```
function f(%0) : %1
var %0, %1, %2, %3
entry f6
exit f0
f6: l i %1, 0 =>f5
f5: blez %0 =>f4, f3
```

La traduction la plus simple de la conditionnelle consiste à évaluer la condition vers un pseudo-registre, qui contient alors 0 ou 1, puis à utiliser (par exemple) l'instruction bgtz.

Une conditionnelle peut parfois être traduite sans évaluer explicitement la condition: c'est ce que permettent les instructions spécialisées bgez, bgtz, blez, bltz, ble, bne.

L'élimination des « sous-expressions communes» vise à supprimer certains calculs redondants.
Considérons par exemple le fragment de code suivant:

```
x := t[ i ] ;
t[ i ] := t[ i−1] ;
t[ i−1] := x ;
```

Que vont produire les traductions de PP vers UPP puis RTL ?
Une traduction naive calcule quatre fois $a0 + 4 × $a3:

```
sll $v0, $a3, 2
addu $v0, $a0, $v0
lw $a2, 0($v0)
sll $v0, $a3, 2
addu $a1, $a0, $v0
sll $v0, $a3, 2
addu $v0, $a0, $v0
lw $v0, −4($v0)
sw $v0, 0($a1)
sll $v0, $a3, 2
addu $v0, $a0, $v0
sw $a2, −4($v0)
```

Ce calcul redondant est celui de l'adresse que l'on pourrait écrire, en C, sous la forme t + i.

On préférerait obtenir ceci:

```
sll $v0, $a2, 2
addu $a1, $a0, $v0
lw $a3, 0($a1)
lw $v0, −4($a1)
sw $v0, 0($a1)
sw $a3, −4($a1)
```

La multiplication et l'addition ne sont effectuées qu'une fois et leur résultat, à savoir l'adresse $a1, est utilisée quatre fois.

Fonction factorielle (en récursif)

```
f (n : integer) : integer
if n = 0 then
        f : = 1
else
        f : = n * f (n * 1)
```

Traduction en ERTL

```
procedure f (1)
var %0;%1;%2;%3;%4;%5;%6
entry f 11
f 11 : newframe => f 10
f 10 : move %6; $ra => f 9
f 9 : move %5; $s1 => f 8
f 8 : move %4; $s0 => f 7
f 7 : move %0; $a0 => f 6
f 6 : li %1; 0 => f 5
f 5 : blez %0 => f 4; f 3
f 3 : addiu %3;%0;1 => f 2
f 2 : j => f 20
f 20 : move $a0;%3 => f 19
f 19 : call f (1) => f 18
f 18 : move %2; $v0 => f 1
f 1 : mul %1;%0;%2 => f 0
f 0 : j => f 17
f 17 : move $v0;%1 => f 16
f 16 : move $ra;%6 => f 15
f 15 : move $s1;%5 => f 14
f 14 : move $s0;%4 => f 13
f 13 : delframe => f 12
f 12 : jr $ra
f 4 : li %1; 1 => f 0
```

Version terminale

```
f (n : integer)(acc : integer) : integer
if n = 0 then
f : = acc
else
f : = f (n - 1; n * acc)
```

Appel avec f (n, 1).

Traduction optimisée de l'appel terminal en MIPS

```
mul $a1 , $a0 , $a1 # calcul des arguments
addiu $a0 , $a0 , -1
lw $ra , 0( $sp ) # restauration des callee - save
addiu $sp , 4 # désallocation de la pile
j fact # appel de fact
```

Traduction encore plus optimisée en MIPS

```
begin : addiu $sp , $sp , -4
sw $ra , 0( $sp)
loop : blez $a0 , base
mul $a1 , $a0 , $a1
addiu $a0 , $a0 , -1
j loop
end : lw $ra , 0( $sp)
```

```
addiu $sp , $sp , 4
jr $ra
base : move $v0 , $a1
j end
```

Programme équivalent

```
f (n : integer)(acc : integer) : integer
while n > 0 do
(acc : = n * acc;
n : = n - 1);
f : = acc
```

Fonction factorielle (en rÃ©cursif)

```
f (n : integer) : integer
if n = 0 then
        f : = 1
else
        f : = n * f (n * 1)
```

Traduction en LTL

```
procedure f (1)
var 8
entry f 11
f 11 : newframe => f 10
f 10 : sets local (0); $ra => f 9
f 9 : j => f 8
f 8 : sets local (4); $s0 => f 7
f 7 : move $s0; $a0 => f 6
f 6 : j => f 5
f 5 : blez $s0 => f 4; f 3
f 3 : addiu $a0; $s0;-1 => f 2
f 2 : j => f 20
f 20 : j => f 19
f 19 : call f => f 18
f 18 : j => f 1
f 1 : mul $v0; $s0; $v0 => f 0
f 0 : j => f 17
f 17 : j => f 16
f 16 : gets $ra; local(0) => f 15
f 15 : j => f 14
f 14 : gets $s0; local(4) => f 13
f 13 : delframe => f 12
f 12 : jr $ra
f 4 : li $v0; 1 => f 0
```

Fonction factorielle (en rÃ©cursif)

```
f (n : integer) : integer
if n = 0 then
        f : = 1
else
        f : = n * f (n * 1)
```

Traduction en LIN

```
procedure f (1)
var 8
f 11 :
newframe
sets local (0); $ra
sets local (4); $s0
move $s0; $a0
blez $s0; f 4
addiu $a0; $s0;-1
call f
mul $v0; $s0; $v0
f 16 :
gets $ra; local(0)
gets $s0; local(4)
delframe
jr $ra
f 4 :
li $v0; 1
j f 16
```

Fonction factorielle (en rÃ©cursif)

```
f (n : integer) : integer
if n = 0 then
        f : = 1
else
        f : = n * f (n * 1)
```

Traduction en MIPS

```
f17 : addiu $sp , $sp , -8
sw $ra , 4( $sp)
sw $s0 , 0( $sp)
move $s0 , $a0
blez $s0 , f4
addiu $a0 , $s0 , -1
jal f17
mul $v0 , $s0 , $v0
f28 : lw $ra , 4( $sp )
lw $s0 , 0( $sp)
addiu $sp , $sp , 8
jr $ra
f4: li $v0 , 1
j f28
```