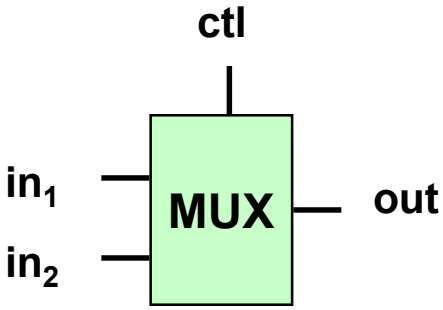
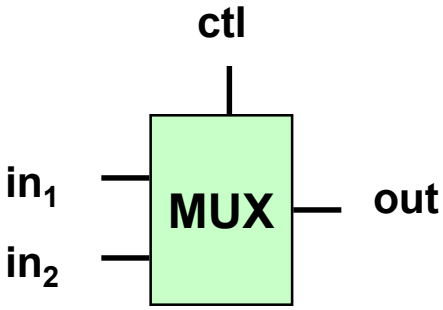
Elementary Circuits



Multiplexer

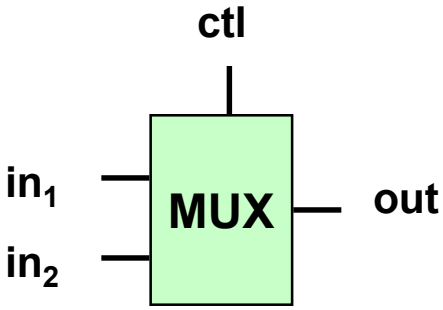
Elementary Circuits



Multiplexer

```
MUX(ctl,in1,in2,out) <==>
 ∀ t .  out t = if (ctl t) then (in1 t)
                           else (in2 t)
```

Elementary Circuits

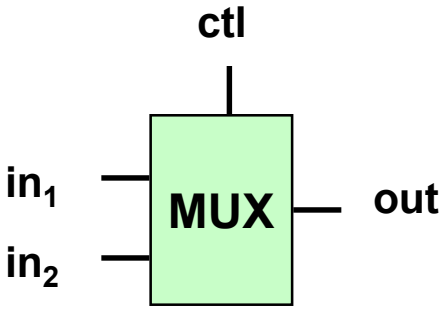


Multiplexer

```
MUX(ctl,in1,in2,out) <==>
 ∀ t .  out t = if (ctl t) then (in1 t)
                           else (in2 t)
```

Delay

Elementary Circuits



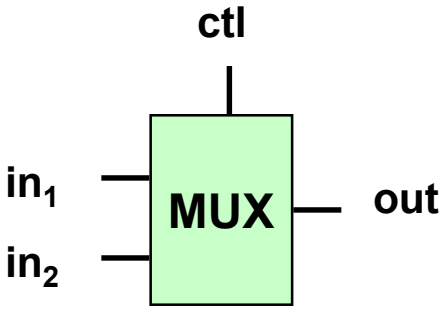
Multiplexer

```
MUX(ctl,in1,in2,out) <==>
 ∀ t .  out t = if (ctl t) then (in1 t)
                           else (in2 t)
```

Delay

```
DELAY(in,out) <==>
                  ∀ t . out (t+1) = in t
```

Elementary Circuits

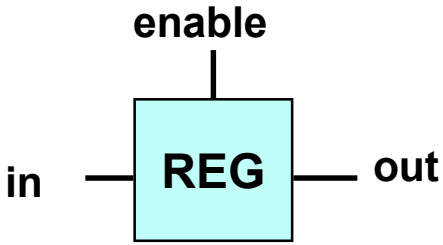


Multiplexer

```
MUX(ctl,in1,in2,out) <==>
 ∀ t .  out t = if (ctl t) then (in1 t)
                           else (in2 t)
```

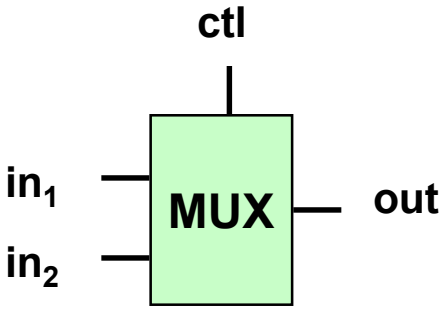
Delay

```
DELAY(in,out) <==>
                  ∀ t . out (t+1) = in t
```



Register

Elementary Circuits

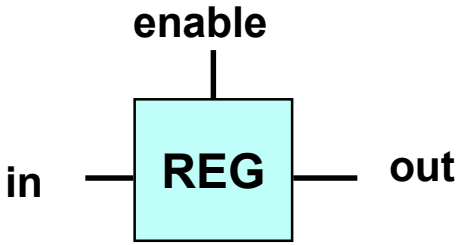


Multiplexer

```
MUX(ctl,in1,in2,out) <==>
 ∀ t .  out t = if (ctl t) then (in1 t)
                           else (in2 t)
```

Delay

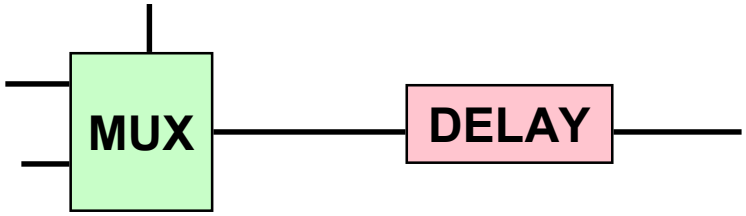
```
DELAY(in,out) <==>
                  ∀ t . out (t+1) = in t
```



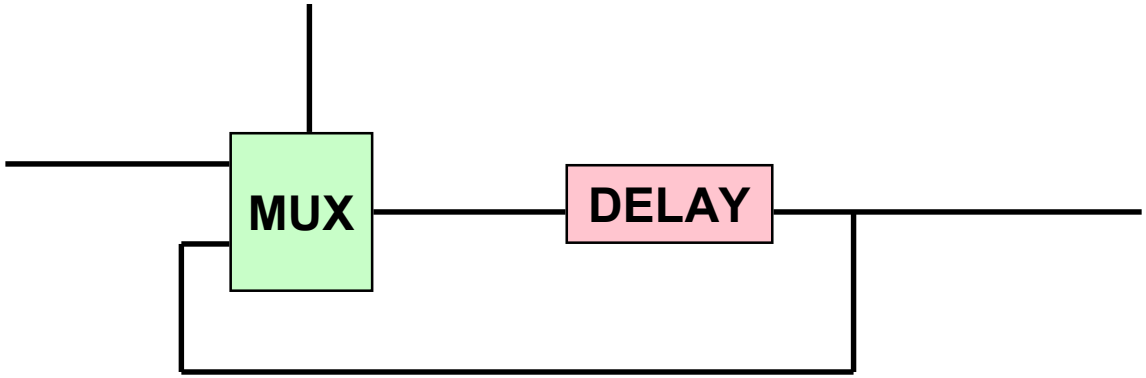
Register

```
REG(in,enable,out) <==>
  ∀ t . out (t+1) =
           if (enable t) then (in t)
                         else (out t)
```

