i,t_i),...,(S_j,m_j,t_j),...),\sigma> \rightarrow^{\xi} <PAR(...,(S_i',m_i,t),...,(S_j',m_j,t),...),\sigma'>}$$

where $\sigma'=\sigma[x:=e_\sigma]$, $t=max(t_i,t_j)+1$ and $\xi=(\tau{:}e_\sigma,(m_i,m_j),t)$ if m_i is the input-process or $\xi=(\tau{:}\underline{e}_\sigma,(m_j,m_i),t)$ if m_j is the input process and $S_i'=rest(S_i)$, $S_j'=rest(S_j)$ as explained before

R3 $$\frac{<(g_i,m,t),\sigma> \rightarrow^{\xi} <(g_i',m,t'),\sigma>}{<ALT(...,(g_i\; S_i,m,t),...),\sigma> \rightarrow^{\xi} <ALT(...,(g_i'\; S_i,m,t'),...),\sigma>}$$

R4 $<IF((b_1\; S_1,m,t),...),\sigma> \;-\!(\epsilon,m,t+1)\!\rightarrow\; <(S_1,m,t+1),\sigma>$ if $\underline{b}_\sigma=tt$

$<IF((b_1\; S_1,m,t),...),\sigma> \;-\!(\epsilon,m,t+1)\!\rightarrow\; <IF((E,m,t+1),...),\sigma>$ if $\underline{b}_\sigma=ff$

R5 $<WHILE\;(b\; S,m,t),\sigma> \;-\!(\epsilon,m,t+1)\!\rightarrow\; <(E,m,t+1),\sigma>$ if $\underline{b}_\sigma=ff$

$<WHILE\;(b\; S,m,t),\sigma> \;-\!(\epsilon,m,t+1)\!\rightarrow\; <SEQ((S,m,t),(WHILE\; b\; S,m)),\sigma>$ if $\underline{b}_\sigma=tt$

The replication of processes is already treated.
The other cases, procedure invocation, prioritised and placed composition are omitted.

DEFINITION: (*mapping computations to event-sequences*)
Let A_{roc} be the alphabet of elementary action-labels, augmented with the record of communication label indicating the transfer of a value v over a channel c, let $M \subseteq \mathbb{N}^+$ be the set of valid process-numbers, and let $E_0 \subseteq A_{roc} \times (M \cup M \times M) \times \mathbb{N} \times \Sigma$ be such that

$<\xi,m,t,\sigma>$	denotes that action ξ of process m ended at time t in state σ
$<\tau_c{:}v,\,(m_i,m_j),t,\sigma>$	denotes the transfer of value v from the output process m_j to the input process m_i at time t over channel c leading to state σ
$<\epsilon_0,m,t,\sigma>$	denotes some control-action that is unobservable and hence will be removed when collecting the meaning of the program

For $T, <S,\sigma> \vdash <S_0,\sigma_0> \rightarrow^{l_1} \cdots \rightarrow^{l_n} <S_n,\sigma_n> \rightarrow \cdots$ a computation define the corresponding event-sequence to be

$$T, <S,\sigma> \Vdash <\epsilon_0,m_0,t_0,\sigma_0> \rightarrow \cdots \rightarrow <\xi_n,m_n,t_n,\sigma_n> \rightarrow \cdots$$

where $l_i=<\xi_i,m_i,t_i>$ is a plain label or a *roc*-label, with the provision that if the computation is blocked (which now includes failure to communicate) than *FAIL* is appended to the event-sequence.

Definition: (*ϵ_0-restriction, τ-restriction, m-restriction, projection*)
For w an event-sequence let $w \setminus \epsilon_0$ be as defined previously the event-sequence with empty moves (except the first) eliminated, and let $w{\upharpoonright}\tau$ be the restriction of w to the events involving communication. Let $\Pi_m(w)$ be the restriction to the events in which all $\overline{m}' \geqslant m$ participate, with $\geqslant$ the prefix-ordering on process-numbers. Finally define $(e)_i\ ,i=0,\cdots,3$ to be the projection of e to its i-th component and similarly for $(\underline{w})_i$ extended to sequences over E_0

Definition: (*operational semantics*)

(0) behavioral:	$w \in \mathcal{O}_0[\![S]\!](\sigma)$ iff $T, <S,\sigma> \Vdash w$
(1) observable:	$\underline{\overline{w}} \in \mathcal{O}_1[\![S]\!](\sigma)$ iff $\exists w'.T, <S,\sigma\overline{>} \Vdash w'$ & $\underline{w} = w' \setminus \epsilon_0$
(2) Σ-sequence:	$\underline{\overline{w}} \in \mathcal{O}_2[\![S]\!](\sigma)$ iff $\exists \overline{w}'.T, <S,\sigma> \Vdash \overline{w}'$ & $\overline{w}_\Sigma \overline{=} (w' \setminus \epsilon_0)_3$
(3) τ-history:	$\underline{\overline{w}} \in \mathcal{O}_3[\![S]\!](\sigma)$ iff $\exists \underline{\overline{w}}'.T, <S,\sigma> \Vdash \underline{\overline{w}}'$ & $\underline{\overline{w}}_\tau = \underline{\overline{w}}'{\upharpoonright}\tau$

Example: $S_0 = SEQ(x:=1, PAR(SEQ(x:=2, c?x), c!3), x:=4)$
Let $S_1 = PAR(S_3, S_4)$ with $S_3 = SEQ(x:=2, c?x)$ and $S_4 = c!3$.
Let $\sigma_i = \sigma[x:=i]$ for $i=1,...,4$ and assume execution is started at time 0 and process 0.

(1) $<(S_0,0,0),\sigma> \rightarrow <SEQ((x:=1,0,0),(S_1,0),(x:=4,0)),\sigma>$ by M1 and T1

(2) $<(x:=1,0,0),\sigma> \xrightarrow{(\alpha,0,0)} <(E,0,1),\sigma_1>$ by A1

(3) $<(S_0,0,0),\sigma> \xrightarrow{(\alpha,0,1)} <SEQ((S_1,0,1),(x:=4,0)),\sigma_1>$ from (1),(2) by R1 and E1

(4) $<(S_1,0,1),\sigma_1> \rightarrow <PAR((S_3,01,1),(S_4,02,1)),\sigma_1>$ by M2 and T2

(5) $<(S_3,01,1),\sigma_1> \rightarrow <SEQ((x:=2,01,1),(c?x,01)),\sigma_1>$ by M1 and T1

(6) $<(x:=2,01,1),\sigma_1> \xrightarrow{(\alpha,01,2)} <(E,01,2),\sigma_2>$ by A2

(7) $<SEQ((x:=2,01,1),(c?x,01)),\sigma_1> \xrightarrow{(\alpha,01,2)} <SEQ((c?x,01,2)),\sigma_2>$ by (6), R1 and E1

(8) $<PAR((S_3,01,1),(c!3,02,1)),\sigma_1> \xrightarrow{(\alpha,01,2)} <PAR(SEQ((c?x,01,2)),(c!3,02,1)),\sigma_2>$
by (7) and R2(a)

(9) $<PAR(SEQ((c?x,01,2)),(c!3,02,1)),\sigma_2> \xrightarrow{(\tau:3,(01,02,3)}$
$<PAR((E,01,3),(E,02,3)),\sigma_3>$ by R2(b), E1 and R2(a)

(10) $<PAR((E,01,3),(E,02,3)),\sigma_3> \rightarrow <(E,01,3),\sigma_3>$ by E2

(11) $<SEQ((S_1,0,1),(x:=4,0)),\sigma_1> \xrightarrow{(\alpha,01,2)} \cdots \xrightarrow{(\tau:3,(01,02),3)} \cdots$
$<SEQ((E,01,3),(x:=4,0)),\sigma_3>$ by (4),(8) and (9)

(12) $<SEQ((E,01,3),(x:=4,0)),\sigma_3> \rightarrow <SEQ(((x:=4,0,3)),\sigma_3>$

etcetera.
So omitting empty moves:

$$<S_0,\sigma> \Vdash <\epsilon_0,0,0,\sigma> \rightarrow <\alpha,0,1,\sigma_1> \rightarrow <\alpha,01,2,\sigma_2> \rightarrow <\tau_c:3,(01,02),3,\sigma_3> \rightarrow <\alpha,0,4,\sigma_4>$$

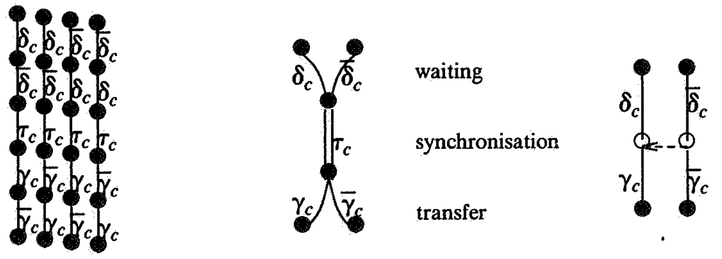
and, for example, the one element sequence $<\tau_c:3,(01,02),3,\sigma_3> \in \mathcal{O}_3[\![S_0]\!](\sigma)$.

3. Denotational semantics

Processes were introduced in (BZ1) for the denotational semantics of concurrency, synchronisation and synchronisation with value passing.
In (BKMOZ) the relation of this type of semantics with the operational semantics by means of transition-systems is further elucidated. A notable difference between the language with synchronisation and value-passing treated in (BKMOZ) with Occam is the absence of recursion in Occam and the guardedness of alternative composition. Otherwise the operational semantics given in the previous section closely resembles the one given in (BKMOZ), except for process-numbering and timing. Also the denotational semantics will be set up as in (BKMOZ).
The problem faced here is, similarly as for the operational semantics, how to include process-numbering and timing in the domain of processes. As will be explained in more detail later the domain of processes is given as the solution of a domain-equation of the form

$$\mathbb{P}=\{p_0\} \cup \Sigma\to\mathcal{P}_c(\Sigma\times\mathbb{P})$$

where $(\mathbb{P},d)$ is the metric space constructed as the completion of the union of finite processes and their associated metrics. This type of processes can be regarded as synchronisation trees under a certain equivalence relation, that is processes are commutative trees with sets rather than multi-sets as successors.
Later an extension will be formulated in which processes resemble event-structures, or partial orders, instead of trees.
For both types of processes, due to the introduction of process-numbering and timing, the structure of communication is slightly more complex than when these aspects are omitted from the treatment.



(a) (b) (c)

Simplifying matters the event of communication could be pictured as above. Three phases, both for the input and output process, are involved: *waiting*, *synchronisation*, and *value-transfer*.
In (a) the interleaving of these component-events is pictured, in (b) synchronisation is seen to be common to both the input and the output process, and in (c) synchronisation is pictured as some event connecting the input and output process, so to speak, at the system-level instead of the process-level. The actual representation used will shift from (a) in this section to (c) in the next section.
However due to process-numbering and timing they can both be regarded equivalent to (b).
Actually the role of the δ-component is different than pictured here. Intuitively (c) appears to be the best representation since the open nodes can be considered as allowing a choice between different communications. The extension with respect to the treatment of communication in (BKMOZ) consists of an explicit distinction between transfer (or synchronisation assignment for the input process) and synchronisation itself. In this setup the counterpart of synchronisation assignment for the

output-process is just skip.

DEFINITION: (*(processes)*)
Let C be the set of channel-names.

$$A_{roc}=\{\alpha,\epsilon\} \cup \{\delta_c,\bar{\delta}_c,\gamma_c,\bar{\gamma}_c,\tau_c{:}v \mid c\in C, v\in V\},$$

$M\subseteq\mathbb{N}^+$ the set of process-numbers as introduced previously,
$\underline{M}\subseteq M\cup M\times M$ and $\underline{\mathbb{N}}\subseteq\mathbb{N}\cup\mathbb{N}\times\mathbb{N}$ then the equation

$$\mathbb{P}=\{p_o\}\cup M\times\mathbb{N}\times\Sigma\to\mathcal{P}_c(A_{roc}\times\underline{M}\times\underline{\mathbb{N}}\times\Sigma\times(\mathbb{P}\cup V\to\mathbb{P}\cup V\times\mathbb{P}))$$

is given by:

$$(0)\quad \mathbb{P}_0=\{p_0\}\ ,\ d_0(p',p'')=0 \text{ for } p',p''\in\mathbb{P}_0$$

$$(n+1)\quad \mathbb{P}_{n+1}=\{p_o\}\cup M\times\mathbb{N}\times\Sigma\to\mathcal{P}_c(A_{roc}\times\underline{M}\times\underline{\mathbb{N}}\times\Sigma\times(\mathbb{P}_n\cup V\to\mathbb{P}_n\cup V\times\mathbb{P}_n))$$

with the associated distance

$$d_{n+1}(p',p'')=\begin{cases}1 \text{ if } p'=p_0\ \&\ p''\neq p_0 \quad\text{or}\quad p''=p_0\ \&\ p'\neq p_0\\ \sup_m\sup_t\sup_\sigma \quad d_{n+1}(p'(m,t,\sigma),p''(m,t,\sigma))\end{cases}$$

where $d_{n+1}(p'(m,t,\sigma),p''(m,t,\sigma))$ is the Hausdorff distance induced by

$$d_{n+1}(<\xi,m,t,\sigma,\rho>,<\xi',m',t',\sigma',\rho'>)=\begin{cases}1 \text{ if } (\xi,m,t,\sigma)\neq(\xi',m',t',\sigma')\\ \tfrac{1}{2}d_n(\rho,\rho')\end{cases}$$

with for $\rho,\rho'\in V\to\mathbb{P}_n$ $d_n(\rho,\rho')=\sup_v d_n(\rho(v),\rho'(v))$ and for $\rho,\rho'\in V\times\mathbb{P}_n$ $d_n((v,p),(v',p'))=1$ if $v\neq v'$ and $d_n(p,p')$ if $v=v'$. If ρ and ρ' are of different 'types' then $d_n(\rho,\rho')=1$.
For each $n\geqslant 0$ $(\mathbb{P}_n,d_n)$ is a metric space. The union of these spaces $(\mathbb{P}_\omega,d_\omega)=(\cup\mathbb{P}_n,\cup d_n)$.
Now $(\mathbb{P},d)$ is taken to be the completion of $(\mathbb{P}_\omega,d_\omega)$

The *degree* of a process is defined by $degree(p_0)=0$, $degree(p_n)=n$ if $p\in\mathbb{P}_n\setminus\mathbb{P}_{n-1}$ for some $n\geqslant 1$ and $degree(p)=\infty$ otherwise.

Operators on processes are commonly defined by induction on their *degree*. For infinite processes $p=\lim_n p_n$, $q=\lim_n q_n$ an operator *op* is defined by $p\ op\ q=\lim_n(p_n\ op\ q_n)$.
Continuity-results for several operators on several domains are provided in (BZ1). In the definition of operators an explicit mention of the infinite case will mostly be omitted.

A 5-tuple of the form $<\xi,m,t,\sigma,\rho>$ with $\rho\in(\mathbb{P}\cup V\to\mathbb{P}\cup V\times\mathbb{P})$ is called a *configuration* in which ξ denotes the label of the step leading to the configuration, m the process it belongs to, t the (local synchronised) time it took place and ρ the resumption that is either a plain process, a process that has a value as an extra parameter or a value-process pair in case it is the resumption of a communication-intention.

Possible configurations and their functions are:

$<\alpha,m,t,\sigma,p>$	local assignment
$<\delta_c,m,t,\sigma,\underline{\theta}>$	input-intention with $\theta \in V \rightarrow \mathbb{P}$
$<\overline{\delta}_c,m,t,\sigma,\overline{\theta}>$	output-intention with $\overline{\theta} \in V \times \mathbb{P}$
$<\tau_c:v,(m,m'),(t,t'),\sigma,p>$	synchronisation
$<\gamma_c,m,t,\sigma,p>$	communication-assignment (input)
$<\overline{\gamma}_c,m,t,\sigma,p>$	communication value transfer (output)
$<\epsilon,m,t,\sigma,p>$	empty move (wait, skip, test)
$<\circ,m,t,\sigma,p>$	guard-control
$<\bullet,m,t,\sigma,p>$	to be explained later

With the auxiliary process-number and the time-fields in the configuration it is possible to keep track of the activity of one process and the history of communication between processes.

As an example the denotation of an assignment-statement in a particular environment is

$$\llbracket x:=e \rrbracket(\gamma) = \lambda mt\sigma.\{<\alpha,m,t+1,\sigma[x:=\underline{e}_\sigma],p_0>\}$$

which states that an assignment takes unit-time and changes the state it is given with respect to the value for x.

