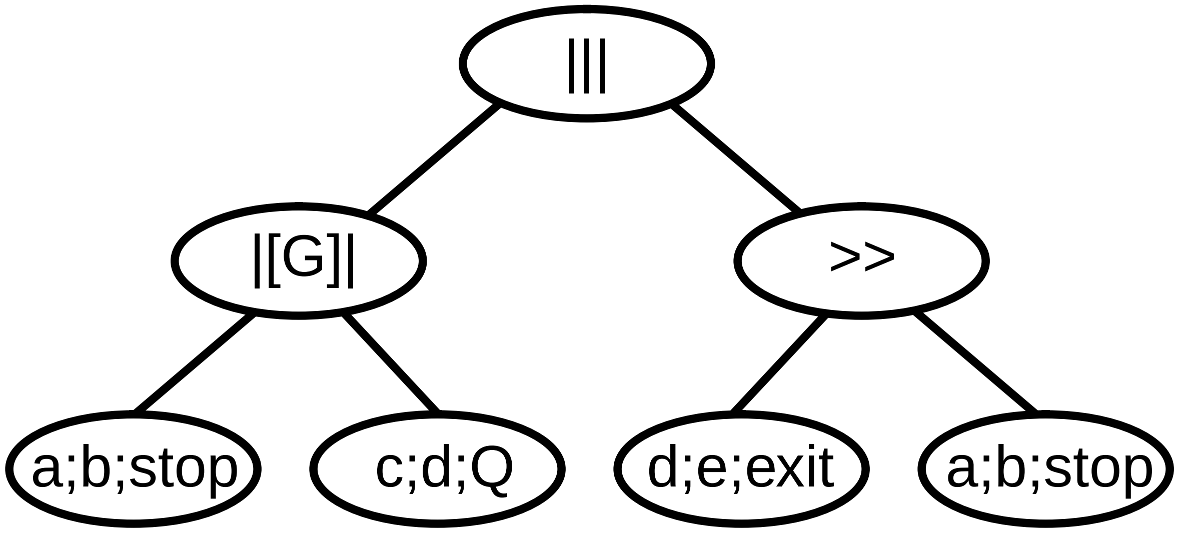
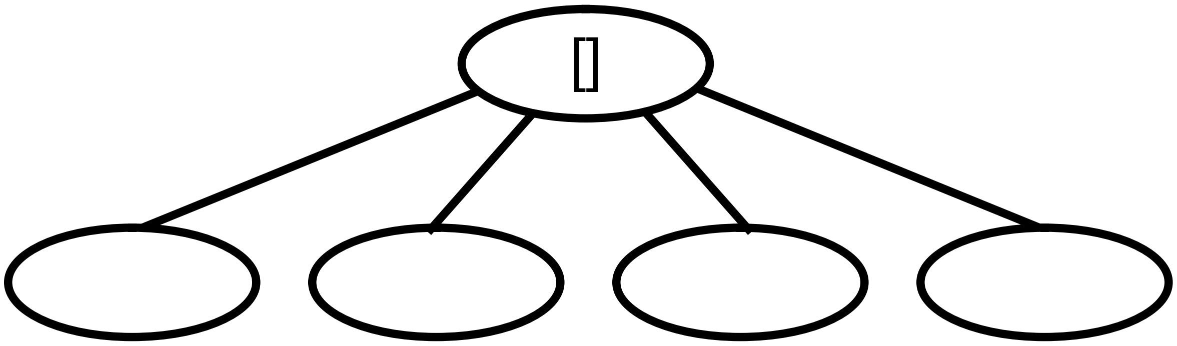
Equality and Expansion

THE IDEA OF THE EXPANSION LAWS

Let us see a behavior expression as a syntactic tree having operators as nodes:

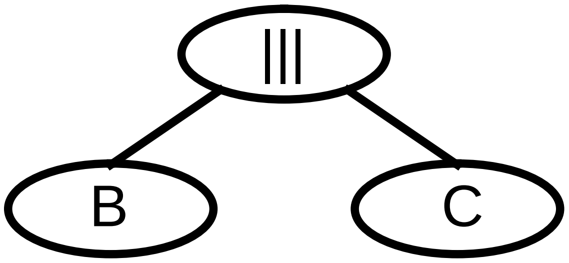


The expansion laws allow to transform this behavior expression into another one where **[]** is the top operator:

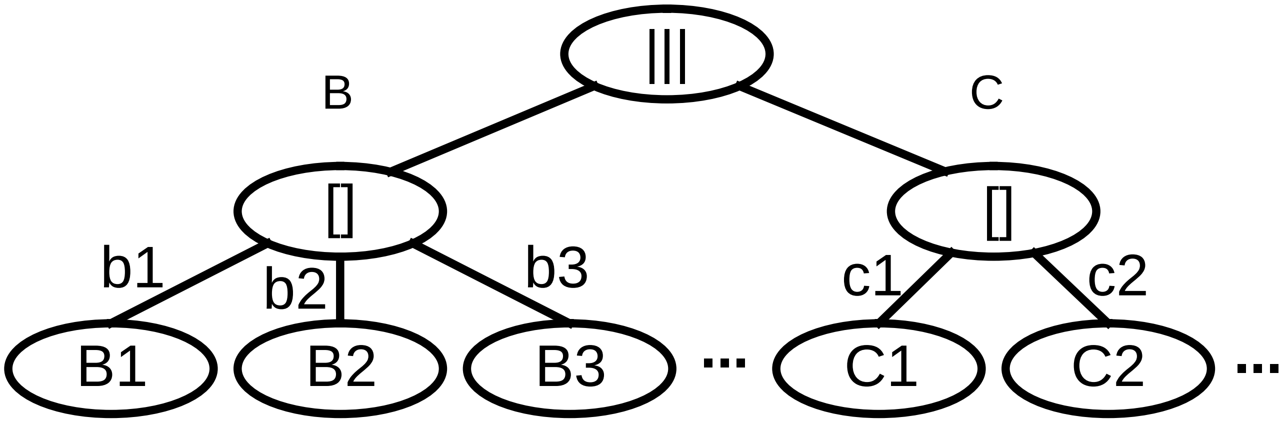


By doing this repeatedly, one can *push down* all operators different from ‘**[]**’ and ‘**;**’. It is possible that at the end we obtain a fully expanded expression, or the process can run forever.

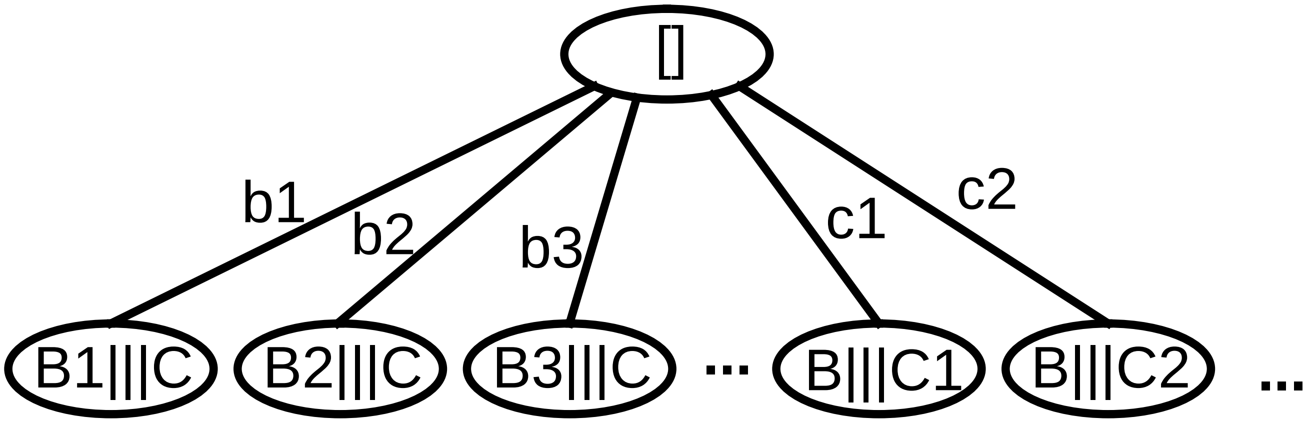
Example for the **|||** operator:



assume that B and C are already part. expanded, e.g.



the expansion law for **|||** is applied:



SOME BASIC EQUALITIES AND NOTATION FOR []

