Introductory note to "Object-Oriented Algebraic Specification"

Jan Bergstra
Jan Heering
Jan Willem Klop

The following CWI report proposes a notation for OOAS (Object-Oriented Algebraic Specification). It is one of four related formalisms in the area of algebraic specification that were conceived around 1984 at CWI. The other ones were ACP (Algebra of Communicating Processes), ASF (Algebraic Specification Formalism), and BMA (Basic Module Algebra). Whereas these have generated and still generate a significant volume of research, OOAS was considered of minor importance and, apart from its use in [1], no further study of it was made by CWI researchers.

In retrospect, this is unfortunate. When Banâtre *et al.* [2] independently introduced multiset programming, which in turn led Berry and Boudol [3] to the Chemical Abstract Machine (CHAM), the underlying concepts and definitions turned out to be very close to OOAS. Since then the French researchers have made substantial progress, and the CHAM has become an important theoretical tool.

We respectfully dedicate this account of the vagaries of scientific work to Cor Baayen on the occasion of his retirement as scientific director from CWI.

References

1. J.C.M. Baeten, J.A. Bergstra, and J.W. Klop, An operational semantics for process algebra, in: *Mathematical Problems in Computation Theory*, Banach Center Publications, Vol. 21, PWN—Polish Scientific Publishers, Warsaw, 1988, pp. 47–81.
2. J.-P. Banâtre, A. Coutant, and D. Le Métayer, A parallel machine for multiset transformation and its programming style, *Future Generations Computer Systems*, **4** (1988), pp. 133–144.
3. G. Berry and G. Boudol, The chemical abstract machine, in: *Conference Record of the Seventeenth ACM Symposium on Principles of Programming Languages (POPL '90)*, ACM, 1990, pp. 81–94.

OBJECT-ORIENTED ALGEBRAIC SPECIFICATION: PROPOSAL FOR A NOTATION AND 12 EXAMPLES

J.A. BERGSTRA, J. HEERING, J.W. KLOP
Centre for Mathematics and Computer Science, Amsterdam

A notation is introduced for expressing the dynamic behaviour of configurations of objects. At each instant of time a configuration is just a multi-set of objects which themselves are points (values) from some algebraically specified abstract data type. Several examples should convince the reader of the attractive expressive power of our notation.

1980 MATHEMATICS SUBJECT CLASSIFICATION: 68C01, 68F20.
1982 CR CATEGORIES: F.1.1, F.3.2.
KEY WORDS & PHRASES: object-oriented specification, algebraic specification, configuration transition system, transformation rule.
NOTE: This report will be submitted for publication elsewhere.

Report CS-R8411
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

1. INTRODUCTION

This note has the following aim: to propose a *notation* compatible with the well-known notations for *algebraic data type specification* which captures the concept of an *object*.

The reasons for doing so are many; we list some reasons in arbitrary order:

(a) There is an increasing interest in object-oriented approaches to software design. See Cox [4], Jamsa [6], Jonkers [7] for some discussions of object-oriented programming.

(b) The discussion on what constitutes an object and what constitutes a value is not yet settled. See Cohen [3] and MacLennan [9] for two very interesting expositions about the nature of objects.

(c) From the point of view of abstract data types (and their algebraic specification) it is hard to understand what an object is. The history of the subject is confusing indeed. The Simula class is meant as a class of objects. Abstract data types in the ADJ tradition are modules of structured values. In the survey by Goguen & Meseguer [5] an option to augment data types with states is discussed, thus regaining some of the dynamic aspects that were somehow lost in the "initial algebra = abstract data type" stage.

(d) We feel that a workable distinction between objects and values can be made, taking algebraic abstract data type specifications as a point of departure.

2. AN ORGANISATION OF NOTIONS

Let Σ be a (many-)sorted algebraic signature, let $A \in Alg(\Sigma)$ be an algebra of type (signature) Σ. A is called an *abstract data type*. For (algebraic) specification of abstract data types, we refer to the literature collected in Kutzler & Lichtenberger [8].

The signature Σ is a triple $\$(\Sigma)$, $\mathbb{F}(\Sigma)$, $C(\Sigma)$ (sorts, functions and constants) of Σ. For $s \in \$(\Sigma)$, A_s is the interpretation of sort s in A. An element of A_s will be called a *point*. A_s itself will also be called a *data space*. (See Figure 1.) A point $p \in A_s$ may play two roles:

(i) p may represent a value,

(ii) p may represent an object (with a particular state).

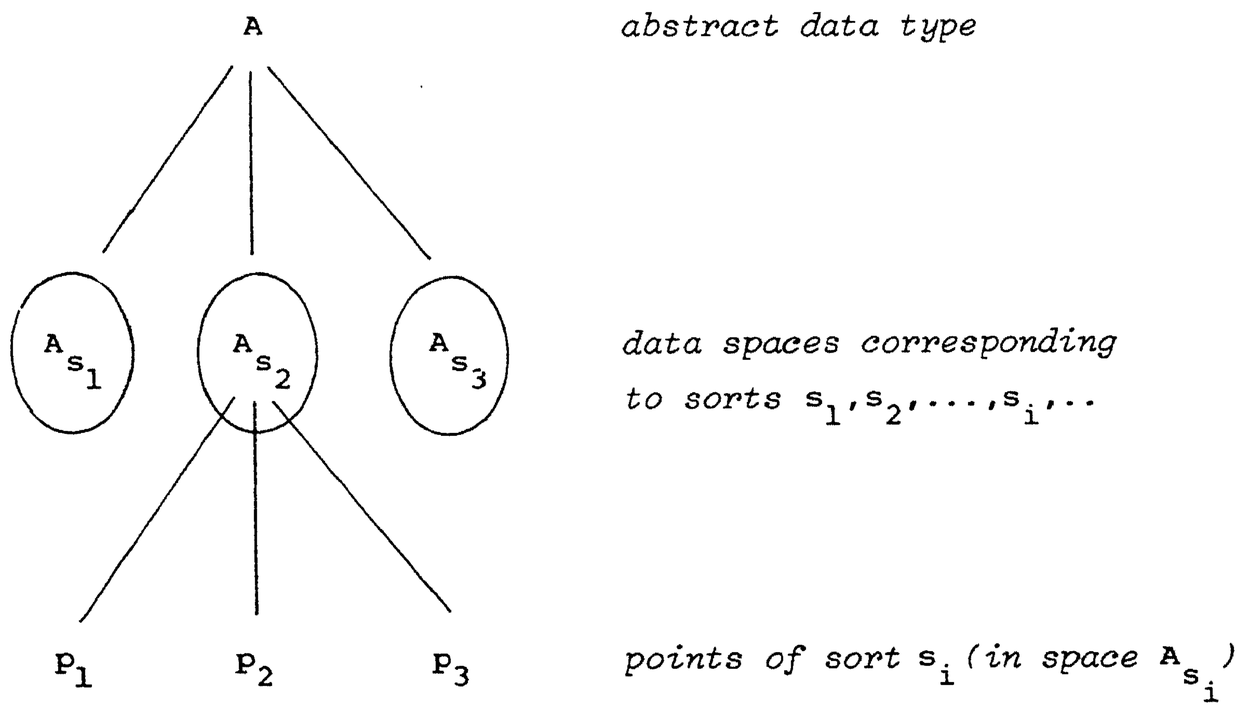


Figure 1.