DEFINITION: (*denotational semantics*)

$$\llbracket x:=e \rrbracket(\gamma) = \lambda mt\sigma.\{<\alpha,m,t+1,\sigma[x:=\underline{e}_\sigma],p_0>\}$$

$$\llbracket c?x \rrbracket(\gamma) = \lambda mt\sigma.\{<\delta_c,m,t,\sigma,\lambda v.\lambda m't'\sigma'.\{<\gamma_c,m,t'+1,\sigma'[x:=v],p_0>\}>\}$$

$$\llbracket c!e \rrbracket(\gamma) = \lambda mt\sigma.\{<\overline{\delta}_c,m,t,\sigma,(v,\lambda m't'\sigma'.\{<\gamma_c,m,t'+1,\sigma',p_0>\}>\}$$

$$\llbracket WAIT\ e \rrbracket(\gamma) = \lambda mt\sigma.\{<\epsilon,m,t+1,\ldots\lambda m_n t_n \sigma_n.\{<\epsilon,m,t+\underline{e}_\sigma,\sigma,p_0>\}\ \ldots\ >\}$$

$$\llbracket SKIP \rrbracket(\gamma) = \lambda mt\sigma.\{<\epsilon,m,t+1,\sigma,p_0>\}$$

$$\llbracket SEQ(S_1,\ldots,S_n) \rrbracket(\gamma) = \llbracket S_1 \rrbracket(\gamma);\cdots;\llbracket S_n \rrbracket(\gamma)$$

$$\llbracket PAR(S_1,\ldots,S_n) \rrbracket(\gamma) = \lambda mt\sigma.\llbracket S_1 \rrbracket(\gamma)(m\hat{\ }1,t,\sigma)\ \|\ \ldots\ \|\ \lambda mt\sigma.\llbracket S_n \rrbracket(\gamma)(m\hat{\ }n,t,\sigma)$$

$$\llbracket b\ S \rrbracket(\gamma) = b \supset \llbracket S \rrbracket(\gamma), \varnothing$$

$$\llbracket ALT(g_1\ S_1,\ldots,g_n\ S_n) \rrbracket = \Delta(\llbracket g_1 \rrbracket(\gamma));\llbracket S_1 \rrbracket(\gamma) + \ \ldots\ + \Delta(\llbracket g_n \rrbracket(\gamma));\llbracket S_n \rrbracket(\gamma)$$

$$\llbracket IF(b_1\ S_1,\ldots,b_n\ S_n) \rrbracket(\gamma) = b_1 \supset \llbracket S_1 \rrbracket(\gamma),\ldots,b_n \supset \llbracket S_n \rrbracket(\gamma),p_0$$

$$\llbracket WHILE\ b\ S \rrbracket(\gamma) = \lim_i\ ,\ p_0 = p_0\ \ p_{i+1} = b \supset \llbracket S \rrbracket(\gamma);p_i,p_0$$

$$\llbracket OP\ i=[e_1\ FOR\ e_2](X) \rrbracket =$$

$$\lambda mt\sigma.\llbracket OP(X)(\gamma)(m,t,\sigma[i:=\underline{e_1}_\sigma])\ op\ \ \ldots\ \ op\ \lambda mt\sigma.\llbracket OP(X)(\gamma)(m,t,\sigma[i:=\underline{e_1}_\sigma+\underline{e_2}_\sigma-1])$$

for $OP \in \{SEQ,PAR,ALT,IF\}$ and corresponding $op \in \{;,\|,+,\supset\}$

$$\llbracket p(c_1,\ldots,c_n) \rrbracket(\gamma) = \gamma(p)(c_1,\ldots,c_n)$$

Remarks:

- The definitions for prioritised and placed statements are omitted as not to burden the configurations with even more fields.

- The slightly awkward definition for the *PAR*-construct shows the assignment of process-numbers in a functional fashion, that is the process-number given as an argument is modified as indicated. Assigning process-numbers when applying the operator for parallel composition has the disadvantage that the operator would no longer be associative and commutative as required.
- The definitions for *IF* and *WHILE* make use of the operator $\supset : Bexp \times \mathbb{P} \times \mathbb{P} \to \mathbb{P}$ to be defined shortly. The operator is used to facilitate the treatment of timing.
- The clause for the alternative construct will be explained later.

DEFINITION: (*operators on processes*)

Let $\theta \in V \to \mathbb{P}$, $(v,p) \in V \times \mathbb{P}$ and $\rho \in (\mathbb{P} \cup V \to \mathbb{P} \cup V \times \mathbb{P})$

(1) sequential composition; $p_0;p = p;p_0$. For $p,q \neq p_0$

$$p;q = \lambda mt\sigma.(p(m,t,\sigma);\lambda m't'\sigma'.q(m,t',\sigma')\rangle$$

$$X;q = \{x;q : x \in X\}$$

$$<\xi,m,t,\sigma,p_0>;q = <\xi,m,t,\sigma,\lambda m't'\sigma'.q(m',t,\sigma')>$$

$$<\xi,m,t,\sigma,\rho>;q = <\xi,m,t,\rho;q>$$

$$\theta;q = \lambda v.(\theta(v);q) \qquad (v,p);q = (v,p;q)$$

(2) alternative composition: $p_0 + p = p + p_0 = p$. For $p,q \neq p_0$

$$p+q = \lambda mt\sigma.(p(m,t,\sigma) \cup q(m,t,\sigma))$$

(3) parallel composition: $p_0 \| p = p \| p_0 = p$. For $p,q \neq p_0$

$$p \| q = \lambda mt\sigma.(p(m,t,\sigma) \| \lambda m't'\sigma'.q(m,t,\sigma') + q(m,t,\sigma) \| \lambda m't'\sigma'.p(m,t,\sigma') + p((m,t,\sigma) \mid q(m,t,\sigma))$$

$$X \| q = \{x \| q : x \in X\}$$

$$<\xi,m,t,\sigma,\rho> \| q = <\xi,m,t,\sigma,\rho \| q>$$

$$\theta \| q = \lambda v.(\theta(v) \| q) \qquad (v,p) \| q = (v,p \| q)$$

$$X \mid Y = \cup\{x \mid y : x \in X, y \in Y\} \quad \text{with}$$

$$<\delta_c,m,t,,\sigma,\theta> \mid <\bar{\delta}_c,m',t',\sigma,(v,p)> = \{<\tau_c:v,(m,m'),(t,t'),\sigma),\lambda m''t''\sigma''.\theta(v)(m,\max(t,t'),\sigma'') \| \lambda m''t''\sigma''.p(m',\max(t,t'),\sigma'')>\}$$

and $x \mid y = \varnothing$ otherwise

(4) conditional composition: $\supset : Bexp \times \mathbb{P} \times \mathbb{P} \to \mathbb{P}$ is defined by

$$b \supset p,q = \lambda mt\sigma.\text{if } \underline{b}_\sigma \text{ then } \{<\epsilon,m,t+1,\sigma,p_0>\};p \text{ else } \{\epsilon,m,t+1,\sigma,p_0>\};q \text{ fi}$$

Also $(b_1 \supset p_1,\ldots,b_n,p_n,p_0) \supset (b_1' \supset p_1',\ldots,b_m' \supset p_m',p_0)$ is defined to be

$$b_1 \supset p_1,\ldots,b_n \supset p_n, b_1' \supset p_1',\ldots,b_m' \supset p_m',p_0$$

(5) restriction: $\backslash C : \mathbb{P} \rightarrow \mathbb{P}$ where C is an arbitrary set of channel-names

$$p_0 \backslash C = p_0$$

$$(\lambda mt\sigma.X) \backslash C = \lambda mt\sigma.X \backslash C$$

$$X \backslash C = \cup \{ x \backslash C : x = <\xi, \cdots > \; \& \; \xi \neq \delta_c \textit{ or } \bar{\delta}_c \textit{ for any } c \in C \}$$

$$<\xi,m,t,\sigma,p> \backslash C = \{<\xi,m,t,\sigma,p \backslash C>\}$$

Remarks:

- The definition of sequential composition is not satisfactory with respect to timing. A process continuing after a parallel branching in a sequential composition takes over the local clock-value of an arbitrary component. This is however not as harmful as it may seem since all possibilities are covered due to the 'arbitrariness' of interleaving.
  Further treatment of this aspect is postponed until a suitable *timing − operator* is introduced.
- Note that under the assumption that the correct timing is delivered the synchronisation-step $<\tau,...>$ also contains the record of communication (*input,output,value*) and the respective wait-times.
- The treatment of guarded commands also has to wait for the definition of a proper timing-operator.
- The restriction-operator eliminates communication-intentions $<\delta_c,...>$, $<\bar{\delta}_c,...>$. It is introduced here to facilitate the treatment of examples.

THEOREM:
(1) || and ; are associative.
(2) || is commutative
(3) ||, ;, + and ⊃ are continuous.

Proof: Omitted.

EXAMPLE: (*synchronisation merge*)

To calculate the meaning of $PAR(SEQ(x:=1, c?x), c!2)$ let

$$p_1 = \lambda mt\sigma.\{<\alpha,m,t+1,\sigma_1,p_2>\} \quad \text{with} \quad \sigma_1 = \sigma[x:=1]$$

$$p_2 = \lambda m't'\sigma'.\{<\delta_c,m,t+1,\sigma',\theta>\}$$

$$\text{with} \quad \theta = \lambda v.\lambda m''t''\sigma''.\{<\gamma_c,m,t''+1,\sigma''[x:=v],p_0>\}$$

and

$$q_1 = \lambda \underline{mt\sigma}.\{\bar{\delta}_c,\underline{m},\underline{t},\sigma\_,(2,q_2)>\}$$

$$q_2 = \lambda \underline{m't'\sigma'}.\{<\bar{\gamma}_c,\underline{m},\underline{t'}+1,\underline{\sigma'},p_0>\}$$

then

$$[\![PAR(SEQ(x:=1,c?x),c!2)]\!](\gamma)(0,0,\sigma) = (p \,||\, q)(0,0,\sigma)$$

with $p = \lambda mt\sigma.p_1(m\hat{}1,t,\sigma)$, $q = \lambda mt\sigma.q_1(m\hat{}2,t,\sigma)$ and

$$\begin{aligned} p \,||\, q = \lambda mt\sigma.(&p_1(m\hat{}1,t,\sigma) \,||\, \lambda \underline{\underline{mt\sigma}}.q_1(m\hat{}2,t,\underline{\underline{\sigma}}) + \\ &q_1(m\hat{}2,t,\sigma) \,||\, \lambda \underline{\underline{mt\sigma}}.p_1(m\hat{}1,t,\underline{\underline{\sigma}}) + \\ &p_1(m\hat{}1,t,\sigma) \,|\, q_1(m\hat{}2,t,\sigma)) \end{aligned}$$

and hence (applying the restriction operator)

$$
\begin{aligned}
&((p \parallel q)\setminus C)(0,0,\sigma)\\
&\quad = (\{<\alpha,01,1,\sigma_1,p_2>\} \parallel \lambda\underline{\underline{mt}}\sigma.\{<\bar{\delta}_c,02,0,\underline{\underline{\sigma}},p_0>\} = \\
&\qquad \{<\bar{\delta}_c,02,0,\sigma,p_0>\} \parallel \lambda\underline{\underline{mt}}\sigma.\{<\alpha,01,1,\underline{\underline{\sigma_1}},p_2>\})\setminus C\\
&\quad = (\{<\alpha,01,1,\sigma_1,p_2 \parallel \lambda\underline{\underline{mt}}\sigma.\{<\bar{\delta}_c,02,0,\underline{\underline{\sigma}},p_0>\}.\} + \{<\bar{\delta}_c, \cdots >\})\setminus C\\
&\qquad (\{<\alpha,o1,1,\sigma_1,\lambda m't'\sigma'.(\{<\delta_c, \cdots >\}+\{<\bar{\delta}_c, \cdots >\}+\\
&\qquad\quad \{<\tau_c{:}2,(01,02),(1,0),\sigma',\lambda m''t''\sigma''.\theta(2)(m'',1,\sigma'') \parallel \lambda m''t''\sigma''.q_2(m'',1,\sigma'')>\})>\} +\\
&\qquad\quad \{<\bar{\delta}_c, \cdots >\})\setminus C\\
&\quad = \{<\alpha,01,1,\sigma_1,\lambda m't'\sigma'.\{<\tau_c{:}2,(01,02),(1,0),\sigma', \cdots >\}>\}
\end{aligned}
$$

Note that the resumption after the synchronisation between process 01 and 02 is itself an interleaving:

$$
\begin{aligned}
&\lambda\underline{mt}\sigma.\{<\gamma_c,01,2,\sigma\_[x:=2],\lambda\underline{\underline{mt}}\sigma.\{<\bar{\gamma}_c,02,2,\underline{\underline{\sigma}},p_0>\}>,\\
&\qquad <\bar{\gamma}_c,02,2,\sigma\_,\lambda\underline{\underline{mt}}\sigma.\{\gamma_c,01,2,\underline{\underline{\sigma}}[x:=2],p0>\}>\}
\end{aligned}
$$

This example shows that when the merge takes place, process-numbers and times are fixed at the 'top-level', an update of the local times occurs at the event of synchronisation.

In order to compare the denotational semantics with the operational semantics given in the previous chapter the operator $paths{:}\mathbb{P}\setminus\{p_0\}\to(M\times\mathbb{N}\times\Sigma\to\mathbb{E})$ is needed.

DEFINITION: (*paths*)

Let $E_0 \subseteq A_{roc}\times M\times N\times\Sigma$, $\mathbb{E} = E_0^* \cup E_0^\omega \cup E_0^*\cdot\{FAIL\}$ and $\cdot$ the concatenation operator for sequences in $\mathbb{E}$.
Then the operator *paths* is defined by:

$$
\begin{aligned}
&paths(\lambda mt\sigma.X) = <\epsilon_0,m,t,\sigma>\cdot paths(X)\\
&paths(X) = \ \cup\ \{\, paths(x) : x\in X \,\}\ ,\ X\neq\varnothing\\
&\qquad\qquad = \{FAIL\}\ ,\ \ X=\varnothing\\
&paths(<\xi,m,t,\sigma,p_0>) = \{<\xi,m,t,\sigma>\}\\
&paths(<\xi,m,t,\sigma,p>) = <\xi,m,t,\sigma>\cdot paths(p(m,t,\sigma))\\
&paths(<\delta_c,\ldots>) = paths(<\bar{\delta}_c,\ldots>) = \{FAIL\}
\end{aligned}
$$

CONJECTURE: (*relation-ship event-sequences and paths*)

$$<S,\sigma> \vdash \underline{w} \;\; \& \;\; \underline{w}' = \underline{\hat{w}} \setminus \epsilon_0 \;\Leftrightarrow\; \underline{w}' \in paths([\![S]\!](\gamma) \setminus C)(0,0,\sigma)$$

with $\underline{\hat{w}}$ the appropriate modification of $\underline{w}$ with respect to events of synchronisation and transfer.

Remarks:

- The definition of communication in the transition-system collapsed synchronisation and transfer into one event. This definition could be adapted in a rather straightforward manner.
- The conjecture is not correct with respect to alternative commands.

To retrieve the state transforming function of a process the *yield* p^+ of a process is defined as in (BKMOZ).
Put $\Sigma_\perp = \Sigma \cup \{FAIL\} \cup \{\perp\}$. The order on $\Sigma_\perp$ is defined by putting $\sigma_1 \sqsubseteq \sigma_2$ iff $\sigma_1 = \perp$ or $\sigma_1 = \sigma_2$.
Let $\mathfrak{T} = \mathfrak{P}(\Sigma_\perp)$ and for $T_1,\ T_2 \in \mathfrak{T}$ define
$T_1 \sqsubseteq_{\mathfrak{T}} T_2$ iff $\perp \in T_1$ & $T_1 \setminus \{\perp\} \subseteq T_2$ or $T_1 = T_2$ (the Egli-Milner order).
Then $(\mathfrak{T}, \sqsubseteq_{\mathfrak{T}}, \{\perp\})$ is a CPO.