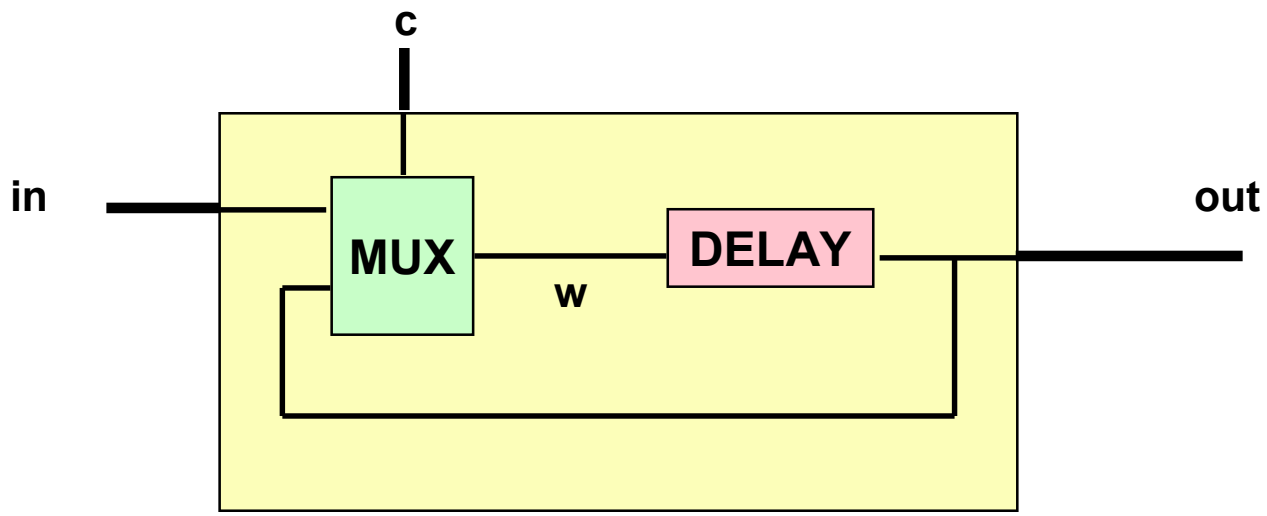
Construction of a circuit



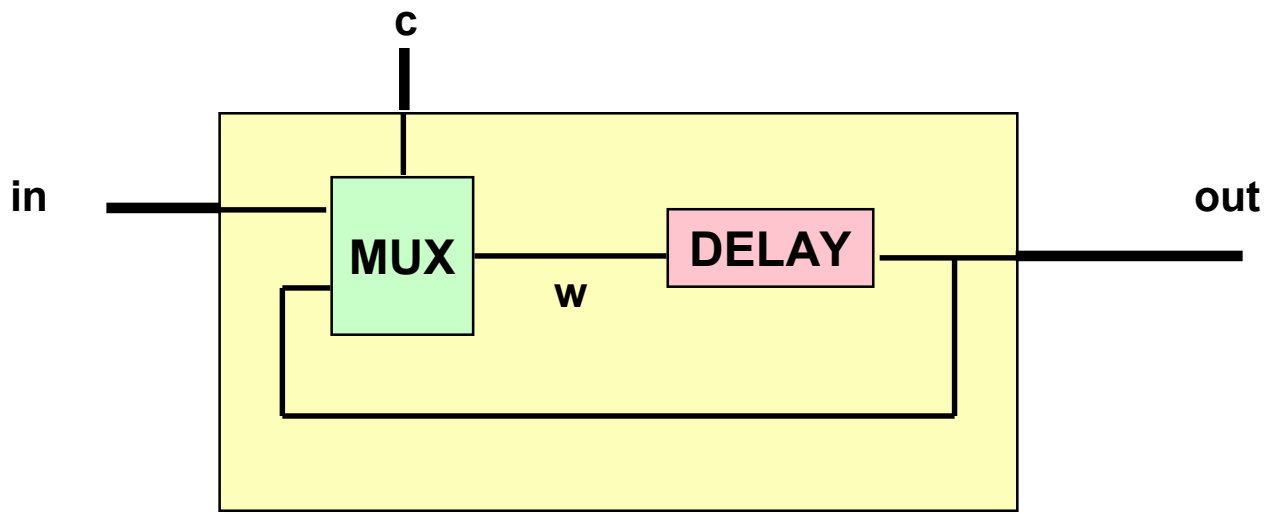
Construction of a circuit



Construction of a circuit



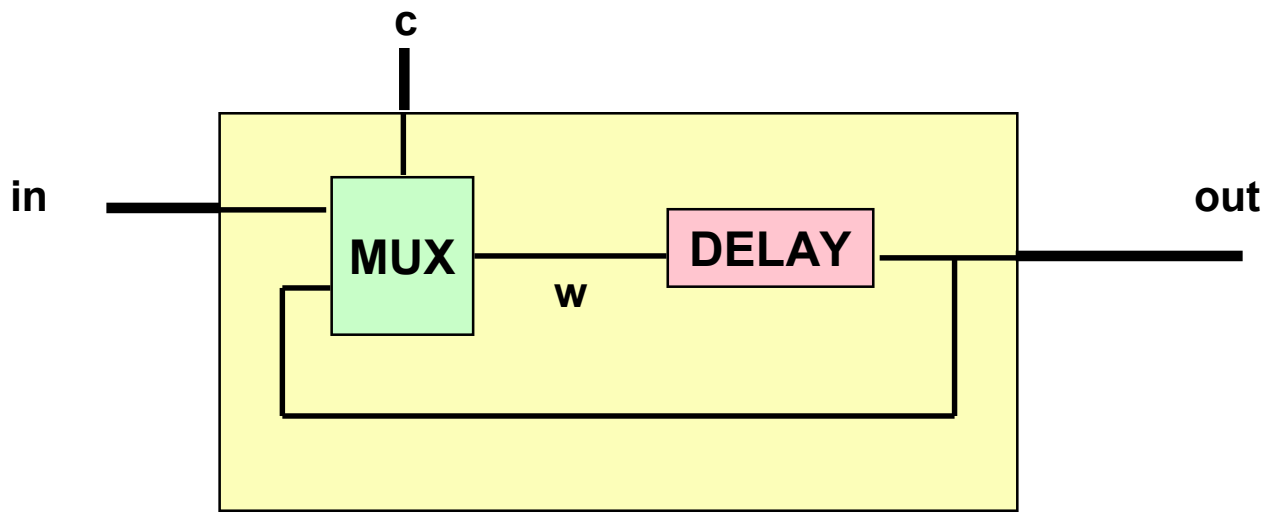
Construction of a circuit



Description:

```
Circ(c,in,out) <==> ∃ w . MUX(c,in,out,w) ∧ DELAY(w,out)
```

Construction of a circuit

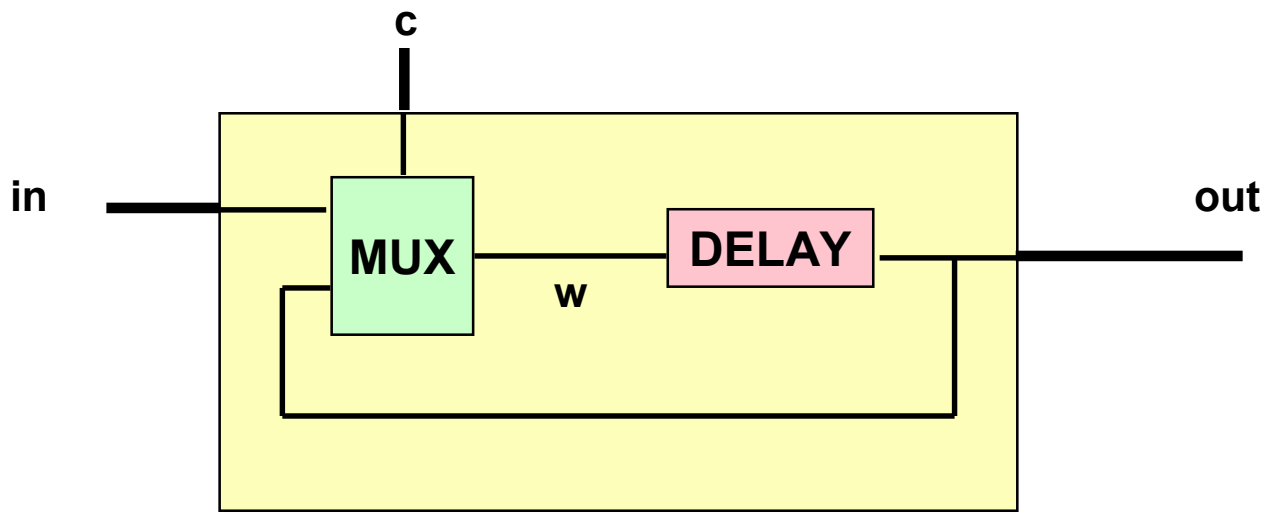


Description:

$$\mathtt{Circ(c,in,out)} \iff \exists\ \mathtt{w}\ .\ \mathtt{MUX(c,in,out,w)} \wedge \mathtt{DELAY(w,out)}$$

Claim: $\mathtt{Circ(c,in,out)} \vdash \mathtt{REG(in,c,\ out)}$

Construction of a circuit



Description:

```
Circ(c,in,out) <==> ∃ w . MUX(c,in,out,w) ∧ DELAY(w,out)
```

Claim:

```
Circ(c,in,out) |- REG(in,c, out)
```

Sequent :

```
MUX(c,in,out,w), DELAY(w,out) |- REG(in,c, out)
```

Sequential Calculus

Sequent :

$$C_1, C_2, \ldots, C_n \;\; |\!-- \;\; D$$

" From the assumptions $C_1, C_2, \ldots, C_n$ derive D "

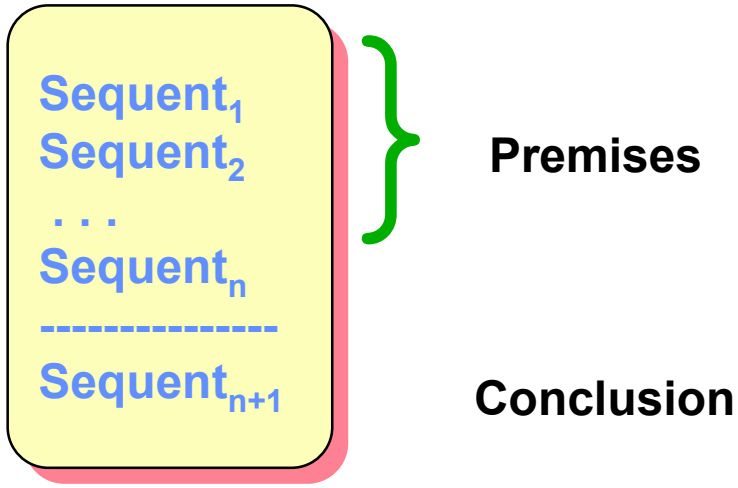
Interpretation in circuit design :

" The components $C_1, C_2, \ldots, C_n$ satisfy specification D "

Example :

```
MUX(c,in,out,w), DELAY(w,out) |- REG(in,c, out)
```

Rules for Sequential Calculus



Examples :

$$\frac{\Gamma \vdash P \qquad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q}$$

$$\frac{P, \Gamma \vdash Q}{\Gamma \vdash P \rightarrow Q}$$

$$\frac{}{P, \Gamma \vdash P}$$

Proof of the register

Claim : `MUX(c,in,out,w), DELAY(w,out) |- REG(in,c, out)`

Proof : `MUX(c,in,out,w), DELAY(w,out) |- w = w`

`=> ∃ t, MUX(c,in,out,w), DELAY(w,out) |- w t = w t`

MUX-Rule :

```
Γ, MUX(c,u,v,w), Δ  |--  C#(w t)
-------------------------------------------------
Γ, MUX(c,u,v,w), Δ  |--  C#(if(c t)then(u t)else(v t))
```