A multi-set of objects (i.e. a multi-set of points seen as objects) is called a *configuration*. Configurations exhibit dynamic behaviour. In particular, configurations may perform (or allow) *transformation steps*

$$C \xrightarrow[R]{} C'.$$

Transformation steps are generated from *transformation rules*. In Section 3 we will present syntax and semantics of a *notation* for transformation rules.

Suppose that we know what a rule is for a given signature Σ. Let T be a collection of transformation rules, A a Σ-algebra. Then the pair <A,T> determines a *configuration transition system*.

If $A = T_I(\Sigma,E)$, i.e. (Σ,E) is an initial algebra specification of A, and T is a collection of transformation rules for Σ, then

$$<(\Sigma,E),T>$$

is an *object-oriented algebraic specification* which specifies a *configuration transition system*.

3. TRANSFORMATION RULES

Informally, a transformation rule is a notation of the following kind:

$$\textit{rule name (parameter list)} \left[\frac{\textit{configuration before transformation}}{\textit{configuration after transformation}} \right]$$

Often it is convenient to divide the parameter list in three parts: one part associated with the rule name, the other two parts consisting of input values and output values respectively. This suggests the following notation:

$$\textit{rule name (par. list)} \left[\begin{array}{l|l} \textit{configuration before transformation} & \textit{input values} \\ \hline \textit{configuration after transformation} & \textit{output values} \end{array} \right]$$

The input values constitute a multi-set of points which are consumed during the transformation and the output values constitute a multi-set of points which are produced during the transformation. It is understood that a configuration may be transformed inside a context (a larger configuration). So if $C_1 \subseteq C_1 \cup C_2$ is a sub-configuration of $C_1 \cup C_2$ (where $\subseteq$ denotes inclusion between multi-sets and $\cup$ their union), and

$$R = \underline{\text{name}}\ (\vec{p}) \left[\begin{array}{c|c} C_1 & \vec{a} \\ \hline C_1' & \vec{b} \end{array} \right]$$

is an instance of the rule with name name, then $C_1 \cup C_2 \xrightarrow[R]{} C_1' \cup C_2$ is a transformation step. (For a more elaborate explanation, see Section 9.)

Example: an instantiation R of the transformation rule

$$\underline{\text{add}} \left[\begin{array}{c|c} x & y \\ \hline x+y & \end{array} \right]$$

used in the example below, is: $R = \underline{\text{add}} \left[\begin{array}{c|c} 3 & 5 \\ \hline 8 & \end{array} \right]$. (Here 3 is short for (1+1)+1, etc.) In this example $\vec{p}$, $\vec{b}$ are empty, and $C_1 = \{3\}$, $C_1' = \{8\}$. Now we have the transformation step

$$\{3\} \xrightarrow[R]{} \{8\}$$

and also e.g. for $C_2 = \{7,1\}$, the step

$$\{3,7,1\} \xrightarrow[R]{} \{8,7,1\}.$$

Such steps can be composed into transformation sequences; e.g. if R' is the instantiation: $\underline{\text{add}} \left(\begin{array}{c|c} 7 & 6 \\ \hline 13 & \end{array}\right)$, we have

$$\{3,7,1\} \xrightarrow[R]{} \{8,7,1\} \xrightarrow[R']{} \{8,13,1\}.$$

Here we would like to point out the relation to Plotkin [10], which addresses similar issues, where system behaviour is systematically described by means of transition relations.

The following two very simple examples will help to further explain the notation. Consider the following specification of the initial algebra A:

$$\Sigma \quad \left|\begin{array}{l} \$: \mathbf{N} \\ \quad \mathbf{ER} \\ \mathbb{F}: +: \mathbf{N} \times \mathbf{N} \to \mathbf{N} \\ \quad \cdot: \mathbf{N} \times \mathbf{N} \to \mathbf{N} \\ ¢: 0 \in \mathbf{N} \\ \quad 1 \in \mathbf{N} \\ \quad \perp \in \mathbf{ER} \end{array}\right.$$

$$E \quad \left|\begin{array}{l} x + 0 = x \\ x + (y + 1) = (x + y) + 1 \\ x \cdot 0 = 0 \\ x \cdot (y + 1) = x \cdot y + x \end{array}\right.$$

Now $A = T_I(\Sigma,E)$. We will now present two different collections T_1 and T_2 of transformation rules for configurations over A.

$$T_1 \quad \left|\begin{array}{ll} \underline{\text{succ}} \left(\begin{array}{c|c} x & \\ \hline x+1 & \end{array}\right) & T_{1,1} \\ \underline{\text{add}} \left(\begin{array}{c|c} x & y \\ \hline x+y & \end{array}\right) & T_{1,2} \\ \underline{\text{subtract}} \left(\begin{array}{c|c} x+y & y \\ \hline x & \end{array}\right) & T_{1,3} \end{array}\right.$$

$$\underline{\text{subtract}}\left(\begin{array}{c|c} x & x+y+1 \\ \hline x & \bot \end{array}\right) \qquad T_{1,4}$$

If one starts with the initial configuration {0}, then T_1 describes the behaviour of a single counter with some actions (transformations) on it; part of this behaviour is as in Figure 2.

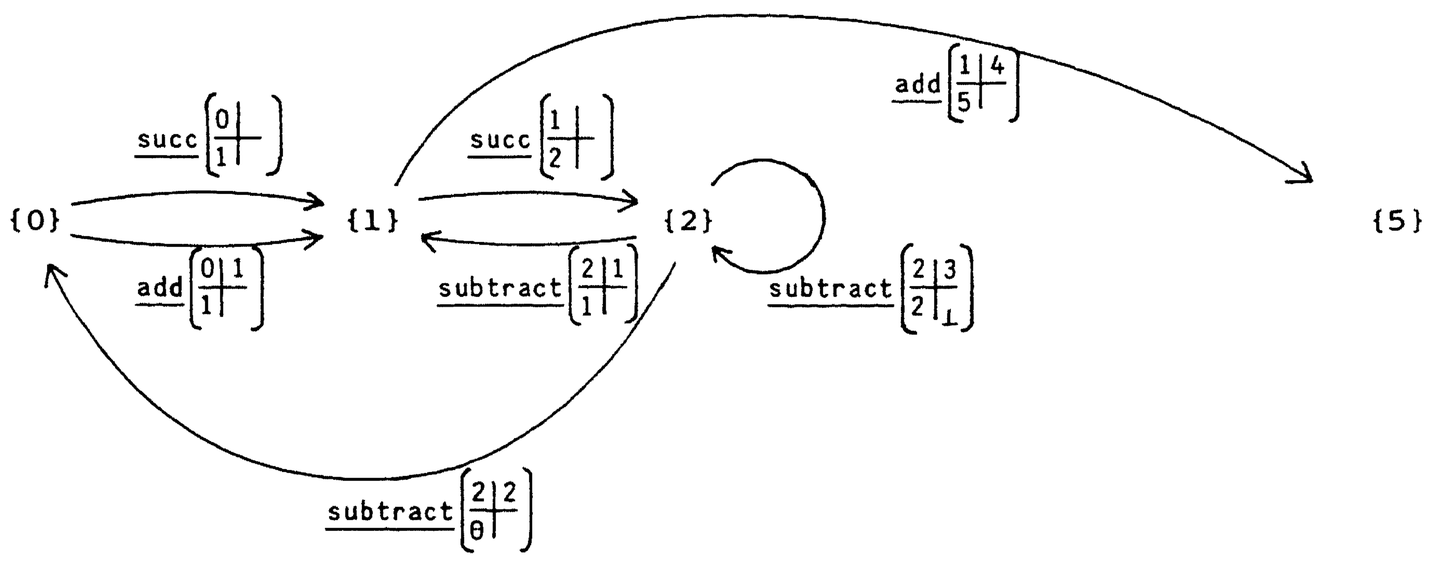


Figure 2.

