Technical Report

UCAM-CL-TR-144
ISSN 1476-2986

Number 144

# An operational semantics for occam

Juanito Camilleri

August 1988

This is an extended version of UCAM-CL-TR-125, in which we include the operational semantics of priority alternation.

ISSN 1476-2986

# An Operational Semantics for occam[*]

(EXTENDED VERSION)

Juanito Camilleri
University of Cambridge,
Computer Laboratory,
New Museum 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 5]. 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 10] and are applied in [Plotkin 11] where an operational semantics for CSP [Hoare 4] was presented. The operational semantics for a subset of Ada is defined in [Li 8], where tasking and exception handling are modeled. For simplicity only a subset of* **occam** *is defined. Timing, 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.*

---

[*] This is an extended version of technical report nº.125 from the Computing Lab, University of Cambridge. In this extended version we include the operational semantics of priority alternation as described in [Inmos 5].

## 1 Introduction

This work was originally intended as an exercise in operational semantics [Plotkin 10]. In the first version of this report [Camilleri 2], we chose to define the main constructs of the programming language **occam**, but, various features of the language were omitted.

One of the constructs which wasn't included in the original version of this report, is priority alternation [Inmos 5]. What exactly is priority alternation? One can argue that the semantics of priority alternation is very similar to the semantics of simple alternation. In fact the only extra condition that is imposed on priority alternation is that at any instant in the computation, if more than one guarded command can be executed in a PRIALT construct, then the one which comes first in textual order is executed. This can be rephrased as follows:-

"A guarded command can be executed in a PRIALT construct if none of the guarded commands which occur previously in textual order can be executed."

In order to formalise the above statement we need a notion of which guarded commands can be executed. In this paper we augment the operational semantics as presented in [Camilleri 2] by including the operational semantics of the PRIALT construct. To this aim, this paper is divided into the following sections, namely:

i. The introduction of the syntactic categories (i.e. sets) and syntax of the language to be defined.

ii. The definition of a transition system.

iii. The definition of the static semantics.

iv. The definition of the dynamic semantics.

v. An Appendix to define certain notations used in the paper.

## 2 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 | ranged over by | cc. |
| 10. | guarded commands | Gcom | " " " " " | gc. |
| 11. | commands | Com | " " " " " | c. |
| 12. | declarations | Dec | " " " " " | d. |
| 13. | actual parameters | Acts | " " " " " | acts. |
| 14. | formal parameters | Forms | " " " " " | forms. |

## 3 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 $\lambda$forms.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 $\underline{s}$ denotes a finite list (possibly empty) of elements of S. Therefore $\underline{s} = (s_1, \ldots, s_n)$ where for all $1 \leqq i \leqq n$, $s_i \in S$. Note ( ) denotes the empty list.

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

Iexp

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

Bexp

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

Ccom

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

Gcom

gc : : = chi ? X $\rightarrow$ c I 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 |

PRIALT 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
        c0 ; c1      denotes        c0
                                    c1
```

<u>and</u>

