Technical Report

UCAM-CL-TR-125
ISSN 1476-2986

Number 125

An operational semantics for Occam

Juanito Camilleri

February 1988

ISSN 1476-2986

An Operational Semantics for occam

Juanito Camilleri
University of Cambridge
Computer Laboratory
New Museums Site
Pembroke Street
Cambridge CB2 3QG

Abstract

occam is a programming language designed to support concurrent applications, especially those implemented on networks of communicating processors. The aim of this paper is to formulate the meaning of the language constructs of occam by semantic definitions which are intended as a direct formalisation of the natural language descriptions usually found in programming language manuals [Inmos 3]. This is done by defining a syntax directed transition system, where the transitions associated to a phrase are a function of the transitions associated to its components. This method is by no means novel. The concepts used here were introduced in [Plotkin 8] and are applied in [Plotkin 9] where an operational semantics for CSP [Hoare 2] was presented. The operational semantics for a subset of Ada is defined in [Li 6], where tasking and exception handling are modelled. For simplicity only a subset of occam is defined. Timing, priority, replicators and BYTE subscription are omitted. Other features of occam which deal with the association of components of an occam program with a set of physical resources (i.e. configurations) are also omitted since they do not effect the semantic interpretation of a program.

Contents

1 The Syntactic Categories.

1.	numbers	**Num**	ranged over by	**N.**
2.	variable identifiers	**Id**	" " " " "	**X.**
3.	locations	**Loc**	" " " " "	**l.**
4.	channel identifiers	**Chid**	" " " " "	**chi.**
5.	channels	**Chan**	" " " " "	**ch.**
6.	abstractions	**Abs**	" " " " "	**abs.**
7.	integer expressions	**Iexp**	" " " " "	**a.**
8.	boolean expressions	**Bexp**	" " " " "	**b.**
9.	conditional commands	**Ccom**	" " " " "	**cc.**
10.	guarded commands	**Gcom**	" " " " "	**gc.**
11.	commands	**Com**	" " " " "	**c.**
12.	declarations	**Dec**	" " " " "	**d.**
13.	actual parameters	**Acts**	" " " " "	**acts.**
14.	formal parameters	**Forms**	" " " " "	**forms.**

2 The Syntax.

We assume that the syntactic categories **Num, Id, Chid** are given. **Id** is an infinite set of variable identifiers while **Chid** is an infinite set of channel identifiers. Locations (**Loc**) can be thought of as "abstract addresses". Channels (**Chan**) can be viewed as "abstract channels" via which communication takes place. We do not want to commit ourselves to any machine architecture, but, only to the needed intuitive properties. A better way to think about locations or channels, is as entities which have a lifetime (or extent); they are created by a declaration and they continue to exist throughout the execution sequence, unless their existence is terminated by block exit. Finally an abstraction (**abs**) takes the form λforms.c which is a syntactic representation of a set of formal parameters together with the body of the abstraction c.

Notation

Suppose S is some syntactic class with typical element s then $\underline{\underline{s}}$ denotes a finite list (possibly empty) of elements of S. Therefore $\underline{\underline{s}} = (s_1, \ldots, s_{\underline{\underline{n}}})$ where for all $1 \leqq i \leqq n$, $s_i \in S$. Note ( ) denotes the empty list.

The following rules illustrate the nature of the syntactic categories (7-14).

Iexp

a ::= N | X | a + a | a - a | a * a | ...

Bexp

b ::= TRUE | FALSE | a = a | b OR b | NOT b | b AND b | a <= a | a >= a | ...

Ccom

cc ::= b $\rightarrow$ c | cc [] cc

Gcom

gc ::= chi ? X $\rightarrow$ c | gc [] gc

Acts

acts ::= $\underline{\underline{a}}$, $\underline{\underline{X}}$, $\underline{\underline{\text{chi}}}$

Com

c ::= skip | stop | X := a | c ; c | c || c | IF cc | WHILE b c | ALT gc |

chi ? X | chi ! a | d ; c | X(acts)

Dec

d ::= DEF X = a | VAR X | CHAN chi | PROC X (forms) c | d ; d

Forms

forms ::= VALUE $\underline{\underline{X}}$, VAR $\underline{\underline{X}}$, CHAN $\underline{\underline{\text{chi}}}$

For ease of presentation the **occam** syntax has been altered in this paper. For example:

		SEQ				PAR
$c_0 ; c_1$	<u>denotes</u>	c_0	<u>and</u>	$c_0 \| c_1$	<u>denotes</u>	c_0
		c_1				c_1

We are replacing n-ary combinators with binary ones and have altered the syntax of a list of parameters so that VALUE, VAR and CHAN parameters occur in that order.

3 Definition of a transition system

<u>Definition 1</u>: A transition system is a triple $<\Gamma, T, -->>$ where:

Γ is the set of configurations.

$T \subseteq \Gamma$ is the set of terminal configurations.

$--> \subseteq \Gamma^2$ is the transition relation such that $\forall \beta \in T\ \forall \beta' \in \Gamma\ \neg(\beta --> \beta')$.

4 Static Semantics

The aim of the static semantics is to distinguish the well-formed commands from those commands which are not well-formed. A command is <u>not</u> well-formed if:-

(a) it consists of two processes running in parallel such that they both can write to a common variable. For example $y := 10 \parallel y := 5$. (None the less two processes can read from a common variable).

(b) there are two communicating processes which do not conform with the concept that a channel is a unidirectional and indivisible means of communication. That is

   i. A channel should not be used for input and output by the same process.

   ii. The same channel cannot act as an input (or output) to more than one process.

(c) it contains a call to a process such that the actual parameters do not conform in nature and number to the formal parameters.

Since a call to a process must conform to its declaration we require a command to be well-formed relative to some static environment SEnv which associates process names with their declarations as defined hereafter. Let

$\mathrm{SEnv}_I = I \longrightarrow \mathbf{Abs}$ where $I \subseteq_{fin} \mathbf{Id}$ then we can define

$\mathrm{SEnv} = \sum_{I \subseteq_{fin} \mathrm{Id}} \mathrm{SEnv}_I \quad (\simeq \mathbf{Id} \longrightarrow_{fin} \mathbf{Abs})$

We shall use α to range over SEnv.

