A PETRI NET SEMANTICS OF OCCAM 2

J.G. Hall, R.P. Hopkins, O. Botti and F. De Cindio

TECHNICAL REPORT SERIES
No 329 October, 1991
A Petri Net Semantics of occam 2

J.G. Hall, R.P. Hopkins, O. Botti and F. De Cindio

Abstract

We describe a Petri Net model (the box model) which allows the production of a true concurrency semantics for a range of concurrent programming languages. In particular we give a true concurrency semantics for a large subset of occam 2 including data and priorities.
Bibliographical details

PETRI NET SEMANTICS OF OCCAM 2
A Petri net Semantics of occam 2
[By] J.G. Hall [and others].
Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Laboratory, 1991.
(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 329)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 329
HALL, Jon George

Abstract

We describe a Petri Net model (the box model) which allows the production of a true concurrency semantics for a range of concurrent programming languages. In particular we give a true concurrency semantics for a large subset of occam 2 including data and priorities.

About the author

Jon Hall is a Research Assistant in the Computing Laboratory.

Dr. R.P. Hopkins has been a lecturer in the Computing Laboratory since October, 1983, prior to which he was a Research Associate.

Fiorella De Cindio is a professor of Computing Science at the University of Milano, Italy.

Oliver Botti is a student of Computer Science at the University of Milano, Italy

All the authors are involved in the ESPRIT BRA DEMON

Suggested keywords

OCCAM 2
PETRI NETS
SEMANTICS

Suggested classmarks (primary classmark underlined)
Dewey (18th):
001.6424 511.5
U.D.C. 519.767 519.711
A Petri Net Semantics of occam 2

Jon Hall and Richard Hopkins
Computing Laboratory
Newcastle upon Tyne University
Newcastle upon Tyne
NE1 7RU
j.g.hall@uk.ac.newcastle
r.p.hopkins@uk.ac.newcastle

Oliver Botti and Fiorella De Cindio
Dipartimento di Scienze dell'Informazione
Università Statale di Milano
Milano

Abstract

We describe a Petri Net model (the box model) which allows the production of a
true concurrency semantics for a range of concurrent programming languages. In
particular we give a true concurrency semantics for a large subset of occam 2 in-
cluding data and priorities.

1. Introduction

We describe a Petri net based model (the box model) which may be used to give a Petri net semantics
to programs expressed in concurrent programming languages. In particular we give Place/Transition net
semantics to a subset of occam 2 for both the control flow and the data and control flow present therein.
The box model may then be considered as some form of consistency between the two models and also
with the higher level net models investigated in [Poigné90].

The subset of occam 2 [INMOS88] modelled in this paper is the atomic actions of STOP, SKIP, assign-
ment, and input and output communication, together with sequential (SEQ) and (prioritised) parallel
composition (PAR and PRI PAR), deterministic and (prioritised) non-deterministic choice (IF, ALT and
PRI ALT), iteration (WHILE), and the declaration and use of simple data and untyped channel variables
(INT, BOOL and CHAN). The parts of occam 2 not modelled are the expression of time within pro-
cesses, the use of data structures and typed channels.

There are two main variants of parallel composition within occam 2, PAR and PLACED PAR. The arms
of a PLACED PAR are assigned to distinct processors (if the hardware is sufficient), whereas the un-
adorned construct makes a single processor a shared resource. In the latter case an interleaving model
should be adequate as there is no true concurrency. Within this paper we regard occam 2 as a purely
concurrent language in that model of parallel composition is that of PLACED PAR. However only minor
modifications to the model would be required to model both.

The preservation of causality in sequential composition of two nets is by place multiplication. This ensures
that the firing of any initial transition within the second net be dependent on the firing of each final transition
in the first net. The placing of a silent (or τ) transition at the interface of two processes would have
a similar effect. This is described in [Botti/deCindio90].

1 this document is part of the first year report of the ESPRIT Basic Research Action No. 3148, (DEMON)
Operands of the ALT and PAR operators may be prioritised by prefixing the ALT or PAR by PRI. These modified operators are then non–commutative, in that the first operand has lower priority than the second, which has lower priority than the third, etc. The abstraction of priorities into the box model is by expressing them as a relation between transitions – the explicit representation of them is by methods such as those described in [Best/Koutny90]. The problems of modelling priorities within higher level net models is discussed in [Poigné90].

The box abstraction has similarities both to the work of Milner [Milner80] and Kotov [Kotov78].

2. Notation and Definitions

2.1. Notation

The following notation is used throughout this document:

- $2^A$ is the powerset of the set $A$
- $A^*$ is the set of (finite) strings on the set $A$
- $\cdot$ is the concatenation operator between strings
- $A \times B$ represents the Cartesian Product of the sets $A$ and $B$
- $\prod_{i=1}^{n} A_i = A_1 \times \ldots \times A_n$, an element of which is $(a_1, \ldots, a_n)$.
- $\langle a_1, \ldots, a_n \rangle_i = a_i$
- $\emptyset$ represents the empty set.

Functions are written as sets of maplets, where a maplet is a pair, representing an element of the domain and its image under the function. $f(e \rightarrow i)$, where $e \notin \text{dom } f$, is the map $f$ extended by the maplet $[e \rightarrow i]$. If the domains of two functions $f_1$ and $f_2$ are disjoint then their union, $f_1 \cup f_2$, is also a function.

$A \text{ del } f$, where $A$ is a set and $f$ a function, represents map deletion, such that $A \text{ del } f$ is a map such that $\text{dom} (A \text{ del } f) = \text{dom } f \setminus A$, whose values coincide with $f$ on the restricted domain. $f_1 \uplus f_2$, where $f_1, f_2$ are maps, represents map overwrite, that is $f_1 \uplus f_2$ is a map with domain $\text{dom } f_1 \cup \text{dom } f_2$ such that

$$(f_1 \uplus f_2)(x) = \begin{cases} f_2(x) & \text{if } x \in \text{dom } f_2 \\ f_1(x) & \text{otherwise} \end{cases}$$

$Ide$ is a set of identifiers. $Uid$, the set of unique identifiers resolve identifier ambiguities. $Exp$ are expressions with a syntax similar to that in [INMOS88], (except that variables may be represented either by identifiers or as unique identifiers).

$$\text{var} : \text{Exp} \rightarrow 2^{Ide}$$
$$\text{uid} : \text{Exp} \rightarrow 2^{Uid}$$

are functions which respectively: take an expression and return all identifiers which occur in the expression; take an expression and return all unique identifiers which occur in the expression.

