ESTEREL on FPGA via ECLAT (WIP)

Loïc Sylvestre[1]

[1]Université de Toulouse - IRIT (TRACES team)

November 28 2025 - SYNCHRON '25 à Aussois

Compilation scheme for ESTEREL in the ECLAT compiler

$$\left(\!\left| e \right|\!\right)^{\ell,\, x}_{\rho,\, \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) => rdy_ℓ := false ; s
t_1 ⋯ t_n
end
```

where

$$\left[\!\left[e \right]\!\right]^{\ell, x, k}_{\rho,\, \alpha} =$$

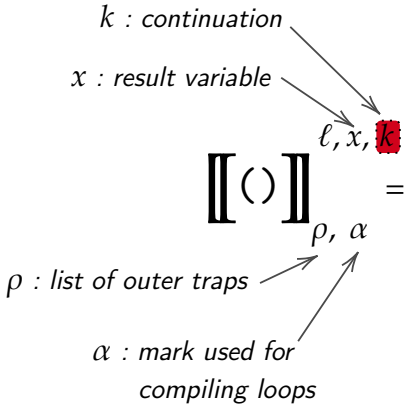
```
case state_ℓ is
when enum(IDLE_ℓ) => s
t_1 ⋯ t_n
end
```

k = ($\texttt{rdy}_\ell$:= true ; $\texttt{state}_\ell$:= **enum**($\texttt{IDLE}_\ell$))

naming convention:
- ℓ (id of the current FSM)
- $\texttt{state}_\ell$ (state variable)
- $\texttt{IDLE}_\ell$ (initial state)
- $\texttt{rdy}_\ell$ (when result is rdy)

invariant: once rdy_ℓ is true, the FSM reenters in the initial state **enum**(IDLE_ℓ) at the next instant

Compiling "nothing"




```
case state_ℓ is
when enum(IDLE_ℓ) =>
    k
end
```

Compiling valued signals

$[\![\ \texttt{emit}\ y(a)\ ]\!]^{\ell,x,k}_{\rho,\alpha} =$

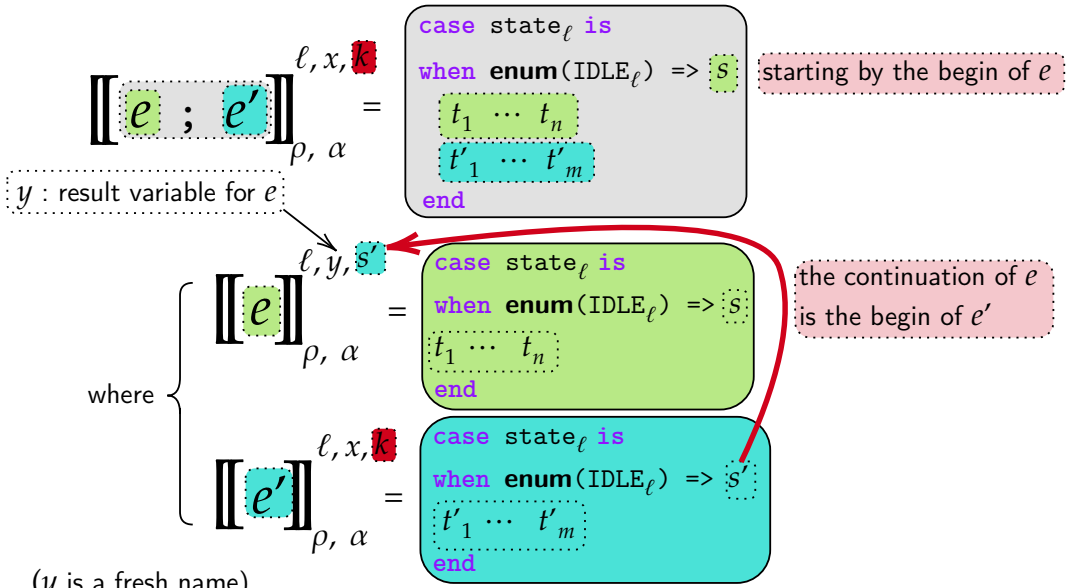
```
case state_ℓ is
when enum(IDLE_ℓ) =>
        y <= a ; k
end
```

x : *result variable*

$[\![\ ?y\ ]\!]^{\ell,x,k}_{\rho,\alpha} =$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
        x :=?y ; k
end
```