We need to define a transition system $<\Gamma_{stat}, T_{stat}, \dashrightarrow_{stat}>$ which elaborates the static environment whenever a procedure declaration is encountered. We extend **Dec** by adding the production rule:

$$d ::= \alpha$$

What this means is that the abstract syntax of <u>declaration configurations</u> includes static environments; it does not mean that the abstract syntax of <u>declarations</u> does so.

$\Gamma_{stat} = \{<d>\}$

$T_{stat} = \{<\alpha>\}$

$\dashrightarrow_{stat} \subseteq \Gamma_{stat} \times \Gamma_{stat}$

Rule

$\alpha \vdash \; < \underline{\text{PROC}}\ X\ (\text{forms})\ c > \; \dashrightarrow_{stat} \; < \alpha[X \mapsto \lambda \text{forms}.c] >$

The above rule is read — Given the static environment α the definition of the abstraction PROC X (forms) c is well-formed and yields the augmented environment $\alpha[X \mapsto \lambda\text{forms.c}]$.

Let the property of being well-formed be denoted by $\vdash$.

For example $\alpha \vdash c$ means that c is well-formed relative to the static environment α.

Before defining $\vdash$ on the structure of the syntax let us define the following functions which will be required in the definition of $\vdash$.

RI(c)	—	is the set of read variables of command c.
WI(c)	—	is the set of write variables of command c.
INCH(c)	—	is the set of channel identifiers being used for input in c.
OUTCH(c)	—	is the set of channel identifiers being used for output in c.

We also need to formalise the meaning of actual parameters conforming in nature and number to formal parameters. Consider the command **X(acts)** such that $\alpha(X) = \lambda$**forms.c**. Note that **acts** is a list of actual parameters and **forms** is a list of formal parameters. These two lists should have the same length (say **n**). Then for all $1 \leq i \leq n$, the nature of $a_i \in$ **acts** must conform to the type expected by $f_i \in$ **forms**. We use $\models$ **acts** $\uparrow$ **forms** to denote that the actual parameters conform to the corresponding formal parameters as defined hereafter.

$\models () \uparrow ()$	where ( ) is the empty list.
$\models a \uparrow \text{VALUE } X'$	where a is an integer expression and X' is a VALUE parameter.
$\models X \uparrow \text{VAR } X''$	where X is an identifier and X'' is a VAR parameter.
$\models chi \uparrow \text{CHAN } chi'$	where chi is a channel identifier and chi' is a CHAN parameter.

$$\frac{\models a_1 \uparrow f_1 \qquad \models \underline{\underline{act}} \uparrow \underline{\underline{form}}}{\models (a_1, \underline{\underline{act}}) \uparrow (f_1, \underline{\underline{form}})}$$

The following is a definition of RI, WI, INCH, OUTCH by structural induction, expressed in tabular form, instead of using the format of rules, to keep the definitions concise.

For integer expressions

	n	X	a_0 op a_1
RI	$\varnothing$	{X}	$RI(a_0) \cup RI(a_1)$

where op $\in$ { + , - , * , ... }
WI(a) , INCH(a) , OUTCH(a) are all $\varnothing$.

For boolean expressions

	t	NOT b	b_0 bop b_1	a_0 relop a_1
RI	$\varnothing$	RI(b)	$RI(b_0) \cup RI(b_1)$	$RI(a_0) \cup RI(a_1)$

where t $\in$ { TRUE , FALSE }

relop $\in$ { = , <= , >= , ...} and bop $\in$ { AND , OR }

WI(a) , INCH(a) , OUTCH(a) are all $\varnothing$.

For conditional commands

	$b \rightarrow c$	cc_0 [] cc_1
RI	$RI(b) \cup RI(c)$	$RI(cc_0) \cup RI(cc_1)$
WI	WI(c)	$WI(cc_0) \cup WI(cc_1)$
INCH	INCH(c)	$INCH(cc_0) \cup INCH(cc_1)$
OUTCH	OUTCH(c)	$OUTCH(cc_0) \cup OUTCH(cc_1)$

For guarded commands

	$chi?X \rightarrow c$	$gc_0 [] gc_1$
RI	RI(c)	$RI(gc_0) \cup RI(gc_1)$
WI	$\{X\} \cup WI(c)$	$WI(gc_0) \cup WI(gc_1)$
INCH	$\{chi\} \cup INCH(c)$	$INCH(gc_0) \cup INCH(gc_1)$
OUTCH	OUTCH(c)	$OUTCH(gc_0) \cup OUTCH(gc_1)$

For commands

	skip	stop	chi ? X	chi ! a	$c_0 ; c_1$
RI	$\varnothing$	$\varnothing$	$\varnothing$	RI(a)	$RI(c_0) \cup RI(c_1)$
WI	$\varnothing$	$\varnothing$	$\{X\}$	$\varnothing$	$WI(c_0) \cup WI(c_1)$
INCH	$\varnothing$	$\varnothing$	$\{chi\}$	$\varnothing$	$INCH(c_0) \cup INCH(c_1)$
OUTCH	$\varnothing$	$\varnothing$	$\varnothing$	$\{chi\}$	$OUTCH(c_0) \cup OUTCH(c_1)$

	$c_0 \parallel c_1$	IF cc	X : = a
RI	$RI(c_0) \cup RI(c_1)$	RI(cc)	RI(a)
WI	$WI(c_0) \cup WI(c_1)$	WI(cc)	$\{X\}$
INCH	$INCH(c_0) \cup INCH(c_1)$	INCH(cc)	$\varnothing$
OUTCH	$OUTCH(c_0) \cup OUTCH(c_1)$	OUTCH(cc)	$\varnothing$