$e[a/\mathfrak{f}]$ where $e \in Exp, \mathfrak{f} \in Ide$, is the simultaneous textual replacement of all occurrences of the variable identifier $\mathfrak{f}$ in $e$ by $a$. If $\mathfrak{f}$ does not appear in $e$, $e$ is not changed.

The (component–wise disjoint) union of nets is the pairwise union of the components of the nets.
2.2. Place/transition systems and safeness

2.2.1. Definition [P/T net]

A triple \( \Sigma = (S, T; F) \) is called a P/T net iff

- \( S \cap T = \emptyset \)
- \( F \subseteq (S \times T) \cup (T \times S) \)

All definitions involving P/T nets allow for the possibility of isolated places. The traditional definition may be found in [Best/Fernandez87].

2.2.2. Definition [Place/transition system]

A six-tuple \( \Sigma = (S, T; F, K, W, M_0) \) is called a place/transition system (P/T system) iff:

- \( (S, T; F) \) is a P/T net.
- \( K : S \to \mathbb{N}^+ \cup \{\infty\} \) a capacity function
- \( W : F \to \mathbb{N}^+ \) a weight function
- \( M_0 : S \to \mathbb{N} \) an initial marking function satisfying
  \[ M_0(s) \leq K(s) \quad \text{for all} \quad s \in S. \]

The capacity function \( K \) gives the maximum number of tokens which can occupy a place at any time. The weight function \( W \) gives the number of tokens which must flow along an arc. Throughout this paper we assume infinite capacity and unitary arc weight. In fact the nets we generate are all I-safe in that although the capacity is infinite there can only be a single token on any place at any one time.

Because of this and as a notational convenience we will omit the \( K \) and \( W \) fields of P/T nets, writing them as four-tuples, \( \Sigma = (S, T; F, M_0) \)

2.2.3. Definition [pre-set, post-set, marking, enabled, firing, \( M[A]M' \), \( M[H]M' \), reachable markings]

Let \( \Sigma = (S, T; F, M_0) \) be a P/T system.

- For \( x \in S \cup T \) define
  \[ x^* = \{ y \mid (y, x) \in F \} \quad \text{(pre-set of } x) \]
  \[ x^\ast = \{ y \mid (x, y) \in F \} \quad \text{(post-set of } x) \]
- A function \( M : S \to \mathbb{N} \) is called a marking of \( \Sigma \).
- A non-empty set of transitions \( A \subseteq T \) is enabled at \( M \) if
  \[ \forall s \in T : M(s) \geq |s^\ast \cap A| \]
- If \( A \subseteq T \) is a non-empty set of transitions which is enabled at a marking \( M \) then \( A \) is said to be a single step from \( M \) to \( M' \) if
  \[ \forall s \in S : M'(s) = M(s) - |s^\ast \cap A| + |s^* \cap A|. \]
• The **firing** of \( A \) changes the marking \( M \) into the new marking \( M' \); we denote this as \( M[A]M' \).

• If \( A = \{ \epsilon \} \) then we write \( M[\epsilon]M' \).

• The **reachable markings**, denoted by \( [M_0] \), is the smallest set of markings of \( \Sigma \) such that:

\[
M_0 \in [M_0];
\]

if \( M_1 \in [M_0] \) and \( M_1[\tau]M_2 \) for some \( \tau \in T \) then \( M_2 \in [M_0] \).

2.2.4. Definition **[safeness]**

Let \( \Sigma = \langle S, T; F, K, W, M_0 \rangle \) be a P/T system

\[
s \in S \text{ is } n\text{-safe } (n \in \mathbb{N}) \text{ if } \forall M \in [M_0]: M(s) \leq n;
\]

\( \Sigma \) is \( n\)-safe \( (n \in \mathbb{N}) \) if \( \forall s \in S : s \) is \( n\)-safe

All P/T systems we produce are 1-safe allowing us to interpret a marking as the characteristic function of a subset of \( S \).

3. **Syntactic and Semantic Framework**

3.1. **Introduction**

Below appears the **abstract syntax** used throughout the rest of the document. Also defined are the **semantic domains** used in the denotational semantics.

Semantic domains are labelled with acronyms of three letters, members of which are labelled consistently by greek or latin letters, modified as appropriate by subscripts or primes.

3.2. **The Abstract Syntax**

3.2.1. **Definition [domains]**

The basic domains over which the abstract syntax is defined are:

- \( e \in \text{Exp} \) **Expressions**,  
- \( \mathcal{I} \in \text{Ide} \) **Identifiers**, taken from some set, used to represent both variables and channels,  
- \( b \in \text{Bas} \)  
  - \( \text{Bases} = \{ \chi : = e, \, \chi \text{le}, \, \chi?x, \, r, \, \phi \} \)  
  - where \( x, \, \chi \in \text{Ide}, \, e \in \text{Exp} \),  
- \( V \in \text{Vty} \) **Variable Types** = \( \langle \text{BOOL, INT} \rangle \),  
- \( C \in \text{Cty} \) **Channel Types** = \( \langle \text{CHAN} \rangle \),  
- \( T \in \text{Typ} \)  
  - **Types** = \( \text{Vty} \cup \text{Cty} \),  
- \( \Gamma \in \text{Com} \) **Commands**,.

Elements of \( \text{Typ} \) are often identified with the types themselves, for example we may say that \( T \in \text{Typ} \) and \( x \in T \) to mean that \( T \) is a type and \( x \) is an element (or variable) of that type.
Bases are the atomic actions in occam 2 – STOP, SKIP, assignment, channel input and channel output. STOP and SKIP are denoted $\phi$ and $\tau$ respectively.