Proof of the register

Claim : `MUX(c,in,out,w), DELAY(w,out) |- REG(in,c, out)`

Proof : `MUX(c,in,out,w), DELAY(w,out) |- w = w`

`=> ∃ t, MUX(c,in,out,w), DELAY(w,out) |- w t = w t`

MUX-Rule :

$$\frac{\Gamma, \mathtt{MUX(c,u,v,w)}, \Delta \ \mathtt{|--} \ \mathtt{C\#(w\ t)}}{\Gamma, \mathtt{MUX(c,u,v,w)}, \Delta \ \mathtt{|--} \ \mathtt{C\#(if(c\ t)then(u\ t)else(v\ t))}}$$

`=> ∃ t, MUX(c,in,out,w),DELAY(w,out) |- w t = if(c t)then(in t)`
`else(out t)`

Proof of the register

```
=> ∃ t,  MUX(c,in,out,w), DELAY(w,out) |- w t=if(c t) then(in t)
                                          else(out t)
```

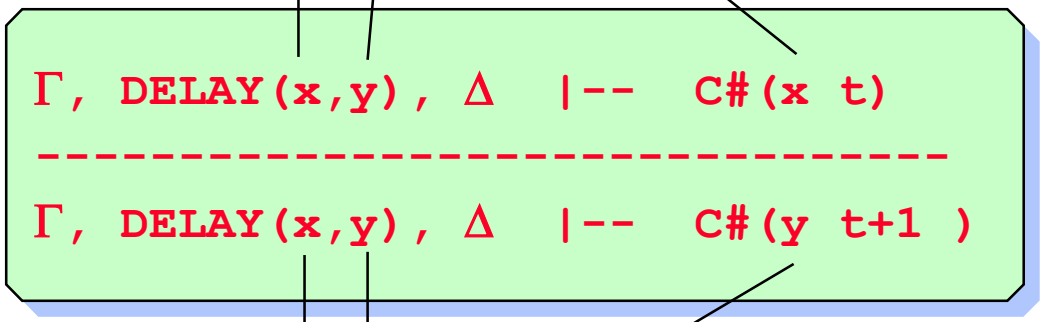
DELAY-Rule :

$$\frac{\Gamma, \mathtt{DELAY(x,y)}, \Delta \ \mathtt{|--} \ \mathtt{C\#(x\ t)}}{\Gamma, \mathtt{DELAY(x,y)}, \Delta \ \mathtt{|--} \ \mathtt{C\#(y\ t+1\ )}}$$

Proof of the register

```
=> ∃ t,  MUX(c,in,out,w), DELAY(w,out) |- w t=if(c t) then(in t)
                                          else(out t)
```

DELAY-Rule :




```
=> ∃ t,  MUX(c,in,out,w), DELAY(w,out) |- out t+1 =
                                          if(c t) then(in t)
                                          else(out t)

=>   MUX(c,in,out,w), DELAY(w,out) |- ∀ t. out t+1 =
                                          if(c t) then(in t)
                                          else(out t)

=>   MUX(c,in,out,w), DELAY(w,out) |- REG(in, c, out)
```

q.e.d.

Synthesis of the register

Components ⊢ Specification

Synthesis of the register

here: Π `|- REG(in, c, out)`

Components |- Specification

Synthesis of the register

$$\Pi \vdash \forall\, t.\ out\ t+1 = if(c\ t)\ then\ (in\ t)\ else(out\ t)$$

here: $\Pi \vdash REG(in, c, out)$

Components |- Specification

Synthesis of the register

$$\exists\ t, \Pi \vdash \texttt{out t+1 = if(c t) then (in t) else(out t)}$$

$$\Pi \vdash \forall\ t.\ \texttt{out t+1 = if(c t) then (in t) else(out t)}$$

here: $\Pi \vdash \texttt{REG(in, c, out)}$

Components |- Specification

Synthesis of the register

$$\frac{\Gamma,\ \mathtt{DELAY(x,y)},\ \Delta \ \vdash\ \mathtt{C\#(x\ t)}}{\Gamma,\ \mathtt{DELAY(x,y)},\ \Delta \ \vdash\ \mathtt{C\#(y\ t+1\ )}}$$

$$\exists\ t, \Pi \ \vdash\ \mathtt{out\ t+1\ =\ if(c\ t)\ then\ (in\ t)\ else(out\ t)}$$

$$\Pi \ \vdash\ \forall\ \mathtt{t.\ out\ t+1\ =\ if(c\ t)\ then\ (in\ t)\ else(out\ t)}$$

here: $\Pi \ \vdash\ \mathtt{REG(in,\ c,\ out)}$

Components |- Specification

Synthesis of the register

$\exists$ t, DELAY(x,out), Δ |- x t=if(c t) then(in t)
else(out t)

Γ, DELAY(x,y), Δ |- C#(x t)

Γ, DELAY(x,y), Δ |- C#(y t+1)

$\exists$ t,Π |- out t+1 = if(c t) then (in t)
else(out t)

Π |- $\forall$ t. out t+1 = if(c t) then (in t)
else(out t)

here: Π |- REG(in, c, out)

Components |- Specification

Synthesis of the register

$\exists$ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)

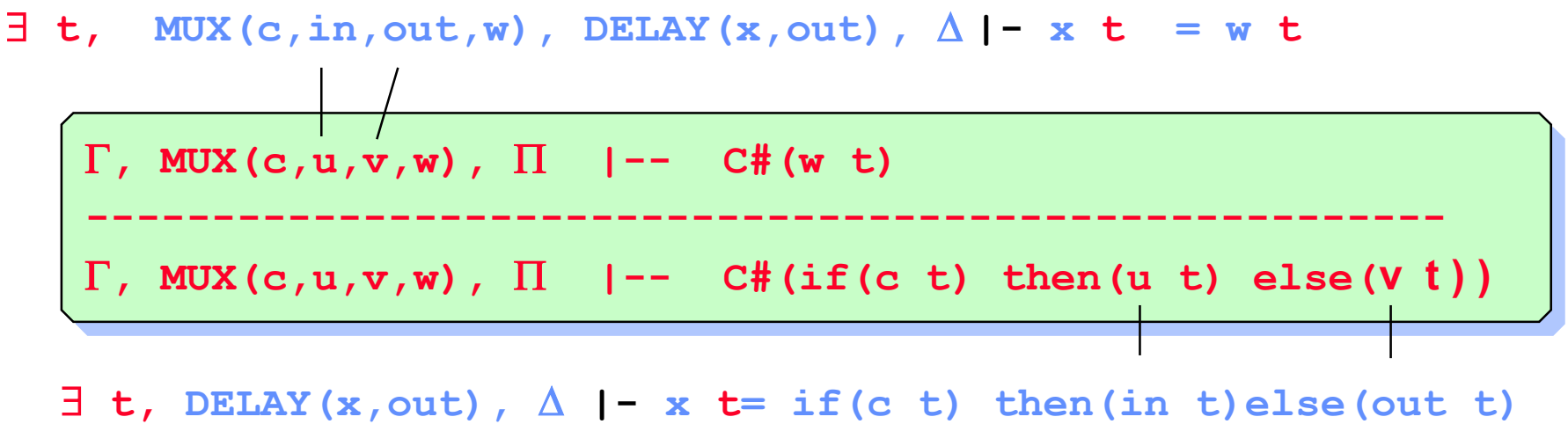
Synthesis of the register

Γ, MUX(c,u,v,w), Π |-- C#(w t)

Γ, MUX(c,u,v,w), Π |-- C#(if(c t) then(u t) else(v t))

$\exists$ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)

Synthesis of the register



Synthesis of the register

MUX(c,in,out,w), DELAY(x,out), Δ |- ∀ t. x t = w t

∃ t, MUX(c,in,out,w), DELAY(x,out), Δ |- x t = w t

Γ, MUX(c,u,v,w), Π |-- C#(w t)

Γ, MUX(c,u,v,w), Π |-- C#(if(c t) then(u t) else(v t))

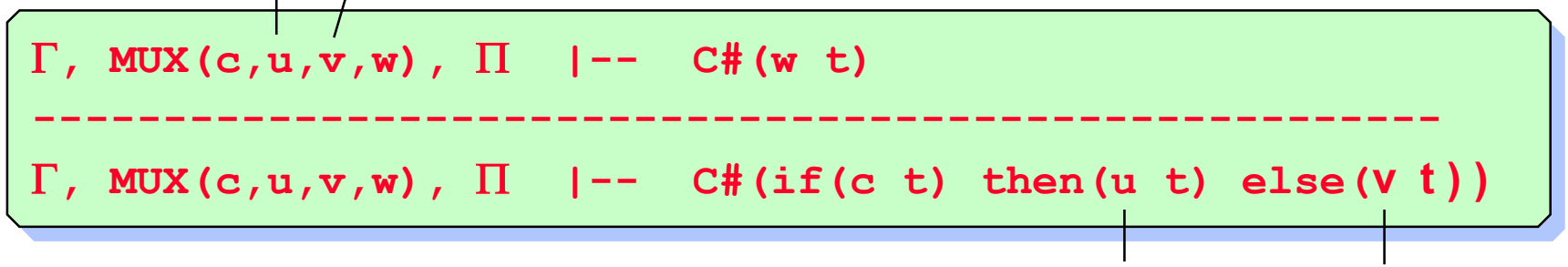
∃ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)

