Synchronous Languages—Lecture 07

Prof. Dr. Reinhard von Hanxleden

Christian-Albrechts Universität Kiel
Department of Computer Science
Real-Time Systems and Embedded Systems Group

20 Nov. 2018

Last compiled: November 18, 2018, 16:50 hrs

Esterel V—The Constructive Circuit Semantics

The 5-Minute Review Session

1. What is the derivative (*Ableitung*) of a program?

The 5-Minute Review Session

1. What is the derivative (*Ableitung*) of a program?
2. How is the *program transition* of an Esterel program defined?

The 5-Minute Review Session

1. What is the derivative (*Ableitung*) of a program?
2. How is the *program transition* of an Esterel program defined?
3. How do program transitions express logical coherence?

The 5-Minute Review Session

1. What is the derivative (*Ableitung*) of a program?
2. How is the *program transition* of an Esterel program defined?
3. How do program transitions express logical coherence?
4. Which semantics for Esterel exist?

The 5-Minute Review Session

1. What is the derivative (*Ableitung*) of a program?
2. How is the *program transition* of an Esterel program defined?
3. How do program transitions express logical coherence?
4. Which semantics for Esterel exist?
5. What are the *constructive coherence laws*, how do they differ from the logical coherence law?

Overview

Translating Esterel to Circuits

- Can consider Esterel programs as SW or HW descriptions
- As it turns out, the HW-equivalent of constructiveness is that the synthesized circuit is delay-independent
  - This gives a firm, physical base for the constructive semantics we just considered

Translating Esterel to Circuits

- Can consider Esterel programs as SW or HW descriptions
- As it turns out, the HW-equivalent of constructiveness is that the synthesized circuit is delay-independent
  - This gives a firm, physical base for the constructive semantics we just considered
- Can in turn simulate this synthesized HW-circuit in SW
  - This is just what the Esterel v5 compiler does
  - Can then also take advantage of HW optimization techniques
  - Use BDD-based techniques to check constructiveness

Circuit Semantics—Introduction

```
module P1:
input I;
output O;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit O end
end signal
end module
```

≡

Circuit Semantics—Introduction

```
module P1:
input I;
output O;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit O end
end signal
end module
```

$\equiv$

circuit C1:
$S1 = I$
$S2 = \neg S1$
$O = S2$

Circuit Semantics—Introduction

```
module P1:
input I;
output O;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit O end
end signal
end module
```

$\equiv$

circuit C1:
$S1 = I$
$S2 = \neg S1$
$O = S2$

- Resulting circuit is acyclic

Circuit Semantics—Introduction

```
module P1:
input I;
output O;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit O end
end signal
end module
```

$\equiv$

circuit C1:
$S1 = I$
$S2 = \neg S1$
$O = S2$

- Resulting circuit is acyclic
- Hence always stabilizes
- Reactive and deterministic

Circuit Semantics—Introduction

```
module P3:
output O;
present O else emit O end
end module
```

≡

Circuit Semantics—Introduction

```
module P3:
output O;
present O else emit O end
end module
```

$\equiv$

circuit C3:
$O = \neg O$

Circuit Semantics—Introduction

```
module P3:
output O;
present O else emit O end
end module
```

$\equiv$

circuit C3:
$O = \neg O$

- Resulting circuit never stabilizes
- Not reactive

Circuit Semantics—Introduction

```
module P4:
output O;
present O then emit O end
end module
```

≡

Circuit Semantics—Introduction

```
module P4:
output O;
present O then emit O end
end module
```

$\equiv$

circuit C4:
$O = O$

Circuit Semantics—Introduction

```
module P4:
output O;
present O then emit O end
end module
```

$\equiv$

circuit C4:
$O = O$

- Resulting circuit can stabilize at different values
- Not deterministic

Circuit Semantics—Introduction

```
module P9:
[
  present O1 then emit O1 end
||
  present O1 then
    present O2 else emit O2 end
  end
]
```

Circuit Semantics—Introduction

```
module P9:
[
  present O1 then emit O1 end
||
  present O1 then
    present O2 else emit O2 end
  end
]
```

$\equiv$

circuit C9:

$O1 = O1$

$O2 = O1 \wedge \neg O2$

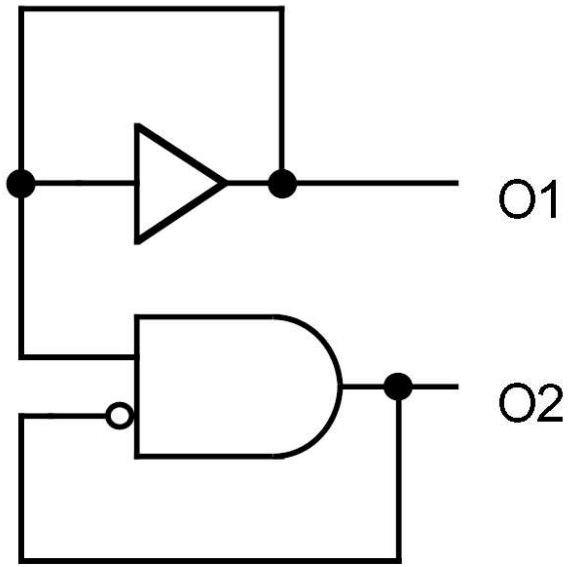
Circuit Semantics—Introduction

```
module P9:
[
  present O1 then emit O1 end
||
  present O1 then
    present O2 else emit O2 end
  end
]
```