Further comments on the rules of T_1:

(i) If one of the compartments of the 'matrix' is left empty, this means that the empty multi-set $\emptyset$ of values (or objects) is meant.

(ii) Note the difference between rule $T_{1,2}$ and the rule

$$\underline{\text{add}}\left(\begin{array}{cc|} x & y \\ \hline x+y & \end{array}\right);$$

in $T_{1,2}$ we focus on the transformation of one object, while in the displayed rule the *fusion* of two objects is embodied.

(iii) The rules $T_{1,3}$ and $T_{1,4}$ for subtraction exhibit *polymorphism of types:* in $T_{1,3}$ the multi-set of output values is empty, while in $T_{1,4}$ an error message is delivered.

In the second example the same initial algebra A as above is used. The set T_2 of transformation rules for configurations over A will describe the behaviour of a fixed number n_0 of counters. The k-th counter ($k \in \{0,..,n_0-1\}$) with content x can conveniently be represented (coded) by the natural number $k + n_0 x$. Below, k,ℓ,m vary over $\{0,...,n_0-1\}$.

T_2

$$\underline{\text{create}}(k)\left(\begin{array}{c|c} & x \\ \hline k+n_0x & \end{array}\right) \qquad T_{2,1}$$

$$\underline{\text{add}}(k,\ell,m)\left(\begin{array}{c|c} k+n_0x,\ \ell+n_0y & \\ \hline m+n_0(x+y) & \end{array}\right) \qquad T_{2,2}$$

$$\underline{\text{mult}}(k,\ell,m)\left(\begin{array}{c|c} k+n_0x,\ \ell+n_0y & \\ \hline m+n_0xy & \end{array}\right) \qquad T_{2,3}$$

$$\underline{\text{succ}}(k)\left(\begin{array}{c|c} k+n_0x & \\ \hline k+n_0(x+1) & \end{array}\right) \qquad T_{2,4}$$

$$\underline{\text{read}}(k)\left(\begin{array}{c|c} k+n_0x & \\ \hline & x \end{array}\right) \qquad T_{2,5}$$

$$\underline{\text{compare}}(k)\left(\begin{array}{c|c} k+n_0(x+y) & x \\ \hline k+n_0(x+y) & 0 \end{array}\right) \qquad T_{2,6}$$

$$\underline{\text{compare}}(k)\left(\begin{array}{c|c} k+n_0x & x+y+1 \\ \hline k+n_0x & 1 \end{array}\right) \qquad T_{2,7}$$

$$\underline{\text{skip}}(k)\left(\begin{array}{c|c} k+n_0x & \\ \hline & \end{array}\right) \qquad T_{2,8}$$

$$\underline{\text{copy}}(k,\ell)\left(\begin{array}{c|c} k+n_0x & \\ \hline k+n_0x,\ \ell+n_0x & \end{array}\right) \qquad T_{2,9}$$

Comments: (i) The rules $T_{2,6}$ and $T_{2,7}$ for compare(k) compare the content a of counter k with some given number b; if $a \geqslant b$ the output is 0, otherwise 1.
(ii) Note that the copy(k,ℓ) rule can lead to confusion (in the sense that two indiscernible objects may arise) if it is applied while an object of the form $\ell + n_0 x$ is present (which can be avoided by first performing skip(ℓ) or read(ℓ)).
(iii) The empty configuration is an adequate initial configuration for this system. Clearly $T_{2,1-9}$ offer only limited facilities (subtraction is absent etc.). Moreover explicit naming might be a preferable alternative to the coding trick, which represents "counter k with content x" as $k + n_0 x$, if natural number objects are to be maintained.

4. THE STACK

In this section we consider object-oriented specifications of the stack. We formulate four different specifications of the dynamic behaviour of a single stack. This raises the following
Question: is it possible to express this rich variety of operational possibilities without the object-oriented approach (i.e. in terms of the original algebraic framework)?
We will leave this question unanswered.

Σ		
	\$:	A
		S
		ER
		B
	IF:	push: $A \times S \to S$
	¢:	$a_1, \ldots, a_n \in A$
		$\perp \in ER$
		$\emptyset \in S$
		$T \in B$
		$F \in B$

$E = \emptyset$

As data space we use $T_I(\Sigma,\emptyset)$.

T_3

$\underline{\text{push}}\left[\begin{array}{c\|c} x & a \\ \hline \text{push}(a,x) & \end{array}\right]$	$T_{3,1}$
$\underline{\text{pop}}\left[\begin{array}{c\|c} \text{push}(a,x) & \\ \hline x & a \end{array}\right]$	$T_{3,2}$
$\underline{\text{pop}}\left[\begin{array}{c\|c} \emptyset & \\ \hline \emptyset & \perp \end{array}\right]$	$T_{3,3}$

The initial configuration is $\{\emptyset\}$. At each time the configuration will be a singleton.

T_4

$\underline{\text{push}}\left[\begin{array}{c\|c} x & a \\ \hline \text{push}(a,x) & \end{array}\right]$	$T_{4,1}$
$\underline{\text{pop}}\left[\begin{array}{c\|c} \text{push}(a,x) & \\ \hline x & \end{array}\right]$	$T_{4,2}$
$\underline{\text{pop}}\left[\begin{array}{c\|c} \emptyset & \\ \hline \emptyset & \perp \end{array}\right]$	$T_{4,3}$
$\underline{\text{top}}\left[\begin{array}{c\|c} \text{push}(a,x) & \\ \hline \text{push}(a,x) & a \end{array}\right]$	$T_{4,4}$
$\underline{\text{top}}\left[\begin{array}{c\|c} \emptyset & \\ \hline \emptyset & \perp \end{array}\right]$	$T_{4,5}$

As in the previous case $\{\emptyset\}$ should be taken as the initial configuration.

T_5

$\underline{\text{create}}\left[\begin{array}{c\|c} & \\ \hline \emptyset & \end{array}\right]$	$T_{5,1}$
$\underline{\text{push}}\left[\begin{array}{c\|c} x & a \\ \hline \text{push}(a,x) & \end{array}\right]$	$T_{5,2}$

$$
\left|\begin{array}{ll}
\underline{\text{pop}}\left(\begin{array}{c|c} \text{push}(a,x) & \\ \hline x & a \end{array}\right) & T_{5,3} \\[2ex]
\underline{\text{pop}}\left(\begin{array}{c|c} \emptyset & \\ \hline & \perp \end{array}\right) & T_{5,4}
\end{array}\right.
$$

In the case of T_5, pop is destructive on $\emptyset$. Hence after $\perp$ has been observed an empty stack must be created again. Care must be taken not to create two or more stacks at the same time, because this would lead to non-deterministic effects of pop.