Synthesis of the register

MUX(c,in,out,w), DELAY(x,out), Δ |- x = w

MUX(c,in,out,w), DELAY(x,out), Δ |- ∀ t. x t = w t

∃ t, MUX(c,in,out,w), DELAY(x,out), Δ |- x t = w t



∃ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)

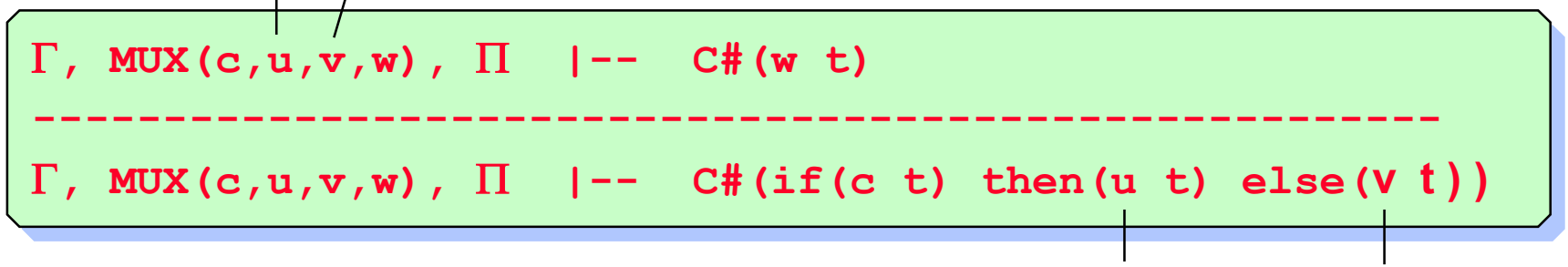
Synthesis of the register

MUX(c,in,out,w), DELAY(w,out), x = w, Δ |- x = w

MUX(c,in,out,w), DELAY(x,out), Δ |- x = w

MUX(c,in,out,w), DELAY(x,out), Δ |- ∀ t. x t = w t

∃ t, MUX(c,in,out,w), DELAY(x,out), Δ |- x t = w t



∃ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)

Synthesis of the register

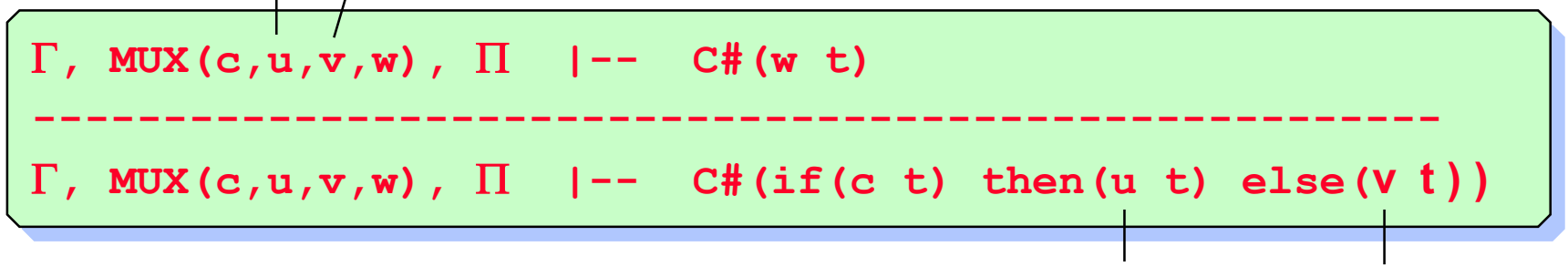
```
MUX(c,in,out,w), DELAY(w,out), x = w |- x = w
```

```
MUX(c,in,out,w), DELAY(w,out), x = w, Δ |- x = w
```

```
MUX(c,in,out,w), DELAY(x,out), Δ |-  x = w
```

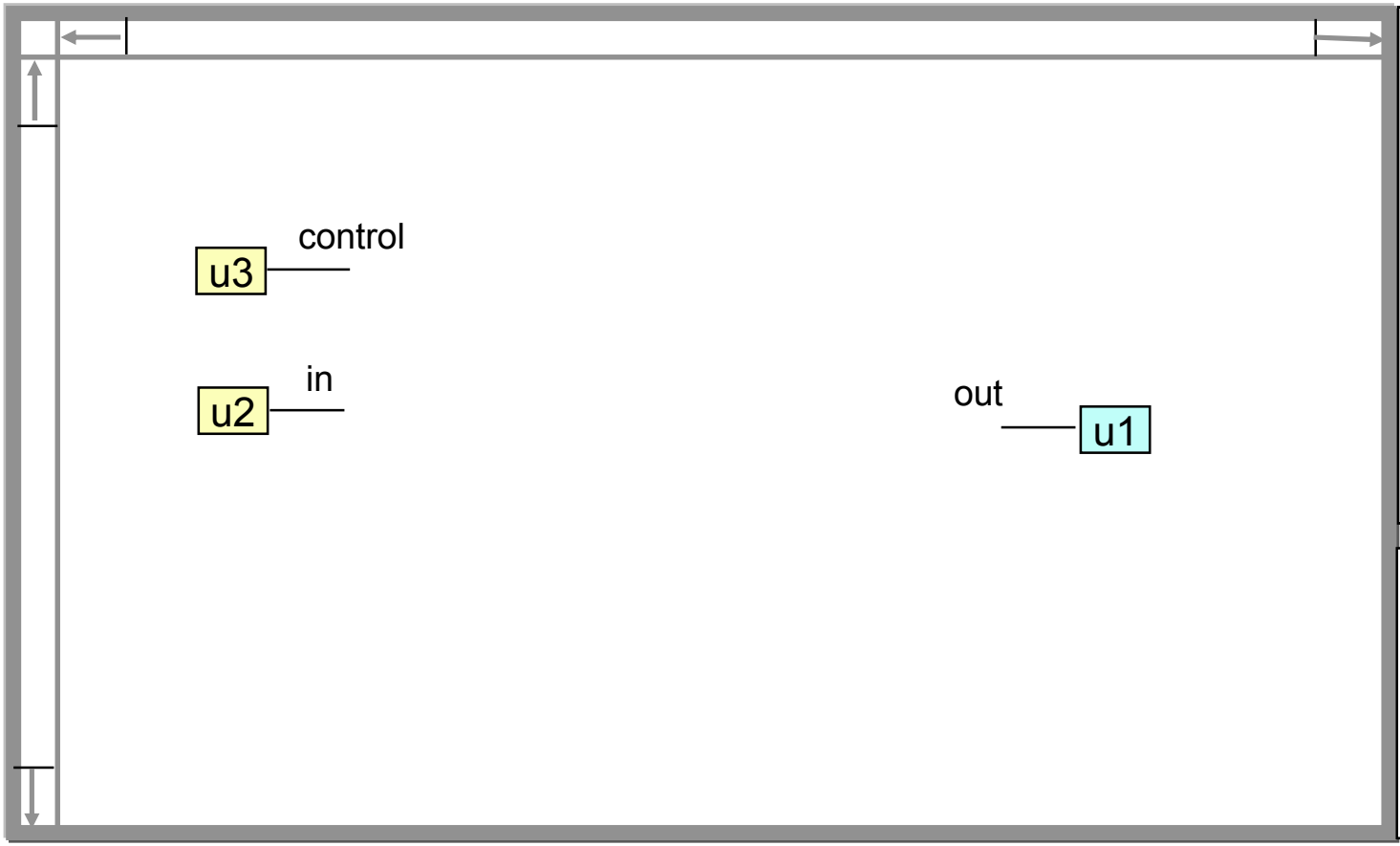
```
MUX(c,in,out,w), DELAY(x,out), Δ |- ∀ t. x t  = w t
```

```
∃ t,  MUX(c,in,out,w), DELAY(x,out), Δ |- x t  = w t
```




```
∃ t, DELAY(x,out), Δ |- x t= if(c t) then(in t)else(out t)
```