≡

circuit C9:
$O1 = O1$
$O2 = O1 \wedge \neg O2$

≡



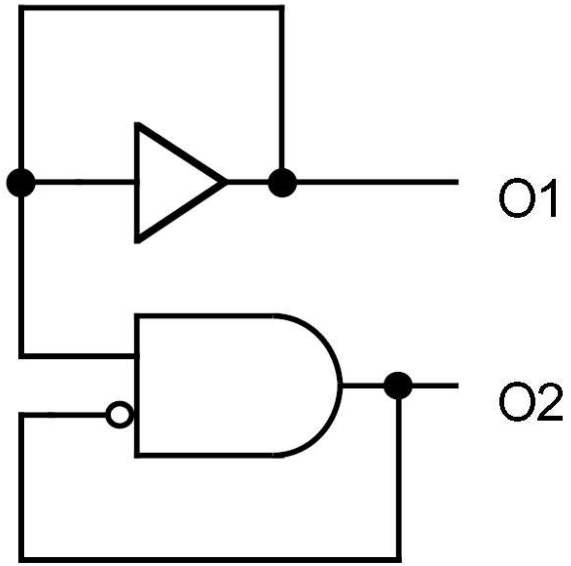
Circuit Semantics—Introduction

```
module P9:
[
  present O1 then emit O1 end
||
  present O1 then
    present O2 else emit O2 end
  end
]
```

≡

circuit C9:
$O1 = O1$
$O2 = O1 \wedge \neg O2$

≡



- Reactive and deterministic

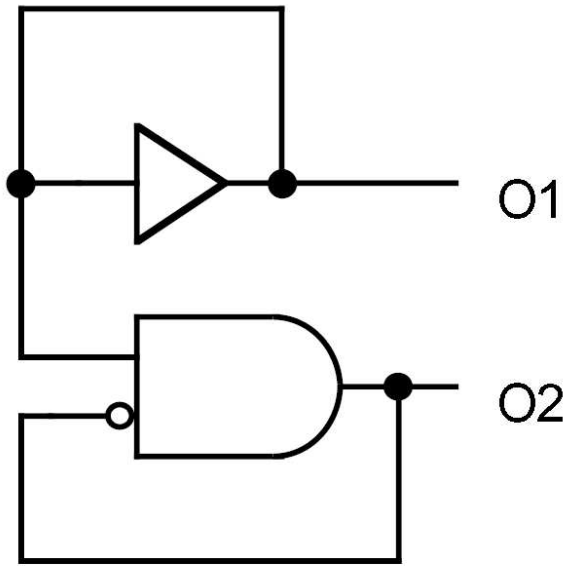
Circuit Semantics—Introduction

```
module P9:
[
  present O1 then emit O1 end
||
  present O1 then
    present O2 else emit O2 end
  end
]
```

≡

circuit C9:
$O1 = O1$
$O2 = O1 \wedge \neg O2$

≡



- Reactive and deterministic
- But not constructive!

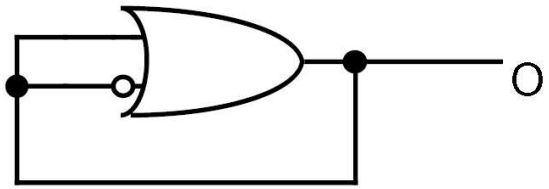
Circuit Semantics—Introduction

```
module P12:
present O then
  emit O;
else
  emit O
end
```

≡

circuit C12:
$O = O \vee \neg O$

≡



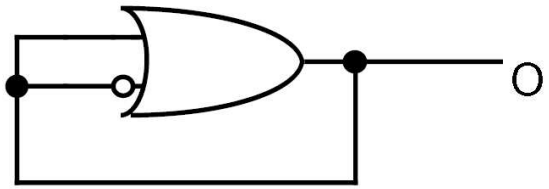
Circuit Semantics—Introduction

```
module P12:
present O then
  emit O;
else
  emit O
end
```

≡

circuit C12:
$O = O \vee \neg O$

≡



- Reactive and deterministic

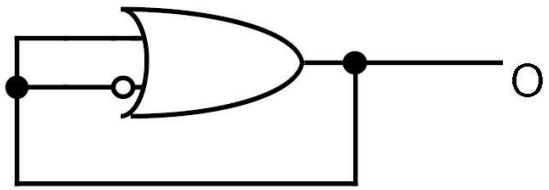
Circuit Semantics—Introduction

```
module P12:
present O then
  emit O;
else
  emit O
end
```

≡

circuit C12:
$O = O \vee \neg O$

≡



- Reactive and deterministic
- Meaning: If it stabilizes, there is only one possible value for each wire's voltage

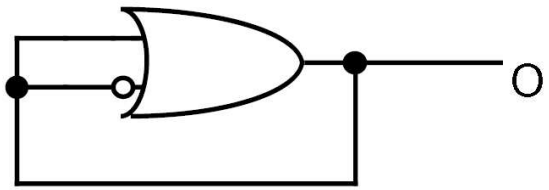
Circuit Semantics—Introduction

```
module P12:
present O then
  emit O;
else
  emit O
end
```

≡

circuit C12:
$O = O \vee \neg O$

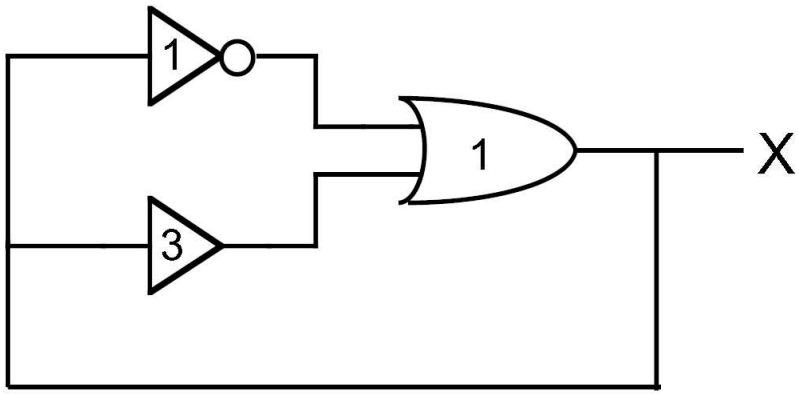
≡



- Reactive and deterministic
- Meaning: If it stabilizes, there is only one possible value for each wire's voltage
- But: Does it always stabilize?