Compiling sequences



Exemple

$$[\![\ \texttt{emit}\ y(42);\ \texttt{emit}\ z(43)\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   y <= 42; z <= 43; k
end
```

where

$$[\![\ \texttt{emit}\ y(42)\ ]\!]^{\ell, x_0, s'}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   y <= 42; s'
end
   s
```

$$[\![\ \texttt{emit}\ z(43)\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   z <= 43; k
end
   s'
```

(x_0 is a fresh name)

Compiling conditionals

$$[\![\ \texttt{if}\ a\ \texttt{then}\ e\ \texttt{else}\ e'\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
  if a then s else s'
  t_1 ··· t_n   t'_1 ··· t'_m
end
```

where

$$[\![\ e\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ)=> s
  t_1 ··· t_n
end
```

NB: in the general case, need to duplicate k

$$[\![\ e'\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) ⇒ s'
  t'_1 ··· t'_m
end
```

k is what remains to do in the instant, e.g.:

```
(rdy_ℓ := true;
 state_ℓ := enum(IDLE_ℓ))
```

Local signals

$$[\![\ \textbf{signal}\ y\ \textbf{in}\ e\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} = [\![\ e\ ]\!]^{\ell, x, k}_{\rho,\ \alpha}$$

NB: assumes that name y is declared only once (renaming by the compiler)

Compiling **exit**

$$[\![\texttt{exit}\ y]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

```
case state_ℓ is
when enum(IDLE_ℓ) =>
    y := true
end
```

note that $y \in \rho$

NB: the continuation k is cut

Compiling **traps**, using an internal state machine

$$[\![\ \texttt{trap}\ y\ \texttt{in}\ e\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

where $\left(\!\left| e \right|\!\right)^{L, x}_{(y \cdot \rho),\ \alpha} =$

```
case state_L is
when enum(IDLE_L) => s
t_1 ··· t_n
end
```

(L and *WAIT* are fresh)

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   y := false ;
   s ; (if disjonction_traps(ρ) then ()
        else if y ∨ rdy_L then k
        else state_L := enum(WAIT) )
when enum(WAIT) =>
   case state_L is
   when enum(IDLE_L) => () t_1 ··· t_n
   end ;
   (if disjonction_traps(ρ) then ()
    else if y ∨ rdy_L then k
    else state_L := enum(WAIT) )
end
```

internal state machine (with id L)

```
disjonction_traps(x_1, ··· x_n)
  := x_1 ∨ ··· ∨ x_n
```

Compiling **suspend**

$$[\![\ \texttt{suspend}\ e\ \texttt{when}\ y\ ]\!]^{\ell, x, k}_{\rho,\ \alpha} =$$

where $(\!|\ e\ |\!)^{L, x}_{\rho,\ \alpha} =$

```
case state_L is
when enum(IDLE_L) => s
t_1 ··· t_n
end
```

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   ( s ; if rdy_L then k else state_ℓ := enum(WAIT))
when enum(WAIT) =>
   if ?y then state_ℓ := enum(WAIT) else ((
      case state_L is
      when enum(IDLE_L) => ()
      t_1 ··· t_n
      end
   ); if rdy_L then k
      else state_ℓ := enum(WAIT))
end
```

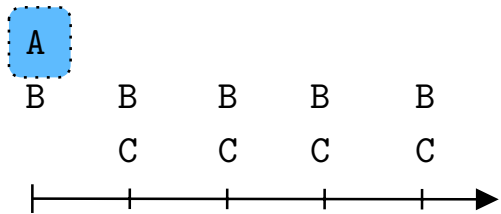
internal state machine (with id L)

(L and *WAIT* are fresh)

Loop in Esterel

```
emit A;
loop
   emit B;
   pause; emit C;
end;
emit D
```

dead code



Compiling loops

$[\![\ \texttt{loop}\ e\ \texttt{end}\ ]\!]^{\ell, x, k}_{\rho,\ \alpha}$ =

```
case state_ℓ is
when enum(IDLE_ℓ) => s
   t_1 ··· t_n   t'_1 ··· t'_m
end
```

α ::=
- | **1** *(not the end of a loop)*
- | **2** *(end of loop)*

inspired by Tardieu's approach for compiling loops

where

$[\![\ e\ ]\!]^{\ell, x, s'}_{\rho,\ \underbrace{\max(\alpha, \mathbf{1})}_{=\alpha}}$ =

```
case state_ℓ is
when enum(IDLE_ℓ) => s
   t_1 ··· t_n
end
```

$[\![\ \texttt{rename}(e)\ ]\!]^{\ell, x, ()}_{\rho, \mathbf{2}}$ =

```
case state_ℓ is
when enum(IDLE_ℓ) => s'
   t'_1 ··· t'_m
end
```

The continuation of the body is the start of the renamed body

reincarnation: renaming traps and signals

Compiling pauses

$[\![\ \texttt{pause}_Q\ ]\!]^{\ell,x,k}_{\rho,\mathbf{1}}$ =

```
case state_ℓ is
when enum(IDLE_ℓ) =>
    state_ℓ := enum(Q)
when enum(Q) => k
end
```

each pause is labeled with a unique name Q (not visible in source programs)
pause labels are not renamed when compiling loops, thus ensuring looping