Lambda : λ

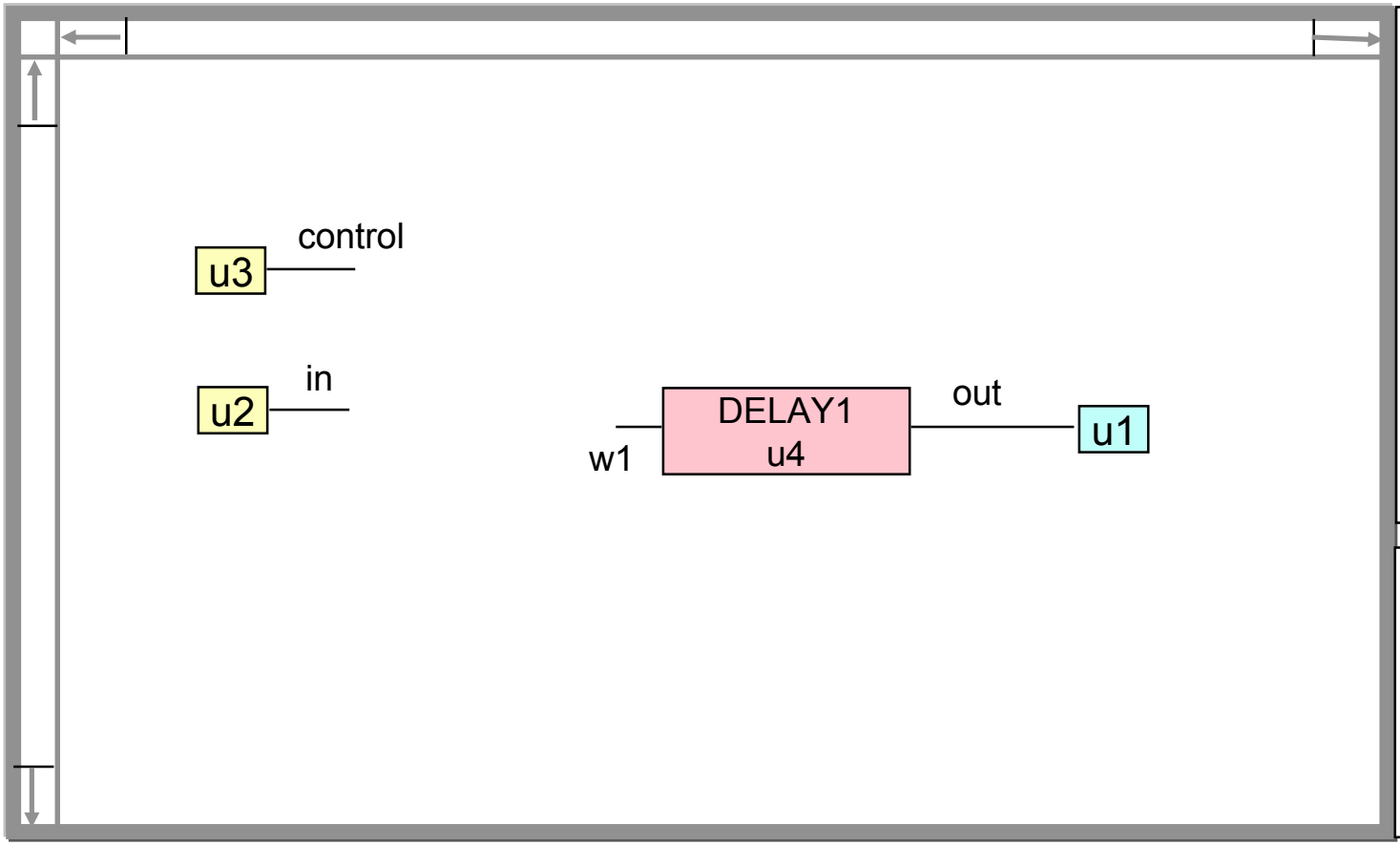


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT

```
E t ... |- out (S t) === (if control t then in t else out t)
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Lambda : λ

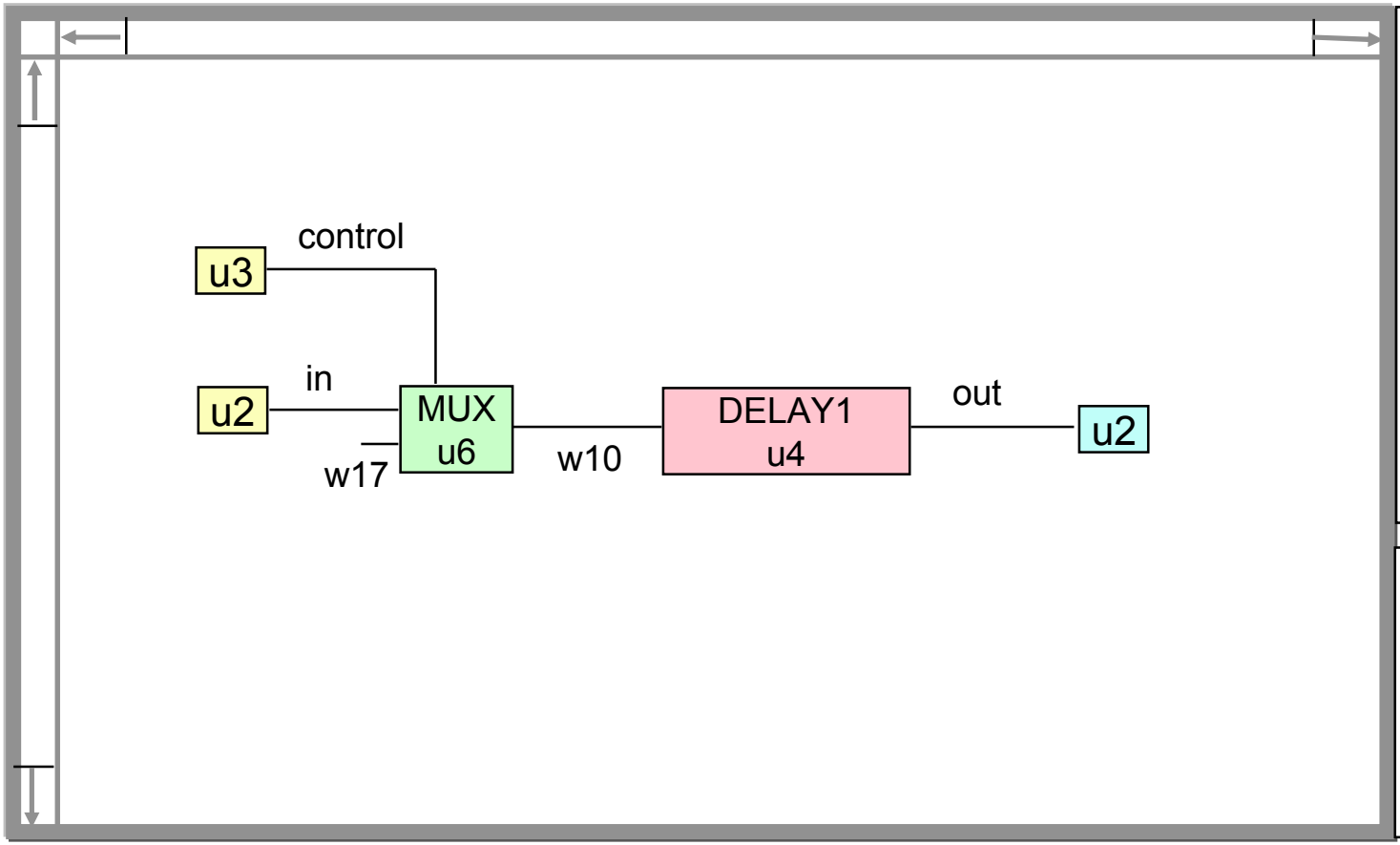


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	DELAY1
u6	MUX

```
E t ... |- w1 t === (if control t then in t else out t)
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Lambda : λ