	WHILE b c	ALT gc	d ; c
RI	RI(b) ∪ RI(c)	RI(gc)	RI(c)
WI	WI(c)	WI(gc)	WI(c)
INCH	INCH(c)	INCH(gc)	INCH(c)
OUTCH	OUTCH(c)	OUTCH(gc)	OUTCH(c)

To define INCH(c), OUTCH(c), RI(c), WI(c) when c is a call to a process P:

Suppose $\alpha(P) = \lambda forms.c$ where the list of formal parameters

$$forms = \text{VALUE } \underline{\underline{X}}', \text{VAR } \underline{\underline{X}}'', \text{CHAN } \underline{\underline{chi}}'.$$

The call to the process P should take the form P(acts) where the list of actual parameters acts = $\underline{\underline{n}}$, $\underline{\underline{X}}$, $\underline{\underline{chi}}$ conforms in nature and number to the list of formal parameters (ie. $\models acts \uparrow forms$ as discussed previously).

	P ($\underline{\underline{a}}$, $\underline{\underline{X}}$, $\underline{\underline{chi}}$)
RI	RI(c [$\underline{\underline{X}}$ / $\underline{\underline{X}}''$])
WI	WI(c [$\underline{\underline{X}}$ / $\underline{\underline{X}}''$])
INCH	INCH(c [$\underline{\underline{chi}}$ / $\underline{\underline{chi}}'$])
OUTCH	OUTCH(c [$\underline{\underline{chi}}$ / $\underline{\underline{chi}}'$])

Finally we can define $\alpha \vdash c$ by structural induction.

We assume that $\alpha \vdash a$ and $\alpha \vdash b$ hold for any integer or boolean expression.

For conditional commands

$$\frac{\alpha \vdash c}{\alpha \vdash b \rightarrow c} \qquad\qquad \frac{\alpha \vdash cc_0 \qquad \alpha \vdash cc_1}{\alpha \vdash cc_0 \,[]\, cc_1}$$

For guarded commands

$$\frac{\alpha \vdash c}{\alpha \vdash chi?X \rightarrow c} \qquad\qquad \frac{\alpha \vdash gc_0 \qquad \alpha \vdash gc_1}{\alpha \vdash gc_0 \,[]\, gc_1}$$

For commands

$\alpha \vdash skip$ $\quad$ $\alpha \vdash stop$ $\quad$ $\alpha \vdash X ::= a$ $\quad$ $\alpha \vdash chi\,?X$ $\quad$ $\alpha \vdash chi\,!\,a$

$$\frac{\alpha \vdash cc}{\alpha \vdash IF\ cc} \qquad \frac{\alpha \vdash c}{\alpha \vdash WHILE\ b\ c} \qquad \frac{\alpha \vdash gc}{\alpha \vdash ALT\ gc} \qquad \frac{\alpha \vdash c_0 \qquad \alpha \vdash c_1}{\alpha \vdash c_0\ ;\ c_1}$$

$$\frac{\alpha \vdash c_0 \qquad \alpha \vdash c_1}{\alpha \vdash c_0 \parallel c_1}$$

if $(WI(c_0) \cup RI(c_0)) \cap WI(c_1) = \varnothing$
and $(WI(c_1) \cup RI(c_1)) \cap WI(c_0) = \varnothing$
and $INCH(c_0) \cap OUTCH(c_0) = \varnothing$
and $INCH(c_1) \cap OUTCH(c_1) = \varnothing$
and $INCH(c_0) \cap INCH(c_1) = \varnothing$
and $OUTCH(c_0) \cap OUTCH(c_1) = \varnothing$

$$\frac{\alpha \vdash d \dashrightarrow_{stat} \alpha' \qquad \alpha' \vdash c}{\alpha \vdash d\ ;\ c}$$

$$\frac{\alpha \vdash c}{\alpha \vdash X(acts)}$$

If $\alpha(X) = \lambda forms.c$ and $\models acts \uparrow forms$

5 Dynamic Semantics

Semantic Domains

Given the syntax and syntactic categories defined earlier the following semantic domains can be constructed to be used in the dynamic semantics. The first semantic domain represents that section of the environment which associates a finite set of identifiers with numbers, locations or abstractions. Let

$\mathrm{LEnv}_I = I \rightarrow (\mathbf{Num} + \mathbf{Loc} + \mathbf{Abs})$ where $I \subseteq_{fin} \mathbf{Id}$ then we can define

$$\mathrm{LEnv} = \sum_{I \subseteq_{fin} \mathrm{Id}} \mathrm{LEnv}_I \quad (\simeq \mathbf{Id} \rightarrow_{fin} (\mathbf{Num} + \mathbf{Loc} + \mathbf{Abs}))$$

We shall use ρ to range over LEnv.

The next semantic category to be introduced represents the remaining section of the environment which associates channel identifiers with abstract channels. Let

$\mathrm{CEnv}_{CH} = \mathbf{CH} \rightarrow \mathbf{Chan}$ where $\mathbf{CH} \subseteq_{fin} \mathbf{Chid}$ then we can define

$$\mathrm{CEnv} = \sum_{CH \subseteq_{fin} \mathrm{Chid}} \mathrm{CEnv}_{CH} \quad (\simeq \mathbf{Chid} \rightarrow_{fin} \mathbf{Chan})$$

We shall use γ to range over CEnv.

The semantic domain which represents the store is defined as follows:

$\mathrm{Stores}_L = \mathbf{L} \rightarrow \mathbf{Num}$ where $\mathbf{L} \subseteq_{fin} \mathbf{Loc}$ then we can define

$$\mathrm{Stores} = \sum_{L \subseteq_{fin} \mathrm{Loc}} \mathrm{Stores}_L \quad (\simeq \mathbf{Loc} \rightarrow_{fin} \mathbf{Num})$$

We shall use σ to range over Stores.

The semantic domains LEnv and CEnv give the notion of an environment when paired to form the semantic domain Env as used hereafter.