3.2.2. Definition [abstract syntax]

The subset of occam 2 modelled is the language accepted by the following grammar. A command $\Gamma$ can be one of:

1. $b$ – a base
2. $\Gamma_1;\Gamma_2$ – sequential composition (SEQ)
3. $\Gamma_1 \parallel \Gamma_2$ – parallel composition (PAR)
4. $\Gamma_1 \ll \Gamma_2$ – prioritized parallel composition (PRI PAR)
5. $e \in \Gamma$ – guarded command (single-arm IF)
6. $e \in \Gamma_1 \lor \Gamma_2$ – deterministic choice (multi-arm IF)
7. $e \&\& b \in \Gamma$ – guarded command (single-arm ALT)
8. $e \&\& b \in \Gamma_1 \triangleleft \Gamma_2$ – non-deterministic choice (multi-arm ALT)
9. $e \&\& b \in \Gamma_1 \ll \Gamma_2$ – prioritised non-deterministic choice (PRI ALT)
10. $*e \in \Gamma$ – guarded iteration (WHILE)
11. $\forall \beta : \Gamma$ – scoping of a data variable
12. $C\beta : \Gamma$ – scoping of a channel variable

Whereas there is no procedural or functional abstraction given in the above syntax, this is due to recursion within occam 2 being limited to instances which allow the compile-time allocation of resources, i.e. iteration. The definition of functions and procedures is limited, in effect, to macro expansion – for any program which uses functions or procedures there is an equivalent program without ([INMOS88]). This abstract syntax does not allow the modelling of time nor the manipulation of data structures. The current state of work on the modelling of these is described in [Hopkins/Hall90].

We use only a binary form of sequential and parallel composition, and only unary and binary choice composition. Other arities are expressible by decomposition into these.

We use the above syntax plus parenthesisation for written examples.

3.2.3. Definition [semantic domains]

The semantic domains over which the denotational semantics is defined are:

- $n \in \mathbb{N}$
- $s \in Pla$
- $t \in Tra$
- $\Sigma \in Nets = 2^{Pla} \times 2^{Tra} \times ((2^{Pla} \times 2^{Tra}) \cup (2^{Tra} \times 2^{Pla}))$
- $Uid = Typ \times [L, R]^* \times \mathbb{N}$

**Non-negative Integers**
**Places**
**Transitions**
**Nets**
**Unique Identifiers**
\[ f \in LcY = \text{Ide} \rightarrow \text{Uid} \]

\[ L \in \text{Lab} = \text{Tra} \rightarrow \text{Exp} \times \text{Bas} \times \text{Lcy} \]

\[ P \in \text{Box} = \text{Nets} \times \text{Lab} \times (2^{\text{Bas}})^3 \times (2^{\text{Lcy}})^3 \]

\[ q \in \text{Env} = \text{Lcy} \times [L,R]^* \times \text{N} \]

**Localities**

**Labellings**

**Boxes**

**Environments**

A *unique identifier* consists of a type, a *string* on \( \{L,R\} \) and a *natural number*. It assigns a unique identity to each variable by recording the point of declaration of that variable as: a string on \( \{L,R\} \) which gives the position within the (binary) parse tree at which the declaration appeared as a *sequence of arms* to take at bifurcations (\( L \) for left, \( R \) for right); a natural number which gives the position within a (possibly multiple) declaration at that point. The type is the type of the variable.

A *locality* is a map from identifiers to unique identifiers.

An *environment* is that in which a program fragment should be regarded. It is constructed from the parse tree (which determines the string and number fields in the same way as for a unique identifier), and a locality which is the current one for the fragment.

The *label* of a transition holds semantic information from statements in the *occam 2* source such as the guard and action, and an environment for that statement.

### 4. The Box Model

#### 4.1. Introduction

An *occam 2* program may interact with its environment in only two ways – by communicating across shared channels, or by altering shared data. We call these actions the *interface* of an *occam 2* process. A *box* is a labelled Petri net together with two distinguished sets of places and two distinguished sets of transitions representing this interface. The Petri net describes the control flow of the *occam 2* process. Places represent local (control) state and transitions represent (atomic) actions. The distinguished sets of places are called *entry* and *exit* places respectively. The box begins execution by having a single token placed on each of its entry places, and has terminated when all of its exit places are marked. The distinguished sets of transitions are *input* and *output* transitions, representing the actions of input and output communication respectively.

#### 4.2. \( \beta \)-expressions

A *\( \beta \)-expression* (proposed in [Best90]) is a pair comprising a boolean expression (or guard) and an atomic action. The behaviour of a \( \beta \)-expression is to allow the action to occur if and only if the boolean guard evaluates to true within the current environment.

**4.2.1. Definition [\( \beta \)-expression]**

A \( \beta \)-expression is a pair \( (e,b) \in \text{Exp} \times \text{Bas} \), where \( e \) is a *boolean guard* and \( b \) is a *base*.

### 4.3. Boxes

#### 4.3.1. Definition [box]

A *box* is a seven tuple \( P = \langle \Sigma, L, R, B, E, I, O \rangle \) where

- \( \Sigma = \{S, T, F\} \)

is a \( P/T \) net
• \( L \in \text{Lab} \) is a labelling function
• \( R \subseteq T \times T \) is a priority relation
• \( B \subseteq S \) Entry Places
• \( E \subseteq S \) Exit Places
• \( I \subseteq T \) Input Transitions
• \( O \subseteq T \) Output Transitions

where
• \( B \neq \emptyset \land E \neq \emptyset \)
• \( I \cap O = \emptyset \)

\( L \) assigns to a transition a \( \beta \)-expression and the locality of the statement from which it was produced. \( R \) is a relation between transitions of the net and represents the pairs of transitions which have had a priority expressed between them. \( R \) is not necessarily anti-symmetric, as would perhaps be expected in the modelling of priorities, as it is possible in occam 2 to specify that processes \( a \) and \( b \) have greater priority than each other simultaneously such as the case of the following occam 2 fragment:

```
PAR
  PRI PAR
    c ! 1
    d ! 2
  PRI ALT
    d ? x & SKIP
    c ? y & SKIP
```

The representation of such a situation in terms of nets is left to the explicit modelling of priorities by, for instance, those in [Best/Koutny90].

Unless otherwise stated all distinct boxes are disjoint, i.e. if \( P_1 \) and \( P_2 \) are two boxes then either \( P_1 = P_2 \) or their components are pairwise disjoint.

### 4.4. Composition Operators on boxes

We define two operators on nets, place and transition multiplication, which are used in the definition of the composition operators on boxes.