In the next example T_6 we replace the create facility by a test on emptiness of the stack.

$$
T_6 \left|\begin{array}{ll}
\underline{\text{push}}\left(\begin{array}{c|c} x & a \\ \hline \text{push}(a,x) & \end{array}\right) & T_{6,1} \\[2ex]
\underline{\text{empty}}\left(\begin{array}{c|c} \text{push}(a,x) & \\ \hline \text{push}(a,x) & F \end{array}\right) & T_{6,2} \\[2ex]
\underline{\text{empty}}\left(\begin{array}{c|c} \emptyset & \\ \hline \emptyset & T \end{array}\right) & T_{6,3} \\[2ex]
\underline{\text{pop}}\left(\begin{array}{c|c} \text{push}(a,x) & \\ \hline a & a \end{array}\right) & T_{6,4} \\[2ex]
\underline{\text{pop}}\left(\begin{array}{c|c} \emptyset & \\ \hline & \perp \end{array}\right) & T_{6,5}
\end{array}\right.
$$

In the case of T_6, $\{\emptyset\}$ is again an appropriate initial configuration. In order to prevent loss of the stack it is useful to do pop only after a test on emptiness. If the stack is not empty, pop may be safely applied; otherwise it should not be applied because in that case the object would be irreversibly destroyed.

5. PROCESS ALGEBRA WITHOUT COMMUNICATION

Let (Σ_{PA}, PA) be the following specification.

Σ_{PA}

\$:	PR
IF:	$+: PR \times PR \to PR$
	$\cdot: PR \times PR \to PR$
	$\|\|: PR \times PR \to PR$
	$\lfloor\!\lfloor : PR \times PR \to PR$
¢:	$a_1, \ldots, a_n \in PR$

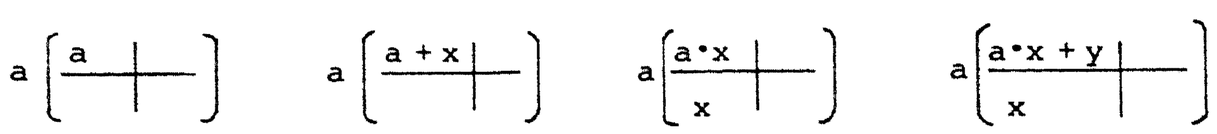
PA

$x + y = y + x$	A1
$(x + y) + z = x + (y + z)$	A2
$x + x = x$	A3
$(x + y) \cdot z = x \cdot z + y \cdot z$	A4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5
$x \| y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor z$	M1
$a \lfloor\!\lfloor x = a \cdot x$	M2
$(a \cdot x) \lfloor\!\lfloor y = a \cdot (x \| y)$	M3
$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	M4

Here 'a' varies over $A = \{a_1, \ldots, a_n\}$. We will write the initial algebra $T_I(\Sigma_{PA}, PA)$ of this specification as $A_\omega(+, \cdot, \|, \lfloor\!\lfloor)$. With $A_\omega(+, \cdot)$ we denote the reduct of $A_\omega(+, \cdot, \|, \lfloor\!\lfloor)$ after forgetting $\|$ and $\lfloor\!\lfloor$. Let $\Sigma_{PA}^{+,\cdot}$ be Σ_{PA} minus $\|, \lfloor\!\lfloor$ and let BPA be A1-5. It can be shown (see Bergstra & Klop [2]) that $A_\omega(+, \cdot) = T_I(\Sigma_{PA}^{+,\cdot}, BPA)$. The axiom system PA was introduced in [2] as the core axiomatisation of process algebra.

When we take $A_\omega(+, \cdot)$ as a data space, and use the $a \in A$ as rule names, the following transformation rules (without inputs and outputs) reflect the operational semantics of + (*choice, alternative composition*) and $\cdot$ (*product, sequential composition*):

$T_{7,1-4}$



Now consider the configuration

$$\{x_1,\ldots,x_k\}.$$

The behaviour of this configuration corresponds to that of the process

$$x_1 \| \ldots \| x_k.$$

Thus the formation of configurations is represented by the operation || of PA. It can be concluded that process algebra is more denotational than object-oriented system specification by means of transformation rules.

6. SETS OF INTEGERS

Let Σ be as follows:

Σ

$:	N
	SN
	B
	ER
IF:	eq: N × N → B
	ins: N × SN → SN
	del: N × SN → SN
	s: N → N
¢:	T ∈ B
	F ∈ B
	0 ∈ N
	Ø ∈ SN
	⊥ ∈ ER

As (conditional) equational specification of the data space we take:

E

eq(0,0) = T
eq(0,s(x)) = F
eq(s(x),0) = F
eq(s(x),s(y)) = eq(x,y)
ins(x,ins(x,X)) = ins(x,X)
ins(x,ins(y,X)) = ins(y,ins(x,X))
del (x,Ø) = Ø

$$del(x,ins(x,Y)) = del(x,Y)$$

$$eq(x,y) = F \;\rightarrow\; del(x,ins(y,X)) = ins(y,del(x,X))$$

We will now describe a configuration transformation system starting from $\{\emptyset\}$ as an initial configuration.

T_8

$$\underline{ins}\left(\begin{array}{c|c} X & a \\ \hline ins(a,X) & \end{array}\right) \qquad T_{8,1}$$

$$\underline{del}\left(\begin{array}{c|c} X & a \\ \hline del(a,X) & \end{array}\right) \qquad T_{8,2}$$

$$\underline{get}\left(\begin{array}{c|c} ins(a,X) & \\ \hline X & a \end{array}\right) \qquad T_{8,3}$$

$$\underline{get}\left(\begin{array}{c|c} \emptyset & \\ \hline \emptyset & \perp \end{array}\right) \qquad T_{8,4}$$

$$\underline{elt}\left(\begin{array}{c|c} ins(a,X) & a \\ \hline ins(a,X) & T \end{array}\right) \qquad T_{8,5}$$

$$\underline{elt}\left(\begin{array}{c|c} del(a,X) & a \\ \hline del(a,X) & F \end{array}\right) \qquad T_{8,6}$$

$$\underline{empty}\left(\begin{array}{c|c} \emptyset & \\ \hline \emptyset & T \end{array}\right) \qquad T_{8,7}$$

$$\underline{empty}\left(\begin{array}{c|c} ins(a,X) & \\ \hline ins(a,X) & F \end{array}\right) \qquad T_{8,8}$$

Remark: note the implicit non-determinism present in $T_{8,3}$. Namely, by the instance

$$R = \underline{get}\left(\begin{array}{c|c} ins(a,ins(b,\emptyset)) & \\ \hline ins(b,\emptyset) & a \end{array}\right)$$

we have the step $\{ins(a,ins(b,\emptyset))\} \xrightarrow[R]{} \{ins(b,\emptyset)\}$. Further, by E we have

ins(a,ins(b,Ø)) = ins(b,ins(a,Ø)), hence the configuration in the LHS of the displayed step can also be transformed to {ins(a,Ø)} by the instance of $T_{8,3}$:

$$R' = \underline{get}\left(\frac{ins(b,ins(a,Ø))\ |\ }{ins(a,Ø)\ |\ b}\right).$$

7. A SIMPLE EDITOR

This example has been taken from Bergstra & Klop [1]. Let $A = \{a_1,\ldots,a_n\}$ be an alphabet of symbols. Consider the following signature:

Σ_F

\$:	F
	Edf
	E
IF:	*: F × F → F
	edobj: F × F → Edf
¢:	ε ∈ F
	a ∈ F (all a ∈ A)
	⊥ ∈ E
	OK ∈ E

with equations

E_F

x*ε = x
ε*x = x
(x*y)*z = x*(y*z)

We use the initial algebra $T_I(\Sigma_F,E_F)$ as data space. With edobj(x,y) we denote a text x*y which is being edited with the cursor between x and y.