Let Env = (LEnv, CEnv) then Env is made up of:

1. the associations between a finite set of identifiers (found in LEnv), with either locations, numbers or abstractions.

2. the association between a finite set of channel identifiers (found in CEnv) with abstract channels.

Env is ranged over by (ρ, γ) with the understanding that ρ ranges over LEnv and γ ranges over CEnv.

In the above model of an environment the distinction between channel identifiers and variable identifiers is explicit (i.e channel identifiers and identifiers are syntactically distinguished). One can adopt a different model where any identifier falls under one class hence:

$$\text{Env} = \textbf{Id} \rightarrow_{\text{fin}} (\textbf{Num} + \textbf{Loc} + \textbf{Abs} + \textbf{Chan})$$

In this case an identifier must be "tagged" with a type (i.e whether it is associated with a channel or otherwise) and "type checking" is required.

Going back to the original model, in the environment (ρ, γ) an identifier X can be associated with :-

(a) a number — when X is defined to be constant. Therefore $\rho(X) = n$.

(b) a location — when X is defined to be a variable. Therefore $\rho(X) = l$. Given a store σ with domain L (denoted by $\sigma::L$) such that $l \in L$ then $\sigma(l) = n$, where n is the value held in location l. However if l is in the range of ρ, but, not in the domain of σ, then the dangling reference problem is encountered.

(c) an abstraction — when X is the name given to a process of the form PROC X (forms) c. That is $\rho(X)$ is denoted by λ forms.c.

On the other hand a channel identifier chi is associated with an abstract channel. That is $\gamma(\text{chi}) = \text{ch}$.

Before proceeding with the dynamic semantics it is useful to discuss the declaration of constants, variables, channels and abstractions at this stage. A declaration in **occam** is used to introduce an identifier for use in the current block. There are channel identifiers. These are introduced by a CHAN declaration and associate a channel identifier with an abstract channel. There are also identifiers which refer to locations (or constant values) which are introduced by a VAR (or DEF) declaration. Finally there are named processes which are introduced via a PROC declaration. When a new identifier X is declared within a block, it has scope only within the block. If X already exists in the environment outside the block of the new declaration then the latter declaration hides the former one. Hence a gap is created in the scope of the former declaration of X. We use the following notation to denote the above.

Notation

Let $\beta :: \mathrm{B}$ mean that β has domain B.

For any $\mathrm{B}_0, \mathrm{B}_1$ and $\beta_0 :: \mathrm{B}_0$, $\beta_1 :: \mathrm{B}_1$ we define $\beta = \beta_0 [\, \beta_1 \,] :: \mathrm{B}_0 \cup \mathrm{B}_1$ by:

$$\beta(\mathrm{X}) = \begin{cases} \beta_1(\mathrm{X}) & \text{if} \quad (\, \mathrm{X} \in \mathrm{B}_1 \,) \\ \beta_0(\mathrm{X}) & \text{if} \quad (\, \mathrm{X} \in \mathrm{B}_0 \setminus \mathrm{B}_1 \,) \end{cases}$$

Using the above definition we denote the updating of environment $(\rho_0, \gamma_0) :: (\mathrm{I}_0, \mathrm{C}_0)$ by $(\rho_1, \gamma_1) :: (\mathrm{I}_1, \mathrm{C}_1)$ as $(\rho_0 [\, \rho_1 \,], \gamma_0 [\, \gamma_1 \,]) :: ((\, \mathrm{I}_0 \cup \mathrm{I}_1 \,), (\mathrm{C}_0 \cup \mathrm{C}_1 \,))$

For each syntactic category we define a transition system.The transition relations are relative to the environment. Therefore if s is an element of some syntactic category we write

$$(\rho, \gamma) \vdash \; <\mathbf{s}, \sigma> \quad \dashrightarrow_{\mathrm{s}} \quad <\mathbf{s}', \sigma'>$$

and read — In a given environment (ρ, γ) one step in the execution of **s** in store σ yields **s'** and store σ'. For the purpose of keeping the rules concise, certain rules deal with more than one possible outcome of an evaluation. For example

$$(\rho, \gamma) \vdash <\mathbf{s}, \sigma> \quad \dashrightarrow_{\mathrm{s}} \quad <\mathbf{s}', \sigma'> \;|\; <\mathbf{s}'', \sigma''>$$

is read — In environment (ρ, γ) one step in the evaluation of **s** in store σ can either yield **s'** and store σ', or, **s''** and store σ''.

Note : In any of the transition rules that follow, failure denotes failure to satisfy a boolean condition while abortion denotes the explicit failure to reach a final state (i.e. the explicit non-termination of a construct which leads to the abortion of a program)

Integer Expressions

For integer expressions we have the transition system $<\Gamma_a, T_a, \dashrightarrow_a>$ such that

$\Gamma_a = \{<a, \sigma>\} \cup Z$

$T_a = Z$

Rules

Evaluation of numbers.

$(\rho, \gamma) \vdash <N, \sigma> \dashrightarrow_a n$ where n is the number representing the piece of syntax N.

Evaluation of identifiers.

$(\rho, \gamma) \vdash <X, \sigma> \dashrightarrow_a \rho(X)$ if $\rho(X)$ is a number or abstraction

$(\rho, \gamma) \vdash <X, \sigma> \dashrightarrow_a \sigma(\rho(X))$ if $\rho(X) = l$ and $\sigma(l)$ is a number

Evaluation of binary operations

Sum

$$\frac{(\rho, \gamma) \vdash <a_0, \sigma> \dashrightarrow_a <a_0', \sigma>}{(\rho, \gamma) \vdash <a_0 + a_1, \sigma> \dashrightarrow_a <a_0' + a_1, \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <a_1, \sigma> \dashrightarrow_a <a_1', \sigma>}{(\rho, \gamma) \vdash <n + a_1, \sigma> \dashrightarrow_a <n + a_1', \sigma>}$$