Interleaved with the definitions are examples of the operators. As we wish to keep the figures clear we refrain from labelling the places and transitions in the interfaces of the processes. However they are distinguished by position – in any net entry places appear higher up the page than exit places, input transitions appear to the left of the page whereas output transitions appear to the right. Any internal transition or place falls within the extremes of the net.

We describe the examples in a CSP type language to allow a compact representation. Note that in occam 2 the examples for alternation and iteration would include boolean guards. We use an overscore to indicate an input communication, while leaving outputs unadorned.

Throughout the rest of this section, let
4.4.1. Definition [place multiplication]

The place multiplication operator

\[ \wedge : \text{Nets} \times ((2^\text{Net}) \times (2^\text{Net})) \to \text{Nets} \]

is such that if \( N = (S, T, F) \), \( S_1 \cap S_2 = \emptyset \) and \( S_i \subseteq S \), \( i \in [1, 2] \) then

\[ N \wedge (S_1, S_2) = (S', T, F') \]

where

- \( S' = S \setminus (S_1 \cup S_2) \cup (S_1 \times S_2) \)
- \( F' = FR \cup NF \)

where

- \( FR = F \cap ((S' \times T) \cup (T \times S')) \)
- \( NF = \{ (t, (s_1, s_2)) \in T \times (S_1 \times S_2) \mid \exists i \in [1, 2] . (t, s_i) \in F \} \)
- \( NF = \{ (s_1, s_2, t) \in (S_1 \times S_2) \times T \mid \exists i \in [1, 2] . (s_i, t) \in F \} \)

![Diagram of place multiplication]

Figure 4.4.2. - an example of place multiplication

In figure 4.4.2. an example of place multiplication is given. The two sets \( \{r_1, r_2\} \) and \( \{s_1, s_2, s_3\} \) are multiplied to give the set of places \( \{\langle r_1, s_1\rangle, \langle r_1, s_2\rangle, \langle r_1, s_3\rangle, \langle r_2, s_1\rangle, \langle r_2, s_2\rangle, \langle r_2, s_3\rangle\} \). Each new place inherits the flow of both components.

4.4.3. Definition [transition multiplication]

The transition multiplication operator
\[ \land : \text{Nets} \times ((2^\text{fin}) \times (2^\text{fin})) \rightarrow \text{Nets} \]

and is defined as in place multiplication, with the roles of places and transitions interchanged. \( \lor \) and \( \land \) associate left to right.

We now define composition operators on boxes. These correspond to those underlying occtam 2.

4.4.4. Definition [sequential composition]

The sequential composition operator on boxes, \( ; \), is defined to be

\[ P_1 ; P_2 = (\Sigma, L_1 \cup L_2, R_1 \cup R_2, B_1, E_2, I_1 \cup I_2, O_1 \cup O_2) \]

where

- \( \Sigma = (\Sigma_1 \cup \Sigma_2) \land (E_1, B_2) \)

\[ a ; (b \parallel c) \]

Figure 4.4.5. Sequential Composition

\( a ; (b \parallel c) \), the sequential composition of \( a \) with the process consisting of \( b \) and \( c \) in parallel.

4.4.6. Definition [choice composition]

The choice composition operator on boxes, \( \square \), is defined to be

\[ P_1 \square P_2 = (\Sigma, L_1 \cup L_2, R_1 \cup R_2, B_1 \times B_2, E_1 \times E_2, I_1 \cup I_2, O_1 \cup O_2) \]

where

- \( \Sigma = (\Sigma_1 \cup \Sigma_2) \land (B_1, B_2) \land (E_1, E_2) \)

\[ a \square b \]

Figure 4.4.7. Choice
a □ b, the non-deterministic choice of a and b in which the entry places and exit places of each process are multiplied.

4.4.8. Definition [prioritised choice composition]

The prioritised choice composition operator on boxes, □<, is defined to be

\[ P_1 □ P_2 = (\Sigma, L_1 \cup L_2, R, B_1 \times B_2, E_1 \times E_2, I_1 \cup I_2, O_1 \cup O_2) \]

where

- \( \Sigma = (\Sigma_1 \cup \Sigma_2) \land (B_1, B_2) \land (E_1, E_2) \)

- \( R = R_1 \cup R_2 \cup \{ (t_1, t_2) \in T_1 \times T_2 \mid \ \cdot \ t_1 \subseteq B_1 \land (B_2 \subseteq B_2) \} \)

The modified R component represents the fact that each initial transition in the second net is given priority over each initial transition in the first net.

4.4.9. Definition [iterative composition]

The iterative operator on boxes, *, is defined to be

\[ * P_1 = (\Sigma, L_1, R_1, B_1 \times E_1, [s_{new}], I_1, O_1) \]

where

- \( \Sigma = (\Sigma_1 \cup [s_{new} \emptyset, \emptyset]) \land (B_1, E_1) \)

\( s_{new} \) is a new place distinct from any in \( S_1 \). If we were to regard the iteration as an infinite sequence of transitions (connected by an infinite sequence of places) then this new exit place represents the 'place at infinity'.

![Figure 4.4.10. Iteration](image)

* a. the iteration of a in which we multiply the entry and exit places of a and append a new place as the entry place.

4.4.11. Definition [communication alphabet]

For every \( ch \in Ide, i \in [1, 2] \) define

\[ ^{\cdot}C_i = \{ t \in L_i \exists x \in Ide \cup Uid, e \in Exp. \quad L_i(t) = ch?x \land \exists t' \in O_{i \rightarrow} L_{i \rightarrow}(t) = ch{:e} \} \]
\[C_i^0 = \{ t \in O_i | \exists x \in \text{Ide} \cup \text{Uid}, a \in \text{Exp} \} \]

\[L_2(t_2) = ch!e \land \exists t' \in I_2. L_3(t_2) = ch?x\]

\[C_i^1 \text{ are those input transitions in } P, \text{ which have corresponding output transitions in } P_{out} \text{ and so would be matched in the parallel composition of the two processes. Similarly for } C_i^2.\]

Let \([c_1, \ldots, c_n] \text{ be the set of those } ch \text{ for which } 5C_i^1 \text{ (and hence } 6C_i^1) \text{ are non-empty, and } [d_1, \ldots, d_n] \text{ the set of those } ch \text{ for which } 6C_i^2 \text{ (and hence } 6C_i^2) \text{ are non-empty.}\]