```
                                  PAR
        c0 || c1     denotes        c0
                                    c1
```

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.

## 4 Definition of a transition system.

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

$\Gamma$ is the set of configurations.

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

$--> \subseteq \Gamma^2$ is the transition relation

such that $\forall \beta \in T \; \forall \beta' \in \Gamma \;\; \neg(\beta --> \beta')$.

## 5 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 not 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 $\alpha$ 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 declaration configurations includes static environments; it does not mean that the abstract syntax of declarations 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 $\alpha$ 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 $\alpha$.

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 $1 \leqq i \leqq n$, 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. |
| $\models a \uparrow$ VALUE X' | where a is an integer expression and X' is a VALUE parameter. |
| $\models X \uparrow$ VAR X'' | where X is an identifier and X'' is a VAR parameter. |
| $\models$ chi $\uparrow$ 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 | $\emptyset$ | {X} | $RI(a_0) \cup RI(a_1)$ |

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

For boolean expressions

| | t | NOT b | $b_0$ bop $b_1$ | $a_0$ relop $a_1$ |
|---|---|---|---|---|
| RI | $\emptyset$ | 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 $\emptyset$.

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 | PRIALT gc | d ; c |
|---|---|---|---|---|
| RI | RI(b) ∪ RI(c) | RI(gc) | RI(gc) | RI(c) |
| WI | WI(c) | WI(gc) | WI(gc) | WI(c) |
| INCH | INCH(c) | INCH(gc) | INCH(gc) | INCH(c) |
| OUTCH | OUTCH(c) | OUTCH(gc) | 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

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

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

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

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**

$$\alpha \vdash \text{skip} \qquad \alpha \vdash \text{stop} \qquad \alpha \vdash X ::= a \qquad \alpha \vdash chi\,?X \qquad \alpha \vdash chi\,!\,a$$

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

$$\frac{\alpha \vdash c_0 \qquad \alpha \vdash c_1}{\alpha \vdash c_0 \parallel c_1} \qquad
\begin{array}{ll}
\text{if} & (WI(c_0) \cup RI(c_0)) \cap WI(c_1) = \varnothing \\
\text{and} & (WI(c_1) \cup RI(c_1)) \cap WI(c_0) = \varnothing \\
\text{and} & INCH(c_0) \cap OUTCH(c_0) = \varnothing \\
\text{and} & INCH(c_1) \cap OUTCH(c_1) = \varnothing \\
\text{and} & INCH(c_0) \cap INCH(c_1) = \varnothing \\
\text{and} & OUTCH(c_0) \cap OUTCH(c_1) = \varnothing
\end{array}$$

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

$$\frac{\alpha \vdash c}{\alpha \vdash X(acts)} \qquad \text{If } \alpha(X) = \lambda forms.c \ \text{ and } \ \vDash acts \uparrow forms$$

## 6 Dynamic Semantics

### (a) 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

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

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

We shall use $\rho$ 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

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

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

We shall use $\gamma$ to range over CEnv.

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

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

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