DEFINITION: (*yield*)

The mapping $+ : \mathbb{P} \setminus \{p_0\} \to (\Sigma \to \mathfrak{T})$ is given by

$$p^+ = \lambda\sigma.p(0,0,\sigma)^+$$
$$X^+ = \bigsqcup\nolimits_{\mathfrak{T},n} X^{(n)}$$

where $\bigsqcup_{\mathfrak{T}}$ is the lub in $(T, \sqsubseteq_{\mathfrak{T}}, \{\perp\})$ and $X^{(n)}$ is defined by

$$X^{(0)} = \{\perp\}$$
$$X^{(n+1)} = \bigcup \{ x^{(n+1)} : x \in X \}$$
$$<\xi, m, t, \sigma, p_0>^{(n+1)} = \{\sigma\}$$
$$<\xi, m, t, \sigma, p>^{(n+1)} = p(m,t,\sigma)^{(n)}$$
$$<\delta_c, \ldots>^{(n+1)} = <\bar{\delta}_c, \ldots>^{(n+1)} = \{FAIL\}$$

In other words the $+$-operator unwinds the process p in σ, neglecting process-numbering and timing.

EXAMPLE:
The yield p^+ for $[\![PAR(SEQ(x:=1, c?x), c!2)]\!](\gamma)$ would be

(0) $\lambda\sigma.p(0,0,\sigma)^{(0)} = \lambda\sigma.\{\perp\}$

(1) $\lambda\sigma.p(0,0,\sigma)^{(1)} = \lambda\sigma.\{<\tau_c, \ldots>\}^{(0)} = \lambda\sigma.\{\perp\}$

(2) $\lambda\sigma.p(0,0,\sigma)^{(2)} = \lambda\sigma.\{<\gamma_c, \ldots>\}^{(0)} \cup \lambda\sigma.\{<\bar{\gamma}_c, \ldots>\}^{(0)} = \lambda\sigma.\{\perp\}$

.

(4) $\lambda\sigma.p(0,0,\sigma)^{(4)} = \lambda\sigma.\{<\bar{\gamma}_c, 02, 2, \sigma[x:=2], p_0>\}^{(1)} \cup \lambda\sigma.\{<\gamma_c, 02, 1, \sigma[x:=2], p_0>\}^{(1)}$
$= \lambda\sigma.\sigma[x:=2]$

Now to cope with guarded commands it seems necessary to introduce an operator that measures the time a statement took to be executed. As a preparation a more general timing operator is introduced first.

DEFINITION: (*timing*)
Let $(\mathbb{N},\leq)$ be the total order of time-values and $M=(\mathbb{N}^+,\leq)$ the partial order of process-numbers with $\leq$ the prefix order and 0 as a least element.
Then the collection of functions $time_{m,t}:\mathbb{P}\rightarrow\mathbb{N})$ can be defined by

$$time_{m,t}(p_0) = t$$

$$time_{m,t}(p) = \lambda m't'\sigma'.time_{m,t}(p(m,t,\sigma'))$$

$$time_{m,t}(X) = \sup_n time_{m,t}(X^{(n)}) \text{ with}$$

$$time_{m,t}(X^{(n)}) = \max\{ time_{m,t}(x^{(n)}) : x\in X \} \text{ if } X\neq\varnothing \text{ and } \infty\text{if } X=\varnothing$$

$$time_{m,t}(<\delta_c,...>^{(n)}) = time_{m,t}(<\overline{\delta}_c,...>^{(n)}) = \infty$$

$$time_{m,t}(<\xi,m',t',\sigma,p>^{(0)}) = t$$

$$time_{m,t}(<\xi,m',t',\sigma,p_0>^{(n+1)}) = t' \text{ if } m\leq m' \text{ and } t \text{ if } \neg m\leq m'$$

$$time_{m,t}(<\xi,m',t',\sigma,p>^{(n+1)}) = time_{m,t'}(p(m,t',\sigma')^{(n)}) \text{ if } m\leq m'$$

$$= time_{m,t}(p(m,t',\sigma')^{(n)}) \text{ otherwise}$$

Notice that in order to estimate the time-behavior of a process communication-intentions should be eliminated.

The timing-operator can be used for passing a more correct clock-value to a successor-component in a sequential composition by defining the operator for sequential composition at the top-level as:

$$p;q = \lambda mt\sigma.(p(m,t,\sigma);\lambda m't'\sigma'.q(m,time_{m,t}(p(m,t,\sigma)\setminus C),\sigma')$$

but this has the drawback that the component has to be assumed closed for further communications. Note however that the component is not actually closed for communication. The timing-operator applied in the way shown gives a reliable lower-bound for the derivation of the execution of a statement.

THE TREATMENT OF GUARDED COMMANDS

A guarded command in Occam can occur as a component of the alternative construct $ALT(g_1\ S_1,...,g_n\ S_n)$. The guard g_i can be of the form b b C or b $WAIT$ e for C an input or output. The convention is that a branch with a trivial guard b is always chosen over branches with non-trivial guards. If there are only non-trivial guards a selection is made between branches whose guards became ready first.
From this discussion it follows that it would suffice to locate the guards that take the least time to get ready. In assessing the denotational semantics the other branches could be restricted away.
The approach followed here is to enclose the guard by auxiliary steps, a step that indicates when the guard starts to be active (○) and a step that indicates that the guard is ready (●).
For convenience assume guarded commands do not occur nested.

DEFINITION: (*guard-enclosure, guard-timing, branch-selection*)

Let $\Delta:\mathbb{P}\rightarrow\mathbb{P}$ be defined by

$$\Delta(p) = \lambda mt\sigma.\{<\circ,m,t,\sigma,p_0>\};p\,;\lambda mt\sigma.\{<\bullet,m,t,\sigma,p_0>\}$$

that is p is surrounded by $\circ$ and $\bullet$.
Define also (the collection of functions)

$$guard\text{ - }time_{m,t}(X) = \min\,\{guard\text{ - }time_{m,t}(x) : x\in X\},\text{ where}$$

$$guard\text{ - }time_{m,t}(<\delta_c,\ldots>) = guard\text{ - }time_{m,t}(<\bar{\delta}_c,\ldots>) = \infty$$

$$guard\text{ - }time_{m,t}(<\bullet,m',t',\sigma,p>) = t\ \text{ if } m=m' \text{ and } \infty \text{ if } m\neq m'$$

$$\begin{aligned} guard\text{ - }time_{m,t}(<\xi,m',t',\sigma,p>) &= \infty \ \text{ if } p=p_0 \\ &= guard\text{ - }time_{m,t'}(p(m',t',\sigma')) \ \text{ if } m=m' \\ &= guard\text{ - }time_{m,t}(p(m',t',\sigma')) \ \text{ if } m\neq m' \end{aligned}$$

Next the operator $select\text{-}branch_{m,t}$ is defined by

$$\begin{aligned} select\text{-}branch_{m,t}(X) = \{\ <\circ,m,t,\sigma,p> \in X\ :\ \neg\exists q.\ <\circ,m,t,\sigma,q> \in X\ \& \\ guard\text{ - }time_{m,t}(q(m,t,\sigma)) < guard\text{ - }time_{m,t}(p(m,t,\sigma))\ \} \end{aligned}$$

Then to get the desired pruning-operator that eliminates non-candidate branches of an alternative composition it suffices to define the restriction operator $prune:\mathbb{P}\rightarrow\mathbb{P}$ by

$$prune(p_0) = p_0$$

$$prune(\lambda mt\sigma.X) = \lambda mt\sigma.prune(X)$$

$$prune(X) = \{\ <\xi,\ldots> \in X : \xi\neq\circ\ \}\ \cup$$

$$\bigcap\ \{\ select\text{-}branch_{m,t}(X') : <\circ,m,t,\sigma,p> \in X' \subseteq X\ \}$$

LEMMA: (*prune is well-defined and gives the desired result*)
Proof: Omitted.

EXAMPLE:

$[\![ALT(true\rightarrow x:=1, WAIT\ 2\rightarrow x:=2)]\!](\gamma)(0,0,\sigma) =$

$\{<\circ,0,0,\sigma,\lambda m't'\sigma'.\{<\epsilon,0,1,\sigma',\lambda m''t''\sigma''.\{<\bullet,0,1,\sigma'',\lambda m'''t'''\sigma'''.\{<\alpha,0,2,\sigma'''[x:=1],p_0>\}>\}>\}>,$

$<\circ,0,0,\sigma,\lambda m't'\sigma'.\{<\epsilon,0,1,\sigma',\lambda m''t''\sigma''.\{<\epsilon,0,2,\sigma'',\lambda m'''t'''\sigma'''.\{<\bullet,0,2,\sigma''',$

$\lambda m''''t''''\sigma''''.\{<\alpha,0,3,\sigma''''[x:=2],p_0>\}>\}>\}>\}>\}$

and when this set is pruned only the first element with the dot on time 1 remains.

4. The concept of alternation

"Alternation is a generalisation of non-determinism in which existential and universal quantifiers alternate during the course of a computation whereas in non-deterministic computation there are only existential quantifiers".(CKS)

4.1. Introduction: an orthogonal treatment of parallelism and nondeterminism

In (CKS) the concept of alternation is used to describe a generalisation of the Turing machine model. Each machine when encountering a parallel or non-deterministic instruction spawns off, to use their phrasing, a collection of machines of which the results are combined in a conjunctive or disjunctive fashion dependent on whether the branching is parallel or non-deterministic. Infinite, that is non-terminating, computations are dealt with, as usual, by finite approximations. The authors wonder why the concept hasn't been applied to the semantics of parallelism outside the context of Turing-machines and complexity and, frankly, so do I.

But before starting to venture in this type of semantics let us shortly inventarise the state of the art concerning the semantics of communicating processes and the problems to expect for the new semantics.

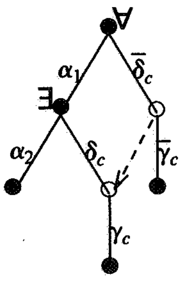
● **linear time and braching time semantics**: (LT,BT)
In (BBKM) it is sufficiently argued that with respect to deadlock behavior a linear time (stream) semantics is unable to cope with the difference between (the uniform processes) $a;b+a;c$ and $a;(b+c)$, where c denotes a communication-intention. A tree representation reflects this difference by indicating at what point the choice between a local action and a communication intention is to be made.
Branching time models of communicating processes are conveniently pictured as a type of synchronisation trees in which the act of synchronisation disappears as a silent non-observable action indicating the confluence of matching communication-intentions. A proper formulation of such models can be done in the metric space of processes as presented in the previous section where the equivalence between processes is dealt with by taking closed sets as successors rather than arbitrary multi-sets. (Cf. BKMOZ) The approach first advocated by Milner can be characterised as a reduction of parallelism to non-determinism under the restriction of synchronisation by expanding a parallel composition into its allowed interleavings.

● **partial order semantics**
(Re) observes that the total order of events as represented by an interleaving semantics is only partially due to the dependencies implicit in the program. Where the implicit order is not total it is a result of the need to order the observations that can be made of the execution of the program and the assumption of a discrete totally ordered time-scale. This assumption limits the fineness with which equivalence between programs can be assessed.
It is claimed by (Re) that a partial order semantics gives the desired (orthogonal) treatment of parallelism and non-determinism. However net-theory, from which the partial order approach originates, gives only a thorough treatment of the relationship between sequential and non-sequential computations. Non-determinism seems, as is indicated in (NPW), to be added later on, giving rise to variant computations and the like. Lacking is a clear integral representation from which both the non-determinism and the parallelism can be read off so to speak. Another drawback of partial order semantics is that no process-domain is constructed that is commonly agreed upon. A possible candidate is the domain of event-structures as described in (Wi1).

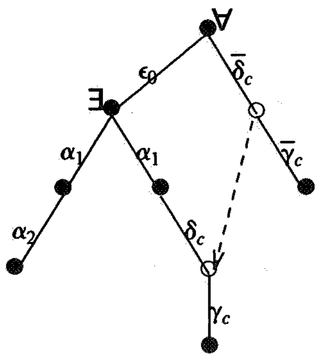
● **diagrams**: (AT)

Inspired by the space-time diagrams of (La) it seems feasible to use an alternating tree as a representation. (Cf. CKS)
Informally an alternating tree is a finitly branching rooted tree consisting of closed and open nodes, such that to each node adheres a quality, universal, existential or atomic corresponding with respectively parallel, alternative and sequential composition. The leaves are to be considered atomic, and open nodes by their virtue of representing an intention to communicate are to be considered existential. See fig 4.1.

I. $(x:=1;(x:=2 + c?x))\|c!3)$



II. $(x:=1;x:=2 + x:=1;x:=2)\|c!3$



III. $(x:=1;c?x;x:=3)\|(y:=2;c!y;y:=4)$

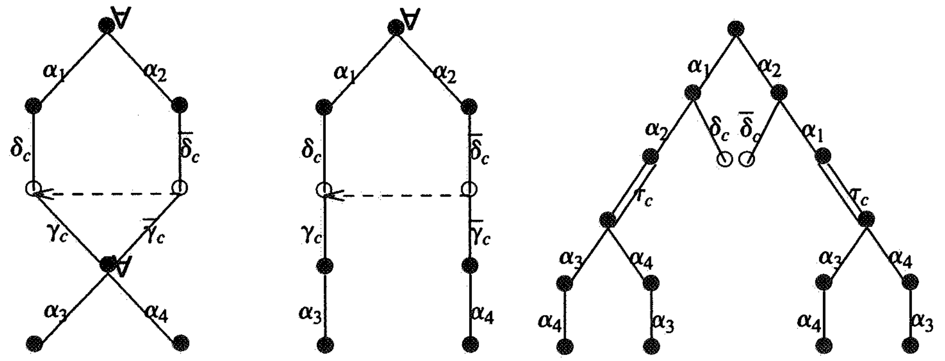
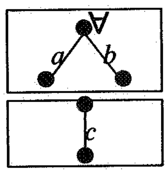


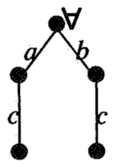
fig 4.1. Example of alternating trees.

The indices for α correspond with the respective assignment-instructions. In III. two versions of an alternating 'tree' are presented that can easily seen to be equivalent. For comparison a synchronisation-tree for the same process is depicted. Note that in I. when choosing between α_2 and δ_c the existence of an open node is immediately visible while this is not the case in II.

As was pointed out by Joost Kok however trees will not do here. For instance consider the program $(a \| b);c$. There is no way to represent the resumption of the execution in c returning from the parallel construct in a tree, since attaching c to b and a is clearly invalid.
The solution to this problem is simply to represent computations as diagrams that can be glued together. For instance the representation of the problem example becomes

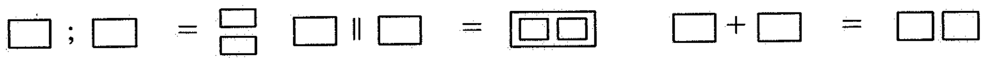


instead of



Such diagrams, as will be explained shortly, can be described as a sequence of alternating trees.

Intuitively the construction-rules for composing diagrams corresponding to the constructors for sequential, parallel and alternative composition are:



where the enclosing box in parallel composition denotes the requirement that both diagrams participate in the computation.

To indicate how to capture such diagrams in a domain-equation recall that the domain-equation for giving the branching-time (BT) semantics was of the form $\mathbb{P}=\{p_0\}\cup\mathcal{P}_c(A\times\mathbb{P})$ where $\mathcal{P}_c(\cdot)$ stood for the power-set of *action-resumption* pairs closed with respect to the metric on $\mathbb{P}$.
In a similar vein the equation $\mathbb{P}=\{p_0\}\cup\mathcal{P}_c(\mathcal{P}_c(A\times\mathbb{P}))\times\mathbb{P}$ should do the job for diagrams.
To explain the equation some auxiliary notation is needed.
Let

$$<a,p> \;\in Conf\subseteq A\times\mathbb{P}$$
$$X \in \exists\text{-}Conf\subseteq\mathcal{P}_c(Conf)$$
$$[X_1,\ldots,X_n] \in\forall\text{-}Conf\subseteq\mathcal{P}_c(\exists\text{-}Conf)$$

then the following cases can arise (where $\|$ denotes parallel composition and $+$ alternative composition):

$+ \backslash \|$	0	1
0	I	II
1	III	IV

I: $[\{<a,p>\}]$	sequential
II: $[\{<a,p>\},\{<b,q>\},\ldots]$	parallel
III: $[\{<a,p>,<b,q>,\ldots\}]$	alternative
IV: $[\{<a,p>,\ldots\},\{<b,q>,\ldots\},\ldots]$	par. & alt.

So case I is a singleton-singleton set which plays an analogous role to the singleton set in the process representation of BT. Now to form a sequential composition with another process the only cases that present any difficulties are case II and IV. In the other cases the process can just be appended to the resumption of the configurations in the set. This motivates the (second) component $\mathbb{P}$ in the $\forall$-$Conf\times\mathbb{P}$ product, as the resumption after parallel branching.
Now assuming that a suitable metric can be found so that the requirement of closure can be fulfilled

the only problem left is to deal with the communications between the distinct parallel branches.
This is done following a suggestion by (GM) by defining an operator that allows for each branch a check on whether it has any communication intentions matching the intentions of other branches. After that a monadic operator can be introduced that establishes the communication potential afterwards by a pairwise treatment of the branches.