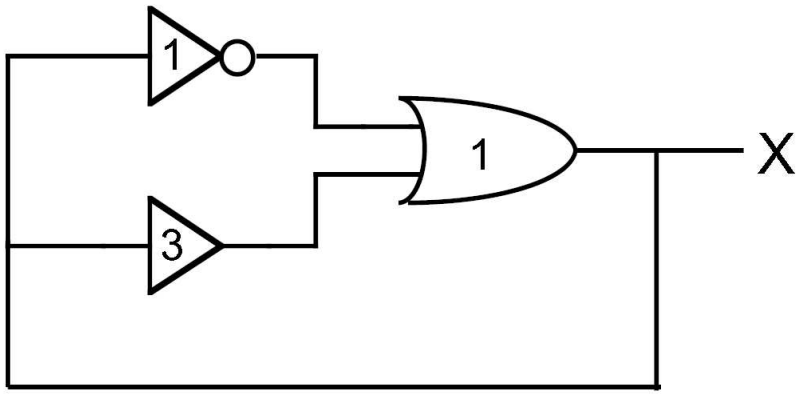
Circuit Semantics—Introduction

- Consider following delay assignment:



Circuit Semantics—Introduction

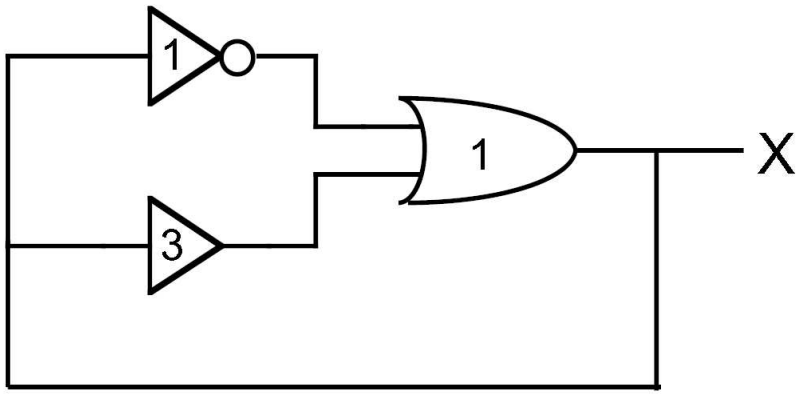
- Consider following delay assignment:



- Circuit is reactive and deterministic (Newtonian model)

Circuit Semantics—Introduction

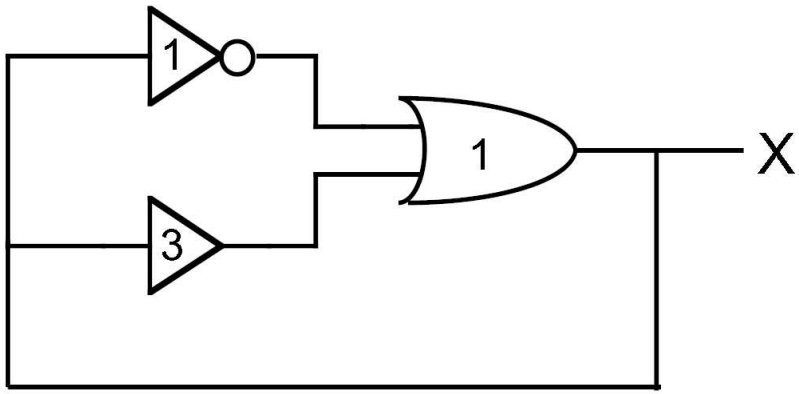
- Consider following delay assignment:



- Circuit is reactive and deterministic (Newtonian model)
- But: Circuit never stabilizes (Vibration model)

Circuit Semantics—Introduction

- Consider following delay assignment:



- Circuit is reactive and deterministic (Newtonian model)
- But: Circuit never stabilizes (Vibration model)
- **Hence: Electrical stabilization is not the conjunction of reactivity and determinism!**

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

≡

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

$\equiv$

circuit C13:

$O1 = I \wedge O2$

$O2 = \neg I \wedge O1$

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

$\equiv$

circuit C13:

$O1 = I \wedge O2$

$O2 = \neg I \wedge O1$

- Reactive and deterministic

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

$\equiv$

circuit C13:

$O1 = I \wedge O2$

$O2 = \neg I \wedge O1$

- Reactive and deterministic
- Cyclic, yet always stabilizes

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

$\equiv$

circuit C13:

$O1 = I \wedge O2$

$O2 = \neg I \wedge O1$

- Reactive and deterministic
- Cyclic, yet always stabilizes
- Hence: Electrical stabilization does not require acyclicity

Circuit Semantics—Introduction

```
module P13:
present I then
  present O2 then emit O1 end
else
  present O1 then emit O2 end
end
```

$\equiv$

circuit C13:

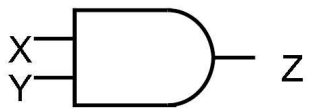
$O1 = I \wedge O2$

$O2 = \neg I \wedge O1$

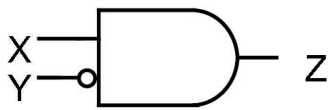
- Reactive and deterministic
- Cyclic, yet always stabilizes
- Hence: Electrical stabilization does not require acyclicity
- In fact: Electrical stabilization equivalent to constructiveness

Constructive Circuits

Basic building blocks



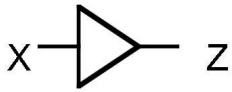
Z = X and Y



Z = X and not Y

Z = X or Y

Z = not (X or Y)



Z = X

Z = reg(X)

- Allow insertion of arbitrary delays
- Registers:
  - $reg(X) = 0 \rightarrow pre(X)$

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not $0 \rightarrow 1$
  - not $1 \rightarrow 0$

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not $0 \rightarrow 1$
  - not $1 \rightarrow 0$
  - 1 or $x \rightarrow 1$
  - x or $1 \rightarrow 1$
  - 0 or $0 \rightarrow 0$

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not $0 \rightarrow 1$
  - not $1 \rightarrow 0$
  - 1 or $x \rightarrow 1$
  - x or $1 \rightarrow 1$
  - 0 or $0 \rightarrow 0$
  - 0 and $x \rightarrow 0$
  - x and $0 \rightarrow 0$
  - 1 and $1 \rightarrow 1$

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 $\rightarrow$ 1
  - not 1 $\rightarrow$ 0
  - 1 or x $\rightarrow$ 1
  - x or 1 $\rightarrow$ 1
  - 0 or 0 $\rightarrow$ 0
  - 0 and x $\rightarrow$ 0
  - x and 0 $\rightarrow$ 0
  - 1 and 1 $\rightarrow$ 1
- There is no law of excluded middle (x or not x $\rightarrow$ 1)!