$(\rho, \gamma) \vdash <n + m, \sigma> \dashrightarrow_a n + m$

Similarly for - * e.t.c

Lists of Integer Expressions

For lists of integer expressions we have the transition system $<\Gamma_{\underline{\underline{a}}}, T_{\underline{\underline{a}}}, \dashrightarrow_{\underline{\underline{a}}} >$ such that

$\Gamma_{\underline{\underline{a}}} = \{<\underline{\underline{a}}, \sigma>\} \cup \underline{\underline{Z}}$

$T_{\underline{\underline{a}}} = \underline{\underline{Z}}$

Rules

$$\frac{(\rho, \gamma) \vdash <a, \sigma> \dashrightarrow_{a} <a', \sigma> \quad | \quad n}{(\rho, \gamma) \vdash <(a, \underline{\underline{a}}), \sigma> \dashrightarrow_{\underline{\underline{a}}} <(a', \underline{\underline{a}}), \sigma> \mid (n, <\underline{\underline{a}}, \sigma>)}$$

$$\frac{(\rho, \gamma) \vdash <\underline{\underline{a}}, \sigma> \dashrightarrow_{\underline{\underline{a}}} \underline{\underline{n}}}{(\rho, \gamma) \vdash <(\underline{\underline{a}}, ()), \sigma> \dashrightarrow_{\underline{\underline{a}}} \underline{\underline{n}}}$$

Boolean Expressions

For boolean expressions we have the transition system $<\Gamma_b, T_b, \dashrightarrow_b >$ such that

$\Gamma_b = \{<b, \sigma>\} \cup \{\text{true}, \text{false}\}$

$T_b = \{\text{true}, \text{false}\}$

Rules

$(\rho, \gamma) \vdash <\text{TRUE}, \sigma> \dashrightarrow_b \text{true}$

$(\rho, \gamma) \vdash <\text{FALSE}, \sigma> \dashrightarrow_b \text{false}$

Relational operators

$$\frac{(\rho, \gamma) \vdash \langle a_0, \sigma\rangle \dashrightarrow_a \langle a_0', \sigma\rangle}{(\rho, \gamma) \vdash \langle a_0 = a_1, \sigma\rangle \dashrightarrow_b \langle a_0' = a_1, \sigma\rangle}$$

$$\frac{(\rho, \gamma) \vdash \langle a_1, \sigma\rangle \dashrightarrow_a \langle a_1', \sigma\rangle}{(\rho, \gamma) \vdash \langle m = a_1, \sigma\rangle \dashrightarrow_b \langle m = a_1', \sigma\rangle}$$

$(\rho, \gamma) \vdash \langle m = n, \sigma\rangle \dashrightarrow_b \text{true}$ if $m = n$

$(\rho, \gamma) \vdash \langle m = n, \sigma\rangle \dashrightarrow_b \text{false}$ if $m \neq n$

Similarly for $>=$ $<=$ e.t.c.

Boolean operators.

NOT

$$\frac{(\rho, \gamma) \vdash \langle b, \sigma\rangle \dashrightarrow_b \text{true} \mid \text{false} \mid \langle b', \sigma\rangle}{(\rho, \gamma) \vdash \langle \text{NOT } b, \sigma\rangle \dashrightarrow_b \text{false} \mid \text{true} \mid \langle \text{NOT } b', \sigma\rangle}$$

AND

$$\frac{(\rho, \gamma) \vdash \langle b_0, \sigma\rangle \dashrightarrow_b \text{false} \mid \text{true} \mid \langle b_0', \sigma\rangle}{(\rho, \gamma) \vdash \langle b_0 \text{ AND } b_1, \sigma\rangle \dashrightarrow_b \text{false} \mid \langle b_1, \sigma\rangle \mid \langle b_0' \text{ AND } b_1, \sigma\rangle}$$

OR

$$\frac{(\rho, \gamma) \vdash \langle b_0, \sigma\rangle \dashrightarrow_b \text{true} \mid \text{false} \mid \langle b_0', \sigma\rangle}{(\rho, \gamma) \vdash \langle b_0 \text{ OR } b_1, \sigma\rangle \dashrightarrow_b \text{true} \mid \langle b_1, \sigma\rangle \mid \langle b_0' \text{ OR } b_1, \sigma\rangle}$$

Conditional Commands

For conditional commands we have the transition system $<\Gamma_{cc}, T_{cc}, \dashrightarrow_{cc}>$ such that

$\Gamma_{cc} = \{<cc, \sigma>\} \cup \{<c, \sigma>\} \cup \{failure\}$

$T_{cc} = \{<c, \sigma>\} \cup \{failure\}$

Rules

$$\frac{(\rho, \gamma) \vdash <b, \sigma> \dashrightarrow_b \quad true \mid false \mid <b', \sigma>}{(\rho, \gamma) \vdash <b \rightarrow c, \sigma> \dashrightarrow_{cc} <c, \sigma> \mid failure \mid <b' \rightarrow c, \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <cc_0, \sigma> \dashrightarrow_{cc} \quad <c, \sigma> \mid <cc_0', \sigma>}{(\rho, \gamma) \vdash <cc_0 \, [] \, cc_1, \sigma> \dashrightarrow_{cc} <c, \sigma> \mid <cc_0' \, [] \, cc_1, \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <cc_0, \sigma> \dashrightarrow_{cc} \quad failure}{(\rho, \gamma) \vdash <cc_0 \, [] \, cc_1, \sigma> \dashrightarrow_{cc} <failure \, [] \, cc_1, \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <cc_1, \sigma> \dashrightarrow_{cc} \quad <c, \sigma> \mid <cc_1', \sigma>}{(\rho, \gamma) \vdash < failure \, [] \, cc_1, \sigma> \dashrightarrow_{cc} \quad <c, \sigma> \mid <failure \, [] \, cc_1', \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <cc_1, \sigma> \dashrightarrow_{cc} \ failure}{(\rho, \gamma) \vdash <failure \, [] \, cc_1, \sigma> \dashrightarrow_{cc} \ failure}$$

Declarations

For Declarations we have the transition system $< \Gamma_d, T_d, \dashrightarrow_d >$. We extend Dec by adding the production:

$$d ::= (\rho, \gamma)$$

As before this means that the abstract syntax of declaration configurations includes environments; it does not mean that the abstract syntax of declarations does so.