Note that the requirement that the set representing the parallel branching is a multi-set is (will be) trivially fulfilled due to the process-numbering that was introduced in the previous section. In this respect the to be proposed semantics is not completely orthogonal in its treatment of parallelism and non-determinism. That is, only parallel branches are assigned *fresh* process-numbers, alternative branches inherit the process-number from their parent.
An alternative to the nesting of choice-branches within parallel branches is its reversal whereby a process consists of a set of possibly parallel branched alternatives.
This was considered in an earlier stage, and rejected as it forces to evaluate parallel composition as a distribution over each of the alternatives when encountered. It was believed that this would imply a commitment to 'local' communications, that is with partners belonging to the other component. This however doesn't seem necessitated after all. Nevertheless, the possibility to postpone the distribution and realisation of communication-potential seems lost.
It should be remarked that the structure proposed is only tentative in the sense that its application to semantics, and its possible alternatives are not fully understood, yet.
It was suggested by Peter van Emde Boas to investigate the algebraic properties of (finite) and/or-structures before encapsulating it in a metric denotational framework as will be sketched shortly. In order to relate the AT-semantics to the BT-semantics given previously however the advice is ignored, leaving the development of an algebra of and/or-structures as an interesting task for the future.

4.2. AT-semantics

DEFINITION: (*domain for AT-semantics*)
Let A_{roc} be the set of action-labels, M the set of process-numbers, $\mathbb{N}$ the natural numbers standing for clock-values, $\underline{M} \subseteq M \cup M \times M$ and $\underline{\mathbb{N}} \subseteq \mathbb{N} \cup \mathbb{N} \times \mathbb{N}$ all as in the previous section.
The metric space of processes $(\mathbb{P}, d)$ satisfying the equation

$$\mathbb{P} = \{p_o\} \cup M \times \mathbb{N} \times \Sigma \to \mathcal{P}_c(\mathcal{P}_c(A_{roc} \times \underline{M} \times \underline{\mathbb{N}} \times \Sigma \times (\mathbb{P} \cup V \to \mathbb{P} \cup V \times \mathbb{P}))) \times \mathbb{P}$$

is constructed by:

$$(0)\quad \mathbb{P}_0 = \{p_0\}\ ,\ d_0(p', p'') = 0 \text{ for } p', p'' \in \mathbb{P}_0$$

$$(n+1)\quad \mathbb{P}_{n+1} = \{p_o\} \cup M \times \mathbb{N} \times \Sigma \to \mathcal{P}_c(\mathcal{P}_c(A_{roc} \times \underline{M} \times \underline{\mathbb{N}} \times \Sigma \times (\mathbb{P}_n \cup V \to \mathbb{P}_n \cup V \times \mathbb{P}_n))) \times \mathbb{P}_n$$

with

$$d_{n+1}(p', p'') = \begin{cases} 0 \text{ if } p' = p'' = p_0 \\ 1 \text{ if } p' = p_0 \ \&\ p'' \neq p_0 \quad \text{or} \quad p'' = p_0 \ \&\ p' \neq p_0 \\ \sup_m \sup_t \sup_\sigma \quad d_{n+1}(p'(m,t,\sigma), p''(m,t,\sigma)) \end{cases}$$

where the distance between processes of the form (P,p) and (Q,q) with P and $Q \in V$-$Conf = \mathcal{P}_c(\mathcal{P}_c(A_{roc} \times \underline{M} \times \underline{\mathbb{N}} \times \Sigma \times (\mathbb{P}_n \cup V \to \mathbb{P}_n \cup V \times \mathbb{P}_n)))$ is the distance given by

$$d_{n+1}((P,p),(Q,q)) = \tfrac{1}{2} \cdot \max\{d_{n+1}(P,Q), d_n(p,q)\}$$

and the distance between P and Q is (informally characterised) the double Hausdorff distance induced by

$$d_{n+1}(<\xi,m,t,\sigma,\rho>,<\xi',m',t',\sigma',\rho'>) = \begin{cases} 1 \text{ if } (\xi,m,t,\sigma)\neq(\xi',m',t',\sigma') \\ \tfrac{1}{2}\cdot d_n(\rho,\rho') \text{ otherwise} \end{cases}$$

with the distance between ρ and ρ' treated identically as in the previous section.
This amounts to taking as the distance between P and Q

$$d_{n+1}([X_1,...,X_l],[Y_1,...,Y_m]) = \max\{\sup_i\inf_j d_{n+1}(X_i,Y_j),\sup_j\inf_i d_{n+1}(X_i,Y_j)\}\ , \text{ and}$$

$$d_{n+1}(\{x_1,...,x_l\},\{y_1,...,y_m\}) = \max\{\sup_i\inf_j d_{n+1}(x_i,y_j),\sup_j\inf_i d_{n+1}(x_i,y_j)\}\ .$$

Now define $(\mathbb{P}_\omega,d_\omega)=(\cup\mathbb{P}_n,\cup d_n)$ and take $(\mathbb{P},d)$ to be the completion of $(\mathbb{P}_\omega,d_\omega)$.

DEFINITION: (*operators on processes*)

(1) sequential composition; $p_0;p=p;p_0$. For $p,q\neq p_0$

$$p;q = \lambda mt\sigma.(p(m,t,\sigma);\lambda m't'\sigma'.q(m,t',\sigma')>$$
$$([X],p_0);q = ([X;q],p_0)$$
$$([X_1,...,X_n],p);q = ([X_1,...,X_n],p;q) \text{ if } p\neq p_0$$
$$X;q = \{\ x;q : x\in X\ \}$$
$$<\xi,m,t,\sigma,p_0>;q = <\xi,m,t,\sigma,\lambda m't'\sigma'.q(m',t,\sigma')>$$
$$<\xi,m,t,\sigma,\rho>;q = <\xi,m,t,\rho;q>$$
$$\theta;q = \lambda v.(\theta(v);q) \qquad (v,p);q=(v,p;q)$$

(2) alternative composition: $p_0+p=p+p_0=p$. For $p,q\neq p_0$

$$p+q = \lambda mt\sigma.(p(m,t,\sigma) \cup q(m,t,\sigma))$$
$$([X],p_0)+([Y],p_0) = ([X\cup Y],p_0)$$

(3) parallel composition: $p_0 \parallel\!| p=p \parallel\!| p_0=p$. For $p,q\neq p_0$

$$p \parallel\!| q = \lambda mt\sigma.(p(m,t,\sigma) \parallel\!| q(m,t,\sigma))$$
$$(P,p_0) \parallel\!| (Q,p_0) = (P \parallel\!| Q,p_0) \quad \text{with}$$
$$[X_1,...,X_n] \parallel\!| [Y_1,...,Y_m] = [X_1,...,Y_m]$$

(4) conditional composition: $\supset : Bexp\times\mathbb{P}\times P\to\mathbb{P}$ is defined by#

$$b\supset p,q = \lambda mt\sigma.\text{if } \underline{b}_\sigma \text{ then } ([\{<\epsilon,m,t+1,\sigma,p_0>\}],p_0);p \text{ else } ([\{\epsilon,m,t+1,\sigma,p_0>\}],p_0);q \text{ fi}$$

Also $(b_1\supset p_1,...,b_n,p_n,p_0)\supset(b_1'\supset p_1',...,b_m'\supset p_m',p_0)$ is defined to be

$$b_1\supset p_1,...,b_n\supset p_n,b_1'\supset p_1',...,b_m'\supset p_m',p_0$$

(5) restriction: $\setminus C:\mathbb{P}\to\mathbb{P}$ where C is an arbitrary set of channel-names

$$p_0\setminus C = p_0$$
$$(\lambda mt\sigma.X)\setminus C = \lambda mt\sigma.X\setminus C$$
$$(P,p)\setminus C = (P\setminus C,p\setminus C)$$

$$[X_1,...,X_n]\setminus C = [X_1\setminus C,...,X_n\setminus C]$$
$$X\setminus C = \cup\ \{\, x\setminus C : x{=}{<}\xi,\cdots{>}\ \&\ \xi{\neq}\delta_c \text{ or } \bar\delta_c \text{ for any } c{\in}C\}$$
$$<\xi,m,t,\sigma,p>\setminus C = \{<\xi,m,t,\sigma,p\setminus C>\}$$

Remarks:

- The exclusion of more complex cases for alternative composition is allowed since Occam knows only of a guarded choice. A more general choice-construct would severely complicate the domain.
- The operator for parallel composition is formulated with a similar restriction as the operator for alternative composition. However there there is no such thing as guarded parallel composition in Occam. The restriction amounts to the assumption that an empy begin statement is prefixed to a (nested) *PAR*-construct.
  So for instance $PAR(SEQ(PAR(S_1,S_2),S_3),SEQ(PAR(S_1',S_2'),S_3'))$ should be read as $PAR(SEQ(SKIP,PAR(S_1,S_2),S_3),SEQ(SKIP,PAR(S_1',S_2'),S_3'))$.

To illustrate this with an example for the uniform case:

$[\![((a\,||\,b);c)\,||\,d]\!] =$

$$([\{<a,p_0>\},\{<b,p_0>\}],([\{<c,p_0>\}],p_0))\ |||\ ([\{<d,p_0>\}],p_0)$$

which in general cannot be composed to be

$$([\{<a,p_0>\},\{<b,p_0>\},\{<d,p_0>\}],([\{<c,p_0>\}],p_0))$$

since c is not a proper resumption of d, for instance the execution of d might require as much time as the construct $(a\,||\,b);c$ in total.
However (prefixing an atomic statement):

$[\![(e\,;(a\,||\,b);c)\,||\,d]\!] =$

$$([\{<e,([\{<a,p_0>\},\{<b,p_0>\}],([\{<c,p_0>\}],p_0))>\}],p_0)\,|||\,([\{<d,p_0>\}],p_0)$$

gives, applying the restricted definition for parallel composition

$$([\{<e,(\cdots)>\},\{<d,p_0>\}],p_0)$$

which is what was wanted.
In the sequel it will be assumed that a *PAR*-construct is preceded by a *SKIP*-statement. The semantics of the *SKIP* is therefore changed. The previous *SKIP* should now be written as *WAIT* 1.

DEFINITION: (*denotational semantics*)

$$[\![x{:=}e]\!](\gamma) = \lambda mt\sigma.([\{<\alpha,m,t+1,\sigma[x{:=}\underline{e}_\sigma],p_0>\}],p_0)$$
$$[\![c?x]\!](\gamma) = \lambda mt\sigma.([\{<\delta_c,m,t,\sigma,\lambda v.\lambda m't'\sigma'.\{<\gamma_c,m,t'+1,\sigma'[x{:=}v],p_0>\}>\}],p_0)$$
$$[\![c!e]\!](\gamma) = \lambda mt\sigma.([\{<\bar\delta_c,m,t,\sigma,(v,\lambda m't'\sigma'.\{<\gamma_c,m,t'+1,\sigma',p_0>\}>\}],p_0)$$
$$[\![WAIT\ e]\!](\gamma) = \lambda mt\sigma.([\{<\epsilon,m,t+\underline{e}_\sigma,p_0>\}],p_0)$$
$$[\![SKIP]\!](\gamma) = \lambda mt\sigma.([\{<\epsilon,m,t,\sigma,p_0>\}],p_0)$$
$$[\![SEQ(S_1,...,S_n)]\!](\gamma) = [\![S_1]\!](\gamma);\ ...\ ;[\![S_n]\!](\gamma)$$
$$[\![PAR(S_1,...,S_n)]\!](\gamma) = \lambda mt\sigma.[\![S_1]\!](\gamma)(m\hat{}1,t,\sigma)\,|||\ \ ...\ \ |||\ \lambda mt\sigma.[\![S_n]\!](\gamma)(m\hat{}n,t,\sigma)$$
$$[\![b\ S]\!](\gamma) = b\supset[\![S]\!](\gamma),\varnothing$$

$[\![ALT(g_1\ S_1,\ldots,g_n\ S_n)]\!] = \Delta([\![g_1]\!](\gamma));[\![S_1]\!](\gamma) + \ \ldots\ + \Delta([\![g_n]\!](\gamma));[\![S_n]\!](\gamma)$

$[\![IF(b_1\ S_1,\ldots,b_n\ S_n)]\!](\gamma) = b_1 \supset [\![S_1]\!](\gamma),\ldots,b_n \supset [\![S_n]\!](\gamma),p_0$

$[\![WHILE\ b\ S]\!](\gamma) = \lim_i\ \ , p_0 = p_0\ \ p_{i+1} = b \supset [\![S]\!](\gamma);p_i,p_0$

$[\![OP\ i=[e_1\ FOR\ e_2](X)]\!] =$

$$\lambda mt\sigma.[\![OP(X)(\gamma)(m,t,\sigma[i:=\underline{e}_{1_\sigma}])\ op\ \ldots\ op\ \lambda mt\sigma.[\![OP(X)(\gamma)(m,t,\sigma[i:=\underline{e}_{1_\sigma}+\underline{e}_{2_\sigma}-1])$$

for $OP \in \{SEQ,PAR,ALT,IF\}$ and corresponding $op \in \{;,\|,+,\supset\}$

$[\![p(c_1,\ldots,c_n)]\!](\gamma) = \gamma(p)(c_1,\ldots,c_n)$

Remarks:
- Prioritised and placed statements are omitted.
- Note that the defined operators do not provide for any effective communication to take place. Communication intentions remain as they are.

Now before we are able to collect the communications some auxiliary operators are needed.
Let for convenience $\perp = FAIL$ and $\Sigma_\perp = \Sigma \cup \{\perp\}$
The order on $\Sigma_\perp$ is defined by putting $\sigma_1 \sqsubseteq \sigma_2$ iff $\sigma_1 = \perp$ or $\sigma_1 = \sigma_2$.
Let as in the previous section $\mathfrak{T} = \mathfrak{P}(\Sigma_\perp)$ and for $T_1,\ T_2 \in \mathfrak{T}$ define
$T_1 \sqsubseteq_{\mathfrak{T}} T_2$ iff $\perp \in T_1\ \&\ T_1 \setminus \{\perp\} \subseteq T_2$ or $T_1 = T_2$ (the Egli-Milner order).
Then $(\mathfrak{T},\sqsubseteq_{\mathfrak{T}},\{\perp\})$ is a CPO.
Define composition of elements in $\Sigma_\perp$ as the usual function-composition satisfying $\sigma\circ\perp = \perp\circ\sigma = \perp$ and $\sigma_1\circ\sigma_2 \neq \perp \Rightarrow \sigma_1 \neq \perp\ \&\ \sigma_2 \neq \perp$. (Cf. B)

DEFINITION: (*yield*)
Let by convention $p_0(m,t,\sigma) =_{df} p_0$ for any m,t,σ then the yield-operator $+ : \mathbb{P} \rightarrow (\Sigma \rightarrow \mathfrak{T})$ is defined by

$$p_0^{\,+} = \lambda\sigma.\{\sigma\}$$
$$p^{\,+} = \lambda\sigma.p(0,0,\sigma)^+$$
$$(P,p)^+ = \bigcup_{\sigma\in P^+} p(0,0,\sigma)^+$$
$$[X_1,\ldots,X_n]^+ = \{\sigma_{\theta 1}\circ \ldots \circ\sigma_{\theta n} : \sigma_i \in X_i^{\,+}\ ,\ \theta \text{ a permutation over } \{1,\ldots,n\}\ \}$$

where X^+ is given by $\bigsqcup_{\mathfrak{T},n} X^{(n)}$ and

$$X^{(0)} = \{\perp\} \qquad X^{(n+1)} = \bigcup\{\ x^{(n+1)} : x \in X\ \}$$
$$<\xi,m,t,\sigma,p_0>^{(n+1)} = \{\sigma\}$$
$$<\xi,m,t,\sigma,p>^{(n+1)} = p(m,t,\sigma)^{(n)}$$
$$<\delta_c,\ldots>^{(n+1)} = <\bar{\delta}_c,\ldots>^{(n+1)} = \{FAIL\}$$

LEMMA:
Define $\| : \mathfrak{T}^2 \rightarrow \mathfrak{T}$ by $T_1 \| T_2 = \{\sigma_1\circ\sigma_2, \sigma_2\circ\sigma_1 : \sigma_1 \in T_1, \sigma_2 \in T_2\}$. Then

(a) $\bigsqcup_n T_1^{(n)} \| \bigsqcup_n T_2^{(n)} = \bigsqcup_n (T_1^{(n)} \| T_2^{(n)})$

(b) $[X_1,\ldots,X_n]_+ = X_1^{\,+} \| \cdots \| X_n^{\,+}$

Proof: Omitted.