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 $\rightarrow$ 1
  - not 1 $\rightarrow$ 0
  - 1 or x $\rightarrow$ 1
  - x or 1 $\rightarrow$ 1
  - 0 or 0 $\rightarrow$ 0
  - 0 and x $\rightarrow$ 0
  - x and 0 $\rightarrow$ 0
  - 1 and 1 $\rightarrow$ 1
- There is no law of excluded middle (x or not x $\rightarrow$ 1)!
- Circuit equations yield solution iff circuit is delay insensitive (i.e., the original Esterel program is constructive)

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 $\rightarrow$ 1
  - not 1 $\rightarrow$ 0
  - 1 or x $\rightarrow$ 1
  - x or 1 $\rightarrow$ 1
  - 0 or 0 $\rightarrow$ 0
  - 0 and x $\rightarrow$ 0
  - x and 0 $\rightarrow$ 0
  - 1 and 1 $\rightarrow$ 1
- There is no law of excluded middle (x or not x $\rightarrow$ 1)!
- Circuit equations yield solution iff circuit is delay insensitive (i.e., the original Esterel program is constructive)
  - Propagation of 1's corresponds to

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 $\rightarrow$ 1
  - not 1 $\rightarrow$ 0
  - 1 or x $\rightarrow$ 1
  - x or 1 $\rightarrow$ 1
  - 0 or 0 $\rightarrow$ 0
  - 0 and x $\rightarrow$ 0
  - x and 0 $\rightarrow$ 0
  - 1 and 1 $\rightarrow$ 1
- There is no law of excluded middle (x or not x $\rightarrow$ 1)!
- Circuit equations yield solution iff circuit is delay insensitive (i.e., the original Esterel program is constructive)
  - Propagation of 1's corresponds to *Must*-analysis

Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 $\rightarrow$ 1
  - not 1 $\rightarrow$ 0
  - 1 or x $\rightarrow$ 1
  - x or 1 $\rightarrow$ 1
  - 0 or 0 $\rightarrow$ 0
  - 0 and x $\rightarrow$ 0
  - x and 0 $\rightarrow$ 0
  - 1 and 1 $\rightarrow$ 1
- There is no law of excluded middle (x or not x $\rightarrow$ 1)!
- Circuit equations yield solution iff circuit is delay insensitive (i.e., the original Esterel program is constructive)
  - Propagation of 1's corresponds to *Must*-analysis
  - Propagation of 0s corresponds to *Cannot*-analysis

The Basic Circuit Translation

- Structural translation

The Basic Circuit Translation

- Structural translation
- Follows state semantics
  - Associate registers with "1" statements (`pause`)
  - Associate combinational logic with all other statements

The Basic Circuit Translation

- Structural translation
- Follows state semantics
  - Associate registers with "1" statements (`pause`)
  - Associate combinational logic with all other statements
  - Build up program-circuit from subcircuits

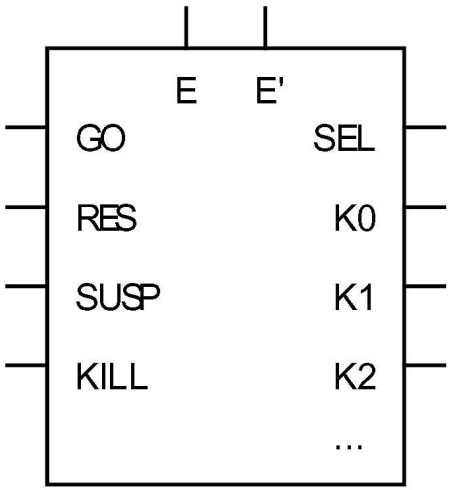
The Basic Circuit Translation

- Structural translation
- Follows state semantics
  - Associate registers with "1" statements (`pause`)
  - Associate combinational logic with all other statements
  - Build up program-circuit from subcircuits
  - Additional boot register to implement initial state

The Basic Circuit Translation

- Structural translation
- Follows state semantics
  - Associate registers with "1" statements (`pause`)
  - Associate combinational logic with all other statements
  - Build up program-circuit from subcircuits
  - Additional boot register to implement initial state
- Basic circuit translation does not address schizophrenia (see later)

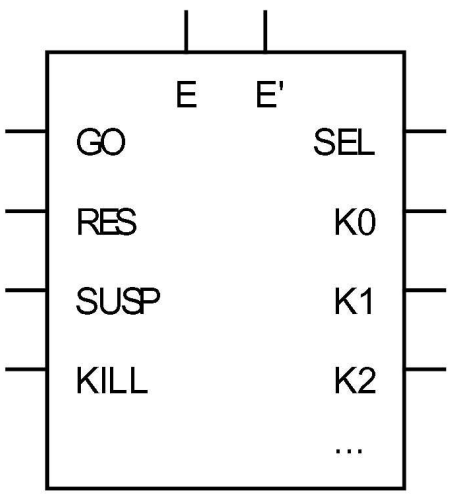
Interface for subcircuits



Inputs:

- GO: Starts statement afresh
- RES: Resumes execution of a selected statement
- SUSP: Suspend execution of the statement
  - Registers keep their current value unless killed because of the KILL input
- KILL: Unsets statement's registers in case of a trap exit

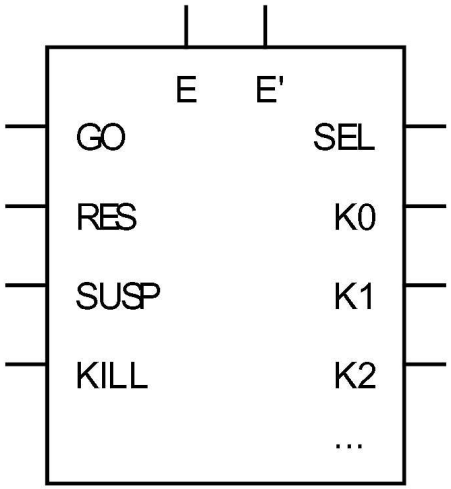
Interface for subcircuits contd.



Outputs:

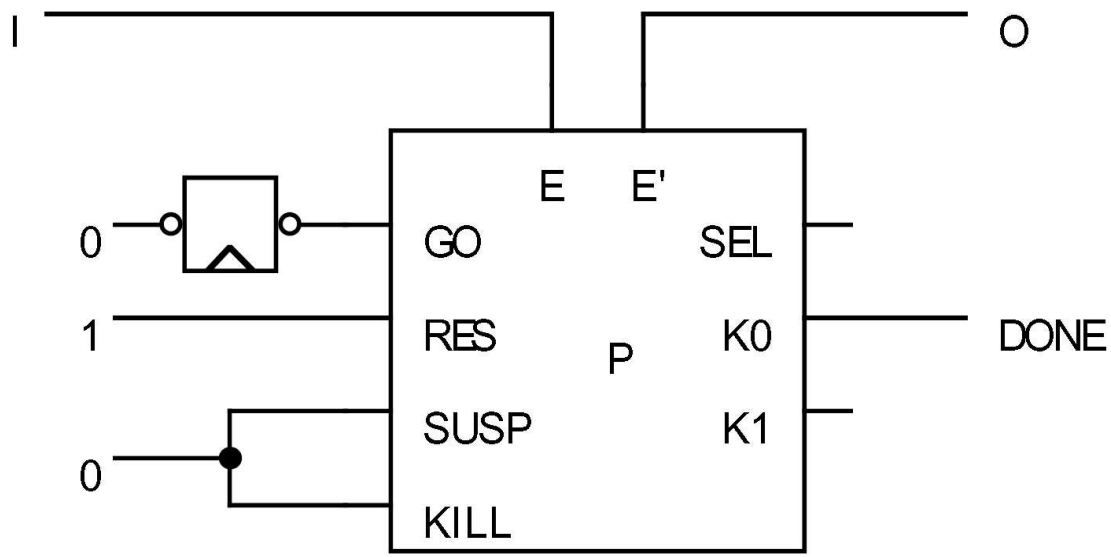
- SEL: Indicates that a state in statement is currently selected for resumption, i.e. that some internal pause register is set
  - Is simply the disjunction of the internal registers.
- K0, K1, ...: Completion codes (1-hot encoding)

Interface for subcircuits contd.

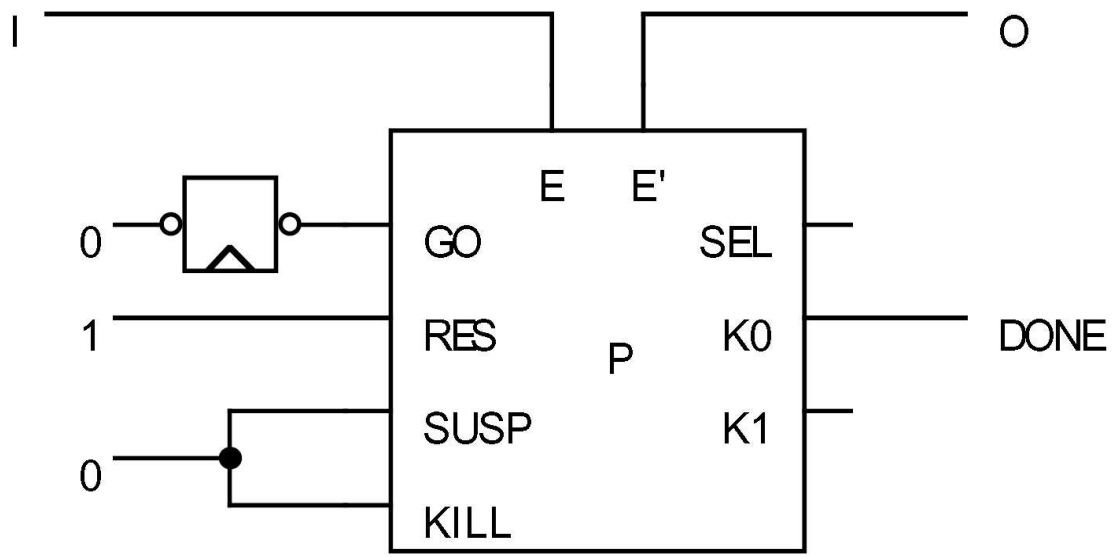


- E and E': input/output signal interface
- Are compound pins or buses
  - Contain one elementary pin per signal visible in the scope of the current statement.
- May freely extract specific signals s or s' out of E or E'.
- As for the K pins, the E' pins are explicitly unset when the statement is not executed
  - I.e. when $\neg($ GO $\vee$ (RES $\wedge$ SEL))

The Global Environment



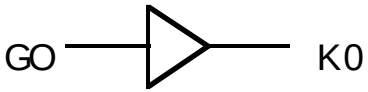
The Global Environment



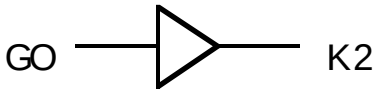
- Boot register sets GO input in initial instant
- At each clock cycle
  - set RES
  - clock the registers