We shall use $\sigma$ 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 $(\rho, \gamma)$ with the understanding that $\rho$ ranges over LEnv and $\gamma$ 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 $(\rho, \gamma)$ an identifier X can be associated with :-

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

(b) a location — when X is defined to be a variable. Therefore $\rho(X) = l$. Given a store $\sigma$ 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 $\rho$, but, not in the domain of $\sigma$, 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 $\lambda$forms.c.

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

## (b) The handling of declarations

Before proceeding with the dynamic semantics, it is useful to discuss the declaration of constants, variables, channels and abstractions. 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.

Let $\beta :: B$ mean that $\beta$ has domain $B$.

For any $B_0, B_1$ and $\beta_0 :: B_0$, $\beta_1 :: B_1$ we define $\beta = \beta_0[\beta_1] :: B_0 \cup B_1$ by:

$$\beta(X) = \begin{cases} \beta_1(X) & \text{if} \quad (X \in B_1) \\ \beta_0(X) & \text{if} \quad (X \in B_0 \setminus B_1) \end{cases}$$

Using the above definition, we denote the updating of environment $(\rho_0, \gamma_0) :: (I_0, C_0)$ by $(\rho_1, \gamma_1) :: (I_1, C_1)$, as $(\rho_0[\rho_1], \gamma_0[\gamma_1]) :: ((I_0 \cup I_1), (C_0 \cup C_1))$

## (c) The transition relations

The transition relations, encountered in the various transition systems to be defined later, are relative to the environment. Therefore if s is an element of some syntactic category we write :

$$(\rho, \gamma) \vdash \; <s, \sigma> \; -->_s \; <s', \sigma'>$$

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

$$(\rho, \gamma) \vdash <s, \sigma> \; -->_s \; <s', \sigma'> \mid <s'', \sigma''>$$

is read as — In environment $(\rho, \gamma)$, one step in the evaluation of s in store $\sigma$ can either yield $s'$ and store $\sigma'$, or, $s''$ and store $\sigma''$.

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)

## (d) The transition systems

### 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 $\rho(X) = I$ and $\sigma(I)$ 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

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

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

## 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

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

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

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

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

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

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

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

**Rules**

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

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

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

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

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

## 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

$$\frac{(\rho, \gamma) \vdash \langle a, \sigma \rangle \dashrightarrow_a \langle a', \sigma \rangle}{(\rho, \gamma) \vdash \langle \underline{\mathrm{DEF}}\ X = a \rangle \dashrightarrow_d \langle \underline{\mathrm{DEF}}\ X = a' \rangle}$$

$$(\rho, \gamma) \vdash \langle \underline{\mathrm{DEF}}\ X = n \rangle \dashrightarrow_d \langle (\rho[X \mapsto n], \gamma) \rangle$$

### Variable declarations

$$(\rho, \gamma) \vdash \langle \underline{\mathrm{VAR}}\ X \rangle \dashrightarrow_d \langle (\rho[X \mapsto l], \gamma) \rangle$$