Note that such a shuffle operator on $\Sigma_\perp$ is allowed only if variable disjointness of the components can be assumed.

Next, similarly as in the previous section a timing operator $time{:}\mathbb{P}\rightarrow M\times\mathbb{N}\times\Sigma\rightarrow\mathbb{N}$ is introduced.

DEFINITION: (*time*)

$$time(p) = \lambda mt\sigma.time(p(m,t,\sigma))$$
$$time(P,p_0) = time(P)$$
$$time(P,p) = \sup_\sigma\ time(p(0,time(P),\sigma)) \quad .\ \sigma\in P^+$$
$$time([X_1,\ldots,X_n]) = \max\ \{\ time(X_i) : 1\leqslant i\leqslant n\ \}$$
$$time(X) = \sup_n\ time(X^{(n)})\ \text{ with}$$
$$time(X^{(n)}) = \max\ \{\ time(x^{(n)}) : x\in X\ \}\ \text{ if } X\neq\varnothing \text{ and } \infty \text{ if } X\neq\varnothing$$
$$time(<\delta_c,\ldots>^{(n)}) = time(<\bar{\delta}_c,\ldots>^{(n)}) = \infty$$
$$time(<\xi,m,t,\sigma,p>^{(0)}) = t$$
$$time(<\xi,m,t,\sigma,p_0>^{(n+1)}) = t$$
$$time(<\xi,m,t,\sigma,p>^{(n+1)}) = time(p(m,t,\sigma)^{(n)})$$

This operator is simpler than the one defined in the previous section because of the absence of interleaving.

DEFINITION: (*extraction of waiting communication-intentions, at time t*)

$$waiting_t(p_0) = \varnothing$$
$$waiting_t(p) = \lambda m't'\sigma'.waiting_t(p(m',t',\sigma'))$$
$$waiting_t(P,p) = waiting_t(P) \cup \bigcup_{\sigma\in(P\setminus C)^+} waiting_t(p(0,time(P\setminus C),\sigma))\ ,\ \perp\notin(P\setminus C)^+$$
$$waiting_t([X_1,\ldots,X_n]) = \bigcup_i waiting_t(X_i)$$
$$waiting_t(X) = \bigcup\{waiting_t(x) : x\notin X\}$$
$$waiting_{t'}(<\xi,m,t,\sigma,p>) = waiting_{t'}(p(m,t,\sigma)) \text{ if } t\leqslant t' \text{ and } \xi\neq\delta_c,\bar{\delta}_c,\ \varnothing \text{ if } t>t'$$
$$waiting_{t'}(<\xi,m,t,\sigma,p>) = \{<\delta,m,t,\sigma,p>\} \text{ if } t\leqslant t'\ ,\varnothing \text{ otherwise}$$

This operator allows to extract communication-intentions from a given process. Note that if the first component in (P,p) is diverging no intensions are added. All intentions collected are fully specific with respect to their process-number and time-value.

DEFINITION: (*insertion of matching communication-intentions*)

Let C be a time-ordered collection of communication-intentions, that is for $C \in COMM$ define

$$\lhd_t : \mathbb{P} \times COMM \to \mathbb{P} \times COMM \text{ , with } p \lhd_t \varnothing = p \text{ by}$$

$$\begin{aligned}
p \lhd_t C &= \lambda m't'\sigma'.((p(m',t',\sigma') \lhd_t C)_0, (p(m',t',\sigma') \lhd_t C)_1) \\
(P,p) \lhd_t C &= ((P \lhd_t C)_0, (p \lhd_t (P \lhd_t C)_1)_0), (p \lhd_t (P \lhd_t C)_1)_1) \\
[X_1,\ldots,X_n] \lhd_t C &= ([(X_1 \lhd_t C)_0,\ldots,(X_n \lhd_t C)_0], \textstyle\bigcap_i (X_i \lhd_t C)_1) \\
X \lhd_t C &= (\textstyle\bigcup \{(x \lhd_t C)_0 : x \in X\}, \bigcap \{(x \lhd_t C)_1 : x \in X\})
\end{aligned}$$

$$<\xi,m,t,\sigma,\rho> \lhd_{t'} C = (\{ <\xi,m,t,\sigma,\rho> \}, C) \text{ if } t'<t$$

$$<\xi,m,t,\sigma,p> \lhd_{t'} C = (<\xi,m,t,\sigma,(p \lhd_{t'} C)_0>, (p \lhd_{t'} C)_1) \text{ if } \xi \neq \delta_c, \overline{\delta}_c$$

$$\begin{aligned}
<\delta_c,m,t,\sigma,\theta> \lhd_{\underline{t}} C = (\{ &<\tau_c{:}v,(m,m'),(t,t'),\sigma,(\lambda m''t''\sigma''.\theta(v')(m,\max(t,t'),\sigma'') \lhd_{\underline{t}} C')_0> : \\
&<\overline{\delta}_c,m',t',\sigma',(v',p')> \in C \ \& \ C' = C \setminus \{ <\overline{\delta}_c,m',t',(v',p')> \} \ \& \\
&\neg\exists t''<t'. <\overline{\delta}_c,m'',t'',\sigma'',(v'',p'')> \in C \}, \textstyle\bigcap \{ (\cdots \lhd_{\underline{t}} C')_1 : \cdots \})
\end{aligned}$$

$$\begin{aligned}
<\overline{\delta}_c,m,t,\sigma,(v,p)> \lhd_{\underline{t}} C = (\{ &<\overline{\tau}_c{:}v,(m',m),(t',t),\sigma,(\lambda m''t''\sigma''.p(m,\max(t,t'),\sigma'') \lhd_{\underline{t}} C')_0> : \\
&<\delta,m',t',\sigma',\theta'> \in C \ \& \ C' = C \setminus \{ <\delta,m',t',\theta'> \} \ \& \\
&\neg\exists t''<t'. <\delta,m'',t'',\sigma'',\theta''> \in C \}, \textstyle\bigcap \{ (\cdots \lhd_{\underline{t}} C')_1 : \cdots \})
\end{aligned}$$

Remarks:
- Each synchronisation consists now of two parts, a $<\tau_c,\ldots>$ for the input-component and a $<\overline{\tau}_c,\ldots>$ for the output-component.
- The effect of the operator is that actual communications are inserted into the process whenever two matching intentions meet.

Having defined an operator for extracting waiting communication-intentions from branches and also an operator to insert communications into a branch, all that is left is to define an operator that real-isesthe potential for communication of a process step by step. Note that the insertion-operator just looks for matching intentions and effects a communication if such an intention is found, leaving the possibility of other communications to take place open. Obviously it would not work to apply a realisation-operator in 'one stroke', since communication-intentions following on another intention are shielded from the the extraction-operator until an effective communication has taken place.

Rather than defining the approximation of realising the potential for communication by induction on the degree of a process the communication-realisation operator is defined by induction on the time a communication can take place.
Remember that the time-domain is discrete and totally ordered.

DEFINITION: (*realisation of communication potential*)

$$\Diamond_t(p_0) = p_0$$

$$\Diamond_t(p) = \lambda m't'\sigma'.\Diamond_t(p(m',t',s'))$$

$$\Diamond_t(P,p) = (\Diamond_t(P),\lambda m't'\sigma'.(\Diamond_t(p(m',time(P \setminus C),\sigma'))$$

$$\Diamond_t[X_1,\dots,X_n] = [X_1{}^{(t)},\dots,X_n{}^{(t)}]\text{with}$$

$$X_i^{(t)} = ((\Diamond_t X_i) \lhd_t \bigcup_{j\neq i} waiting_t(X_j))_0 \quad , \; 1\leqslant i\leqslant n$$

and

$$\Diamond_t(X) = \bigcup \{ \Diamond_t(x) : x\in X \}$$

$$\Diamond_{t'}(<\xi,m,t,\sigma,p>) = \{ <\xi,m,t,\sigma,p> \} \text{ if } t>t'$$

$$= <\xi,m,t,\sigma,\Diamond_t(p)> \text{ if } t\leqslant t' \text{ and } \xi\neq\delta_c,\bar{\delta}_c$$

$$\Diamond_{t'}(<\delta,m,t,\sigma,\rho>) = \{<\delta,m,t,\sigma,\rho>\} \text{ for } \delta=\delta_c,\bar{\delta}_c$$

Now define $\Diamond{:}\mathbb{P}\rightarrow\mathbb{P}$ by $\Diamond = \lim_{t\rightarrow\infty} \Diamond_t \circ \Diamond_{t-1} \circ \cdots \circ \Diamond_0$

EXAMPLE: ($PAR(SEQ(c?x,x:=2,c?x),SEQ(c!1,c!3))$)

Let $p = \lambda mt\sigma.([\{<\delta_c,m,t,\sigma,\theta>\}],p_0)$

$$\theta = \lambda v.\lambda m't'\sigma'.([\{<\gamma_c,m,t'+1,\sigma'[x:=v],p_1>\}],p_0)$$

$$p_1 = \lambda m''t''\sigma''.([\{<\alpha,m,t'+2,\sigma''[x:=2],p_2>\}],p_0)$$

$$p_2 = \lambda m'''t'''\sigma'''.([\{,\delta_c,m,t'+2,\sigma''',\theta'>\}],p_0)$$

$$\theta' = \lambda v'.\lambda m''''t''''\sigma''''.([\{<\gamma_c,m,t''''+1,\sigma''''[x:=v'],p_0>\}],p_0)$$

and

$$q = \lambda mt\sigma.([\{<\bar{\delta}_c,m,t,\sigma,(1,q_1)>\}],p_0)$$

$$q_1 = \lambda m't'\sigma'.([\{<\gamma_c,m,t'+1,\sigma',q_2>\}],p_0)$$

$$q_2 = \lambda m''t''\sigma''.([\{<\bar{\delta}_c,m,t'+1,\sigma'',q_3>\}],p_0)$$

$$q_3 = \lambda m'''t'''\sigma'''.([\{<\bar{\gamma}_c,m,t'''+1,\sigma''',p_0>\}],p_0)$$

then

$[\![\cdots]\!](\gamma)(0,0,\sigma) = ([\{<\delta_c,01,0,\sigma,\theta>\},\{<\bar{\delta}_c,02,0,\sigma,(1,q)>\}],p_0) = (P,p_0)$, say.

Now applying the communication-realisation operator to (P,p_0) gives

$$\Diamond_0(P,p_0) = ([\{<\tau_c{:}1,(01,02),(0,0),\sigma,\lambda m't'\sigma'.([\{<\gamma_c,01,1,\sigma'[x:=1],p_1>\}],p_0)>\},$$
$$\{<\bar{\tau}_c{:}1,(01,02),(0,0),\sigma,\lambda m't'\sigma'.([\{<\bar{\gamma}_c,01,1,\sigma',q_2>\}],p_0)>\}],p_0)$$

$$\Diamond_1 \circ \Diamond_0(P,p_0) = \Diamond_0(P,p_0)$$

$$\Diamond_2 \circ \Diamond_1 \circ \Diamond_0(P,p_0) =$$

$$\begin{array}{l}
([\{<\tau_c{:}1,(01,02),(0,0),\sigma,\lambda m't'\sigma'.([\{<\gamma_c,01,1,\sigma'[x:=1],\\
\quad \lambda m''t''\sigma''.([\{<\alpha,01,2,\sigma''[x:=2],\lambda m'''t'''\sigma'''.([\{<\tau_c{:}3,(01,02),(2,1),\sigma''',\\
\quad \lambda m''''t''''\sigma''''.([\{<\gamma_c,01,3,\sigma''''[x:=3],p_0>\}],p_0)>\}],p_0>\}],p_0>\},\\
\;\{<\overline{\tau}_c{:}1,(01,02),(0,0),\sigma,\lambda m't'\sigma'.([\{<\gamma_6 c,02,1,\sigma',\\
\qquad\qquad\qquad \lambda m''t''\sigma''.([\{<\overline{\tau}_c{:}3,(01,02),(2,1),\sigma'',p_0>\}],p_0)>\}],p_0)>\}],p_0)
\end{array}$$

The removal of communication-intentions participating in a communication appeared to be necessary to avoid 'impossible' communications.

Now supposing communication is handled properly, in order to establish the meaning of a construct and compare the denotational and the operational semantics what remains to be defined is an operator for guard-selection and an operator that produces the possible execution sequences.
It is considered straightforward to define the operators for guard-selection and branch-pruning as in the previous section. The definition of the path-extraction operator however is considered more intricate.

DEFINITION: (*paths*)
Let $E_0 \subseteq A_{roc} \times M \times N \times \Sigma$, $\mathbb{E} = E_0^* \cup E_0^\omega \cup E_0^* \cdot \{FAIL\}$ and $\cdot$ the concatenation operator for sequences in $\mathbb{E}$ and $<>$ the empty sequence.
Let τ_c and $\overline{\tau}_c$ be special elements in E_0 corresponding to $<\tau_c{:}v,(m,m'),(t,t'),\sigma>$ and $<\overline{\tau}_c{:}v,(m,m'),(t,t'),\sigma'>$. τ_c and $\overline{\tau}_c$ are said to *match* if they are equal in v, (m,m') and (t,t').
Let τ_* be a special element corresponding with both τ_c and $\overline{\tau}_c$.
For $w_1\tau_1 w_1'$, $w_2\tau_2 w_2'$ sequences in $\mathbb{E}_\tau$, that is $\mathbb{E}$ extended with special elements for τ_c and $\overline{\tau}_c$, w_1 and w_2 τ-free, define the associative and commutative *synchronised shuffle* operator

$$\|_\tau : \mathbb{E} \times \mathbb{E} \to \mathcal{P}(\mathbb{E}^*) \text{ where } \mathbb{E}^* \text{ is } \mathbb{E} \text{ extended with } \tau_*$$

by

$$\begin{array}{l}
w \,\|\, <> \;=\; <> \,\|\, w \;=\; w \text{ and}\\
w_1\tau_1 w_1' \;\|_\tau\; w_2\tau_2 w_2' \;=\; (w_1 \,\|\, w_2)\cdot <\tau_*{:}v,(m,m'),(t,t'),\sigma>\cdot(w_1' \,\|_\tau w_2')\\
\qquad \text{if } \tau_i \;=\; <\tau_c{:}v,(m,m'),(t,t'),\sigma>\\
\qquad \text{and } \tau_j \;=\; <\overline{\tau}_c{:}v,(m,m'),(t,t'),\sigma'> \;,(i,j)=(1,2) \text{ or } (i,j)=(2,1)\\
\qquad\qquad\qquad = \varnothing \text{ otherwise}
\end{array}$$

where $\|$ is the ordinary shuffle on strings.
Extend $\|_\tau$ to $\|_\tau : \mathcal{P}(\mathbb{E}) \times \mathcal{P}(\mathbb{E}) \to \mathcal{P}(\mathbb{E})$ by: $X \|_\tau Y = \bigcup \{ x \|_\tau y \;, y \|_\tau x : x \in X,\; y \in Y \}$
Next define the *chaining-operator* $\hat{}\,{:}\mathbb{E}^* \times (M \times \mathbb{N} \times \Sigma \to \mathbb{E}^*) \to \mathcal{P}(\mathbb{E}^*)$ by

$$w \cdot <\xi,m,t,\sigma>\hat{}\;\lambda mt\sigma.w' \;=\; w \cdot w'$$

and similarly extend $\hat{}$ such that $X\hat{}\,Y = \{ x\hat{}\,y : x \in X,\; y \in Y \}$.
Then $paths{:}\mathbb{P} \to M \times \mathbb{N} \times \Sigma \to \mathbb{E}^*$ is given by:

$$\begin{array}{l}
paths(\lambda mt\sigma.(P,p)) \;=\; \lambda mt\sigma.paths(P)\hat{}\,paths(p)\\
paths([X_1,\ldots,X_n]) \;=\; paths(X_1) \,\|_\tau \cdots \|_\tau paths(X_n)\\
paths(X) \;=\; \cup \{ paths(x) : x \in X \} \;,\; X \neq \varnothing\\
\qquad\qquad = \{FAIL\} \;,\; X = \varnothing
\end{array}$$

$$paths(<\xi,m,t,\sigma,p_0>) = \{<\xi,m,t,\sigma>\}$$

$$paths(<\xi,m,t,\sigma,p>) = <\xi,m,t,\sigma>\cdot paths(p(m,t,\sigma)) \quad p \neq p_0$$

$$paths(<\delta_c, \cdots >) = paths(<\overline{\delta}_c, \cdots >) = \{FAIL\}$$

Remarks:
· Chaining of paths to parametrised paths is necessary to effect a proper concatenation.
· Note that the definition for $paths([X_1,...,X_n])$ would not work if $\|_\tau$ were not commutative.