B_1 [] B_2 = B_2 [] B_1
B_1 [] (B_2 [] B_3) = (B_1 [] B_2) [] B_3
B_1 [] stop = stop [] B_1 = B

These can be proven on the basis of the inference rules.
As a consequence of the second one, we can write

B1 [] B2 [] B3...

without brackets. In fact, as a consequence of the first two, the following notation can be used:

Notation: [] will be the prefix n-ary [] operator over sets of beh.expr., similar to $\sum$ for $+$, i.e.
[]{B_1,...,B_n} = B_1[] ... [] $B_{n.}$

Then **stop** $=_{def}$ []{}   [{} : empty set of beh.exp.]

Also, []{B_1,...,B_m} [] []{C_1,...,C_n} = []{B_1,...,B_m,C_1,...,C_n}

Expansion Laws

The expansion laws are the *algebraic counterpart* of the inference rules. They exist for each operator, except of course [] and ; They can be proven as theorems from the inference rules.

Theorem (simplified expansion of |||).

Let B= $[]\{b_k;B_k \mid k\in K\}$ and C= $[]\{c_j; C_j \mid j\in J\}$.

Then B|||C = $[]\{b_k;(B_k|||C) \mid k\in K\}$

$[]$ $[]\{c_j;(B|||C_j) \mid j\in J\}$

Proof.

By the inference rules all derivations from B are of the form B-b_k->B_k ($k\in K$) and similarly for those from C. Then all derivations from B|||C will be of the form

B|||C-b_k->B_k|||C ($k\in K$) or B|||C-c_j->B|||C_j ($j\in J$).

These two sets of derivations are exactly those of

$[]\{b_k;(B_k|||C) \mid k\in K\}$ [] $[]\{c_j;(B|||C_j) \mid j\in J\}$

B|[G]|C

Expansion of ***Parallel Composition*** *(general case):*

if B = $[]\,\{b_k; B_k \mid k \in K\} \quad []\, []\,\{\textbf{exit}_n \mid n \in N\}$
 C = $[]\,\{c_m; C_m \mid m \in M\}\, []\, []\,\{\textbf{exit}_j \mid j \in J\}$

then B|[G]|C =

$$[]\,\{g; (B_k \,|[G]|\, C_m) \mid \exists k \in K \;\; m \in M \;\; g \in G,\ b_k = c_m = g \neq \mathbf{i}\}$$
$$[][]\,\{b_k; (B_k \,|[G]|\, C) \mid \exists k \in K\ b_k \notin G \text{ or } b_k = \mathbf{i}\}$$
$$[][]\,\{c_m; (B \,|[G]|\, C_m) \mid \exists m \in M\ c_m \notin G \text{ or } c_m = \mathbf{i}\}$$
$$[][]\,\{\textbf{exit} \mid \ n \in N,\ \ j \in J\}$$

Note: the last alternative means that exits must ‘pair up’ from the two sides in some way, i.e.

$$\textbf{exit} \,|[G]|\, \textbf{exit} = \textbf{exit}$$

and

$$\textbf{exit} \,|[G]|\, []\{\} = []\{\} \,|[G]|\, \textbf{exit} = []\{\}$$

because none of the alternatives is possible.

Other Equality Laws for |[]|
(can be proven on the basis of the expansion laws)

B_1 |[G]| B_2 = B_2 |[G]| B_1

B_1 |[G]| (B_2 |[G]| B_3) = (B_1 |[G]| B_2) |[G]| B_3

Note: B || stop = stop unfortunately false if B has internal actions as initials.

B_1 ||| B_2 = B_1|[{}]| B_2 [where {} is the empty set of gates]

For a beh. expr. B, let L(B) be the set of gates (*labels*) in B

B_1 |[G]| B_2 = B_1 |[G']| B_2
if $G \cap (L(B1) \cup L(B2)) = G' \cap (L(B1) \cup L(B2))$

B_1 |[G]| B_2 = B_1 || B_2 if $(L(B_1) \cup L(B_2)) \subseteq G$

B |[G]| stop = B if $(G \cup \delta) \not\subset L(B)$
so B ||| stop = B if B does not exit