### Channel declarations

$$(\rho, \gamma) \vdash \langle \underline{\mathrm{CHAN}}\ chi \rangle \dashrightarrow_d \langle (\rho, \gamma[chi \mapsto ch]) \rangle$$

### Procedure declarations

$$(\rho, \gamma) \vdash \langle \underline{\mathrm{PROC}}\ X\ (forms)\ c \rangle \dashrightarrow_d \langle (\rho[X \mapsto \lambda forms.c], \gamma) \rangle$$

### Composition of declarations

$$\frac{(\rho, \gamma) \vdash \langle d_o \rangle \dashrightarrow_d \langle d_o' \rangle}{(\rho, \gamma) \vdash \langle d_o\ ;\ d_1 \rangle \dashrightarrow_d \langle d_o'\ ;\ d_1 \rangle}$$

$$\frac{(\rho[\rho_o], \gamma[\gamma_o]) \vdash \langle d_1 \rangle \dashrightarrow_d \langle d_1' \rangle}{(\rho, \gamma) \vdash \langle (\rho_o, \gamma_o)\ ;\ d_1 \rangle \dashrightarrow_d \langle (\rho_o, \gamma_o)\ ;\ d_1' \rangle}$$

$$(\rho, \gamma) \vdash \langle (\rho_o, \gamma_o)\ ;\ (\rho_1, \gamma_1) \rangle \dashrightarrow_d \langle (\rho_o[\rho_1], \gamma_o[\gamma_1]) \rangle$$

## Commands

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

$\Gamma$ is the set of configurations.

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

$\Phi$ is a set of labels.

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

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

For Commands we have a labelled transition system $<\Gamma_c, T_c, \Phi_c, \dashrightarrow_c>$ where $\Phi_c = \{chi\ ?\ n \mid n \in Num\} \cup \{chi\ !\ n \mid n \in Num\}$. The labels specify the direction of flow of information when a communication takes place. 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 synchronize 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.

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

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

## The Refusal and Acceptance sets

Earlier on we mentioned that in order to define the operational semantics of the PRIALT construct, we need a notion of which guarded commands can be executed. For example, consider the following construct:

$$(\text{PRIALT } chi_0\ ?\ X_0 \longrightarrow c_0 \ [] \ chi_1\ ?\ X_1 \longrightarrow c_1) \parallel c_2$$

According to the semantics of the PRIALT construct, the guarded command $chi_1?X_1 \longrightarrow c_1$ will only execute provided a construct of the form $chi_1!n$ can execute in $c_2$ and a construct of the form $chi_0!n$ cannot execute in $c_2$. In other words we might say that $chi_1?X_1 \longrightarrow c_1$ can be executed provided $c_2$ refuses to communicate via channel $chi_0$, but, does not refuse to communicate via channel $chi_1$. In order to formalise this concept of a piece of syntax refusing communication via some channels, we define the relation $\underline{ref} \subseteq Com \times P_{fin}(Chan)$ by rules (i) - (xiii). If $(\rho, \gamma) \vdash c\ \underline{ref}\ R$ holds, we say c *refuses* R in environment $(\rho, \gamma)$. We call R a *refusal set* of c in environment $(\rho, \gamma)$.

(i) $(\rho, \gamma) \vdash \text{skip}\ \underline{ref}\ R$ for any R. (ii) $(\rho, \gamma) \vdash \text{stop}\ \underline{ref}\ R$ for any R.

(iii) $(\rho, \gamma) \vdash X := a\ \underline{ref}\ R$ for any R. (iv) $(\rho, \gamma) \vdash chi?X\ \underline{ref}\ R$ for any R.

(v) $(\rho, \gamma) \vdash \text{ALT } gc\ \underline{ref}\ R$ for any R. (vi) $(\rho, \gamma) \vdash \text{PRIALT } gc\ \underline{ref}\ R$ for any R.

(vii) $(\rho, \gamma) \vdash$ WHILE b c $\underline{\text{ref}}$ R for any R. (viii) $(\rho, \gamma) \vdash$ IF cc $\underline{\text{ref}}$ R for any R.