EXAMPLE:

$paths(\Diamond(\llbracket PAR(SEQ(c?x,x:=2,c?x),SEQ(c!1,c!3))\rrbracket(\gamma)))(0,0,\sigma)$

$= \{<\tau_c:1,(01,02),(0,0),\sigma>\cdot<\gamma_c,01,\sigma[x:=1]>\cdot<\alpha,01,2,\sigma[x:=2]>\cdot$

$\quad <\tau_c:3,(01,02),(2,1),\sigma[x:=2]>\cdot<\gamma_c,01,3,\sigma[x:=3]>\} \ \|_\tau$

$\quad \{<\overline{\tau}_c:1,(01,02),(0,0),\sigma>\cdot<\overline{\gamma}_c,02,1,\sigma>\cdot<\overline{\tau}_c:3,(01,02),(2,1),\sigma>\cdot<\overline{\gamma}_c,02,3,\sigma>\}$

$= \ <\tau_*:1,(01,02),(0,0),\sigma>\cdot(\{<\gamma_c,01,1,\sigma[x:=1]>\cdot<\alpha,01,2,\sigma[x:=2]>\} \ \| \ \{<\overline{\gamma}_c,02,1,\sigma>\})\cdot$

$\quad <\tau_*:3,(01,02),(2,1),\sigma[x:=3]>\cdot(\{<\gamma_c,01,3,\sigma[x:=3]>\} \| \{<\overline{\gamma}_c,02,3,\sigma>\})$

where τ_* is the special synchronisation symbol for channel c

CONJECTURE: $paths(\llbracket S\rrbracket_{BT}(\gamma) \setminus C) = paths(\Diamond(\llbracket S\rrbracket_{AT}(\gamma)))$

Summarizing, it is conjectured that the *BT*-semantics is equivalent to the *AT*-semantics. Unfortunately, although intuitively appealing, mathematically *AT*-semantics appear hard to define.

RELATIONSHIP $AT - BT$

A more direct relationship between *AT* and *BT* semantics can indicated by defining an interleaving operator that merges the in *AT* juxtaposed component-processes of a parallel composition to their arbitrary interleavings, either before or after applying the communication-realisation operator.

DEFINITION: (*interleaving operator, for finitary processes*)
Let $\|$ be a parallel composition operator to be defined below, then

$$interleave(p) = \lambda mt\sigma.interleave(p(m,t,\sigma))$$

$$interleave(P,p) = interleave(P);interleave(p)$$

$$interleave([X_1,...,X_n]) = [interleave(X_1)\| \cdots \| interleave(X_n)]$$

$$interleave(X) = \bigcup \{ \ interleave(x) : x \in X \ \}$$

$$interleave(<\xi,m,t,\sigma,\rho>) = \{<\xi,m,t,\sigma,interleave(\rho)>\}$$

$$interleave(\theta) = \lambda v.interleave(\theta(v))$$

$$interleave(v,p) = (v,interleave(p))$$

and

· $X \| Y = X \mathbin{\|\!\llcorner} Y \cup Y \mathbin{\|\!\llcorner} X \cup X | Y$ with

- $X \mathbin{|\!\lfloor} Y = \{x \mathbin{|\!\lfloor} Y : x \in X\}$

  $<\xi, m, t, \sigma, \rho> \mathbin{|\!\lfloor} Y = <\xi, m, t, \sigma, \rho \parallel Y>$ where

  $p \parallel Y = \lambda m t \sigma . (p(m,t,\sigma) \parallel Y)$ $\theta \parallel Y = \lambda v . (\theta(v) \parallel Y)$ $(v,p) \parallel Y = (v, p \parallel Y)$ and

  $([X], p_0) \parallel Y = ([X \parallel Y], p_0)$

  which is the only form encountered due to the previously applied interleaving operator, and

- $X \mid Y = \bigcup \{ x \mid y \, , y \mid x : x \in X, y \in Y\}$ with

  $<\delta_c, m, t, \sigma, \theta> \mid <\bar{\delta}_c, m', t', \sigma', (v,p)> =$

  $\{<\tau_c : v, (m,m'), (t,t'), \sigma, \lambda m'' t'' \sigma'' . (\theta(v)(m, \max(t,t'), \sigma'') \parallel p(m', \max(t,t'), \sigma'')>\}$

Now defining the mapping $\lambda m t \sigma . ([X], p_0) \mapsto \lambda m t \sigma . X$ it should be easy to establish the isomorphism $([\![S]\!]_{BT}(\gamma) \setminus C) \approx \mathit{interleave}([\![S]\!]_{AT}(\gamma)) \setminus C$.
However a more interesting conjecture can be stated involving the communication-realisation operator:

$$\mathit{interleave}([\![S]\!]_{AT}(\gamma)) \setminus C \approx \mathit{interleave}(\Diamond [\![S]\!]_{AT}(\gamma))$$

further treatment of which is postponed until the properties of the $\Diamond$operator are better understood.

5. Relationship between the semantics: (LT,BT)

In the previous parts the equivalences between the semantics has only been conjectured. The aim of this part is to provide a more thourough proof of the equivalence of the operational and denotational semantics, thereby at the same time providimg more insight into the respective semantics. The focus will be mainly on the LT-operational semantics and the denotational BT-semantics. The relationship of these to AT-semantics will be elucidated in the next chapter where also a comparison with net-theoretic and partial order semantics is made. Because of the difference in domains a mediating abstraction-operator has to be introduced to relate the two semantics.
The proposition to be proved, taking $[\![\cdot]\!]_{LT}$ for $\mathcal{O}_1(\cdot)$ is

Proposition: $[\![S]\!]_{LT}=\beta([\![S]\!]_{BT}(\gamma))$ for arbitrary S and γ, S closed.

Take $\beta=\lambda p\in\mathbb{P}.paths\circ prune(p\setminus C)$.
First the communication intentions are restricted away, then the branches are pruned of which the guards do not become ready in time, after which from the restricted and pruned process the set of possible execution sequences is formed.

To prove the equivalence the operational semantics is changed with respect to guarded commands and both operational and denotational semantics are changed with respect to communication.

(i)guarded commands

Replace the rules T3 and E3 by resp. T3' and E3'

(T3') $<(ALT((g_1\ S_1,m),\ldots,(g_n\ S_n,m)),t),\sigma> \xrightarrow{(\circ,m,t)} <ALT((g_1\ S_1,m,t),\ldots,(g_n\ S_n,m,t)),\sigma>$

(E3') $<ALT(\ldots,(E\ S,m,t),\ldots),\sigma> \xrightarrow{(\bullet,m,t)} <(S,m,t),\sigma>$

(ii)communication

Redefine the denotational semantics for input and output to

$[\![c?x]\!](\gamma) = \lambda mt\sigma.\{ <\delta_c,m,t,\sigma[x:=v],(c?xv,p_0)> : v\in V \}$

$[\![c!e]\!](\gamma) = \lambda mt\sigma.\{ <\bar{\delta}_c,m,t,\sigma,(c!\underline{e}_\sigma,p_0)> \}$

and change the communication operator to

$<\delta_c,m,t,\sigma,(c?xv,p)> | <\bar{\delta}_c,m',t',\sigma',(c!v,q)> = \{ <\tau_c{:}v,(m,m'),\max(t,t')+1,\sigma\cup\sigma',p_0>;(p\,\|\,q) \}$

with of course adapting the domain and the operators to this modification.

For the operational semantics change the axioms in A2 into the collection of axioms A2':

(A2') $<(c?x,m,t),\sigma> \xrightarrow{(\delta_c,m,t)} <(E,m,t),\sigma[x:=v]:\{c?xv\}> \quad v\in V$

$<(c!e,m,t),\sigma> \xrightarrow{(\bar{\delta}_c,m,t)} <(E,m,t),\sigma:\{c!\underline{e}_\sigma\}>$

This change in the communication axioms imply that after encountering a deadlocking configuration the computation can proceed in an arbitrary direction, provided that the extra component after σ is treated as invisible for any ordinary uses of σ.
The function of these transitions can informally be described as creating a hole in a computation that can later be filled. In the denotational semantics a record of the actual instruction is kept containing as much as an hypothesis concerning the value transmitted over the channel.

With the semantics adapted so that communications and the auxiliary guard-enclosure match it seems that due to the extra time, process-number and action information and the creation of holes in previously deadlocking computations the equivalence proof can be done directly by induction on the structure of the statements. No auxiliary domain as in (BKMOZ) is needed.

Say for S given $w \in LT(S,\sigma)$ if $w \in [\![S]\!]_{LT}(\sigma)$ (the extended operational semantics) and $w \in BT(S,\sigma)$ if $w \in \beta'([\![S]\!]_{BT}(\gamma)(0,0,\sigma))$ for some abstraction operator β'. Then it is to be proven that for all S and σ
 (i) $LT(S,\sigma) \subseteq BT(S,\sigma)$
 (ii) $BT(S,\sigma) \subseteq LT(S,\sigma)$
The proofs proceed by induction on the structure of S. Note that for all S and σ both sequences in LT and BT start with $<\epsilon_0,0,0,\sigma>$. For atomic instructions, except communication instructions the mutual inclusion is immediate. The difficult parts in both proofs are those that involve sequential and parallel composition. Alternative composition and conditional composition are relatively easy. Repetitive composition is in its more general recursive form extensively treated in (BMOZ2). Replicative composition immediately follows from the first four types of composition.
A crucial role is played by the domain of event-sequences

$$\mathbb{E} = E_0^* \cup E_0^\omega \cup E_0^* \cdot \{FAIL\}$$

Before giving (an outline of) the proof some auxiliary operators are defined on (sets of) sequences $w \in \mathbb{E}$.

DEFINITION: (*process-number substitution*)

$$\text{Let } m[m''/m'] = \begin{cases} m''m_2 & \text{if } m=m_1m_2 \ \& \ m_1=m' \\ m & \text{otherwise} \end{cases}$$

and $(m_1,m_2)[m''/m'] = (m_1[m''/m'],m_2[m''/m'])$.

For $e=<\xi,m,t,\sigma>$ define $e[m''/m'] =_{df} <\xi,m[m''/m'],t,\sigma>$
and let $w[m''/m']$ be the obvious extension of the substitution to sequences.

DEFINITION: (*time-substitution*)

$$\text{Let } <\xi,\underline{m},t,\sigma> + [n]_{m'} = \begin{cases} <\xi,\underline{m},t+n,\sigma> & \text{if } \underline{m} \leqslant m' \\ <\xi,\underline{m},t,\sigma> & \text{otherwise} \end{cases}$$

where $(m_1,m_2) \leqslant m$ if $m_1 \leqslant m$ or $m_2 \leqslant m$.
Similarly let $w+[n]_m$ be the extension to sequences.

DEFINITION: (*process-number, time-abstraction*)
For possibly infinite sequences $w=\epsilon_0 w'$ with $\epsilon_0=<\epsilon_0,0,0,\sigma>$ define $\Phi(w) = \lambda mt.w'[m/0]+[t]_m$.
An abstracted sequence, notation $\underline{w} = \lambda mt.w$ is thus 'parametrised' with respect to process-numbering and timing.
The first empty event is cut off.

DEFINITION: (*yield*)
Define the yield w^+ of a sequence w to be the state of the last event such that for $w=w'\cdot FAIL$ finitly blocking $w^+=FAIL$ and for w infinite $w^+=\perp$. Also let $(\lambda mt.w)^+=\lambda mt.w^+$. More generally the retrieval functions m,t,s can be defined delivering respectively the process-number (of the first event) of the sequence and the time and state of the last event. Whenever the state is $\perp$ or $FAIL$ the time-value will be ∞.

All operators can be defined for sets in a distributive fashion such that $op(X)=\{op(x){:}x\in X\}$.

Next some binary operators are defined in order to model the composition due to the constructs in Occam.

Let ϵ denote the empty sequence.

(*) **sequential composition:** $w;\epsilon=\epsilon;w=w$
For w infinite or finitly blocking $w;w'=w$, for w finite succesful

$$w;w'=w\cdot\Phi(w')(m(w),t(w))$$

that is w' is concatenated to w with process-numbering and timing properly modified. In applying the operator care should be taken that the state of the last event of w agrees with the state of the first event of w'.

(*) **parallel composition:** $w\|\epsilon=\epsilon\|w=\{w\}$
For w or w' infinite $w\|w'=\lim_n(w^{(n)}\|w'^{(n)})$ where $w^{(n)}$ is the truncation of the sequence w at length n.
For w and w' finite

$\cdot$ $w\|w' = w \mathbin{⫡} w' \cup w' \mathbin{⫡} w \cup w|w'$, with

$FAIL \mathbin{⫡} w = \{FAIL\}$, $e \mathbin{⫡} w = \{e\cdot w\}$ for $e\neq FAIL$ and $ew \mathbin{⫡} w' = e\cdot(w\|w')$

$\cdot$For $e=<\delta_c,m,t,\sigma{:}\{c?xv\}>$ and $e'=<\bar{\delta}_c,m',t',\sigma'{:}\{c!v\}>$

$ew|e'w' = e'w'|ew = <\tau_c{:}v,(m,m'),t'',\sigma\cup\sigma'>\cdot(w+[t''-t]_m\|w'+[t''-t']_{m'})$

where $t''=\max(t,t')+1$.

Note that the parallel composition of two sequences results in a set of sequences.
The operator is extended to include sequences X and Y such that $X\|Y=\bigcup\{\, w\|w' : w\in X,\ w'\in Y\,\}$
One more operator, to allow the treatment of guarded commands, is needed.

DEFINITION: (*guard-enclosure*)
Define the operator Δ on finite sequences $w=e_0e_1\cdots e_n$
with $e_0=<\epsilon_0,0,0,\sigma>$ and $e_n=<\xi,\underline{m},t,\sigma'>$ by

$$\Delta(e_0e_1\cdots e_n) = e_0\cdot<0,0,0,\sigma>\cdot e_1\cdots e_n\cdot<\bullet,0,t,\sigma'>$$

With these definitions the following lemma stating the compositionality of the (modified) operational semantics can be proved.

LEMMA: (*compositionality of operational semantics*)
Let $\sigma_{i+1} = ([\![S_i]\!]_{LT}(\sigma_i))^+$ for $1 \leqslant i < n$ and $\sigma_0 = \sigma$ then

(i) $[\![SEQ(S_1,...,S_n)]\!]_{LT}(\sigma) = [\![S_1]\!]_{LT}(\sigma_0); \ ... \ ;[\![S_n]\!]_{LT}(\sigma_{n-1})$

(ii) $[\![PAR(S_1,...,S_n)]\!]_{LT}(\sigma) = [\![S_1]\!]_{LT}(\sigma)[01/0] \,\|\, ... \,\|\, [\![S_n]\!]_{LT}(\sigma)[0n/0]$

Moreover defining for a set of sequences X

$$prune(X) = \{ \, w \in X : \neg\exists w' \in X. t(w' \upharpoonright \bullet) < t(w \upharpoonright \bullet) \ \& \ w' \text{ hole-free} \, \}$$

where $w \upharpoonright \bullet$ is the truncation of w after the first event with a $\bullet$-action then also with $\sigma_i = ([\![g_i]\!]_{LT}(\sigma))^+$

(iii) $[\![ALT(g_1 \ S_1,...,g_n \ S_n)]\!]_{LT}(\sigma) =$

$\quad prune(\Delta([\![g_1]\!]_{LT}(\sigma));[\![S_1]\!]_{LT}(\sigma_1) \cup \ ... \ \cup \Delta([\![g_n]\!]_{LT}(\sigma));[\![S_n]\!]_{LT}(\sigma_n))$

The proof of this lemma hinges on two auxiliary lemma's, one concerning the stepwise construction of an event-sequence from a computation and another establishing the non-determinism of communication.