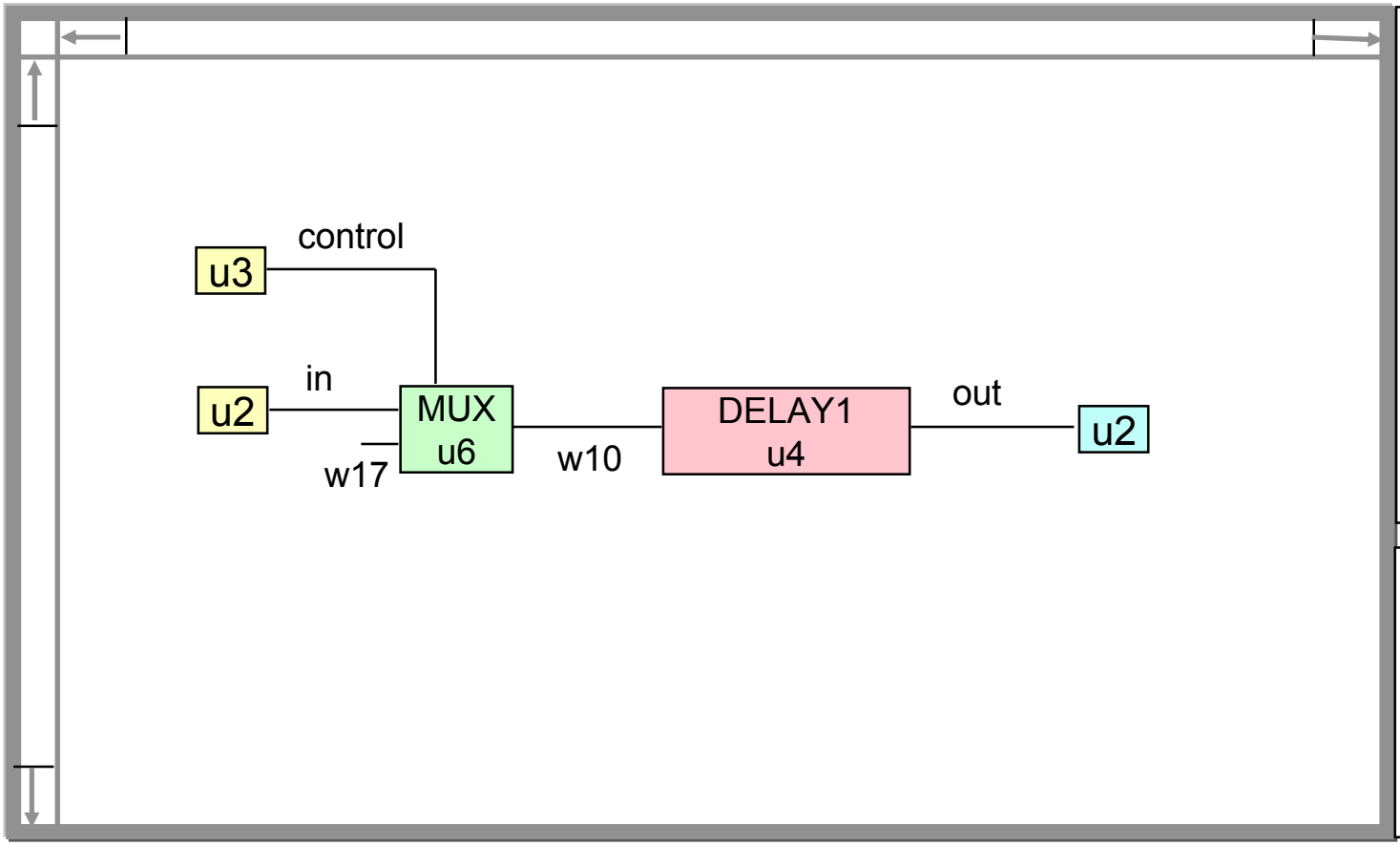


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	DELAY1
u6	MUX

```
E t ... |- w17 t === out t
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Lambda : λ

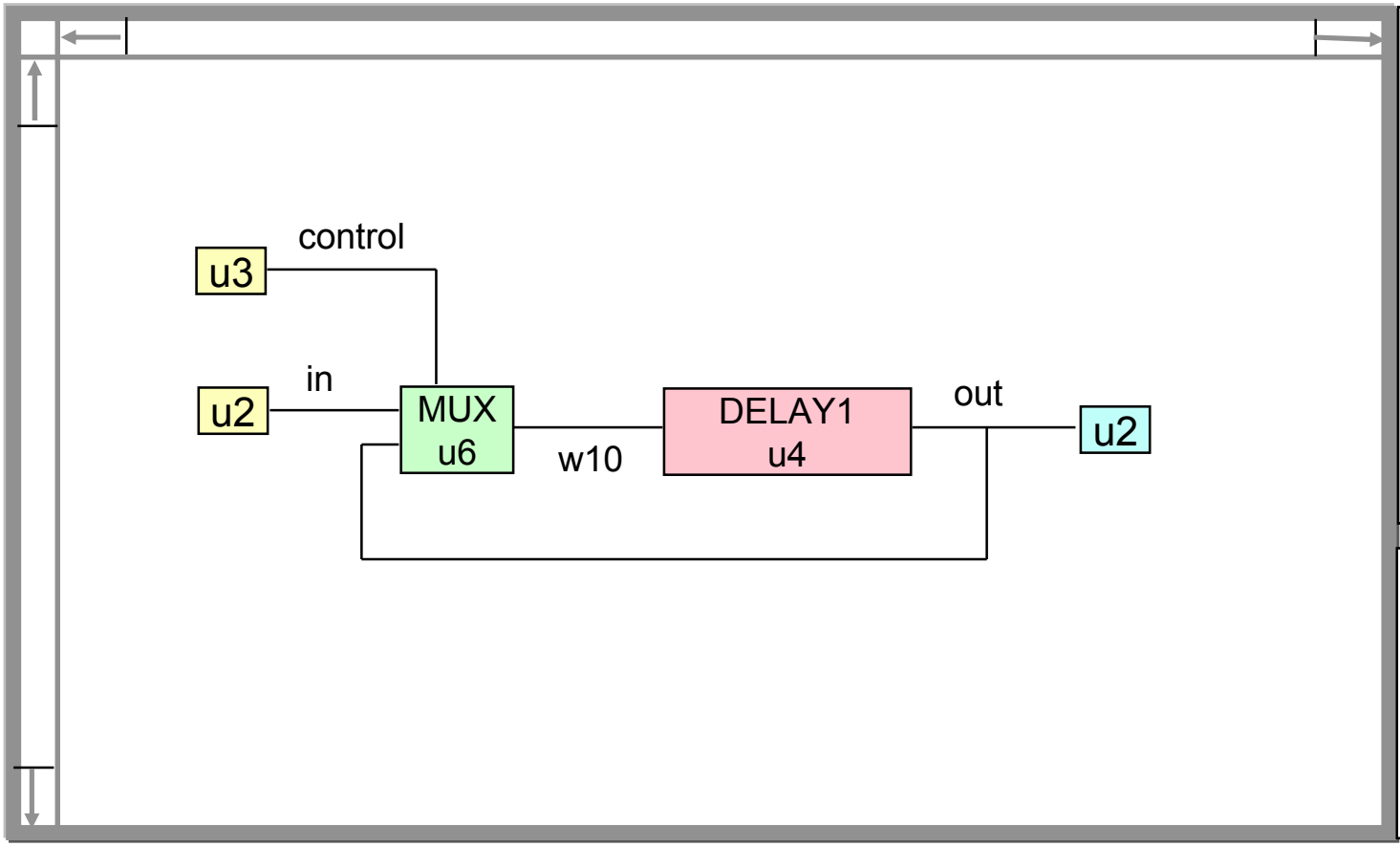


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	DELAY1
u6	MUX

```
 ... |- w17  === out
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Lambda : λ