Let

\[C_1 = \bigcup_{j \in [1, \ldots, n]} (6C_i^1 \times 6C_i^1) \]

\[C_2 = \bigcup_{j \in [1, \ldots, n]} (6C_i^1 \times 6C_i^2) \]

\[C_1 = \bigcup_{j \in [1, \ldots, n]} 5C_i^1 \]

\[C_2 = \bigcup_{j \in [1, \ldots, n]} 5C_i^2 \]

\[C = C_1 \cup C_1 \cup C_2 \cup C_2 \]

\[C \text{ is then the communication alphabet. It is the set of transitions which will be matched in the parallel composition of the two nets and is used, together with the other sets defined above, in the definition of parallel composition.}\]

### 4.4.12. Definition [parallel composition]

The parallel composition operator, \( \parallel \), is defined to be

\[P_1 \parallel P_2 = \langle \Sigma, L, R, B_1 \cup B_2, E_1 \cup E_2, I, O \rangle \]

where \( \Sigma, L, R, I, \) and \( O \) are defined as follows:

For each \( i \in [1, 2] \) and \( \langle t, t' \rangle \in C_i \), define

\[\text{join}(t, t') \equiv x := e \quad \text{where } L_i(t) = ch!x \land L_{3-i}(t') = ch?e\]

for some \( x \in \text{Ide} \cup \text{Uid}, e \in \text{Exp} \)

Define \( G : C_1 \cup C_2 \rightarrow \text{Exp} \times \text{Bas} \times \text{Lcy} \) such that

\[G(t, t') = \langle L_1(t_1) \land L_2(t_2), \text{join}(t, t'), L_1(t_3) \land L_2(t_3) \rangle \]

(where \( \land \) between two boolean expressions is equivalent to the occam \( 2 \land \) operator and so produces an expression which has value the logical \( and \) of the two component expressions).

Then

- \( \Sigma = (\Sigma_1 \cup \Sigma_2) \land \langle 6C_i^1, 6C_i^1 \rangle \land \ldots \land \langle 6C_i^2, 6C_i^2 \rangle \land \langle 5C_i^1, 5C_i^1 \rangle \land \ldots \land \langle 5C_i^2, 5C_i^2 \rangle \)

- \( L = (C \text{ del } (L_1 \land L_2)) \land G \)
4.4.13. Definition [prioritised parallel composition]

The prioritised parallel composition operator, \( \lll \), is defined to be

\[
P_i \lll P_2 = (\Sigma, L, R, B_1 \cup B_2, E_1 \cup E_2, I, O)
\]

where \( \Sigma, L, I, \) and \( O \) are defined as in ordinary parallel composition, \( R \) as follows:

\[
R = \left[ (t_1, t_2) \mid (t_1, t_2), (t_2, t_1) \in (T_1 \times T_2 \cup T_2 \times T_1) \setminus (C_1 \cup C_2) \right]
\]

\[
\cup \left[ \left( (t_1, t_2), t \right) \mid (t_1, t_2) \in C_1 \cup C_2 \land t \in T_1 \cup T_2 \land ((t_1, t) \in R_1 \lor (t_2, t) \in R_2) \right]
\]

\[
\cup \left[ \left( (t_1, t_2), t \right) \mid (t_1, t_2) \in C_1 \cup C_2 \land t \in T_1 \cup T_2 \land ((t_1, t) \in R_1 \lor (t_2, t) \in R_2) \right]
\]

\[
\cup \left[ \left( (r_1, r_2), (t_1, t_2) \right) \mid (r_1, r_2), (t_1, t_2) \in C_1 \cup C_2 \land ((r_1, t_1) \in R_1 \lor (r_2, t_2) \in R_2) \right]
\]

\[
\cup \left[ \left( r, t \right) \mid r \in T_1 \land t \in T_2 \cup C_1 \cup C_2 \right]
\]

The above formulae for \( R \) state: that in the case of unprioritised parallel composition priorities are preserved for transitions not involved in communication and for others the priorities of the communication is inherited from its components. In the case of the prioritised parallel composition, the extra term records that everything in the second box has priority over everything in the first.

\[\bar{a} \parallel (a \boxdot b), \] the parallel composition of two processes. Shared channels are multiplied and become internal and unmatched channels are promoted to be channels of the composed process.
5. Denotational Semantics

5.1. Introduction

We define a denotational semantics which will be used to 'compile' occam 2 processes into boxes. The representation of the control flow of the process is explicitly represented in the structure of the net. Data flow information is represented in transition labelling, which will be expanded later.

5.2. The Denotational Semantics

Throughout the following places or transitions created are taken to be unique.

5.2.1. Definition [valuation functions]

\[ \mathcal{B} : \text{Exp} \times \text{Bas} \rightarrow \text{Env} \rightarrow \text{Box} \]

\[ \mathcal{C} : \text{Com} \rightarrow \text{Env} \rightarrow \text{Box} \]

\[ \mathcal{B} : \text{Box} \rightarrow \text{Ide} \rightarrow \text{Box} \]

\[ \mathcal{I} : \text{Box} \rightarrow \text{Ide} \rightarrow \text{Box} \]

\( \mathcal{B} \) – base valuation – has the effect of combining a guarded atomic action (\( \beta \)-expression) with an environment to produce a box.

\[ \mathcal{B}[(e, b)(f, s, n)] = \langle [s_1, s_2], T, F, L, \emptyset, [s_1], [s_2], I, O \rangle. \]

where

\[
T = \begin{cases} \emptyset & \text{if } b = \phi (\text{STOP}) \\ [t] & \text{otherwise} \end{cases}
\]

\[
F = \begin{cases} \emptyset & \text{if } b = \phi (\text{STOP}) \\ [s_1 \rightarrow t, t \rightarrow s_2] & \text{otherwise} \end{cases}
\]

\[
I = \begin{cases} [t] & \text{if } b = \#e' \\ \emptyset & \text{otherwise} \end{cases}
\]

\[
O = \begin{cases} [t] & \text{if } b = \#_1, \#_2 \\ \emptyset & \text{otherwise} \end{cases}
\]

\[
L = \begin{cases} [t \rightarrow (e, b, f)] & \text{if } T \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}
\]

Note that STOP consists of just two places.
$\mathcal{B}$ - channel hiding - a partial function which maps a box onto a box and has the effect of hiding a channel which has just gone out of scope.

$$\mathcal{B}(P)_\emptyset = \langle S, T', F', L', R', B, E, I, O \rangle$$

where

- $T' = T \setminus \{t \in I \cup O \land \emptyset \in \text{var}(L(t) \cup 2)\}$
- $F' = F \cap ((T' \times S) \cup (S \times T'))$
- $L' = \{t \to (e, b, f') | t \in T' \land t \to (e, b, f) \in L \land f = [\emptyset] \text{ del } f\}$
- $R' = R \cap (T' \times T')$
- $I' = I \cap T'$
- $O' = O \cap T'$

provided $\emptyset \in \text{dom } f$; $\mathcal{B}$ removes all transitions that involve communication on channel $\emptyset$.