LEMMA: (*stepwise construction of an event-sequence*)

$$T \vdash <S,\sigma> \dashv\!(\xi,m,t)\!\rightarrow <S',\sigma'>$$

$$\Leftrightarrow$$

$$T \Vdash <\epsilon_0,0,0,\sigma> \cdot <\xi,m',1,\sigma'>;[\![S']\!]_{LT}(\sigma') \ = \ [\![S]\!]_{LT}(\sigma)$$

for $\xi \in \{\alpha,\epsilon\}$ and some m and m'

LEMMA: (*communication non-determinism*)

$$T \vdash <S,\sigma> \dashv\!(\tau_c{:}v, (m,m'),t)\!\rightarrow <S',\sigma'>$$

$$\Leftrightarrow$$

$$T \vdash <S,\sigma> \dashv\!(\delta_c,m,t')\!\rightarrow <S'',\sigma[x:=v]:\{c?xv\} \ \&$$

$$T \vdash <S,\sigma> \dashv\!(\bar{\delta}_c,m',t'')\!\rightarrow <S''',\sigma:\{c!v\}>$$

for some $(c?x,S''),(c!e,S''') \in <\!\mathit{first,rest}\!>(S)$ with $e_\sigma = v$ and $t = \max(t',t'')+1$. The first lemma states that sequences are built up (mainly) of atomic actions. Unfortunately the lemma does not provide any clue about the relation between the process-numbers. It suffices however to remark that process-numbers are only changed in parallel composition.
The second lemma states that whenever a communication is allowed in a computation the components might also proceed independently provided that suitable holes are created. On the other hand whenever the creation of two complementary holes is possible a communication might occur as well. Basically in merging two sequences what can be considered an omission during the computation, the communication, is being repaired.

Proof of the compositionality-lemma:
For each part assuming a sequence w in either the LHS or the RHS a distinction between three cases has to be made

(a) w is *finite successful*
(b) w is *finitly blocked*
(c) w is infinite

(i) Assume w is in the LHS.

(a) Since w is finite succesfull w is nowhere blocked. Suppose w contains a hole then, since the hole is due to a component in which an input or output occurs some sequence for that single component also must contain a hole. So it suffices to observe that by the definition of sequential composition the process-numbers and times are correctly substituted in the successor-components.

(b) w is finitly blocked. Then there is a blocking alternative construct in some component and the alternative construct will be blocking in that component when taken on its own.

(c) w is infinite due to an infinite subsequence of w resulting from some component, then that component will also result independently in an infinite computation.

The reverse inclusion follows from a similar line of reasoning.

(ii) Follows immediately from the lemma concerning the non-determinism of communication.

The proof of (iii) is omitted.

With the main part of the work for the operational semantics done it remains to show that, roughly, BT-semantics behaves homomorphically over the operators on sequences.

However to effect this, instead of the operator *paths* that projects processes to sets of possibly failing sequences, a different projection operator is to be defined.

DEFINITION: (*projection of processes to sequences*)

The operator $\pi:\mathbb{P}\to M\times\mathbb{N}\times\Sigma\to\mathbb{E}$ is defined by

$\pi(p_0) = \{ \epsilon \}$,with ϵ the empty sequence

$\pi(\lambda mt\sigma.X) = \lambda mt\sigma.\pi(X)$

$\pi(X) = \bigcup \{ \pi(x) : x\in X \}$ if $X\neq\emptyset$ and $\{FAIL\}$ otherwise

Let by convention be $p_0(m,t,s) =_{df} p_0$ then

$\pi(<\xi,\underline{m},t,\sigma,p>) = <\xi,m,t,\sigma>\cdot\pi(p(m,t,\sigma))$ for $\xi\neq\delta_c$ or $\bar{\delta}_c$

$\pi(<\delta_c,m,t,\sigma,(c?xv,p)>) = <\delta_c,m,t,\sigma:\{c?xv\}>\cdot\pi(p(m,t,\sigma))$

$\pi(<\bar{\delta}_c,m,t,\sigma,(c!v,p)>) = <\bar{\delta}_c,m,t,\sigma:\{c!v\}>\cdot\pi(p(m,t,\sigma))$

In order to establish the equivalence also the yield operator has to be adapted.

DEFINITION: (*yield**)
The mapping $+ : \mathbb{P} \setminus \{p_0\} \to (\Sigma \to \mathfrak{I})$ is given by

$$p^+ = \lambda\sigma.p(0,0,\sigma)^+$$
$$X^+ = \bigsqcup_{T,n} X^{(n)}$$

where $\bigsqcup_{\mathfrak{I}}$ is the lub in $(\mathfrak{I},\sqsubseteq_{\mathfrak{I}},\{\bot\})$ and $X^{(n)}$ is defined by

$$X^{(0)} = \{\bot\}$$
$$X^{(n+1)} = \bigcup\{\, x^{(n+1)} : x \in X \,\}$$
$$<\xi,\underline{m},t,\sigma,p_0>^{(n+1)} = \{\sigma\} \quad \text{for } \xi \neq \delta_c \text{ or } \overline{\delta}_c$$
$$<\xi,\underline{m},t,\sigma,p>^{(n+1)} = p(m,t,\sigma)^{(n)}$$
$$<\delta,m,t,\sigma,(c,p_0)> = \{\sigma\} \quad \text{for } \delta = \delta_c \text{ } or \text{ } \overline{\delta}_c \text{ and } c \text{ the communication-intention}$$
$$<\delta,m,t,\sigma,(c,p)> = p(m,t,\sigma)^{(n)}$$

If m is of the form (m',m'') m can be arbitrarily chosen to be m' or m''.
Obviously $yield^*(p) =_{df} p^+$.

LEMMA: (*relation state retrieval functions on sequences and processes*)

$$s(\pi([\![S]\!]_{BT}(\gamma)(0,0,\sigma))) = yield^*([\![S]\!]_{BT}(\gamma)(0,0,\sigma))$$

where $s(\cdot)$ is the previously defined state-retrieval function on sequences. Now it can be proved that

LEMMA: (*compositionality of BT over* $\mathbb{E}$)

(*i*) $\pi([\![S]\!]_{BT}(\gamma);[\![S']\!]_{BT}(\gamma)) = \lambda mt\sigma.\pi([\![S]\!]_{BT}(\gamma)(m,t,\sigma));X$ with

$$X = \bigcup\{\, \pi([\![S']\!]_{BT}(\gamma)(0,0,\sigma')) : \sigma' \in yield^*([\![S]\!]_{BT}(\gamma)(m,t,\sigma)) \,\}$$

(*ii*) $\pi([\![S]\!]_{BT}(\gamma) \,||\, [\![S']\!]_{BT}(\gamma) = \lambda mt\sigma.(\pi([\![S]\!]_{BT}(\gamma)(m,t,\sigma)) \,||\, \pi([\![S']\!]_{BT}(\gamma)(m,t,\sigma)))$

Proof: Straightforward.

COROLLARY:
Let $\sigma_{i+1} = yield^*([\![S_i]\!]_{BT}(\gamma)(0,0,\sigma))$ with $\sigma_0 = \sigma$

(*i*) $\pi([\![SEQ(S_1,...,S_n)]\!]_{BT}(\gamma)(0,0,\sigma)) = \pi([\![S_1]\!]_{BT}(\gamma)(0,0,\sigma_0)); \ldots ;\pi([\![S_n]\!]_{BT}(\gamma)(0,0,\sigma_{n-1}))$

(*ii*) $\pi([\![PAR(S_1,...,S_n)]\!]_{BT}(\gamma)(0,0,\sigma)) = \pi([\![S_1]\!](\gamma)(01,0,\sigma)) \,||\, \ldots \,||\, \pi([\![S_n]\!](\gamma)(0n,0,\sigma))$

Let $X_i = \pi([\![S_i]\!]_{BT}(\gamma)(0,0,\sigma_i)$ with $\sigma_i = yield^*([\![g_i]\!]_{BT}(0,0,\sigma))$ then

(*iii*) $\pi([\![ALT(g_1\ S_1,...,g_n\ S_n)]\!]_{BT}(\gamma)0,),\sigma)) =$

$$\Delta(\pi([\![g_1]\!]_{BT}(\gamma)(0,0,\sigma));X_1 \cup \ldots \cup \Delta(\pi([\![g_n]\!]_{BT}(\gamma)(0,0,\sigma));X_n$$

For a proof of (iii) note that taking care of the respective domains

$$\Delta(\pi([\![S]\!]_{BT}(\gamma))) = \pi(\Delta([\![S]\!]_{BT})(\gamma))$$

A more general statement of the corollary is ommitted since this would involve a more troublesome timing-operator for processes.

THEOREM: $[\![S]\!]_{LT}(\sigma) = \pi([\![S]\!]_{BT}(\gamma))(0,0,\sigma)$

The theorem is easily proved with the help of the previous lemma's. However the proof is not completed since in the modified form both the operational and the the denotational semantics allow to much.
What actually has to be proved is

THEOREM: $\mathcal{O}_1(S)(\sigma) = \mathit{paths}(\mathit{prune}([\![S]\!]_{BT}(\gamma) \setminus C))(0,0,\sigma)$

Only a sketch of the proof is provided here.
Observe that by the definition of $\mathcal{O}_1$ and the priority-rules for communication paths with holes are delivered only if the computation stumbles on a deadlocking configuration. Similarly a deadlocking configuration is detected in the denotational semantics when after restriction there are no alternatives. To equate the semantics it suffices to end a deadlocking computation as failing.
A detailed analysis of the selection-function for matching communication intentions, the priority of application of the rules and the guard-timing operator is necessary to establish the correctness of the pruning operator with respect to the operational semantics of the alternative construct.
Concluding, it can be stated that the extensions that allow a proof of the equivalence of the semantics do not incur a change of meaning to the statements of the language.

6. A COMPARISON WITH OTHER APPROACHES

In the previous section an outline of the proof of operational *linear time* semantics and denotational *branching time* semantics is given by proving a slightly modified operational semantics to be compositional with respect to operators over event-sequences.

It is felt that the relation of these semantics to AT-semantics and the motivation behind the introduction of process-numbering and timing is clarified by relating the respective semantics to other approaches:

(i) partial order semantics (Re,Bes)
(ii) event-structure semantics (Wi,NPW)
(iii) real time semantics (KSRGA,Zij)

Both in (i) and (ii) one of the main issues is real-time justice or weak fairness which informally can be characterised as the absence of infinite delay for ready communication intentions. Roughly this requirement amounts to a $O(1)$ reponse time for communication instructions, that is a reponse time that is independent of the number of distinct processes concurrently active.
It could be argued that this aspect is an issue of implementation and not of semantics. However the language Occam provides for instructions to explicitly allocate processes to processors to influence the response time (Cf. Jo).
Also, in the approach of (i) an assumption of maximality is made whereby each component in a parallel process is allocated to a distinct processor.

The introduction of process-numbering and timing can be motivated as follows:
- It allows a natural translation of event-sequences into partial order computations in the sense of (i).
- Process-numbering appeared to be a necessity in giving a metric denotational semantics that can be directly related to the event-structure semantics of (ii).
- It forms an alternative to the modeling of time by sequences of bags of communication records as is done by (KSRGA). Moreover no commitment to (multiples of) unit time is made in this approach.

It will not be claimed that the resulting semantics are very transparant or easy to use. However it is shown that partial order and real time semantics are not as unrelated to semantics in the tradition of (BZ1) as was previously thought.

The semantics given might even be extended to include other aspects of the mentioned approaches.

- The interleaving operator introduced to relate AT-semantics to BT-semantics might be selectively used to model partial maximality in order to reflect the use of a placed parallel construct. A modification of the time-value exchange is however necessary then such that the time of a branch consisting of merged components is the sum of the times of the components minus the wait-times for communication, after restriction.

- The real time communication model of (KSRGA) in which communication possibilities can be excluded if the wait-time for one of the components exceeds a certain bound can be simulated in the semantics given by eliminating communications of which the difference in local times exceeds this bound.

In what follows the relation of the semantics to partial order semantics will be specified by giving a mapping of event sequences to partial order computations thereby providing the inverse of an interleaving mapping as presented by (Re). After that the relation to the more general event structure

semantics of (Wil) will be indicated.
Some suggestions for further research conclude this section.

6.1. Partial order semantics

In the previous section LT and BT semantics were interrelated on a domain of event sequences.

The process-domain for po-semantics (and event structure semantics) will simply consist of a set of events, a partial order (that is a dependency relation) between events, and later when an extension to general event structures is made a conflict relation, indicating the possibility of choice between mutually incompatible (sequences of) events.

DEFINITION: (*simple conflict-free event-structures)*)
Let $E_0 = A_{roc} \times \underline{M} \times \mathbb{N} \times \Sigma$ be the set of possible events as specified in the previous chapter. Then for $E \subseteq E_0$ the structure $\underline{E} = (E, \leqslant)$ is given as the closure of

$$<\xi, m, t, \sigma> \leqslant <\xi', m', t', \sigma'> \text{ iff } m \leqslant m' \;\&\; t \leqslant t'$$

$$<\xi, m, t, \sigma> \leqslant <\tau_c{:}v, (m', m''), t', \sigma'> \text{ iff } (m \leqslant m' \text{ or } m \leqslant m'') \;\&\; t \leqslant t'$$

$$<\tau_c{:}v, (m', m''), t, \sigma> \leqslant <\xi, m, t', \sigma'> \text{ iff } (m' \leqslant m \text{ or } m'' \leqslant m) \;\&\; t \leqslant t'$$

$$<\tau_c{:}v, (m_1, m_2), t, \sigma> \leqslant <\tau_c'{:}v', (m_1', m_2'), t', \sigma'> \text{ iff } t \leqslant t' \;\&\; (m_1 < m_1' \text{ or } m_1 \leqslant m_2' \text{ or}$$
$$m_2 \leqslant m_1' \text{ or } m_2 \leqslant m_2')$$

Simple event structures suffice only when no conflict among (sequences of) events occur, as is for instance the case in sequences gotten by mapping computations to event sequences for the operational semantics and extracting the event sequences by means of the *paths*-operator for the denotational semantics.

Let for certain S and σ

$$\underline{\mathbb{E}}_{LT(S,\sigma)} = \{\ (E_w, \leqslant))\ :\ e \text{ occurs in } w \in [\![S]\!]_{LT}(\sigma) \Rightarrow e \in E_w\ \}$$

$$\underline{\mathbb{E}}_{BT(S,\sigma)} = \{\ (E_w, \leqslant))\ :\ e \text{ occurs in } w \in paths([\![S]\!]_{BT}(\gamma) \setminus C)(0,0,\sigma) \Rightarrow e \in E_w\ \}$$

$$\underline{\mathbb{E}}_{BT(S,\sigma)} = \{\ (E_w, \leqslant))\ :\ e \text{ occurs in } w \in paths(\diamond[\![S]\!]_{BT}(\gamma))(0,0,\sigma) \Rightarrow e \in E_w\ \}$$

Then if the conjectures about the equivalence (under *path*-abstraction) of the semantics are right it immediately follows that $\underline{\mathbb{E}}_{LT} = \underline{\mathbb{E}}_{BT} = \underline{\mathbb{E}}_{AT}$.

LEMMA: *Simple event-structures form an equivalence relation on event-sequences.*
Proof: Define $w \approx w'$ if $(E_w, \leqslant) = (E_{w'}, \leqslant)$.
That is, two event-sequences are equivalent if they contain the same events. To see that this is the case it suffices to show that if $E_w = E_{w'}$ then w can be transformed to a w'' and w' to a w''' such that $w'' = w'''$.
An auxiliary lemma is needed. Essentially the idea, taken from (Be), is to call two sequences equivalent if they differ only in the order of neighbouring concurrent action-occurrences.

LEMMA: (*interchangeability*)
Define $w \approx_0 w'$ if for $w = w_1 ee' w_2$ and $w' = w_1' e' e w_2'$, $w_1 = w_1'$, $w_2 = w_2'$ and $e \not\lessdot e'$ and $e' \not\lessdot e'$.
This relation states that w and w' differ only in the order of two adjacent unrelated events. That this relation is well-defined follows from assuming the condition that indeed $w \approx w'$.
Omitting w_1, w_1', w_2 and w_2' four cases arise, corresponding to the cases in the definition of $\leqslant$.
(i) $e = <\xi, m, t, \sigma>$, $e' = <\xi', m', t', \sigma'>$
From the definition of $\leqslant$ and interchangeability it follows that ($m \not\lessdot m'$ or $t \not\lessdot t'$) and ($m' \not\lessdot m$ or $t' \not\lessdot t$)
(a)If m has no relation to m' then the computation of e is independent of the computation of e', that is in the derivation the component from which e' results might have been chosen before the component from which e results. Hence $ee' \approx e'e$.
(b)If $t > t'$ then it must be the case that m is unrelated to m' and (a) applies.
The other cases are similar. For finite event-sequences the equivalence relation is defined as the transitive closure of the interchangeability relation: $\approx =_{df} \approx_0^*$. For w, w' infinite $w \approx w'$ iff $w[n] \approx w'[n]$ for almost every $n > 0$ where $w[n]$ is the truncation of a sequence w to a sequence of length n.