Translating the Esterel Kernel

- Completion, with $k \neq 1$:



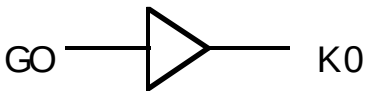
code 0



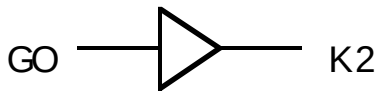
code 2

Translating the Esterel Kernel

- Completion, with $k \neq 1$:



code 0



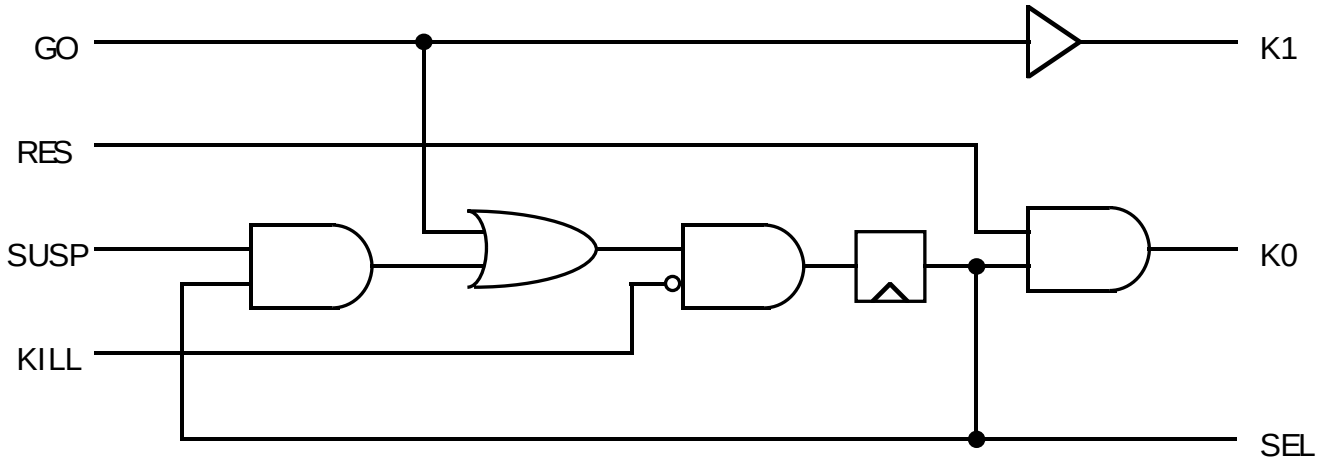
code 2

- $k = 1$ (pause):

Translating the Esterel Kernel

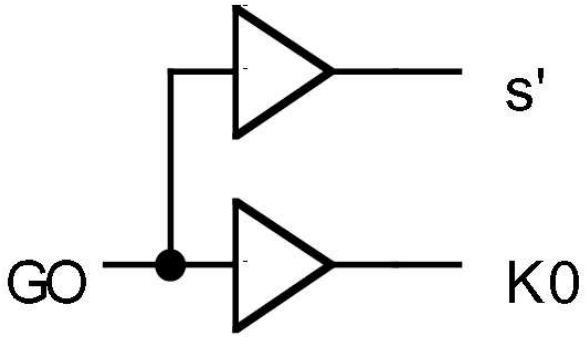
- Completion, with $k \neq 1$:

- $k = 1$ (pause):