$\Gamma_d = \{<d>\}$

$T_d = \{ <(\rho, \gamma)> \}$

Rules

Constant declarations

$$\frac{(\rho, \gamma) \vdash <a, \sigma> \dashrightarrow_a <a', \sigma>}{(\rho, \gamma) \vdash <\underline{\text{DEF}}\ X = a> \dashrightarrow_d <\underline{\text{DEF}}\ X = a'>}$$

$$(\rho, \gamma) \vdash <\underline{\text{DEF}}\ X = n> \dashrightarrow_d <(\rho[X \mapsto n], \gamma)>$$

Variable declarations

$$(\rho, \gamma) \vdash <\underline{\text{VAR}}\ X> \dashrightarrow_d <(\rho[X \mapsto l], \gamma)>$$

Channel declarations

$$(\rho, \gamma) \vdash <\underline{\text{CHAN}}\ chi> \dashrightarrow_d <(\rho, \gamma[chi \mapsto ch])>$$

Procedure declarations

$$(\rho, \gamma) \vdash <\underline{\text{PROC}}\ X\ (forms)\ c> \dashrightarrow_d <(\rho[X \mapsto \lambda forms.c], \gamma)>$$

Composition of declarations

$$\frac{(\rho, \gamma) \vdash <d_0> \dashrightarrow_d <d_0'>}{(\rho, \gamma) \vdash <d_0\ ;\ d_1> \dashrightarrow_d <d_0'\ ;\ d_1>}$$

$$\frac{(\rho[\rho_0], \gamma[\gamma_0]) \vdash <d_1> \dashrightarrow_d <d_1'>}{(\rho, \gamma) \vdash <(\rho_0, \gamma_0)\ ;\ d_1> \dashrightarrow_d <(\rho_0, \gamma_0)\ ;\ d_1'>}$$

$$(\rho, \gamma) \vdash <(\rho_0, \gamma_0)\ ;\ (\rho_1, \gamma_1)> \dashrightarrow_d <(\rho_0[\rho_1], \gamma_0[\gamma_1])>$$

In the case of commands and guarded commands, transitions must be labelled to specify the direction of flow of information when a communication between two parallel processes is taking place via a particular channel.

Definition 2: A labelled transition system is a quadruple $<\Gamma, T, \Phi, \dashrightarrow>$ where:

Γ is the set of configurations.

$T \subseteq \Gamma$ is the set of terminal configurations.

Φ is a set of labels.

$\dashrightarrow \subseteq (\Gamma \times \Phi \times \Gamma) + (\Gamma \times \Gamma)$ is the transition relation.

such that $\forall \beta \in T \ \forall \beta' \in \Gamma \ \forall \phi \in \Phi \quad \neg(\beta \overset{\Phi}{\dashrightarrow} \beta') \ \wedge \ \neg(\beta \dashrightarrow \beta')$

Guarded Commands

For guarded commands we have the labelled transition system $<\Gamma_{gc}, T_{gc}, \Phi_{gc}, \dashrightarrow_{gc}>$. In this case $\Phi_{gc} = \{chi\ ?\ n \mid n \in Num\}$. Note the presence of a label denotes that a communication has taken place.

$\Gamma_{gc} = \{<gc, \sigma>\} \cup \{<c, \sigma>\}$

$T_{gc} = \{<c, \sigma>\}$

Rules

$$(\rho, \gamma) \vdash <chi\ ?X \rightarrow c, \sigma> \overset{\gamma(chi)?n}{\dashrightarrow_{gc}} <c, \sigma[\rho(X) \mapsto n]>$$

$$\frac{(\rho, \gamma) \vdash <gc_0, \sigma> \overset{\gamma(chi)?n}{\dashrightarrow_{gc}} <c, \sigma'>}{(\rho, \gamma) \vdash <gc_0\ []\ gc_1, \sigma> \overset{\gamma(chi)?n}{\dashrightarrow_{gc}} <c, \sigma'>}$$

$$\frac{(\rho, \gamma) \vdash <gc_1, \sigma> \overset{\gamma(chi)?n}{\dashrightarrow_{gc}} <c, \sigma'>}{(\rho, \gamma) \vdash <gc_0\ []\ gc_1, \sigma> \overset{\gamma(chi)?n}{\dashrightarrow_{gc}} <c, \sigma'>}$$

Commands

For Commands we have the transition system $< \Gamma_c , T_c , \Phi c, \dashrightarrow_c >$ where $\Phi c = \{chi ? n \mid n \in Num\} \cup \{chi ! n \mid n \in Num\}$. One understands the execution of $c_0 \parallel c_1$ as the interleaved execution of "grains of action" of c_0 and c_1. For example when c_0 starts a grain of action, it completes it before c_1 can interrupt by interleaving. When two processes running in parallel synchronise to communicate via some channel chi, (i.e. one process inputs from channel chi, while simultaneously the other outputs to the same channel), this is considered as an atomic (unlabelled) transition.