The following set of rules T_9 presents an object-oriented specification of an editor. Here it is assumed that there are some means to inspect the object being edited; i.e. the fact that the user is watching the string being edited, is not explicitly modeled by these transformation rules. A possibility for modeling this would be to output x*_*y whenever edobj(x,y) is formed, where '_' is some new symbol denoting the cursor (by putting x*_*y in the lower-righthand corner of the appropriate rule).

T_9

editor	$\left[\begin{array}{c\|c} & x \\ \hline \text{edobj}(\varepsilon,x) & \text{OK} \end{array}\right]$		$T_{9,1}$
quit	$\left[\begin{array}{c\|c} \text{edobj}(x,y) & \\ \hline & x*y \end{array}\right]$		$T_{9,2}$
left	$\left[\begin{array}{c\|c} \text{edobj}(\varepsilon,y) & \\ \hline \text{edobj}(\varepsilon,y) & \perp \end{array}\right]$		$T_{9,3}$
left	$\left[\begin{array}{c\|c} \text{edobj}(x*a,y) & \\ \hline \text{edobj}(x,a*y) & \end{array}\right]$	$(a \in A)$	$T_{9,4,a}$
right	$\left[\begin{array}{c\|c} \text{edobj}(x,\varepsilon) & \\ \hline \text{edobj}(x,\varepsilon) & \perp \end{array}\right]$		$T_{9,5}$
right	$\left[\begin{array}{c\|c} \text{edobj}(x,a*y) & \\ \hline \text{edobj}(x*a,y) & \end{array}\right]$	$(a \in A)$	$T_{9,6,a}$
delete	$\left[\begin{array}{c\|c} \text{edobj}(x,a*y) & \\ \hline \text{edobj}(x,y) & \end{array}\right]$	$(a \in A)$	$T_{9,7,a}$
delete	$\left[\begin{array}{c\|c} \text{edobj}(x,\varepsilon) & \\ \hline \text{edobj}(x,\varepsilon) & \perp \end{array}\right]$		$T_{9,8}$
insert	$\left[\begin{array}{c\|c} \text{edobj}(x,y) & a \\ \hline \text{edobj}(x*a,y) & \end{array}\right]$	$(a \in A)$	$T_{9,9,a}$

Taking care that at most one edobj is active at any time this will work. Note that $T_{9,3-9}$ constitute the heart of the matter. These rules describe the editing activities proper.

The next step is to describe a storage and retrieval mechanism for files. Consider the following signature:

Σ_{FSR}

$:	FD	(file directory)
	F	(texts/files)
	FN	(file names)
	P	(pairs)
	B	(booleans)
IF:	present: FN × FD → FD	(introduction of name)
	absent: FN × FD → FD	(deletion of name)
	contents: FN × F × FD → FD	(constructor of the file directories)
	pair: FN × FD → P	
	*: F × F → F	(concatenation on files)
	$\bar{*}$: FN × FN → FN	(concatenation on names)
	eq: FN × FN → B	(equality test on names)
¢:	T ε B	(true)
	F ε B	(false)
	Ø ε FD	(empty structure)
	a1,...,an ε F	(alphabet for file)
	b1,...,bm ε FN	(alphabet for names)
	ε ε F	
	$\bar{\varepsilon}$ ε FN	
Variables:	x,y,z ε F	
	u,v,w ε FN	
	X ε FD	

(Conditional) equations:

E_{FSR}

$$(x * y) * z = x * (y * z)$$

$$x * \varepsilon = x$$

$$\varepsilon * x = x$$

$$u \bar{*} (v \bar{*} w) = (u \bar{*} v) \bar{*} w$$

$$u \bar{*} \bar{\varepsilon} = u$$

$$\bar{\varepsilon} \bar{*} u = u$$

$$eq(\bar{\varepsilon},\bar{\varepsilon}) = T$$

$$eq(bi \bar{*} x,\ bi \bar{*} y) = eq(x,y) \qquad (i \varepsilon \{1,...,m\})$$

$$
\begin{array}{|l}
eq(bi \bar{*} x, bj \bar{*} y) = F \qquad (i \neq j,\ i,j \in \{1,\ldots,m\}) \\
eq(\bar{\epsilon}, bi \bar{*} x) = F \qquad (i \in \{1,\ldots,m\}) \\
eq(bi \bar{*} x, \bar{\epsilon}) = F \qquad (i \in \{1,\ldots,m\}) \\
contents(u,x,contents(u,y,X)) = contents(u,x,X) \\
eq(u,v) = F \rightarrow contents(u,x,contents(v,y,X)) = \\
\qquad\qquad contents(v,y,contents(u,x,X)) \\
present(u,\emptyset) = contents(u,\epsilon,\emptyset) \\
present(u,contents(u,x,X)) = contents(u,x,X) \\
eq(u,v) = F \rightarrow present(u,contents(v,x,X)) = \\
\qquad\qquad contents(v,x,present(u,X)) \\
absent(u,\emptyset) = \emptyset \\
absent(u,contents(u,x,X)) = absent(u,X) \\
eq(u,v) = F \rightarrow absent(u,contents(v,x,X)) = \\
\qquad\qquad contents(v,x,absent(u,X))
\end{array}
$$

The initial algebra $T_I(\Sigma_{FSR}, E_{FSR})$ is an appropriate data space for the permanent environment of the editor. Working in

$$T_I((\Sigma_{FSR}, E_{FSR}) \cup (\Sigma_F, E_F))$$

we can specify the system as follows (with $\{\emptyset\}$ as an initial configuration):