(ix) $(\rho, \gamma) \vdash$ X(acts) $\underline{\text{ref}}$ R for any R. (x) $(\rho, \gamma) \vdash$ d ; c $\underline{\text{ref}}$ R for any R.

(xi) $(\rho, \gamma) \vdash$ chi ! a $\underline{\text{ref}}$ R for any R provided $a \neq n$.

(xii) $(\rho, \gamma) \vdash$ chi ! n $\underline{\text{ref}}$ R for any R provided $\gamma(\text{chi}) \notin R$.

(xiii) $$\frac{(\rho, \gamma) \vdash c_0 \ \underline{\text{ref}}\ R}{(\rho, \gamma) \vdash c_0 ; c_1 \ \underline{\text{ref}}\ R}$$ for any R.

(xiv) $$\frac{(\rho, \gamma) \vdash c_0 \ \underline{\text{ref}}\ R \qquad (\rho, \gamma) \vdash c_1 \ \underline{\text{ref}}\ R}{(\rho, \gamma) \vdash c_0 \parallel c_1 \ \underline{\text{ref}}\ R}$$ for any R.

Given a construct of the form (PRIALT $gc_0$ [] $gc_1$) II $c_2$ we will only allow $gc_1$ to be evaluated provided $c_2$ refuses communication via any of the channels included in $gc_0$. Therefore we need to know which channels are included in $gc_0$. In order to formalise this notion we define the relation $\underline{\text{acc}} \subseteq \textbf{GCom} \times P_{\text{fin}}(\textbf{Chan})$ by rules (i) - (ii) below. If $(\rho, \gamma) \vdash$ gc $\underline{\text{acc}}$ A holds, we say gc *accepts* A in environment $(\rho, \gamma)$. We call A the *acceptance set* of gc in environment $(\rho, \gamma)$.

(i) $(\rho, \gamma) \vdash$ chi?X $\twoheadrightarrow$ c $\underline{\text{acc}}$ {ch} where $\gamma(\text{chi})$ = ch.

(ii) $$\frac{(\rho, \gamma) \vdash gc_0 \ \underline{\text{acc}}\ A \qquad (\rho, \gamma) \vdash gc_1 \ \underline{\text{acc}}\ A'}{(\rho, \gamma) \vdash gc_0 \ [\,]\ gc_1 \ \underline{\text{acc}}\ A \cup A'}$$

For example, in an environment $(\rho, \gamma)$, if $\gamma(\text{chi}_0) = \text{ch}_0$ and $\gamma(\text{chi}_1) = \text{ch}_1$ then (PRIALT $\text{chi}_0$ ? $X_0 \twoheadrightarrow c_0$ [] $\text{chi}_1$ ? $X_1 \twoheadrightarrow c_1$) accepts $\{\text{ch}_0, \text{ch}_1\}$.

When we are dealing with the syntactic category of commands, c, the transition relation takes the form:

$$(\rho, \gamma) \vdash_R \ <c, \sigma> \ -->_c \ <c', \sigma'>$$

This means $(\rho, \gamma) \vdash <c, \sigma> -->_c <c', \sigma'>$, provided the following restriction holds; if **c** is executing in parallel with any construct $\mathbf{c_0}$, then $\mathbf{c_0}$ refuses **R**.

Similarly $(\rho, \gamma) \vdash_R <c, \sigma> \overset{\phi}{-->}_c <c', \sigma'>$ means $(\rho, \gamma) \vdash <c, \sigma> \overset{\phi}{-->}_c <c', \sigma'>$ provided the same restriction as mentioned above holds. (Note $\phi$ denotes a label)

Rules

STOP

$$(\rho, \gamma) \vdash_R \langle \mathrm{stop}, \sigma \rangle \dashrightarrow_c \text{ abortion}$$

SKIP

$$(\rho, \gamma) \vdash_R \langle \mathrm{skip}, \sigma \rangle \dashrightarrow_c \langle \sigma \rangle$$

Assignment

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

SEQ

$$\frac{(\rho, \gamma) \vdash_R \langle c_0, \sigma \rangle \dashrightarrow_c \langle c_0', \sigma' \rangle \mid \langle \sigma' \rangle \mid \text{abortion}}{(\rho, \gamma) \vdash_R \langle c_0 ; c_1, \sigma \rangle \dashrightarrow_c \langle c_0' ; c_1, \sigma' \rangle \mid \langle c_1, \sigma' \rangle \mid \text{abortion}}$$

IF

$$\frac{(\rho, \gamma) \vdash \langle cc, \sigma \rangle \dashrightarrow_{cc} \langle c, \sigma \rangle \mid \langle cc', \sigma \rangle \mid \text{failure}}{(\rho, \gamma) \vdash_R \langle \mathrm{IF}\ cc, \sigma \rangle \dashrightarrow_c \langle c, \sigma \rangle \mid \langle \mathrm{IF}\ cc', \sigma \rangle \mid \text{abortion}}$$

WHILE