$\mathcal{J}_\emptyset$ - variable hiding - a partial function which maps a box onto a box and has the effect of hiding a variable which has just gone out of scope.

$$\mathcal{J}_\emptyset(P)_\emptyset = \langle S, T, F, R, L', B, E, I, O \rangle$$

where

- $L' = \{t \to (e', b', f') | t \to (e, b, f) \in L \land e' = e[f(\emptyset)/\emptyset] \land b' = b[\emptyset]/\emptyset \land f' = [\emptyset] \text{ del } f\}$

provided $\emptyset \in \text{dom } f$. $\mathcal{J}_\emptyset$ replaces all references to $\emptyset$ by its unique identifier.

$\mathcal{E}$ - command evaluation - takes a command and an environment and produces a box according to the command being one of:

1. $\mathcal{E}[b]_\emptyset = \mathcal{B}[(\text{TRUE}, b)]_\emptyset$
2. $\mathcal{E}[\Gamma_1 \upharpoonright \Gamma_2][f, s, n] = \mathcal{E}[\Gamma_1][f, s, (L), 0] \cup \mathcal{E}[\Gamma_2][f, s, (R), 0]$
3. $\mathcal{E}[\Gamma_1 \parallel \Gamma_2][f, s, n] = \mathcal{E}[\Gamma_1][f, s, (L), 0] \parallel \mathcal{E}[\Gamma_2][f, s, (R), 0]$
4. $\mathcal{E}[\Gamma_1 \mid \Gamma_2][f, s, n] = \mathcal{E}[\Gamma_1][f, s, (L), 0] \mid \mathcal{E}[\Gamma_2][f, s, (R), 0]$
5. $\mathcal{E}[\ast e \vdash \Gamma]_\emptyset = \mathcal{B}[(\lnot e, r)]_\emptyset \cup \ast [e \vdash \Gamma]_\emptyset$
6. $\mathcal{E}[\forall \emptyset : \Gamma][f, s, n] = \mathcal{J}_\emptyset \mathcal{E}[\Gamma][f, s, (V, s, n), s, n + 1]_\emptyset$
7. $\mathcal{E}[\exists \emptyset : \Gamma][f, s, n] = \mathcal{B}[\Gamma][f, s, (C, s, n), s, n + 1]_\emptyset$
8. $\mathcal{E}[e \vdash \Gamma]_\emptyset = \mathcal{B}[(e, r)]_\emptyset : \mathcal{E}[\Gamma]_\emptyset$
9. \( C[e \vdash \Gamma_1 \cup \Gamma_2]_{f,s,n} = C[e \vdash \Gamma_1]_{f,s,(L),0} \sqcap C[\neg e \vdash \Gamma_2]_{f,s,(R),0} \)

10. \( C[e \& b \vdash \Gamma]_0 = \exists ! \{(e, b)\}_0 ; C[\Gamma]_0 \)

11. \( C[e \& b \vdash \Gamma_1 \cap \Gamma_2]_{f,s,n} = C[(e \& b) \vdash \Gamma_1]_{f,s,(L),0} \sqcap C[\Gamma_2]_{f,s,(R),0} \)

12. \( C[e \& b \vdash \Gamma_1 \sqsubseteq \Gamma_2]_{f,s,n} = C[(e \& b) \vdash \Gamma_1]_{f,s,(L),0} \sqsubseteq C[\Gamma_2]_{f,s,(R),0} \)

More informally:

1. if a base then produce a box directly by passing the base and the environment to the base valuation function.

2. if the sequential composition of two commands then produce the net which is the sequential composition of the nets of the two commands, each evaluated in the current environment, recording the split in the parse tree by adding to s.

3. if the parallel composition of two commands then produce the net which is the parallel composition of the nets of the two commands, each evaluated in the current environment, recording the split in the parse tree by adding to s.

4. if the prioritised parallel composition of two commands then produce the net which is the prioritised parallel composition of the nets of the two commands, each evaluated in the current environment, recording the split in the parse tree by adding to s.

5. if the iteration of a guarded command then produce the net which is the choice of either the iteration of the guarded command, or SKIP; each guarded by the negation of the guard corresponding to the continuation or termination of the iteration respectively.

6. if a variable declaration followed by a command then produce the net which is that of the command evaluated within the environment modified by the declaration, with the variable just declared hidden.

7. if a channel declaration followed by a command then produce the net which is that of the command evaluated within the environment modified by the declaration, with the channel just declared hidden.

8. if a boolean guarded command then produce the net consisting of a SKIP guarded by the boolean composed sequentially with the net of the command.

9. if a deterministic choice consisting of a guarded command and a command then produce the net consisting of the choice between the two arms (the second having an extra guard – the negation of the first’s guard).

10. if an (input) guarded command then produce the net consisting of the input guarded command composed sequentially with the net of the command.

11. if an (input) guarded command in a non–deterministic choice with a command then produce the net which is the choice of the input guarded command and the command.

12. if a prioritised (input) guarded command in a non–deterministic choice with a command then produce the net which is the prioritised choice of the input guarded command and the command.

5.3. Example of the translation of occam 2 to boxes

We are now in a position to be able to present a small example of the translation from occam 2 to boxes (Figure 5.3.1). As in the previous examples members of the interfaces are distinguished only by position.

6. Mapping to Data and Control flow

With the box constructed by section 5 we may perform the final stages of the translation.
6.1. Closure

For the rest of the document we will assume that all occam 2 processes are closed.

6.1.1. Definition [closure of an occam 2 process]

An occam 2 process is said to be closed if each channel and variable identifier within the process has been scoped by a channel or variable declaration.