---

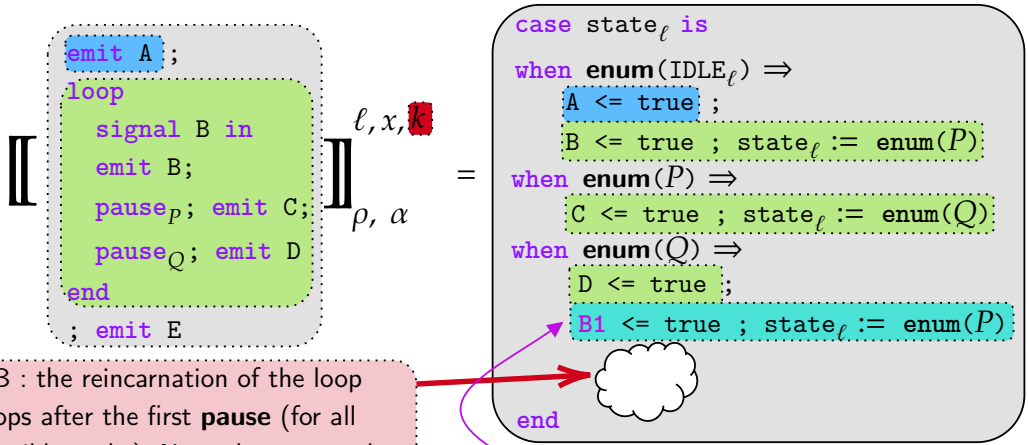
$[\![\ \texttt{pause}_Q\ ]\!]^{\ell,x,k}_{\rho,\mathbf{2}}$ =

(end of loop)

```
case state_ℓ is
when enum(IDLE_ℓ) =>
   state_ℓ := enum(Q)
end
```

NB: the transition associated to Q already have been compiled in the first part of the current loop

Example of compilation



Parallel execution (1/3)

$$[\![\ \texttt{emit A ;}\ \texttt{[}\ e\ \texttt{||}\ e'\ \texttt{]}\ ]\!]^{L,x,k}_{\rho,\ \alpha} =$$

```
case state_L is
when enum(IDLE_L) =>
     A <= true ;
     (e)^{ℓ,y}_{ρ,α}
     ;
     (e')^{ℓ',y'}_{ρ,α}
     ;
     if rdy_ℓ & rdy_ℓ' then x_0 := (x,x'); k
     else state_L := enum(IDLE_L) ;
end
```

(reexecuting the past)

WRONG !

NB: by construction, each time rdy_ℓ [*resp.* $\text{rdy}_{\ell'}$] is true, the FSM returns to state **enum**(IDLE_ℓ) [*resp.* **enum**($\text{IDLE}_{\ell'}$)]

(ℓ, ℓ', y, y' are fresh)

Parallel execution (2/3)

$$[\![\ \texttt{emit A ;}\ [\ e\ ||\ e'\ ]\ ]\!]^{L,x,k}_{\rho,\alpha} =$$

```
case state_L is
when enum(IDLE_L) =>
     A <= true ; s ; s' ;
     if rdy_ℓ & rdy_ℓ' then k
     else state_L := enum(IDLE_L)
when enum(WAIT) =>
     case state_ℓ is
     when enum(IDLE_ℓ) => () t_1 ··· t_n
     end ;
     case state_ℓ' is
     when enum(IDLE_ℓ') => () t'_1 ··· t'_m
     end ;
     if rdy_ℓ & rdy_ℓ' then k
     else state_L := enum(WAIT)
end
```

(ℓ, ℓ', y y', *WAIT* are fresh)

Parallel execution (3/3)

```
disjonction_traps(x_1, ··· x_n)
  := x_1 ∨ ··· ∨ x_n
```

$$[\![\; [\; e \;||\; e' \;] \;]\!]^{L,\,x,\,k}_{\rho,\,\alpha} =$$

```
case state_L is
when enum(IDLE_L) =>
  s ; s' ; if rdy_ℓ & rdy_ℓ' then
           (if disjonction_traps(ρ) then () else k)
           else state_L := enum(IDLE_L)
when enum(WAIT) =>
    case state_ℓ is
    when enum(IDLE_ℓ) => () t_1 ··· t_n
    end ;
    case state_ℓ' is
    when enum(IDLE_ℓ') => () t'_1 ··· t'_m
    end ;
    if rdy_ℓ & rdy_ℓ' then
    (if disjonction_traps(ρ) then () else k)
    else state_L := enum(WAIT)
end
```

(ℓ, ℓ', y y', *WAIT* are fresh)