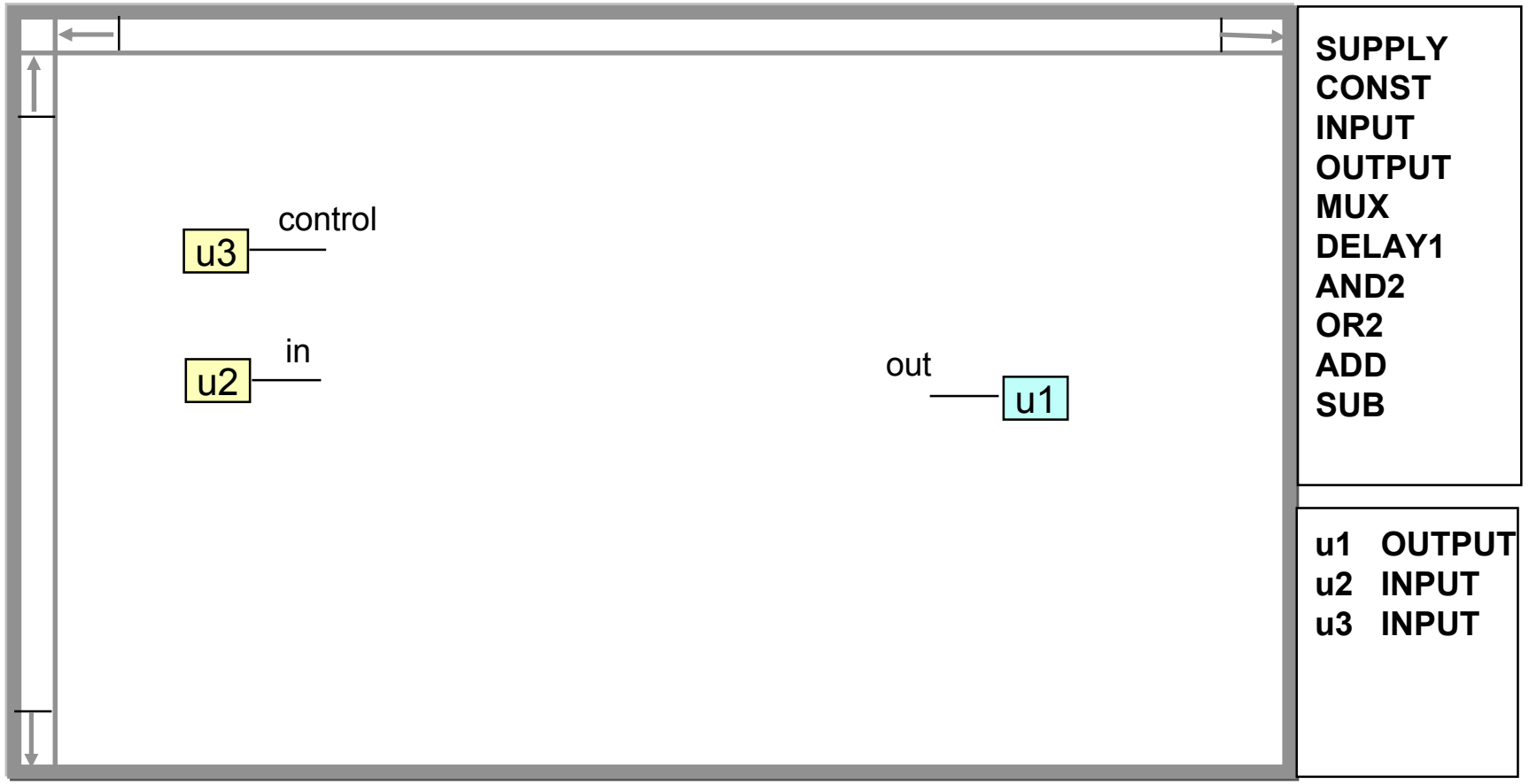


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	DELAY1
u6	MUX

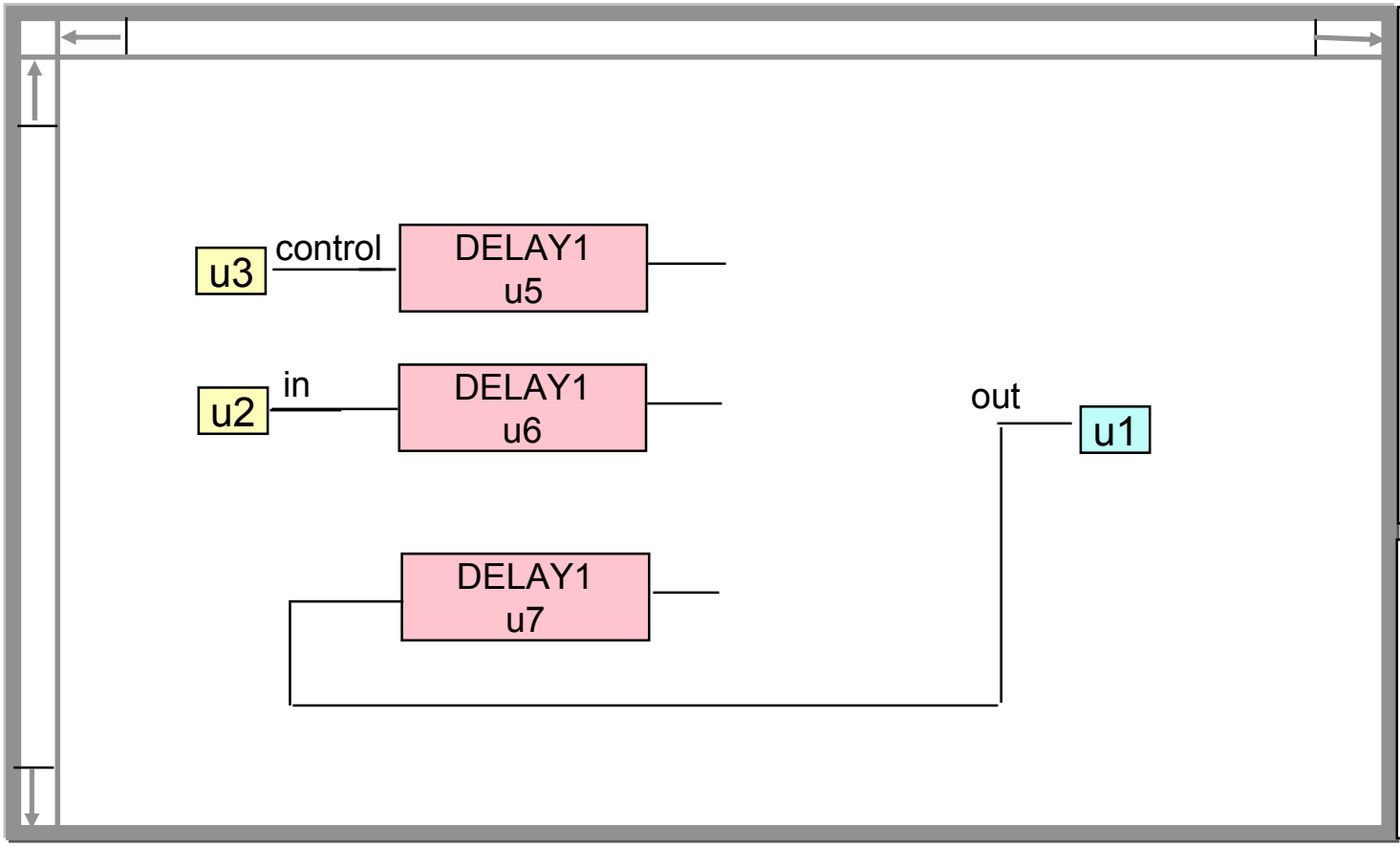
```
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Another Solution




```
E t ... |- out (S t) === (if control t then in t else out t)
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Another Solution

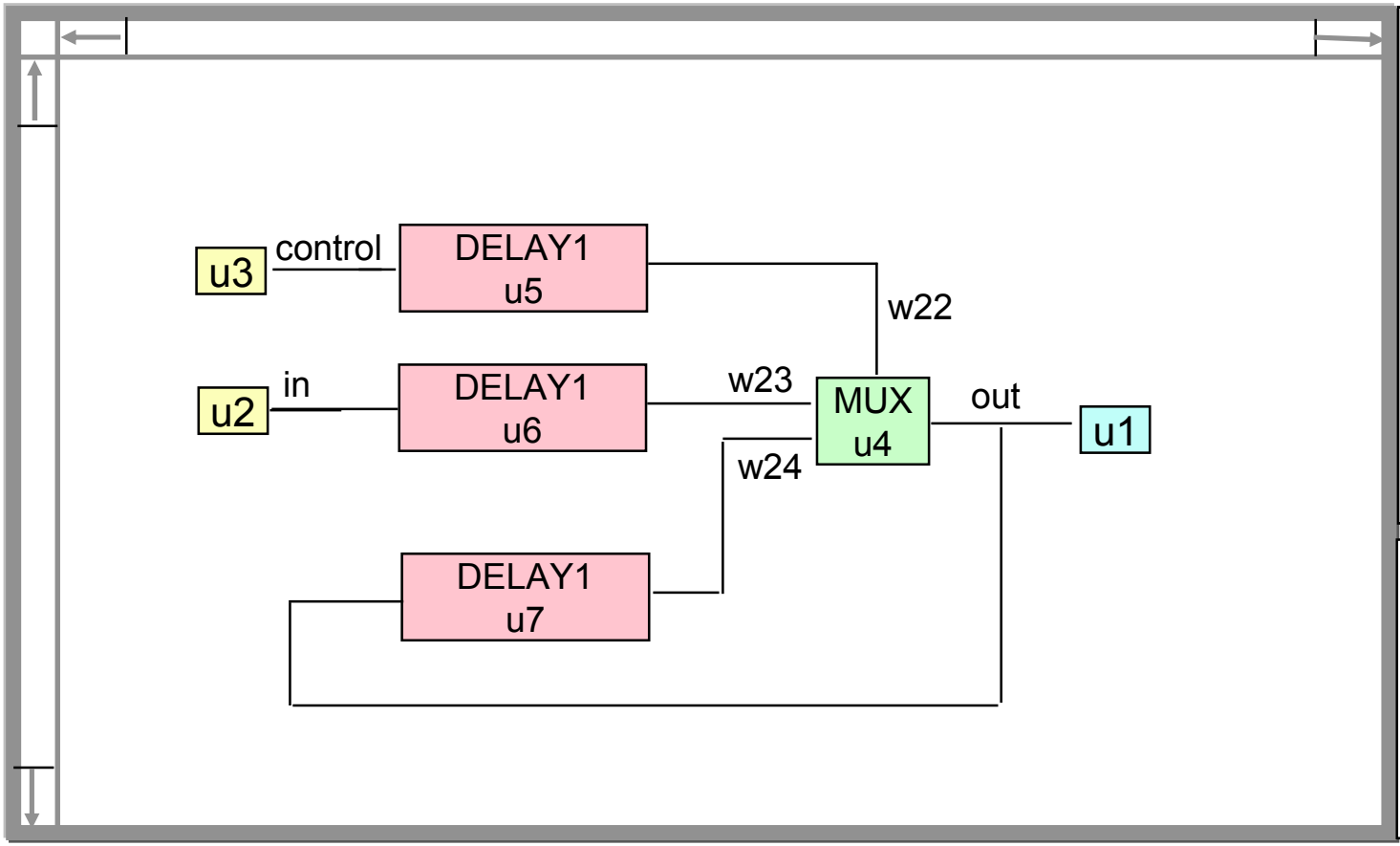


SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	MUX
u5	DELAY1
u6	DELAY1
u7	DELAY1

```
E t, ... |- out (S t) === (if control (S t) then in (S t) else out (S t) )
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Another Solution



SUPPLY
CONST
INPUT
OUTPUT
MUX
DELAY1
AND2
OR2
ADD
SUB

u1	OUTPUT
u2	INPUT
u3	INPUT
u4	MUX
u5	DELAY1
u6	DELAY1
u7	DELAY1

```
...
-----------------------------------
...  |-  forall t. out (S t) === (if control t then in t else out t )
```

Specification Language : ML

Objects

```
datatype natural = 0
                 | S of natural   ;

   type   zeit       =  natural      ;
   type   α signal   =  zeit --> α   ;
   type   boolsig    =  bool signal  ;
   type   word       =  bool list    ;
   type   wordsig    =  word signal  ;
```

Specification Language : ML

Functions

```
fun   add      0 y = y
      add (S x) y = S (add x y)

fun rest e t = e (S t)
```

... resp. ...

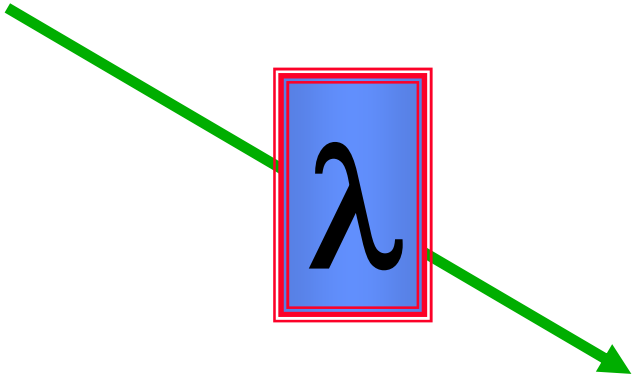
```
fun rest e = fn t => e (S t)

fun register in c =
        fn (S t) => if (c t) then (in t)
                    else (register in c t)
```

Proof Rules for Types

Every type definition leads to an induction scheme