6.1.2. Definition [closure of a box]

A box is said to be closed if the occam 2 process of which it is the image under the denotational semantics was closed.

A process is closed only when each identifier has been typed. This allows the creation of the places representing the values of the variables. While closure is not strictly necessary as a prerequisite of the modelling of control flow, we will require it for consistency between the two models.

6.2. Control Net

The control net of a box is the projection of that box onto the underlying net.

6.2.1. Definition [control flow projection]

Given a box \( P = \langle \Sigma, L, R, B, E, I, O \rangle \) define the control net of \( P \), to be the P/T system
\[ cf(P) = (\Sigma, M_0) \]

where

- \( M_0 = B \)

6.3. Data and Control Flow Net

There are three sections to creating the data and control flow net from the box model – the creation of places representing data, the determination of an initial marking, and the refinement (or \( \beta\)-splitting) of transitions to take account of their dependence on data.

For the data places we create a single place with no connections for each possible value for each unique identifier that appears in the box labelled as a value/unique identifier pair. Closure ensures that this is possible for each transition that appears in the box.

The initial marking of a box is the set of its initial places. In the initial marking of the data places we make the assumption that variables are ‘own’ variables – each time a declaration is encountered, other than the first, the variable is given the value it had the last time it passed out of scope. In the case when all variables are explicitly initialised this strategy is adequate in that it does not introduce any extra non-determinism into a program.

When performing \( \beta\)-splitting we can be sure that transitions in our annotated net do not contain references to channel communications. This is for one of three reasons:

- they have been removed by \( B \) as there are variables which were used within them but were not declared
- because they have been hidden by \( B \), as the channel name has passed out of scope
- because communication on that channel has been matched and so the labelling on that transition has changed to an assignment.

So without loss of generality we may assume that the transitions labels have action field either \( \tau \) (\( SKIP \)) or assignment.

Also as the process is closed there are no explicit references to variables within the transition labels as all have been replaced by their unique identifiers by the hiding operators. We use variables to mean unique identifiers. Within an annotation an action may contain variables to which there is both read/write access, although it is only in the case of assignment when write access is needed and then only to a single variable.

\( \beta\)-splitting is accomplished by splitting each transition into transitions for each possible combination of values of variables that appear within that transition’s annotation and for which the guard evaluates to \( true \). Each of the new transitions has the same control flow linkage as that of the original transition. There are flow loops from it to each data value place which is used in the expressions and, in the case of assignment, flow arcs from the original values of the data places representing the target of the assignment to the transition and from the transition to the new value of the target of the assignment. We treat the cases of read access and read and write access seperately.

6.3.1. Definition [data places]

The data places of a box \( P \) are defined as

\[ D(P) = \{ (v, \lambda) | \lambda \in uid(P) \land v \in \lambda \downarrow 1 \} \]

Note that for infinite data domains we have infinite nets.
6.3.2. Definition [initial marking of data places]

For a box \( P = (\Sigma, L, R, B, E, I, O) \) define the initial marking of data places

\[
\text{Init}(P) = \{ (\nu, \lambda) \in D(P) \}
\]

where

\[
\forall (\nu_1, \lambda_1), (\nu_2, \lambda_2) \in \text{Init}(P) . \ (\lambda_1 = \lambda_2 \Rightarrow \nu_1 = \nu_2)
\]

Notice that by this definition the choice of initial marking is not deterministic.

6.3.3. Definition [\( \beta \)-splitting of assignment]

Let \( P \) be a closed box. For each transition \( t \) in \( P \) such that

\[
L(t) = \langle e_1, x := e_2, f \rangle
\]

Define the \( \beta \)-splitting of a transition \( t \), \( \beta(t) \), by the following: Let

\[
\{(x_1, \ldots, x_m, b_1, \ldots, b_m) | x_i, b_j \in (uid(e_1) \cup uid(e_2)) \setminus \{x\} \land x_i \downarrow 1 = \text{INT} \land b_j \downarrow 1 = \text{BOOL} \}
\]

be the set of unique identifiers which appear in \( e_1 \) and \( e_2 \) (with the exception of \( x \) if it appears in either).

Suppose \( x \downarrow 1 = T \), (i.e. the variable represented \( x \) has type \( T \)) then for every \( c, k \in T \) define

\[
\alpha(t, c, k) = \begin{cases} 
\{ (\nu_1, \ldots, \nu_m, \beta_1, \ldots, \beta_m) | \nu_i \in \text{INT} \land \beta_j \in \text{BOOL} \} & \text{if } |uid(e_1) \cup uid(e_2) \setminus \{x\}| = 0 \\
\emptyset & \text{otherwise}
\end{cases}
\]

\[
\text{if } e_1[v_i/x, \beta_i/b_i, c/x] = \text{TRUE} \land e_2[v_i/x, \beta_i/b_i, c/x] = k
\]

\[
\text{if } e_1[c/x] = \text{TRUE} \land e_2[c/x] = k
\]

For each \( a \in \alpha(t, c, k) \) create a transition \( \beta(t, a, c, k) \) with connectivity

\[
\cdot \beta(t, a, c, k) = \cdot t \cup \bigcup_{i \in \{1, \ldots, m\}} [(v_i, x_i) \cup \bigcup_{i \in \{1, \ldots, m\}} [(\beta_i, b_i)] \cup [(c, x)] \\
\beta(t, a, c, k) = t \cdot \bigcup_{i \in \{1, \ldots, m\}} [(v_i, x_i) \cup [(\beta_i, b_i)] \cup [(k, x)]
\]

where \( (v_i, x_i) \in \{1, \ldots, m\}, (\beta_i, b_i) \in \{1, \ldots, m\}, (c, x), (k, x) \in D(P) \).