$$\frac{(\rho, \gamma) \vdash \langle b, \sigma \rangle \dashrightarrow_b \text{true} \mid \langle b', \sigma \rangle}{(\rho, \gamma) \vdash_R \langle \mathrm{WHILE}\ b\ c, \sigma \rangle \dashrightarrow_c \langle c ; \mathrm{WHILE}\ b\ c, \sigma \rangle \mid \langle \mathrm{WHILE}\ b'\ c, \sigma \rangle}$$

$$\frac{(\rho, \gamma) \vdash \langle b, \sigma \rangle \dashrightarrow_b \text{false}}{(\rho, \gamma) \vdash_R \langle \mathrm{WHILE}\ b\ c, \sigma \rangle \dashrightarrow_c \langle \sigma \rangle}$$

ALT

$$(\rho, \gamma) \vdash_{R} \langle \mathrm{ALT\ chi\ ?\ X} \rightarrow c, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c, \sigma[\rho(X) \mapsto n] \rangle$$

$$\frac{(\rho, \gamma) \vdash_{R} \langle \mathrm{ALT}\ gc_0, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_0, \sigma' \rangle}{(\rho, \gamma) \vdash_{R} \langle \mathrm{ALT}\ gc_0\ []\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_0, \sigma' \rangle}$$

$$\frac{(\rho, \gamma) \vdash_{R} \langle \mathrm{ALT}\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_1, \sigma' \rangle}{(\rho, \gamma) \vdash_{R} \langle \mathrm{ALT}\ gc_0\ []\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_1, \sigma' \rangle}$$

PRIALT

$$(\rho, \gamma) \vdash_{R} \langle \mathrm{PRIALT\ chi\ ?\ X} \rightarrow c, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c, \sigma[\rho(X) \mapsto n] \rangle$$

$$\frac{(\rho, \gamma) \vdash_{R} \langle \mathrm{PRIALT}\ gc_0, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_0, \sigma' \rangle}{(\rho, \gamma) \vdash_{R} \langle \mathrm{PRIALT}\ gc_0\ []\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_0, \sigma' \rangle}$$

$$\frac{(\rho, \gamma) \vdash_{R} \langle \mathrm{PRIALT}\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_1, \sigma' \rangle \qquad (\rho, \gamma) \vdash gc_0\ \underline{\mathbf{acc}}\ \mathbf{A}}{(\rho, \gamma) \vdash_{R} \langle \mathrm{PRIALT}\ gc_0\ []\ gc_1, \sigma \rangle \xrightarrow{\gamma(\mathrm{chi})?n}_{c} \langle c_1, \sigma' \rangle} \quad \text{if } \mathbf{A} \subseteq \mathbf{R}$$

It is evident that the rules for ALT and PRIALT are very similar. The only difference is that in the last rule above we have added the condition that $gc_1$ can only be executed in a context where all the elements of the acceptance set of $gc_0$ are refused.

PAR

(i)

$$\frac{(\rho,\gamma)\vdash_{\mathbf{R}}\langle c_0,\sigma\rangle \dashrightarrow_c \langle c_0',\sigma'\rangle \mid \langle\sigma'\rangle \mid \text{abortion} \qquad (\rho,\gamma)\vdash c_1\ \underline{\mathbf{ref}}\ \mathbf{R}}{(\rho,\gamma)\vdash_{\mathbf{R}}\langle c_0 \parallel c_1,\sigma\rangle \dashrightarrow_c \langle c_0'\parallel c_1,\sigma'\rangle \mid \langle c_1,\sigma'\rangle \mid \langle c_1;\mathrm{stop},\sigma\rangle}$$

(ii)

$$\frac{(\rho,\gamma)\vdash_{\mathbf{R}}\langle c_1,\sigma\rangle \dashrightarrow_c \langle c_1',\sigma'\rangle \mid \langle\sigma'\rangle \mid \text{abortion} \qquad (\rho,\gamma)\vdash c_0\ \underline{\mathbf{ref}}\ \mathbf{R}}{(\rho,\gamma)\vdash_{\mathbf{R}}\langle c_0 \parallel c_1,\sigma\rangle \dashrightarrow_c \langle c_0\parallel c_1',\sigma'\rangle \mid \langle c_0,\sigma'\rangle \mid \langle c_0;\mathrm{stop},\sigma\rangle}$$

(iii)

$$\frac{\begin{array}{l}(\rho,\gamma)\vdash_{\mathbf{R_0}}\langle c_0,\sigma\rangle \overset{\gamma(\mathrm{chi})?n}{\dashrightarrow_c} \langle c_0',\sigma'\rangle \qquad (\rho,\gamma)\vdash c_0\ \underline{\mathbf{ref}}\ \mathbf{R_1}\\ \qquad\qquad (\rho,\gamma)\vdash_{\mathbf{R_1}}\langle c_1,\sigma\rangle \overset{\gamma(\mathrm{chi})!n}{\dashrightarrow_c} \langle c_1',\sigma\rangle \qquad (\rho,\gamma)\vdash c_1\ \underline{\mathbf{ref}}\ \mathbf{R_0}\end{array}}{(\rho,\gamma)\vdash_{\mathbf{R_0}\cup\mathbf{R_1}}\langle c_0\parallel c_1,\sigma\rangle \dashrightarrow_c \langle c_0'\parallel c_1',\sigma'\rangle}$$