DEFINITION: (*linearisation*)
Given a simple event-structure $E = (E, \leqslant)$ in the domain of event-structures $\underline{\mathbb{E}}$ let $\mathfrak{L}: \underline{\mathbb{E}} \to \underline{\mathbb{E}}$ be the mapping that assigns a total order to E respecting the order of $(E, \leqslant)$.
Let $\mathfrak{S}: \mathbb{E} \to \mathbb{E}: w \mapsto (E_w, \leqslant)$ be the mapping that forms a simple event-structure of a given sequence w.
Then it is easy to see that

$$\mathfrak{S} \circ \mathfrak{L}((E, \leqslant)) = (E, \leqslant), \text{ and}$$

$$\mathfrak{L} \circ \mathfrak{S}(w) \neq w$$

however by the definition of equivalence

$$\mathfrak{L} \circ \mathfrak{S}(w) \approx w$$

6.3. General event structures

The use of simple event-structures necessitates to model the non-deterministism that is either explicitly introduced by an alternative construct or implicitly by a choice of communication by forming sets of simple event structures.
This is not the most elegant way.
Conceptually it seems preferable to extend the simple event-structures with a conflict relation indicating the possibility of choice as in (Wil). Instead of repeating the definitions given in (Wil) it will be informally described what the approach amounts to. One definition however seems necessary.

DEFINITION: (*prime algebraic partial order*)
Let $(D, \sqsubseteq)$ be a partial order. An element $p \in D$ is a *complete prime* iff for all $X \subseteq D$ when the lub $\bigsqcup X$ exists and $p \sqsubseteq \bigsqcup X$ then $p \sqsubseteq x$ for some $x \in X$.
Say D is *prime algebraic* iff $\forall x \in D . x = \bigsqcup \{ p \sqsubseteq x : p$ is a *complete prime* $\}$

A general event-structure is a pair (E,F) where E is a set of events and $F \subseteq \mathcal{P}(E)$ is a family of configurations that satisfy certain criteria. Intuitively each configuration in F records the history of past events on which it is dependent. The criteria ensure that when an event is in some configuration its occurrence has depended on a unique set of events. The dependence will be a partial order since it is required that for two distinct events there exists a configuration in F that contains one but not the other event.

For $x \in F$ a dependency-relation $\leqslant_x$ on x can be given by

$$e \leqslant_x e' \Leftrightarrow \forall y \in F. y \subseteq x \Rightarrow (e' \in y \Rightarrow e \in y)$$

that states that e' is dependent on e if the occurrence of e' in a configuration implies the occurrence of e in that configuration.
For $e \in x$ define $[e]_x = \{ e' \in x : e' \leqslant_x e \}$, then it follows that $[e]_x = \bigcap \{ z \in F : e \in z \subseteq x \}$. In other words the elements $[e]_x$ form the complete primes in $(F, \subseteq)$ in the sense of the definition given before. An event-structure (E,F) is said to be *prime* iff every event e is contained in some configuration in F and whenever an event e is contained in two configurations x and y then their histories with respect to these configurations are equal, that is $[e]_x = [e]_y$

For (E,F) a *prime* event structure, the relation $\leqslant$ (the dependency relation) and # (the conflict relation) on E can be defined by:

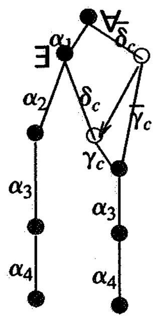
$$e' \leqslant e \;\; \textit{iff} \;\; \forall x \in F. e \in x \Rightarrow e' \in x$$

$$e' \# e \;\; \textit{iff} \;\; \forall x \in F. e \in x \Rightarrow e' \notin x$$

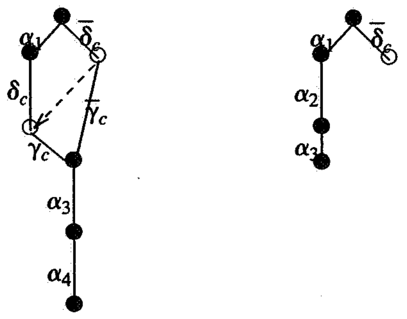
Then $\leqslant$ is a partial order such that $[e] =_{df} \{e' \in E : e' \leqslant e\}$ is finite for all $e \in E$ and # is a binary irreflexive symmetric relation such that $e \# e' \leqslant e'' \Rightarrow e \# e''$ for all $e, e', e'' \in E$.
The configurations F are precisely the left closed consistent (conflict free) subsets of E w.r.t. $\leqslant$ and #, i.e. $x \in F$ iff $x \subseteq E \;\&\; \forall e, e'. e' \leqslant e \in x \Rightarrow e' \in x \;\&\; \forall e, e' \in x. \neg(e \# e')$.
Obviously the configurations are uniquely determined by the sequence of actions leading to them.
From the definitions it follows that prime event-structures (E,F) are in $1-1$ correspondence with extended event-structures $(E, \leqslant, \#)$.

Rather than further elaborating on the definitions an example (fig 6.1) is given to suggest the possibility of relating event structure semantics directly to AT-semantics. The example will be followed by some suggestions for further research.

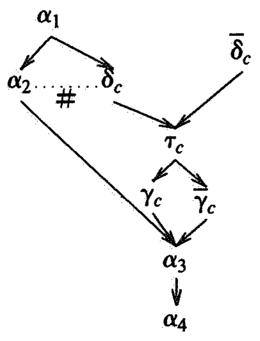
Example: $x \underset{\alpha_1}{:=} 1;(x \underset{\alpha_2}{:=} 2 + c\underset{\delta_c}{?}x);x \underset{\alpha_3}{:=} x+1 \| c\underset{\bar{\delta}_c}{!}4s);x \underset{\alpha_4}{:=} x+3$



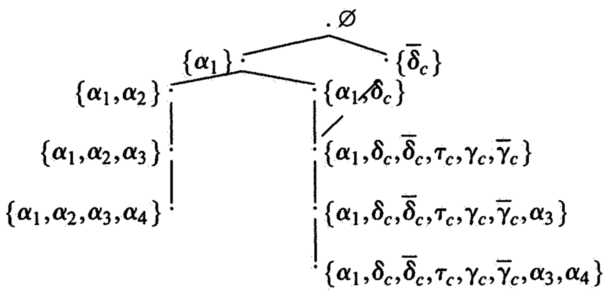
(a) AT-diagram



(b) simple event-occurrence graphs



(c) extended event-structure



$\emptyset$
$\{\alpha_1\}$
$\{\bar{\delta}_c\}$
$\{\alpha_1,\alpha_2\}$
$\{\alpha_1,\bar{\delta}_c\}$
$\{\alpha_1,\alpha_2,\alpha_3\}$
$\{\alpha_1,\delta_c,\bar{\delta}_c,\tau_c,\gamma_c,\bar{\gamma}_c\}$
$\{\alpha_1,\alpha_2,\alpha_3,\alpha_4\}$
$\{\alpha_1,\delta_c,\bar{\delta}_c,\tau_c,\gamma_c,\bar{\gamma}_c,\alpha_3\}$
$\{\alpha_1,\delta_c,\bar{\delta}_c,\tau_c,\gamma_c,\bar{\gamma}_c,\alpha_3,\alpha_4\}$

(d) prime event-structure

Fig. 6.1. Example.

6.3. Suggestions for further research

Supposing that an event structure semantics E for Occam can be given in a direct way, which is not as trivial as it seems since Occam requires a non-uniform approach that is not explored neither in (Wil) nor (GM), then the following layout for future research, suggested by the diagram below can be given.

$$\begin{tikzcd}
LT \arrow[r, dash] \arrow[dr, dash] & BT \arrow[r, dash] \arrow[d, dash] & AT \arrow[dl, dash] \\
 & E &
\end{tikzcd}$$

- The relation BT-AT is treated by means of an interleaving operator to be applied to processes in AT with the conjecture that

$$\mathit{interleave}(\llbracket S \rrbracket_{AT}(\gamma)) \setminus C = \mathit{interleave}(\lozenge \llbracket S \rrbracket_{AT}(\gamma))$$

- A mapping from AT to E is to be defined, say $AT \xrightarrow{\mathfrak{E}_{AT}} E$, such that

$$\mathfrak{E}(\mathit{paths}(\lozenge \llbracket S \rrbracket_{AT}(\gamma))) = \mathfrak{E}_{AT}(\lozenge \llbracket S \rrbracket_{AT}(\gamma))$$

  where $\mathfrak{E}$ is the embedding of sequences in structures as described before.

Variations on this scheme are imaginable. However before undertaking any action in the direction sketched it seems preferable to work out the indicated relations for the uniform case first. With this remark the treatment of semantics for Occam is concluded.

Acknowledgments: I thank Jaco de Bakker, Joost Kok, Jan Rutten, Frank de Boer, John Jules Meyer, Erik de Vink and Peter Van Emde Boas for their continuous intellectual and moral support.

REFERENCES

[Ap]. K. APT (Jan 1983). Formal justification of a proof system for communicating processes. *JACM30.1*, 197-216.

[dB]. J.W. DE BAKKER (1980). *Mathematical theory of program correctness*, Prentice Hall.

[BBKM]. J.W. DE BAKKER, J.A. BERGSTRA, J.W. KLOP, and J.-J.CH. MEYER (1984). Linear time and branching time semantics for recursion with merge. *TCS.35*, 135-156.

[BKMOZ]. J.W. DE BAKKER, J.N. KOK, J.-J.CH. MEYER, E.R. OLDEROG, and J.I. ZUCKER (Jan 1986). *Contrasting themes in the semantics of imperative concurrency,* CWI Report CS-R8603.

[BMOZ1]. J.W. DE BAKKER, J.-J.CH. MEYER, E.R. OLDEROG, and J.I. ZUCKER. *Transition-systems, infinitary languages and the semantics of uniform concurrencyin*: Proceedings 17th ACM STOC, Providence R.I. 1985

[BMOZ2]. J.W. DE BAKKER, J.-J.CH. MEYER, E.R. OLDEROG, and J.I. ZUCKER. *Transition-systems, metric spaces and ready-sets in the semantics of uniform concurrency,* preprint SUNY at Buffalo, (full version of BMOZ1), to appear.

[BZ1]. J.W. DE BAKKER and J.I. ZUCKER (1982). Processes and the denotational semantics of concurrency. *Information and Control.54*, 70-120.

[BZ2]. J.W. DE BAKKER and J.I. ZUCKER. Processes and a fair semantics for the ADA rendez-vous. *LNCS 154*, in: Proc. 10th ICALP.

[B]. D.B. BENSON (1982). In Scott-Strachey style denotational semantics, parallelism implies nondeterminism. *Math. Systems Theory.15*, 267-275.

[BK]. J.A. BERGSTRA and J.W. KLOP (1985). Algebra of communicating processes with abstraction. *TCS.37*, 77-121.

[BKT]. J.A. BERGSTRA, J.W. KLOP, and J.V. TUCKER. Process-algebra with asynchronous communication mechanisms. *LNCS 197.*

[Be]. A.J. BERNSTEIN (April 1980). Output guards and non-determinism in communicating processes. *ACM TOPLAS*, 234-238.

[Bes]. E. BEST. Concurrent behaviour: sequences, processes and axioms. *LNCS 197*, in: Seminar on Concurrency.

[BHR]. S.D. BROOKES, C.A.R. HOARE, and A.W. ROSCOE (1984). A theory of communicating sequential processes. *JACM.31*, 560-599.

[BRW]. S.D. BROOKES, A.W. ROSCOE, and G. WINSKEL (EDS.). Seminar on concurrency. *LNCS 197.*

[BR]. S.D. BROOKES and W.C. ROUNDS. Behavioral equivalence relations induced by programming logics. *LNCS 154*, in: Proc. 10th ICALP.

[Ca]. L. CARDELLI. Real time agents. *LNCS 140*, ICALP 1982.

[CKS]. A.K. CHANDRA, D.C. KOZEN, and L.J. STOCKMEYER (Jan 1981). Alternation. *JACM28.1*, 114-137.

[De]. DETHMER (April 1985). Occam and the transputer. *Electronics and Power.*

[FO]. J.P. FINANCE and M.S. OUERGHI (1983). *On the algebraic specification of concurrency and communication*IEEE

[FLP]. N. FRANCEZ, D.J. LEHMAN, and A. PNUELI (1984). A linear history semantics for distributed programming. *TCS.32*, 25-46.

[Ge]. R. GERTH. *Denotational semantics for DNP-R,* Report Univ. Utrecht.

[GR]. W.G. GOLSON and W.C. ROUNDS (1983). Connections between two theories of concurrency: metric spaces and synchronisation-trees. *Inf. and Control57*, 102-124.

[GM]. U. GOLTZ and A. MYCROFT (1984). On the relationship of CCS and Petri-nets. *LNCS 172*, 196-208.

[HH]. E.C.R. HEHNER and C.A.R. HOARE (1983). A more complete model of communicating sequential processes. *TCS26*, 105-120.

[HP1]. M. HENNESY and G.D. PLOTKIN (1979). Full abstraction for a simple programming language. *LNCS 74*, 108-120, in: Proc. 8th MFCS.

[HP2]. M. HENNESY and G.D. PLOTKIN (1980). A term model for CCS. *LNCS 88*, in: Proc. 9th MFCS.
[Ho]. C.A.R. HOARE (1980). Communicating sequential processes. *CACM21*, 666-677.
[HR]. C.A.R. HOARE and A.W. ROSCOE (1984). *Programs as executable predicates*in: Proc. FGCS
[In]. INMOS LTD (1984). *The Occam Programming Manual*, Prentice Hall International, London.
[Jo]. G. JONES (1985). *Programming in Occam*, Oxford TM PRG-43.
[KSRGA]. R. KOYMANS, R.K. SHYAMESUNDAR, W.P. DE ROEVER, R. GERTH, and S. ARUN-KUMAR. *Compositional semantics for real-time distributed computing*, Report Univ. v. Utrecht.
[La]. L. LAMPORT (1978). Time, clocks and the ordering of events in distributive systems. *CACM7*.
[Ma]. A. MAZURKIEWICZ (1984). *Traces, histories, graphs: instances of a process-monoid*11th MFCS LNCS 176
[Mi1]. R.MILNER (1982). *Four combinators for concurrency*, Ottawain: Proc ACM Sigact Sigops
[Mi2]. R. MILNER (1983). Calculi for synchrony and asynchrony. *TCS34*, 83-134.
[Ni]. R. DE NICOLA (1984). Models and operators for non-deterministic processes. *LNCS*, in: Proc MFCS.
[NPW]. M. NIELSEN, G. PLOTKIN, and G. WINSKEL (1981). Petri-nets, event-structures and domains. *TCS13*, 85-108.
[OH]. E.R. OLDEROG and C.A.R. HOARE. Specification oriented semantics for communicating processes. *LNCS 154*, 561-572, in: Proc ICALP.
[Plo]. G.D. PLOTKIN (1983). An operational semantics for CSP. D.BJORNER (eds.). *Formal Description of Programming Concepts II*, 199-223, North Holland, Amsterdam.
[Pn]. A. PNUELI (1981). The temporal semantics of concurrent programs. *TCS13*, 45-60.
[Pr]. V. PRATT. The pomset model of parallel processes: unifying the temporal and the spatial. *LNCS 197*, 199-223, in: Seminar on Concurrency.
[Re]. W. REISIG. *Partial order semantics for CSP-like languages and its impact on fairness.*
[Ro]. A.W. ROSCOE. Denotational semantics for Occam. *LNCS 197*, in: Seminar on Concurrency.
[SM]. A. SALWICKI and T. MULDNER. *On the algoritmic properties of concurrent programs.*
[Sch]. F.B. SCHNEIDER. *Synchronisation in distributed programs*Toplas 1982
[Schw]. G.S. SCHWARZ. Denotational semantics of parallelism. *LNCS 70*, 191-202.
[Wi1]. G. WINSKEL (1982). Event-structure semantics for CCS and related languages. *LNCS 140*, 9th ICALP.
[Wi2]. G. WINSKEL (1984). Synchronisation-trees. *TCS34*, 33-82.
[Wi3]. G. WINSKEL (1985). Categories of models for concurrency. *LNCS 197*, in: Seminar on Concurrency.
[Zij]. E. ZIJLSTRA (1984). *Semantics of real time systems*doct. thesis UvA