Then

\[
\beta(t) = \{ \beta(t, a, c, k) | c, k \in T, a \in \alpha(t, c, k) \}
\]
6.3.4. Definition [β-splitting of τ]

Let $P$ be a closed box. For each transition $τ$ in $P$ such that

$$L(τ) = \{e_1, τ, f\}$$

Define the β-splitting of a transition $τ$, $β(τ)$, by the following: Let

$$[(x_1, \ldots, x_n, b_1, \ldots, b_m) | x_i, b_j ∈ \text{uid}(e_1) \land x_i \downarrow 1 = \text{INT} \land b_j \downarrow 1 = \text{BOOL}]$$

be the set of unique identifiers which appear in $e_1$, let

$$α(τ) = \begin{cases} 
[(v_1, \ldots, v_n, β_1, \ldots, β_m) | v_i ∈ \text{INT} \land β_j \in \text{BOOL}] & \text{if } |\text{uid}(e_1)| ≠ 0 \land e\{v_i/x_i, β_j/b_j\} = \text{TRUE} \\
[β] & \text{if } |\text{uid}(e_1)| = 0 \land e_1 \equiv \text{TRUE} \\
\emptyset & \text{otherwise}
\end{cases}$$

For each $a \in α(τ)$ create a transition $β(τ, a)$ with connectivity

$$\cdot β(τ, a) = \cdot τ \cup \bigcup_{i \in [1, \ldots, n]} [(v_i, x_i)] \cup \bigcup_{i \in [1, \ldots, m]} [(β_i, b_i)]$$

$$β(τ, a) \cdot = τ \cdot \bigcup_{i \in [1, \ldots, n]} [(v_i, x_i)] \cup \bigcup_{i \in [1, \ldots, m]} [(β_i, b_i)]$$

where $[(v_i, x_i)_{i \in [1, \ldots, n]}, (β_i, b_i)_{i \in [1, \ldots, m]}] \in D(τ)$.

Then

$$β(τ) = [β(τ, a) | a \in α(τ)]$$

6.3.5. Definition [data and control flow net]

Given a box $P = \{Σ, L, R, B, E, I, O\}$ where $Σ = \{S, T; F\}$ define the data flow net of $P$,

$$df(P) = \langle S', T'; F'; M_0 \rangle$$

where

- $S' = S \cup [(v, λ) | λ \in \text{uid}(P) \land v \in λ]$  
- $T \in \bigcup_{i ∈ T} β(t)$
\[ F' = \{ (s,t) | \exists t' \in T . t \in \beta(t') \land \exists a \in \alpha(t') \land s \in \beta(t', a) \} \\
\quad \cup \{ (t,s) | \exists t' \in T . t \in \beta(t') \land \exists a \in \alpha(t') \land s \in \beta(t', a) \} \]

\[ M_0 = B \cup \text{Init}(P) \]

The priority relation, \( R \), can either be discarded in the case that an unprioritised net is needed, or retained and used in the modelling of the priorities. Its structure is that required by [Best/Koutny90].

7. Applications of the Petri Net semantics

We have a well-defined mapping from a subset of occam 2 to Petri nets. We list below, in no particular order and no great detail, some of the applications we see of a Petri net semantics of occam 2.

Deadlock within an occam 2 process can come from a number of sources; it could be the explicit encounter of a \textit{STOP} within the process; there could an attempt to communicate over an unmatched channel; or an \textit{insufficiency of communication} between two processes in which one process prevents another from communicating by never enabling the communication. The detection of deadlock within Petri nets is well studied [Jensen87] and by producing our net semantics we allow this work to be carried into the detection of deadlock within occam 2.

Other properties of occam 2 processes which may also be analysed using a Petri net semantics include liveness, divergence-freeness, correctness conditions such as the invariants of a process, and satisfaction of a specification.

We also believe that the optimal simulation work of Janicki and Koutny [Janicki/Koutny89] applies to the nets we produce.

An axiomatisation of occam 2 appears in [Roscoe/Howe86]. Our model is rich enough to be able to verify such laws within our framework. It is also possible (see below) that new laws may arise due to the true concurrency semantics.

The addition of the \textit{PLACED} modifier in the \textit{PAR} construct is said not to alter the logical behaviour of an occam 2 process. This claim is perhaps necessary when only an interleaving semantics is available, so that arbitrary interleaving can describe the concurrency aspects of both operators adequately (which is, by assumption so). However, others appear to believe that \textit{PLACED PAR} may alter the semantics of a process [Barrett89]. Perhaps the most important outcome of this paper is to produce a true concurrency semantics of occam 2. This will (with future work) allow us to make a real distinction between these two variants of parallel composition and so investigate differences between them. There may also be extended laws based on [Roscoe/Howe86], applying to \textit{PLACED PAR} which can be proved only within the framework of true concurrency, and of course it should be true that existing laws be provable within this new framework.

8. Acknowledgments

The authors would like to thank the following people for their contributions to this document: Eike Best, Maciej Koutny and Axel Poigné.
References


research Action DEMON (Design Methods Based On Nets) #3148.

Richard Hopkins has been a lecturer in computing science at the University of Newcastle since 1983. Before that he was first a student, and then a research Associate there.

All the authors are involved in the ESPRIT BRA DEMON.

Attachments: MailNote
Jon Hall

To: Debbie.Fawcett@newcastle.ac.uk

<J.G.Hall@uk.ac.newcas.tle:stargate>:NclAcUk

unixxfg:CompLab:NclAcU

Subject: Re: Technical Report

In Reply To: J.G.Hall@uk.ac.newcastle:stargate:NclAcUk

Note: Received: from newcastle (cheviot.ncl.ac.uk) by uk.ac.newcastle.bygate; Tue, 29 Oct 91 14:22:17 GMT
Received: from turing by uk.ac.newcastle; Tue, 29 Oct 91 14:27:11 GMT
Original Date: Tue, 29 Oct 91 14:27:13 WET
In-Reply-To: <AA15939.9110041424.bygate@uk.ac.newcastle>;
from "Debbie.Fawcett@newcastle.ac.uk" at Oct 4, 91 3:24 pm
X-Mailer: ELM [version 2.3 PL11]

>
> Jon,
>
> "A Petri Net Semantics of occam 2"
> I also need a bit of info about yourself, eg. how long you have
> been
> here.
>
> Debbie
>
>
I'm sorry it has taken soooo long, after the conference I've had so much
to do and this seems to have always been pushed to the bottom of the
pile. Anyway perhaps you could use this:

Fiorella De Cindio is a professor of Computing Science at the University of
Milano, Italy.

Oliver Botti is a student of computer science at the University of
Milano.

Jon Hall has a post of Research Assistant in the Computing Laboratory
at the University of Newcastle, England. He is funded from ESPRIT Basic