```
datatype natural = 0
                 | S of natural   ;
```

Type



Rule

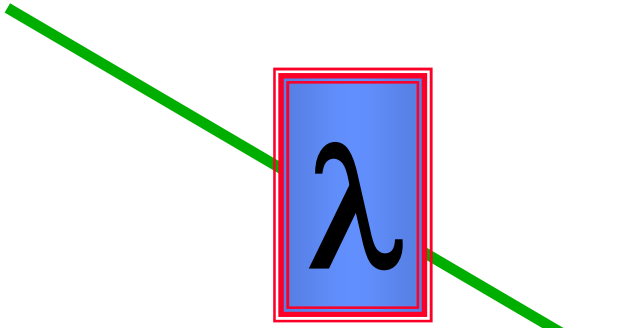
$$\frac{\Gamma \vdash P\#(0) \qquad \exists w.\ P\#(w), \Gamma \vdash P\#(S\ w)}{\Gamma \vdash \forall w.\ P\#(w)}$$

Proof Rules for Functions

Every function definition leads to a replacement rule

```
fun add 0 y = y
  | add (S x) y = S ( add x y)
```

Function



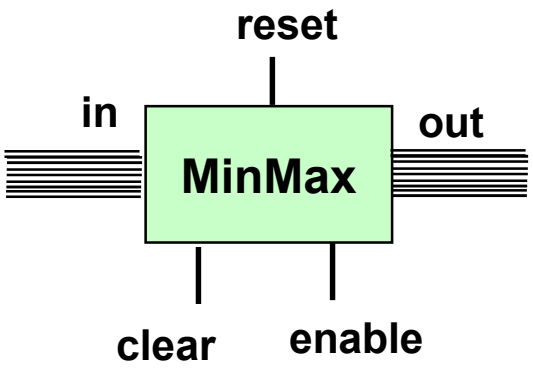
Rules

$$\frac{\Gamma \vdash P\#(\text{add } 0\ y)}{\Gamma \vdash P\#(y)}$$

$$\frac{\Gamma \vdash P\#(\text{add } (S\ x)\ y)}{\Gamma \vdash P\#(S\ (\text{add } x\ y))}$$

The MinMax Problem

The following specification was introduced as a benchmark for verification and synthesis tools :



- in, out $\in$ {0 .. 256}
- **clear** = true => **out**=0
- if the last time **reset** became false was at ***t=c*** then

$$out(t) = \frac{\max_{t>c}[in(t)] + \min_{t>c}[in(t)]}{2}$$

- ... etc. ...

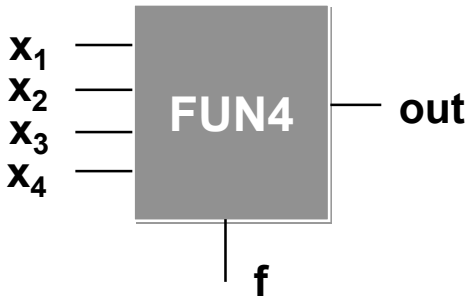
MinMax Specification in ML

The specification is translated into ML ...

```
fun  MinMax (cl, en, re, in) t =
       if (cl t) then (zero 8)
       else if (not reset) then
            mean(max(cl,en,re,in),min(cl,en,re,in))
       else if ....
           ...
fun max (cl,en,re,   in)     0  = 0
    max (cl,en,True ,in) (S t) = 0
    max (cl,en,False,in) (S t) =
                  maximum(max(cl,en,False,in) t,in)
        ... etc. ...
```

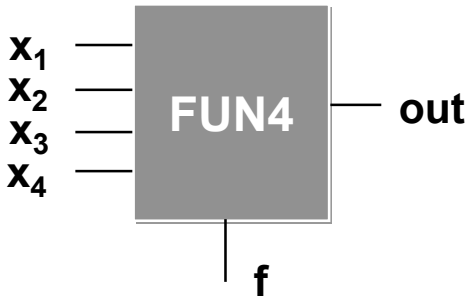
Modularization

**In order to break up complexity, abstract circuits are introduced .
Abstract circuits lead to higher order specifications.**



Modularization

In order to break up complexity, abstract circuits are introduced .
Abstract circuits lead to higher order specifications.

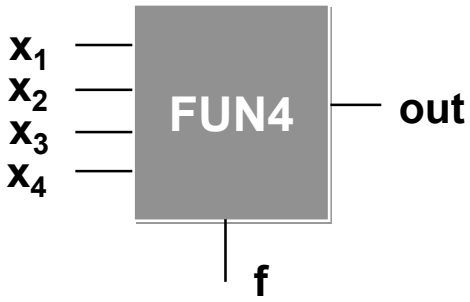


Specification :

$$\mathtt{FUN4}(f, x_1, x_2, x_3, x_4, out) \iff \forall\ t\ .\ out\ t = f(x_1, x_2, x_3, x_4)\ t$$

Modularization

In order to break up complexity, abstract circuits are introduced .
Abstract circuits lead to higher order specifications.



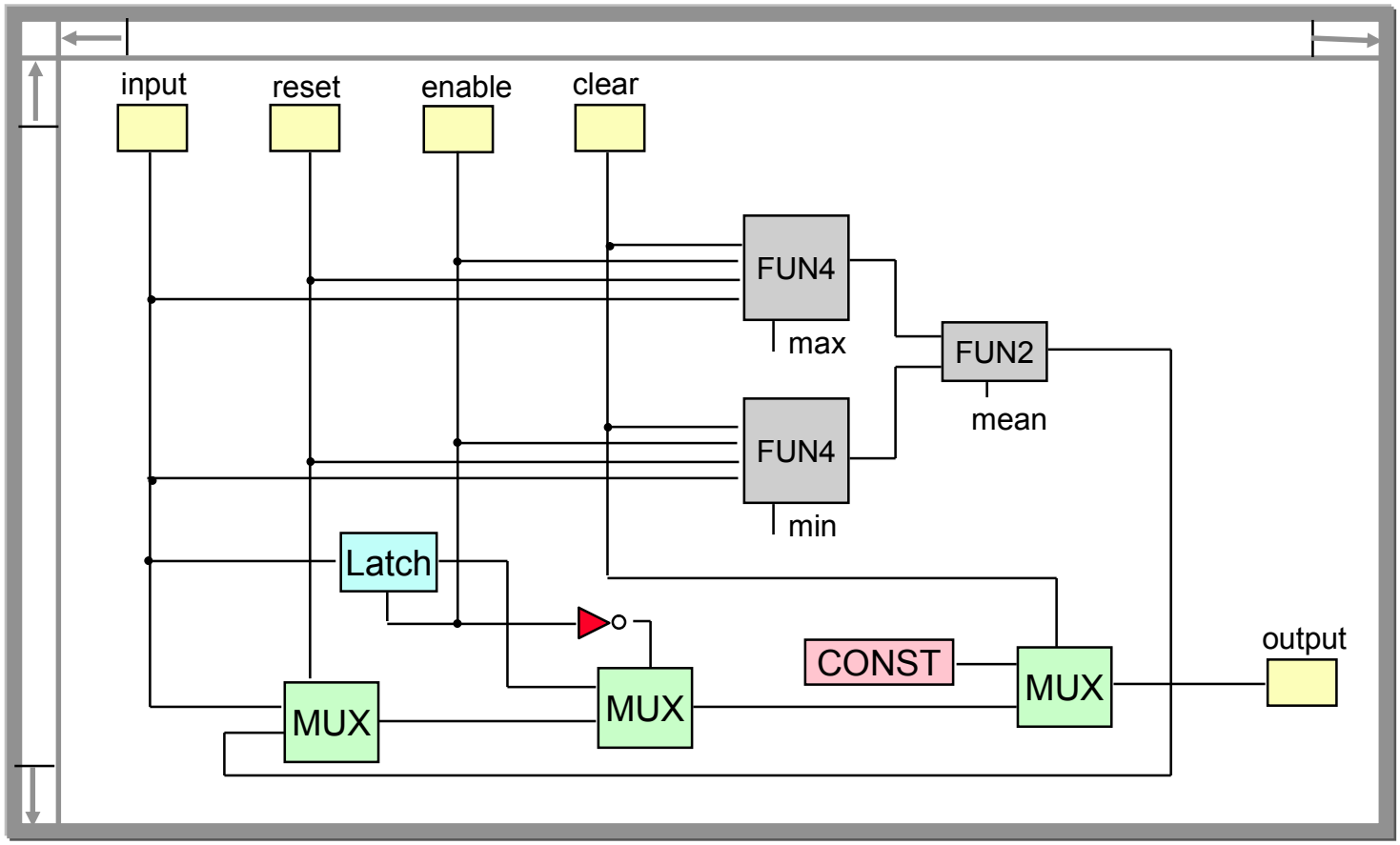
Specification :

$$\texttt{FUN4(f,}x_1\texttt{,}x_2\texttt{,}x_3\texttt{,}x_4\texttt{,out)} \iff \forall\ \texttt{t .\ out\ t = f(}x_1\texttt{,}x_2\texttt{,}x_3\texttt{,}x_4\texttt{)\ t}$$

Replacement rule

$$\frac{\Gamma,\ \texttt{FUN4(f,}x_1\texttt{,}x_2\texttt{,}x_3\texttt{,}x_4\texttt{,out)},\ \Delta\ \vdash\ \texttt{P\#(out t)}}{\Gamma,\ \texttt{FUN4(f,}x_1\texttt{,}x_2\texttt{,}x_3\texttt{,}x_4\texttt{,out)},\ \Delta\ \vdash\ \texttt{P\#(f(}x_1\texttt{,}x_2\texttt{,}x_3\texttt{,}x_4\texttt{)\ t)}}$$

MinMax




```
...
-----------------------------------
...   |-  minmax#(clear,enable,reset,input,output)
```