T_{10}

$$
\begin{array}{lll}
\underline{\text{introduce}} & \left[\begin{array}{l|l} absent(u,X) & u \\ \hline contents(u,\epsilon,X) & OK \end{array}\right] & T_{10,1} \\[2ex]
\underline{\text{introduce}} & \left[\begin{array}{l|l} present(u,X) & u \\ \hline present(u,X) & \perp \end{array}\right] & T_{10,2} \\[2ex]
\underline{\text{skip}} & \left[\begin{array}{l|l} present(u,X) & u \\ \hline absent(u,X) & OK \end{array}\right] & T_{10,3} \\[2ex]
\underline{\text{skip}} & \left[\begin{array}{l|l} absent(u,X) & u \\ \hline absent(u,X) & \perp \end{array}\right] & T_{10,4} \\[2ex]
\underline{\text{edit}} & \left[\begin{array}{l|l} contents(u,x,X) & u \\ \hline edobj(\epsilon,x),pair(u,X) & OK \end{array}\right] & T_{10,5}
\end{array}
$$

$$\underline{\text{edit}}\ \left\{\begin{array}{l|l} \text{absent(u,X)} & \text{u} \\ \hline \text{absent(u,X)} & \perp \end{array}\right\} \qquad T_{10,6}$$

$$\underline{\text{save}}\ \left\{\begin{array}{l|l} \text{edobj(x,y), pair(u,X)} & \\ \hline \text{contents(u, x * y, X)} & \end{array}\right\} \qquad T_{10,7}$$

(plus:) $T_{9,3\text{-}9}$

8. A MULTI-USER ENVIRONMENT FOR THE SIMPLE EDITOR

We now consider the following organisation:

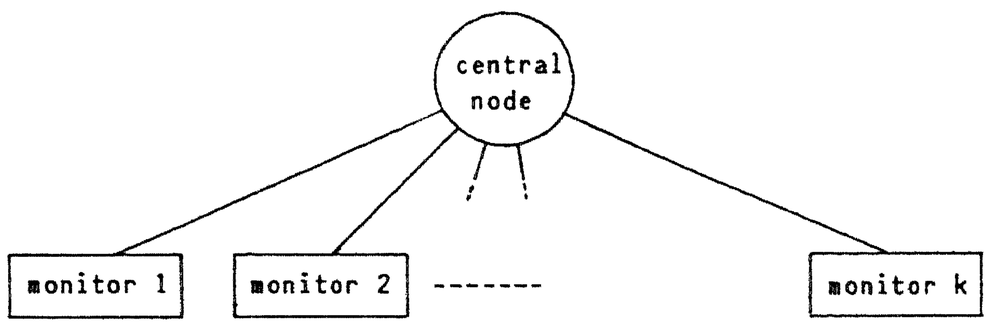


Figure 3.

At monitor k edit sessions act on an object edobj(k,x,y). A user must log in at a terminal with a user name which should be known to the system (by having been introduced at the central node). Each user name is also the index of a file in the permanent central file directory. This file is updated after each edit session.

As before we start with a signature and a specification for the data space. Like in example 7 we proceed in two phases. The central file directory is introduced in the second phase.

First phase.

Σ_{KME}

\$:	F	(files)
	Edf	(files being edited)
	MN	(monitor names)
	AMO	(active monitor objects)
	PMO	(passive monitor objects)
	B	(booleans)
	UN	(user names)
	E	(signals)

$$
\begin{array}{|ll|l}
\mathbb{F}: & *: F \times F \to F & \\
 & \bar{*}: UN \times UN \to UN & \\
 & \text{edobj}: MN \times F \times F \to Edf & \\
 & \text{amo}: MN \times UN \to AMO & \\
 & \text{pmo}: MN \to PMO & \\
 & \text{eq}: UN \times UN \to B & \\
 & & \\
\mathbb{C}: & T \in B & \\
 & F \in B & \\
 & \epsilon \in F & \\
 & a1,\ldots,an \in F & \\
 & \bar{\epsilon} \in UN & \\
 & b1,\ldots,bm \in UN & \\
 & 1,\ldots,k \in MN & \\
 & \perp \in E & \\
 & OK \in E & \\
 & & \\
\text{Variables:} & x,y,z \in F & \\
 & u,v,w \in UN & \\
 & k \in MN & \\
 & & \\
E_{KME} & (x * y) * z = x * (y * z) & \\
 & x * \epsilon = x & \\
 & \epsilon * x = x & \\
 & u \bar{*} (v \bar{*} w) = (u \bar{*} v) \bar{*} w & \\
 & u \bar{*} \bar{\epsilon} = u & \\
 & \bar{\epsilon} \bar{*} u = u & \\
 & \text{eq}(\bar{\epsilon},\bar{\epsilon}) = T & \\
 & \text{eq}(bi \bar{*} x,\ bi \bar{*} y) = \text{eq}(x,y) & (i \in \{1,\ldots,m\}) \\
 & \text{eq}(bi \bar{*} x,\ bj \bar{*} y) = F & (i \neq j,\ i,j \in \{1,\ldots,m\}) \\
 & \text{eq}(\bar{\epsilon},\ bi \bar{*} x) = F & (i \in \{1,\ldots,m\}) \\
 & \text{eq}(bi \bar{*} x,\ \bar{\epsilon}) = F & (i \in \{1,\ldots,m\})
\end{array}
$$

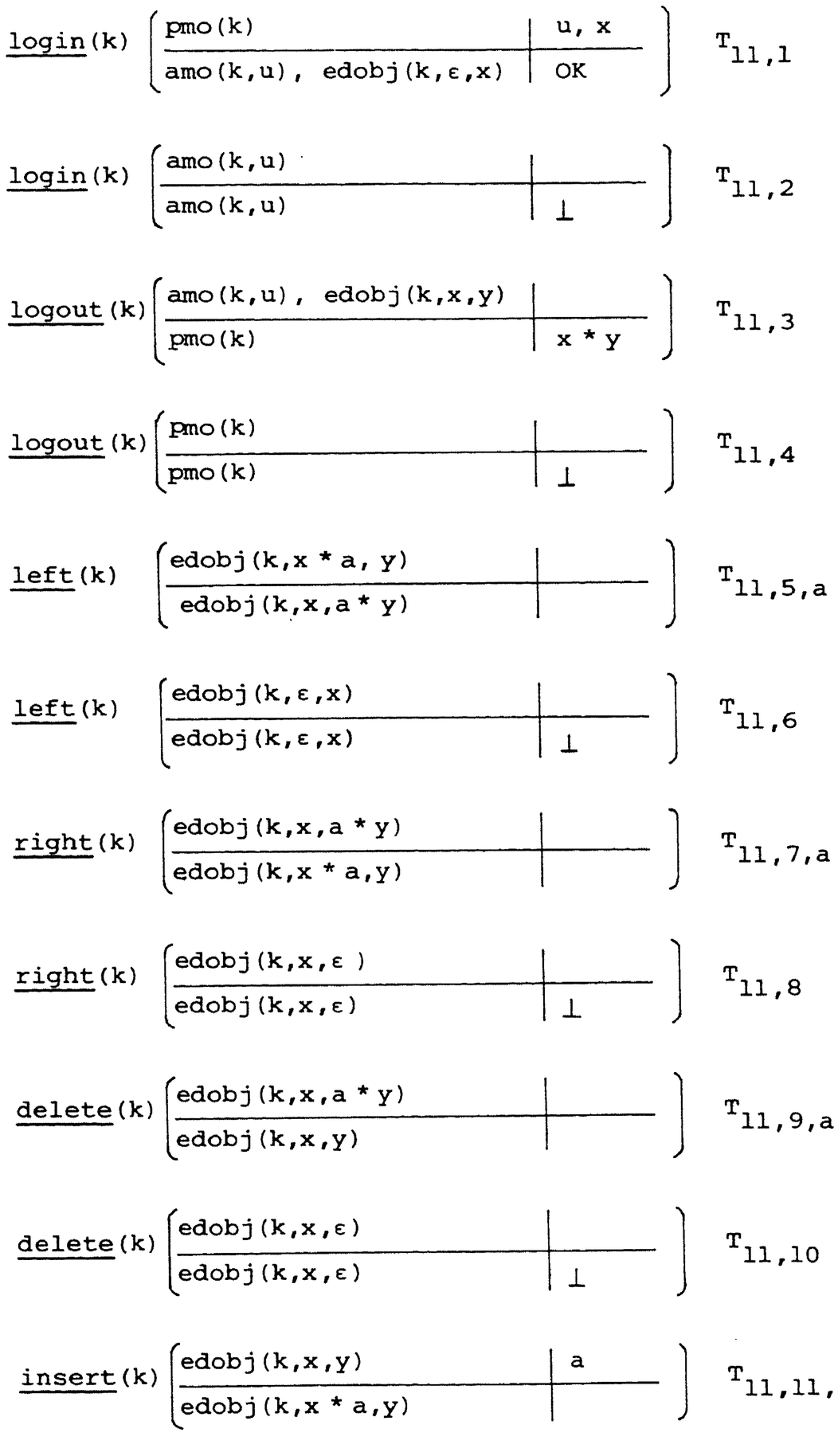
As before we work in $T_I(\Sigma_{KME}, E_{KME})$. As initial configuration we assume