$\Gamma_c = \{<c,\sigma>\} \cup \{<\sigma>\} \cup \{abortion\}$

$T_c = \{<\sigma>\} \cup \{abortion\}$

Rules

STOP

$(\rho, \gamma) \vdash <stop, \sigma> \dashrightarrow_c$ abortion

SKIP

$(\rho, \gamma) \vdash <skip, \sigma> \dashrightarrow_c <\sigma>$

Assignment

$$\frac{(\rho, \gamma) \vdash <a, \sigma> \dashrightarrow_a n \mid <a', \sigma>}{(\rho, \gamma) \vdash <X := a, \sigma> \dashrightarrow_c <\sigma[\rho(X) \mapsto n]> \mid <a', \sigma>}$$

SEQ

$$\frac{(\rho, \gamma) \vdash <c_0, \sigma> \dashrightarrow_c <c_0', \sigma'> \mid <\sigma'> \mid abortion}{(\rho, \gamma) \vdash <c_0 ; c_1, \sigma> \dashrightarrow_c <c_0' ; c_1, \sigma'> \mid <c_1, \sigma'> \mid abortion}$$

IF

$$\frac{(\rho, \gamma) \vdash <cc, \sigma> \dashrightarrow_{cc} <c, \sigma> \mid <cc', \sigma> \mid failure}{(\rho, \gamma) \vdash <IF\ cc, \sigma> \dashrightarrow_c <c, \sigma> \mid <IF\ cc', \sigma> \mid abortion}$$

WHILE

$$\frac{(\rho, \gamma) \vdash <b, \sigma> \ \text{-->}_b \quad true \quad | \quad <b', \sigma>}{(\rho, \gamma) \vdash <WHILE\ b\ \ c, \sigma> \text{-->}_c <c ; WHILE\ b\ \ c, \sigma> \ | \ <WHILE\ b'\ \ c, \sigma>}$$

$$\frac{(\rho, \gamma) \vdash <b, \sigma> \ \text{-->}_b \quad false}{(\rho, \gamma) \vdash <WHILE\ b\ \ c, \sigma> \text{-->}_c \ <\sigma>}$$

ALT

$$\frac{(\rho, \gamma) \vdash <gc, \sigma> \ \text{-->}_{gc} \ <c, \sigma'>}{(\rho, \gamma) \vdash <ALT\ gc, \sigma> \ \text{-->}_c \ <c, \sigma'>}$$

PAR

(i)

$$\frac{(\rho, \gamma) \vdash <c_0, \sigma> \text{-->}_c <c_0', \sigma'> \ | \ <\sigma'> \ | \ abortion}{(\rho, \gamma) \vdash <c_0 \ || \ c_1, \sigma> \text{-->}_c <c_0' \ || \ c_1, \sigma'> \ | <c_1, \sigma'> | \ <c_1 ; stop, \sigma>}$$

(ii)

$$\frac{(\rho, \gamma) \vdash <c_1, \sigma> \text{-->}_c <c_1', \sigma'> \ | \ <\sigma'> \ | \ abortion}{(\rho, \gamma) \vdash <c_0 \ || \ c_1, \sigma> \text{-->}_c <c_0 \ || \ c_1', \sigma'> \ | <c_0, \sigma'> | \ <c_0 ; stop, \sigma>}$$

(iii)

$$\frac{(\rho, \gamma) \vdash <c_0, \sigma> \overset{\gamma(chi)?n}{\text{-->}_c} <c_0', \sigma'> \qquad\qquad (\rho, \gamma) \vdash <c_1, \sigma> \overset{\gamma(chi)!n}{\text{-->}_c} <c_1', \sigma>}{(\rho, \gamma) \vdash <c_0 \ || \ c_1, \sigma> \text{-->}_c <c_0' \ || \ c_1', \sigma'>}$$

(iv)

$$\frac{(\rho, \gamma) \vdash <c_0, \sigma> \overset{\gamma(chi)?n}{\text{-->}_c} <\sigma'> \qquad\qquad (\rho, \gamma) \vdash <c_1, \sigma> \overset{\gamma(chi)!n}{\text{-->}_c} <\sigma>}{(\rho, \gamma) \vdash <c_0 \ || \ c_1, \sigma> \text{-->}_c <\sigma'>}$$

(v)

$$\frac{(\rho, \gamma) \vdash \langle c_0 , \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle \sigma' \rangle \qquad (\rho, \gamma) \vdash \langle c_1, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle c_1', \sigma \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \longrightarrow_{c} \langle c_1', \sigma' \rangle}$$

(vi)

$$\frac{(\rho, \gamma) \vdash \langle c_0, \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle c_0', \sigma' \rangle \qquad (\rho, \gamma) \vdash \langle c_1, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle \sigma \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \longrightarrow_{c} \langle c_0', \sigma' \rangle}$$

There are similar rules corresponding to rules (iii) - (vi) when c_0 is performing an output and c_1 is performing an input.

(vii)

$$\frac{(\rho, \gamma) \vdash \langle c_0, \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle c_0', \sigma' \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle c_0' \parallel c_1, \sigma' \rangle}$$

(viii)

$$\frac{(\rho, \gamma) \vdash \langle c_0, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle c_0', \sigma \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle c_0' \parallel c_1, \sigma \rangle}$$

(ix)

$$\frac{(\rho, \gamma) \vdash \langle c_0, \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle \sigma' \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \xrightarrow{\gamma(chi)?n}_{c} \langle c_1, \sigma' \rangle}$$

(x)

$$\frac{(\rho, \gamma) \vdash \langle c_0, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle \sigma \rangle}{(\rho, \gamma) \vdash \langle c_0 \parallel c_1, \sigma \rangle \xrightarrow{\gamma(chi)!n}_{c} \langle c_1, \sigma \rangle}$$

There are similar rules corresponding to rules (vii) - (x) when c_1 is taking part in a communication.

Input

$$(\rho, \gamma) \vdash \langle chi\,?\,X, \sigma\rangle \xrightarrow{\gamma(chi)?n}_c \langle \sigma[\rho(X) \mapsto n]\rangle$$

Output

$$\frac{(\rho, \gamma) \vdash \langle a, \sigma\rangle \dashrightarrow_a \langle a', \sigma\rangle}{(\rho, \gamma) \vdash \langle chi\,!\,a, \sigma\rangle \dashrightarrow_c \langle chi\,!\,a', \sigma\rangle}$$

$$(\rho, \gamma) \vdash \langle chi\,!\,n, \sigma\rangle \xrightarrow{\gamma(chi)!n}_c \langle \sigma\rangle$$

Block

Informally, to execute d ; c from σ :

1. Expand d from σ given (ρ, γ) yielding (ρ_o, γ_o) and store σ'.
2. Execute c from σ' given $(\rho[\rho_o], \gamma[\gamma_o])$ yielding σ'' which is the result of the execution.