(iv)

$$\frac{\begin{array}{l}(\rho,\gamma)\vdash_{\mathbf{R_0}}\langle c_0,\sigma\rangle \overset{\gamma(\mathrm{chi})?n}{\dashrightarrow_c} \langle\sigma'\rangle \qquad (\rho,\gamma)\vdash c_0\ \underline{\mathbf{ref}}\ \mathbf{R_1}\\ \qquad\qquad (\rho,\gamma)\vdash_{\mathbf{R_1}}\langle c_1,\sigma\rangle \overset{\gamma(\mathrm{chi})!n}{\dashrightarrow_c} \langle\sigma\rangle \qquad (\rho,\gamma)\vdash c_1\ \underline{\mathbf{ref}}\ \mathbf{R_0}\end{array}}{(\rho,\gamma)\vdash_{\mathbf{R_0}\cup\mathbf{R_1}}\langle c_0\parallel c_1,\sigma\rangle \dashrightarrow_c \langle\sigma'\rangle}$$

(v)

$$\frac{\begin{array}{l}(\rho,\gamma)\vdash_{\mathbf{R_0}}\langle c_0,\sigma\rangle \overset{\gamma(\mathrm{chi})?n}{\dashrightarrow_c} \langle\sigma'\rangle \qquad (\rho,\gamma)\vdash c_0\ \underline{\mathbf{ref}}\ \mathbf{R_1}\\ \qquad\qquad (\rho,\gamma)\vdash_{\mathbf{R_1}}\langle c_1,\sigma\rangle \overset{\gamma(\mathrm{chi})!n}{\dashrightarrow_c} \langle c_1',\sigma\rangle \qquad (\rho,\gamma)\vdash c_1\ \underline{\mathbf{ref}}\ \mathbf{R_0}\end{array}}{(\rho,\gamma)\vdash_{\mathbf{R_0}\cup\mathbf{R_1}}\langle c_0\parallel c_1,\sigma\rangle \dashrightarrow_c \langle c_1',\sigma'\rangle}$$

(vi)

$$\frac{(\rho,\gamma) \vdash_{R_0} <c_0, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <c_0', \sigma'> \qquad (\rho,\gamma) \vdash c_0 \ \underline{\textbf{ref}}\ R_1 \qquad (\rho,\gamma) \vdash_{R_1} <c_1, \sigma> \xrightarrow{\gamma(chi)!n}_{c} <\sigma> \qquad (\rho,\gamma) \vdash c_1 \ \underline{\textbf{ref}}\ R_0}{(\rho,\gamma) \vdash_{R_0 \cup R_1} <c_0 \parallel c_1, \sigma> \dashrightarrow_{c} <c_0', \sigma'>}$$

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

(vii)

$$\frac{(\rho,\gamma) \vdash_{R} <c_0, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <c_0', \sigma'> \qquad (\rho,\gamma) \vdash c_1 \ \underline{\textbf{ref}}\ R}{(\rho,\gamma) \vdash_{R} <c_0 \parallel c_1, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <c_0' \parallel c_1, \sigma'>}$$

(viii)

$$\frac{(\rho,\gamma) \vdash_{R} <c_0, \sigma> \xrightarrow{\gamma(chi)!n}_{c} <c_0', \sigma> \qquad (\rho,\gamma) \vdash c_1 \ \underline{\textbf{ref}}\ R}{(\rho,\gamma) \vdash_{R} <c_0 \parallel c_1, \sigma> \xrightarrow{\gamma(chi)!n}_{c} <c_0' \parallel c_1, \sigma>}$$

(ix)

$$\frac{(\rho,\gamma) \vdash_{R} <c_0, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <\sigma'> \qquad (\rho,\gamma) \vdash c_1 \ \underline{\textbf{ref}}\ R}{(\rho,\gamma) \vdash_{R} <c_0 \parallel c_1, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <c_1, \sigma'>}$$

(x)

$$\frac{(\rho,\gamma) \vdash_{R} <c_0, \sigma> \xrightarrow{\gamma(chi)!n}_{c} <\sigma> \qquad (\rho,\gamma) \vdash c_1 \ \underline{\textbf{ref}}\ R}{(\rho,\gamma) \vdash_{R} <c_0 \parallel c_1, \sigma> \xrightarrow{\gamma(chi)!n}_{c} <c_1, \sigma>}$$

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

<u>Input</u>

$$(\rho,\gamma) \vdash_{R} <chi\ ?\ X, \sigma> \xrightarrow{\gamma(chi)?n}_{c} <\sigma[\rho(X) \mapsto n]>$$

## Output

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

$$(\rho,\gamma) \vdash_R \langle chi\,!\,n\,,\sigma\rangle \overset{\gamma(chi)!n}{\dashrightarrow_c} \langle\sigma\rangle$$

## Block

Informally, to execute d ; c from $\sigma$:

(i) Expand d from $\sigma$ given $(\rho,\gamma)$ yielding $(\rho_o,\gamma_o)$.

(ii) Execute c from $\sigma$ given $(\rho[\rho_o], \gamma[\gamma_o])$ yielding the resulting state of the execution.

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