$\{pmo(1),\ldots,pmo(k)\}$.

The first system description is T_{11}. The transition rules $T_{11,4\text{-}10}$ describe

the actual working of the editor. The other rules will be replaced in the second phase.

T_{11}



Notice that the monitor objects prevent two or more users from being logged in at the same monitor simultaneously.

Second phase.

In the second phase we add a central file directory for maintaining user names and for the storage and retrieval of each user's own file.

We need a new signature:

Σ_{FD}

$
\begin{array}{ll}
\$: & F \\
 & UN \\
 & FD \\
 & B \\
\mathbb{F}: & known: UN \times FD \to FD \\
 & unknown: UN \times FD \to FD \\
 & active: UN \times FD \to FD \\
 & silent: UN \times F \times FD \to FD \\
 & eq: UN \times UN \to B \\
\text{¢}: & T \in B \\
 & F \in B \\
 & \emptyset \in FD \\
Variables: & x,y,z \in F \\
 & u,v,w \in UN \\
 & X,Y,Z \in FD
\end{array}
$

E_{FD}

$
\begin{array}{l}
active(u, active(u,X)) = active(u,X) \\
active(u, active(v,X)) = active(v, active(u,X)) \\
active(u, silent(u,x,X)) = active(u,X) \\
eq(u,v) = F \to active(u, silent(v,x,X)) = silent(v,x, active(v,X)) \\
silent(u,x, active(u,X)) = silent(u,x,X) \\
silent(u,x, silent(u,y,X)) = silent(u,x,X) \\
eq(u,v) = F \to silent(u,x, silent(v,y,X)) = silent(v,y,silent(u,x,X)) \\
known(v,\emptyset) = silent(v,\epsilon,\emptyset) \\
known(u,active(u,X)) = active(u,X)
\end{array}
$

known(u, silent(u,x,X)) = silent(u,x,X)

eq(u,v) = F → known(u, active(v,X)) = active(v, known(u,X))

eq(u,v) = F → known(u,silent(v,x,X)) = silent(v,x,known(u,x,X))

unknown(u,∅) = ∅

unknown(v, active(u,X)) = unknown(u,X)

unknown(u, silent(u,x,X)) = unknown(u,X)

eq(u,v) = F → unknown(u, active(v,X)) = active(v, unknown(u,X))

eq(u,v) = F → unknown(u, silent(v,x,X)) = silent(v,x, unknown(u,X))

Now let

$$\Sigma_{KME}^{FD} = \Sigma_{KME} \cup \Sigma_{FD}$$

and

$$E_{KME}^{FD} = E_{KME} \cup E_{FD}.$$

We will work in the data space $T_I(\Sigma_{KME}^{FD}, E_{KME}^{FD})$.

Comment. Some remarks about E_{FD} may be in order. Let Z be the "current file directory". If Z = active(u,X), then this expresses that a user with name u is active on some monitor. If Z = known(u,X) this expresses that user name u is known to Z. Similarly if Z = unknown(u,X) this expresses that u is not known to Z. Finally, Z = silent(u,x,X) expresses the fact that the user with name u is not active and that his (her) file is presently containing the text x.

We can now present example T_{12}: a multi-user environment for the simple editor. The system T_{12} contains $T_{11,4\text{-}10}$ (the standard editing operations) and in addition the following transformation rules:

T_{12}

$$\underline{\text{introduce}} \left(\frac{\text{unknown}(u,X) \mid u}{\text{silent}(u,\varepsilon,X) \mid} \right) \qquad T_{12,1}$$

$$\underline{\text{introduce}} \left(\frac{\text{known}(u,X) \mid}{\text{known}(u,X) \mid \perp} \right) \qquad T_{12,2}$$

$$\underline{\text{omit}}\left\{\begin{array}{l|l}\text{known}(u,X) & u\\ \hline \text{unknown}(u,X) & \end{array}\right\} \qquad T_{12,3}$$

$$\underline{\text{omit}}\left\{\begin{array}{l|l}\text{unknown}(u,X) & \\ \hline \text{unknown}(u,X) & \perp\end{array}\right\} \qquad T_{12,4}$$

$$\underline{\text{login}}(k)\left\{\begin{array}{l|l}\text{pmo}(k),\ \text{silent}(u,x,X) & u\\ \hline \text{amo}(k,u),\ \text{edobj}(k,\varepsilon,x),\ \text{active}(u,X) & \text{OK}\end{array}\right\} \qquad T_{12,5}$$

$$\underline{\text{login}}(k)\left\{\begin{array}{l|l}\text{active}(u,X) & u\\ \hline \text{active}(u,X) & \perp\end{array}\right\} \qquad T_{12,6}$$

$$\underline{\text{login}}(k)\left\{\begin{array}{l|l}\text{unknown}(u,X) & u\\ \hline \text{unknown}(u,X) & \perp\end{array}\right\} \qquad T_{12,7}$$

$$\underline{\text{login}}(k)\left\{\begin{array}{l|l}\text{amo}(k,v) & u\\ \hline \text{amo}(k,v) & \perp\end{array}\right\} \qquad T_{12,8}$$

$$\underline{\text{logout}}(k)\left\{\begin{array}{l|l}\text{amo}(k,u),\ \text{edobj}(k,x,y),\ X & \\ \hline \text{pmo}(k),\ \text{silent}(u,\ x * y,\ X) & \end{array}\right\} \qquad T_{12,9}$$

$$\underline{\text{logout}}(k)\left\{\begin{array}{l|l}\text{pmo}(k) & \\ \hline \text{pmo}(k) & \perp\end{array}\right\} \qquad T_{12,10}$$

$$\underline{\text{display}}(k)\left\{\begin{array}{l|l}\text{edobj}(k,x,y) & \\ \hline \text{edobj}(k,x,y) & x * y\end{array}\right\} \qquad T_{12,11}$$

Remarks. (a) Notice that a user can only be omitted when not active. An active user could logout as if nothing has happened and thereafter his or her name would be known to the system again.