$$\frac{(\rho, \gamma) \vdash \langle d\rangle \dashrightarrow_d \langle d'\rangle}{(\rho, \gamma) \vdash \langle d\,;\, c, \sigma\rangle \dashrightarrow_c \langle d'\,;\, c, \sigma\rangle}$$

$$\frac{(\rho[\rho_o], \gamma[\gamma_o]) \vdash \langle c, \sigma\rangle \dashrightarrow_c \langle c', \sigma'\rangle \mid \text{abortion}}{(\rho, \gamma) \vdash \langle (\rho_o, \gamma_o)\,;\, c, \sigma\rangle \dashrightarrow_c \langle (\rho_o, \gamma_o)\,;\, c', \sigma'\rangle \mid \text{abortion}}$$

$$\frac{(\rho[\rho_o], \gamma[\gamma_o]) \vdash \langle c, \sigma\rangle \dashrightarrow_c \langle \sigma'\rangle}{(\rho, \gamma) \vdash \langle (\rho_o, \gamma_o)\,;\, c, \sigma\rangle \dashrightarrow_c \langle \sigma'\rangle}$$

A call to a PROC

$$\frac{(\rho,\gamma) \vdash \langle \underline{\underline{a}}, \sigma \rangle \text{-->}_{a} \underline{\underline{n}} \quad | \quad \langle \underline{\underline{a}}', \sigma \rangle}{(\rho,\gamma) \vdash \langle P(\underline{\underline{a}}, \underline{\underline{X}}, \underline{\underline{chi}}), \sigma \rangle \text{-->}_{c} \langle P(\underline{\underline{n}}, \underline{\underline{X}}, \underline{\underline{chi}}), \sigma \rangle \mid \langle P(\underline{\underline{a}}', \underline{\underline{X}}, \underline{\underline{chi}}), \sigma \rangle}$$

$$(\rho,\gamma) \vdash \langle P(\underline{\underline{n}}, \underline{\underline{X}}, \underline{\underline{chi}}), \sigma \rangle \text{-->}_{c} \langle c[\underline{\underline{n}} / \underline{\underline{X}}'] [\underline{\underline{X}} / \underline{\underline{X}}''] [\underline{\underline{chi}} / \underline{\underline{chi}}'], \sigma \rangle$$

where $\rho(P) = \lambda(\text{VALUE } \underline{\underline{X}}', \text{VAR } \underline{\underline{X}}'', \text{CHAN } \underline{\underline{chi}}').c$ such that $\models$ acts$\uparrow$forms as discussed previously. <u>Note</u>: A **PROC** definition is non-recursive in standard **occam**.

6 Conclusion

In this report an interleaving semantics of the main constructs of **occam** has been presented. The addition of rules for replicators and BYTE subscription should be straight forward. On the other hand it is the author's belief that in order to define the semantics of priority alternation we may require various changes to the model presented here. This point is currently being examined.

Acknowledgements

I express my thanks to Dr. Glynn Winskel for encouraging me to work on the subject and for his advice and comments. I would also like to thank Dr. Alan Mycroft for his helpful feedback. Thanks are also due to Trinity College who are kindly supporting my stay at Cambridge.

7 Appendix

The aim of this appendix is to explain some notation used throughout this paper.

Let A and B be sets.

1. $A \cup B$	denotes the union of A and B.
2. $A + B$	denotes the disjoint union of A and B.
3. $A \subseteq_{fin} B$	denotes that A is a finite subset of B.
4. $A \rightarrow_{fin} B$	denotes the set of all partial functions with domains which are finite subsets of A.
5. $A \rightarrow B$	denotes the set of all total functions with domain A.

Let A_i be sets (for $0 \leqq i \leqq n$) then $\sum_{0 \leq i \leq n} A_i = A_0 + \ldots + A_n$.

<u>Substitution</u>

If $\underline{\underline{a}} = (a_1, \ldots, a_n)$ and $\underline{\underline{b}} = (b_1, \ldots, b_n)$ then we use

$c\ [\underline{\underline{a}} / \underline{\underline{b}}]$ to denote $c\ [a_1 / b_1], \ldots, [a_n / b_n]$

where $c\ [a_i / b_i]$ means substitute all free occurrences of b_i in c by a_i .

8 References

[Dijkstra 1]	Edsger.W.Dijkstra. A Discipline of Programming. Prentice-Hall International Series in Automatic Computation.
[Hoare 2]	C.A.R.Hoare. Communicating Sequential Processes. Prentice-Hall International Series in Computer Science.
[Inmos 3]	INMOS ltd. **occam** Programming Manual. Prentice-Hall International Series in Computer Science.
[Jones 4]	Geraint Jones. Programming in **occam** . Prentice-Hall International Series in Computer Science.
[Khan 5]	Gilles Khan. Natural Semantics. Rapports de Recherche N° 601 INRIA SOPHIA-ANTIPOLIS 06560 Valbonne France. Février 1987.
[Li 6]	Wei Li. An Operational Semantics of Tasking and Exception Handling in Ada. Department of Computer Science University of Edinburgh. December 1981.
[Milner 7]	Robin Milner. A Calculus of Communicating Systems. Lecture notes in Computer Science. Springer-Verlag series N° 92.
[Plotkin 8]	Gordon.D.Plotkin. A Structural Approach to Operational Semantics. Department of Computer Science, Äarhus University Denmark. Sept 1981.
[Plotkin 9]	Gordon.D.Plotkin. An Operational Semantics for CSP. Department of Computer Science, University of Edinburgh. Internal Report CSR-114-82.
[Roscoe - Hoare 10]	A.W.Roscoe, C.A.R Hoare. The laws of **occam** programming. Oxford University. Technical monograph PRG-53.
[Roscoe 11]	A.W.Roscoe. Denotional Semantics for **occam**. Lecture notes In Computer Science. Springer-Verlag series N° 197.