OCCAM 2: TRANSFORMATION RULES (Motivation)

The semantics of very few languages can be formally stated. OCCAM 2 is a noble example. This feature of OCCAM 2 permits to formally specify and verify OCCAM 2 programs. For example it is possible to prove that a particular OCCAM 2 program is (or is not) deadlock free.

Another use for this formalism: To transform one OCCAM 2 program into another, that would be, in some sense, equivalent to the original. Possible motivations:

1. *To change a clear but inefficient program into an efficient but perhaps obscure one.*

2. *To change a sequential program into a parallel one to exploit the existing hardware.*

3. *To change a concurrent program into a sequential one for improved execution on a single processor.*

4. *To change a physically unfeasible program into a physically feasible one.*

OCCAM 2: TRANSFORMATION RULES
(Intuitive introduction)

The rules governing transformations of SEQ, IF, PAR and ALT may be used to exploit processor configurations.

Example 1: The rule for replacing SEQ by PAR says that SEQ can be replaced by PAR if its processes use disjoint sets of variables and channels (or, at least, the common variables or channels are read only).

Consider:

```
SEQ
  Y := F(X)
  Z := G(X)
```

is equivalent to:

```
PLACED PAR
  PROCESSOR 1 <some_type>
    Y := F(X)
  PROCESSOR 2 <some_type>
    Z := G(X)
```


OCCAM 2: TRANSFORMATION RULES
(Intuitive introduction continued)

Example 2: Consider:

```
SEQ
  ch1 ? A
  GetX(A, X)
  Y := X / 2.0
  GetZ(Y, Z)
  ch2 ! Z
```

When introducing a new channel, the above is equivalent to:

```
-- definition of new channel New.Chan
PAR
  SEQ
    ch1 ? A
    GetX(A, X)
    New.Chan ! X / 2.0
  SEQ
    New.Chan ? Y
    GetZ(Y, Z)
    ch2 ! Z
```


OCCAM 2: TRANSFORMATION RULES
(Notation)

To simplify the presentation of OCCAM 2 syntax, we will use brackets. For example, instead of:

```
SEQ
  A
  B
  C
```

we will write:

SEQ *(A, B, C)*

Where a construct is considered to have an arbitrary number of components, say n, we will write:

$$\underset{i=1}{\overset{n}{\mathbf{PAR}}}\ P_i \qquad \underset{i=1}{\overset{n}{\mathbf{SEQ}}}\ S_i \qquad \underset{i=1}{\overset{n}{\mathbf{ALT}}}\ g_i p_i \qquad \underset{i=1}{\overset{n}{\mathbf{IF}}}\ b_i p_i$$

OCCAM 2: TRANSFORMATION RULES

Rule 1: **SEQ** *()* = **SKIP**

Rule 2: **PAR** *()* = **SKIP**

Laws of Associativity:

Rule 3: **SEQ** *(P, Q, R)* = **SEQ** *(P,* **SEQ** *(Q, R))*

Rule 4: **PAR** *(P, Q, R)* = **PAR** *(P,* **PAR** *(Q, R))*

Let $C_i = (b_i, p_i)$ be pairs of Boolean expressions and subprocesses, then:

Rule 5: **IF** *(C_1,* **IF** *(C_2), C_3)* = **IF** *(C_1, C_2, C_3)*

Similarly, let $G_i = (g_i, p_i)$ be pairs of guards and subprocesses, then:

Rule 6: **ALT** *(G_1,* **ALT** *(G_2), G_3)* = **ALT** *(G_1, G_2, G_3)*

From Rule 5: If an **IF** *construct has no Boolean expressions, none can evaluate to TRUE. Similarly (from Rule 6), if an* **ALT** *construct has no guards, none can be ready. Therefore:*

Rule 7: **IF** *()* = **STOP**

Rule 8: **ALT** *()* = **STOP**

Rule 5 applies to one form of nesting **IF** *constructors, where an inner* **IF** *is in place of the usual (b, p). Another form of nesting is derived when an inner* **IF** *is in place of the p subprocess. In such a case:*

OCCAM 2: TRANSFORMATION RULES (Continued)

Rule 9: **IF** *(C, b1,* **IF** *(b2, p))* = **IF** *(C,* **IF** *(b1 AND b2, p))*

NOTE: the above rule is valid if the inner IF is the last subprocess; alternatively, it is valid in any position if the inner IF is complete. A complete IF constructor is one that is guaranteed to have a Boolean expression that evaluates to TRUE. Any IF construct can be made complete by adding at its end:

```
TRUE
  STOP
```

Laws of Symmetry:

Rule 10: $\underset{i=1}{\overset{n}{\mathbf{PAR}}}\ P_i = \underset{i=1}{\overset{n}{\mathbf{PAR}}}\ P_{\Pi(i)}$

Rule 11: $\underset{i=1}{\overset{n}{\mathbf{ALTG}}}\ _i = \underset{i=1}{\overset{n}{\mathbf{ALTG}}}\ _{\Pi(i)}$

where $\Pi(i)$ *is any permutation of the values* $\{1, 2, ..., n\}$

OCCAM 2: TRANSFORMATION RULES (Continued)