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

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

## A call to a PROC

$$\frac{(\rho,\gamma) \vdash \langle \underline{\underline{a}}\,,\sigma\rangle \dashrightarrow_{\underline{\underline{a}}} \underline{\underline{n}} \mid \langle \underline{\underline{a}}',\sigma\rangle}{(\rho,\gamma) \vdash_R \langle P(\underline{\underline{a}}, \underline{\underline{X}}, \underline{\underline{chi}}),\sigma\rangle \dashrightarrow_c \langle P(\underline{\underline{n}}, \underline{\underline{X}}, \underline{\underline{chi}}),\sigma\rangle \mid \langle P(\underline{\underline{a}}', \underline{\underline{X}}, \underline{\underline{chi}}),\sigma\rangle}$$

$$(\rho,\gamma) \vdash_R \langle P(\underline{\underline{n}}, \underline{\underline{X}}, \underline{\underline{chi}}),\sigma\rangle \dashrightarrow_c \langle c[\underline{\underline{n}} / \underline{\underline{X}}'] [\underline{\underline{X}} / \underline{\underline{X}}''] [\underline{\underline{chi}} / \underline{\underline{chi}}'],\sigma\rangle$$

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

## 7 Conclusion

In this report an interleaving semantics of the main constructs of **occam** has been presented. The original version [Camilleri 2] has been extended by the introduction of rules that define the semantics of priority alternation. The addition of rules for replicators and BYTE subscription should be straight forward.

## 8 Acknowledgements

I express my thanks to Prof. 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.

## 9 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. |
| 6. $P_{fin}(A)$ | denotes the finite power set of A. |

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

Substitution

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

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

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

## 10 References

[Boudol - Castellani 1] G.Boudol, I.Castellani. Concurrency and Atomicity. INRIA SOPHIA-ANTIPOLIS 06560 Valbonne, France.

[Camilleri 2] Juanito Camilleri. An operational semantics for **occam**. Computing Lab, University of Cambridge. Technical Report nº 125. February 1988.

[Dijkstra 3] Edsger.W.Dijkstra. A Discipline of Programming. Prentice-Hall International Series in Automatic Computation.

[Hoare 4] C.A.R.Hoare. Communicating Sequential Processes. Prentice-Hall International Series in Computer Science.

[Inmos 5] INMOS ltd. **occam** Programming Manual. Prentice-Hall International Series in Computer Science.

[Jones 6] Geraint Jones. Programming in **occam** . Prentice-Hall International Series in Computer Science.

[Khan 7] Gilles Khan. Natural Semantics. Rapports de Recherche Nº 601 INRIA SOPHIA-ANTIPOLIS 06560 Valbonne, France. Février 1987.

[Li 8] Wei Li. An Operational Semantics of Tasking and Exception Handling in Ada. Department of Computer Science University of Edinburgh. December 1981.

[Milner 9] Robin Milner. A Calculus of Communicating Systems. Lecture notes in Computer Science. Springer-Verlag series nº.92.

[Plotkin 10] Gordon.D.Plotkin. A Structural Approach to Operational Semantics. Department of Computer Science, Äarhus University Denmark. Sept 1981.

[Plotkin 11] Gordon.D.Plotkin. An Operational Semantics for CSP. Department of Computer Science, University of Edinburgh. Internal Report CSR-114-82.

[Roscoe - Hoare 12] A.W.Roscoe, C.A.R Hoare. The laws of **occam** programming. Oxford University. Technical monograph PRG-53.

[Roscoe 13] A.W.Roscoe. Denotational Semantics for **occam**. Lecture notes In Computer Science. Springer-Verlag series nº.197.