B [> C

Expansion of ***Disable****:*

if $B \neq$ **exit** and $B = [] \{b_k; B_k \mid k \in K\}$

then B [> $C = [] \{b_k; (B_k$ [> $C) \mid k \in K\}$ [] C

exit [> C = **exit** [] C

Other equality laws for [>:

stop [> C = C immediately by the expansion and def of stop

B [> stop = B by recursive argument because C= stop

$(B_1$ [> $B_2)$ [] B_2 = B_1 [> B_2

$(B_1$ [> $B_2)$ [> $B_3 = B_1$ [> $(B_2$ [> $B_3)$

B >> C

Expansion of ***Enable****:*

if B = $[]$ {b_k; B_k | k $\in$ K}   [] $[]$ {**exit**$_n$ | n $\in$ N}

then B >> C =

$[]$ {b_k; (B_k >> C) | k $\in$ K }

[] $[]$ { **i** ; C | n $\in$ N}

Other equality laws for >>:

stop >> C = stop
exit >> C = i; C
(B1 >> B2) >> B3 = B1 (B2 >> B3)
B >> stop = B ||| stop if B does not exit

hide G in B

*Expansion of **hiding**:*

if B = $[]$ {b_k; B_k | k $\in$ K} [] $[]$ {**exit**$_n$ | n $\in$ N}

then **hide** G **in** B =

$[]$ {b_k; **hide** G **in** B_k | $b_k \notin$ G and k $\in$ K}

[] $[]$ { **i** ; **hide** G **in** B_k | $b_k \in$ G and k $\in$ K}

[] $[]$ {**exit**$_n$ | n $\in$ N}

Other equality laws for hide:

G, A, A’, A” are sets of gates

hide G in stop = stop
hide A in B = hide A’ in B if A $\cap$ L(B) = A’ $\cap$ L(B)
hide A in B = B if A $\cap$ L(B) = {}
hide A in g; G = g; hide A in G if g $\notin$ A
hide A in B_1 [] B_2 = (hide A in B_1) [] (hide A in B_2)
similar for >> and [>
hide A in B_1 |[A’]| B_2 = (hide A in B_1) |[A’]| (hide A in B_2)
if A$\cap$A’ = {}
hide A in hide A’ in B = hide A” in B, if A” = A$\cup$A’

B[S]

Let B[S] be a relabeling, and S(g) be a renaming function on gates.

*Expansion of **Process Instantiation**:*

if $B[G] = \big[\big] \{b_k; B_k \mid k \in K\} [] \big[\big] \{\mathbf{exit}_n \mid n \in N\}$

then $B[S] = (S(b_k); B_k[S] \mid k \in K\} [] \big[\big] \{\mathbf{exit}_n \mid n \in N\}$

In general, note that if S is one-to-one, for each beh. expr. B it holds that B[S] = B’, where B’ is the result of substituting S(a) for a everywhere in B - in other words, in this case, instantiation-time renaming produces the same results as action-by-action relabeling as defined by the inference rules (examples to the contrary involve many-one relabelings).

Equality laws for relabelling:

S(stop) = stop

S(exit) = exit

S(a; B) = S(a); S(B)

S(B_1[] B_2) = S(B_1) [] S(B_2), same for >> and [>

S(B1|[G]|B2) = S(B1) |[S(G)]| S(B2) if S is one-to-one

S1(S2(B)) = S1.S2(B) where **.** is the composition of maps

INFERENCE RULES VS EXPANSION RULES

Milner’s fundamental contribution was to invent a concurrent language (CCS) which was *executable*, and in which at the same time several useful *algebraic* properties held.

The expansion theorems are the algebraic counterpart of the inference rules.

They can be proven by induction from the inference rules.

An algebraic language such as LOTOS could be defined by giving the expansion rules first, as axioms. The inference rules could then be derived as ‘programs’ for calculating the expansion. ACP, the Algebra of Communicating Processes, was defined in this way.

Therefore, in LOTOS and similar languages *an execution is a proof*: The *LTS* of a process can be identified with the *expansion* of the process.

If, by applying the inference rules on beh. expr. B, one gets an LTS T, and if we know that B’ is another behavior expression whose LTS is also T, then we have a proof that B=B’.

More simply, if by executing inference rules on B we can derive a sequence of actions s, then we have a proof that s is some initial part of a branch of the *expansion* of B.