The standard **IF** *construct cannot be so rearranged. However, if the Boolean expressions are pairwise disjoint, such rearrangement is possible. The Boolean expressions are said to be pairwise disjoint, if only one can be TRUE for any set of values for the associated variables. For example, the following* **IF** *construct has this property:*

```
IF
  x > 0
    y := 1
  x = 0
    y := 0
  x < 0
    y := -1
```

So, we have the law:

Rule 12: $$\underset{i=1}{\overset{n}{\mathbf{IF}}} b_i p_i \;=\; \underset{i=1}{\overset{n}{\mathbf{IF}}} b_{\Pi(i)} p_{\Pi(i)}$$

provided that b_I *AND* b_J *= TRUE implies* $i = j$.

OCCAM 2: TRANSFORMATION RULES (Continued)

Replacement of SEQ by PAR.

Rule 13: **SEQ** *(P, R)* = **PAR** *(P, R)*

provided that P and R use disjoint sets of variables and channels. (Indeed, the replacement can still take place if there are shared variables, as long as P and R both read them).

Rule 14: **PAR** *(c ! x, c ? y)* = *y := x*

or, stating it the other way around:

Rule 15: **SEQ** *(P, x := y, Q)* =
CHAN OF *Proper.Protocol z:*
PAR *(***SEQ** *(P, z ! y),* **SEQ** *(z ? x, Q))*

Examples for the usage of these rules have been given in the intuitive introduction to OCCAM 2 transformation rules.

OCCAM 2: TRANSFORMATION RULES (Continued)

Laws of declarations.

Declarations are associative:

Rule 16: *T a: (T b: P) = T a, b: P*
where P is some process
and T is any valid type.

Declarations can be eliminated if they are not used:

Rule 17: *T a: P = P* *if a is not used in P.*

Declarations can be moved within a program:

Rule 18: **SEQ** *(T a: P, Q) = T a:* **SEQ** *(P, Q)*
provided a is not used in Q.

Rule 19: **SEQ** *(P, T a: Q) = T a:* **SEQ** *(P, Q)*
provided a is not used in P.

Similar laws exist that apply to **ALT, IF** *and* **PAR** *constructs.*

OCCAM 2: TRANSFORMATION RULES (Continued)

Increasing parallelism.

Let **W** *stand for WHILE TRUE (infinite loop) in OCCAM 2, then:*

Rule 20: **W** *(***SEQ** *(P, Q, R) =*
SEQ *(P,* **W** *(***SEQ** *(Q, R, P)))*

To appreciate this law, consider this example:

```
WHILE TRUE
  SEQ
    in ? x
    y := x * x
    out ! y
```

Applying Rules 20 and 13, the code becomes:

```
SEQ  -- after rule 20
  in ? x
  WHILE TRUE
    SEQ
      y := x * x
      out ! y
      in ? x

SEQ  -- after rules 20 and 13
  in ? x
  WHILE TRUE
    SEQ
      y := x * x
      PAR
        out ! y
        in ? x
```


OCCAM 2: TRANSFORMATION RULES (Continued)

Unraveling a replicated SEQ.

Rule 21: $$\underset{i=1}{\overset{n}{\mathbf{SEQ}}}(P_i, Q_i) = \mathbf{SEQ}(P_1, \underset{i=1}{\overset{n-1}{\mathbf{SEQ}}}(Q_i, P_{i+1}), Q_n)$$

The above is a generalization of the Rule 20, which applied to the WHILE TRUE loops.

OCCAM 2: TRANSFORMATION RULES (Continued)

Distributivity.

Rule 22: **PAR** *(***SEQ** *(A, B),* **SEQ** *(C, D)) =*
SEQ *(***PAR** *(A, C),* **PAR** *(B, D))*

To illustrate the power of this law, consider the following code:

```
PAR
  SEQ
    c ! x
    P1
  SEQ
    c ? y
    P2
```

After applying rule 22, the code becomes:

```
SEQ
  PAR
    c ! x
    c ? y
  PAR
    P1
    P2
```

After applying rules 22 and 14, this reduces to:

```
SEQ
  y := x
  PAR
    P1
    P2
```


REFERENCES AND RECOMMENDED READING (On CSP and OCCAM 2)

Hoare, C.A.R.: Communicating Sequential Processes, *Prentice Hall, 1985.*

INMOS Ltd.: OCCAM 2 Reference Manual, *Prentice Hall, 1988.*

Burns, A.: Programming in OCCAM 2, *Addison Wesley, 1988.*

Roscoe, A.W., and Hoare, C.A.R.: The Laws of OCCAM Programming, *Oxford University Programming Research Group, PRG–53, 1986.*

Pountain, R., and May, D.: A Tutorial Introduction to OCCAM Programming, *(second, amended edition), Blackwell Scientific Publications Ltd., 1988.*

Jones, G. and Goldsmith, M.: Programming in OCCAM 2, *Prentice Hall, 1988.*