(b) It is entirely feasible to augment this specification with a mechanism for passwords or other protection mechanisms.

9. SEMANTICAL CONSIDERATIONS

In Section 3 we have given an informal explanation of the semantics of transformation rules. In this section we will elaborate that explanation, in particular, concerning the mechanism by which the *transformation rules* generate the *transformation steps*

$$C \xrightarrow[R]{} C'$$

where C,C' are configurations, i.e. multisets of objects.

Let $A \in Alg(\Sigma)$ be a given data space; then we may write a transformation rule, written above as

$$r(\vec{v})\left(\begin{array}{c|c} X & V \\ \hline Y & W \end{array}\right)$$

in simplified notation as follows:

$$r(\vec{v},V,W): X \longrightarrow Y.$$

Here $\vec{v} = v1,\ldots,vn$ are Σ-terms and V,W,X,Y are finite multisets of Σ-terms. These terms may contain free variables and matching works as usual in term rewrite rules. X,Y themselves are not yet configurations of objects in A; they become so after dividing out therm equality in A. Further, V,W denote multisets of input and output values - properly speaking this is again true after dividing out term equality. The v1,...,vn are parameters of the rule names.

Let us introduce a constant ϕ for the empty configuration and an operator $\cup$ for the union of configurations. The following axioms are obviously valid:

$$X \cup Y = Y \cup X$$
$$X \cup \phi = X$$
$$(X \cup Y) \cup Z = X \cup (Y \cup Z).$$

Note that $\cup$ is represented in process algebra [2] by $\|$, the merge operator. This connection is not quite smooth: there seems to be a difference in level of abstraction between process algebra and behavioural specification via transformation rules.

The propagation of transformations through larger configurations is as follows:

$$\frac{r(\vec{v},V,W)\colon X \longrightarrow Y}{r(\vec{v},V,W)\colon X \cup Z \longrightarrow Y \cup Z}\ .$$

Writing $[\![t]\!]$ for the interpretation of the Σ-term t in the data space A, and $[\![X]\!] = \{[\![t]\!] \mid t \in X\}$ for the multiset of objects in A denoted by the multiset of Σ-terms X, we can now state more precisely what a transformation step is:

if $R = r(\vec{v},V,W)\colon X \cup Z \longrightarrow Y \cup Z$ is obtained from the instance $r(\vec{v},V,W)\colon X \longrightarrow Y$ of some transformation rule, then R allows the *transformation step* of configuration $C = [\![X \cup Z]\!]$ to $C' = [\![Y \cup Z]\!]$; notation: $C \xrightarrow[R]{} C'$. (See Figure 4.)

Such transformation steps can be activated sequentially. In fact, the situation is similar to the case of *term rewriting modulo some given congruence* (apart from the multiset feature).

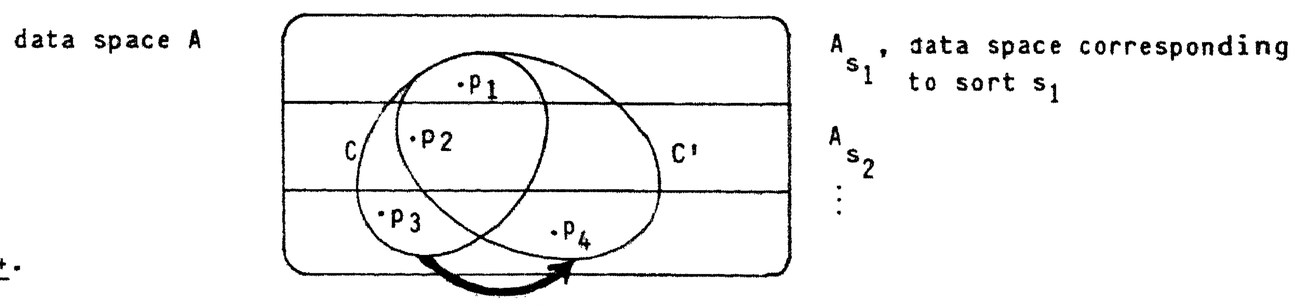


Figure 4.

In other words, the transformation step $C \xrightarrow[R]{} C'$ where $C = \{p_1, p_2, \ldots\}$ is obtained by choosing a *particular representation* of C, e.g. $\{t_1, t_2, \ldots\}$ such that $[\![t_i]\!] = p_i$, and applying some transformation rule on it as explained, to transform this representation into another (of C').

In an intuitive sense, such a representation of a configuration C can be considered as an *aspect* of C. E.g. in the last example (T_{12}), known(v,Ø) is the file directory X = Ø revealing as an aspect that it knows user name v (usually such a fact would have type boolean, here it is of type file directory). And in silent(v,ε,Ø) the same X = Ø reveals another aspect. The transformation rules, then, operate on such aspects.

10. CONCLUDING REMARKS

We feel that the object-oriented notation explained above captures at least a useful fragment of "object-oriented thinking". Clearly we have to pay a

price in terms of manageability of the transformation rules. One can, in view of Section 9, add ϕ and $\cup$, and view the transformation rules as ordinary rewrite rules. From the point of view of algebraic specifications, adding ϕ, $\cup$ and, in general, a type of configurations, leads to the problem that configurations have no fixed type. Any object can be an element of a configuration. In fact, ϕ and $\cup$ are polymorphic operations and this explains their flexibility which is vital for modular and incremental systems design.

REFERENCES

[1] BERGSTRA, J.A. & J.W. KLOP, *Algebraisch programmeren*, (in Dutch), contained in the lecture notes for the PAO course on software engineering, Centrum voor Wiskunde en Informatica, Amsterdam 1984.

[2] BERGSTRA, J.A. & J.W. KLOP, *Process algebra for communication and mutual exclusion*, Report IW218/83, Mathematisch Centrum, Amsterdam 1983.

[3] COHEN, A.T., *Data abstraction, data encapsulation and object-oriented programming*, Sigplan Notices, Vol.19, No.1 (1984).

[4] COX, B.J., *The object-oriented precompiler*, Sigplan Notices, Vol.18, No.1 (1983).

[5] GOGUEN, J.A. & J. MESEGUER, *An initiality primer*, to appear in: Application of Algebra to Language Definition and Compilation (eds.: M. Nivat and J. Reynolds), North-Holland 1983.

[6] JAMSA, K.A., *Object-oriented design versus structured design, a students perspective*, Software Engineering notes, Vol.9, No.1 (1984)

[7] JONKERS, H.B.M., *On the design of an object-oriented design language*, paper presented at the Colloquium 'Van Specificatie tot Implementatie', Centrum voor Wiskunde en Informatica, Amsterdam 1983.

[8] KUTZLER, B. & F. LICHTENBERGER, *Bibliography on abstract data types*, Springer Informatik-Fachberichte, No.68, 1983.

[9] MACLENNAN, B.J., *Values and objects in programming languages*, Sigplan Notices, Vol.17, No.2 (1982).

[10] PLOTKIN, G.D., *A structural approach to operational semantics*, Report Daimi FN-19, Computer Science Dept., Aarhus University, Denmark 1981.