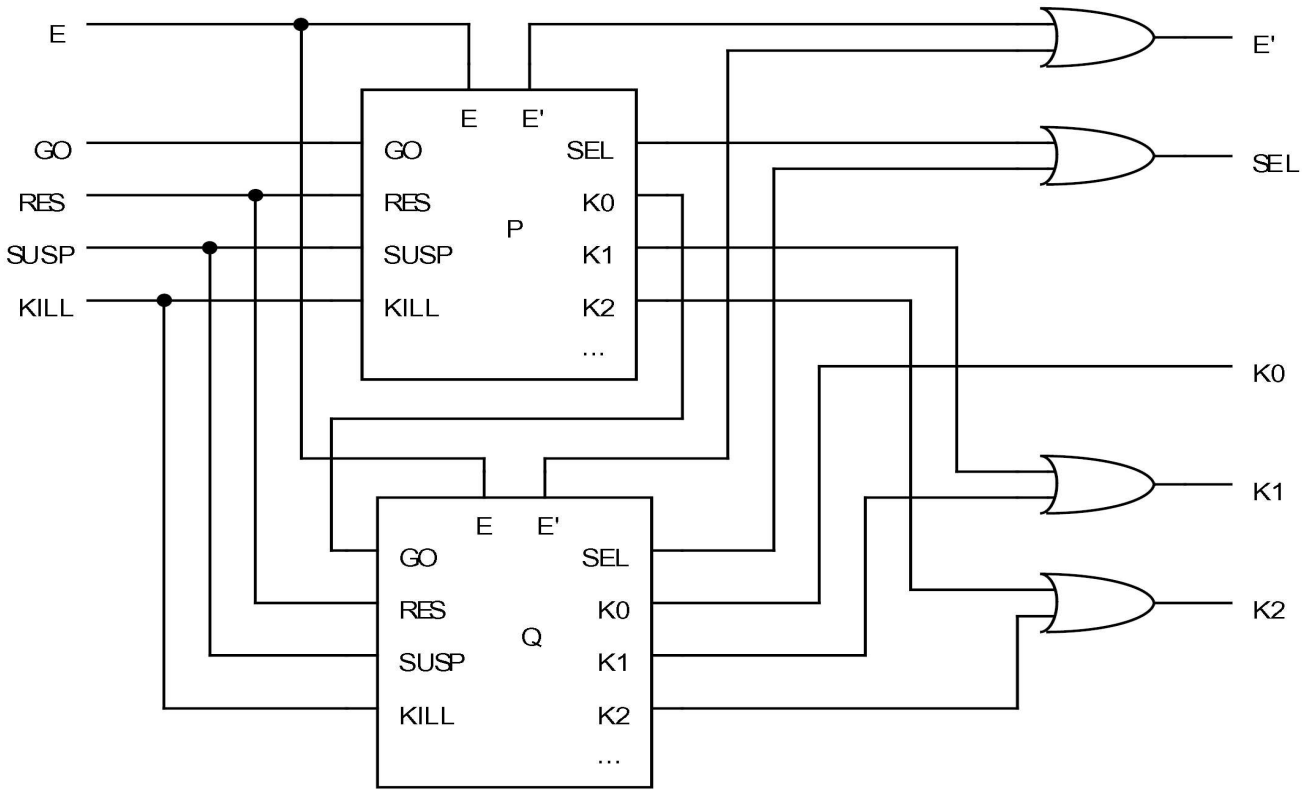
Translating the Esterel Kernel

- $!s$:



Translating the Esterel Kernel

- $p; q$:

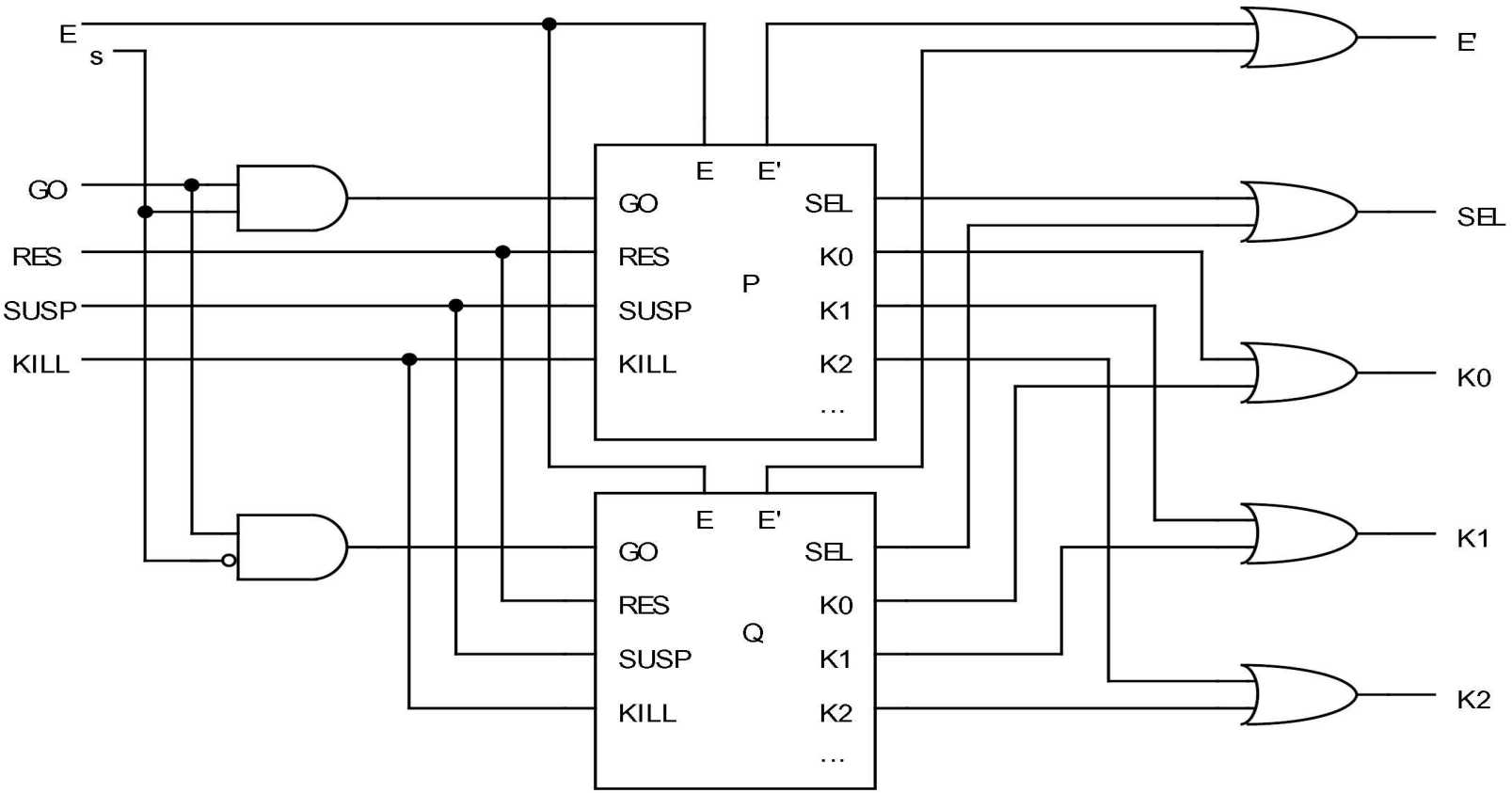


Translating the Esterel Kernel

- $s?p, q$:

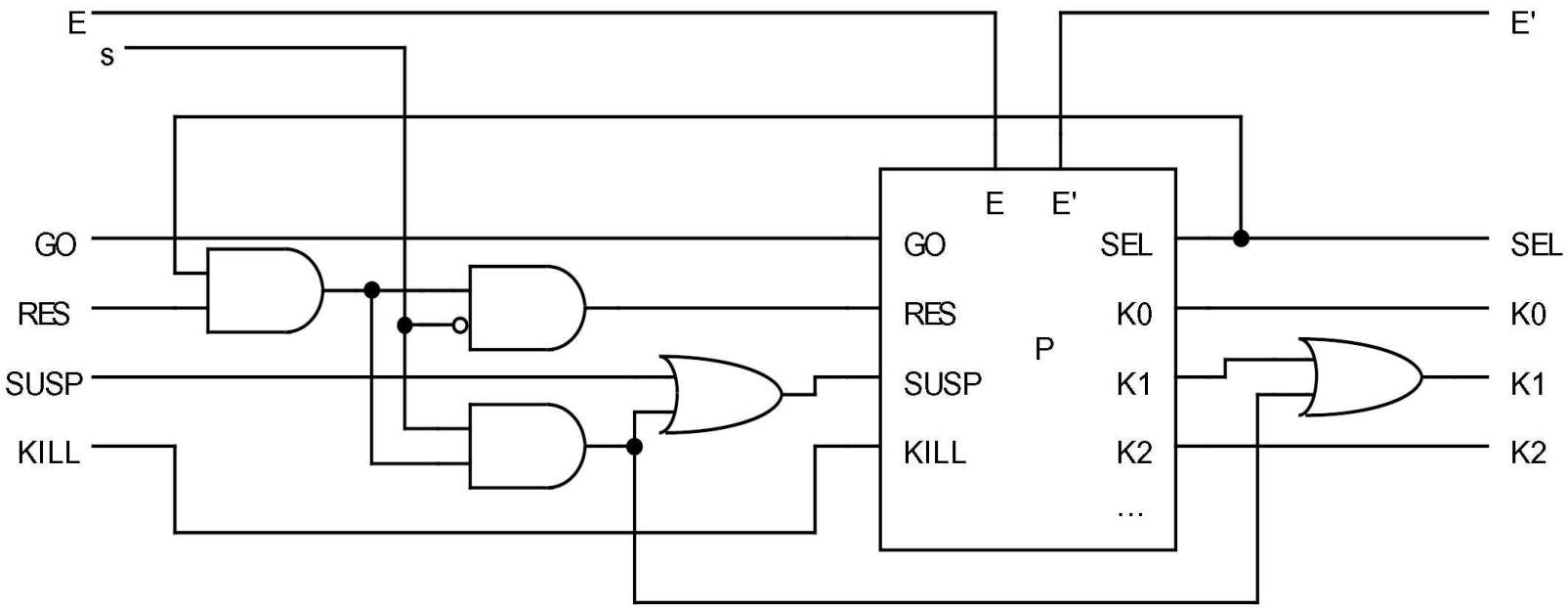
Translating the Esterel Kernel

- $s?p, q$:



Translating the Esterel Kernel

- $s \supset p$:

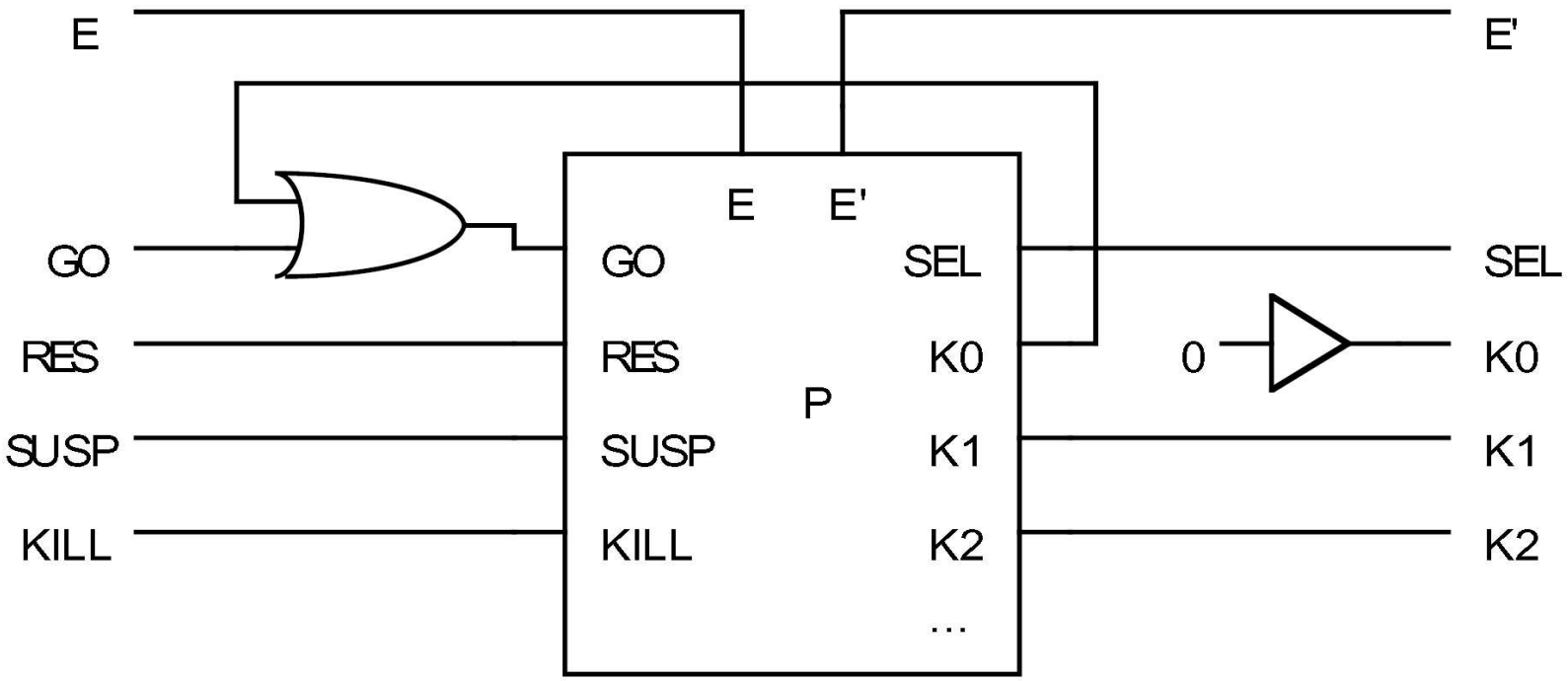


Translating the Esterel Kernel

- $p*$:

Translating the Esterel Kernel

- $p*$:

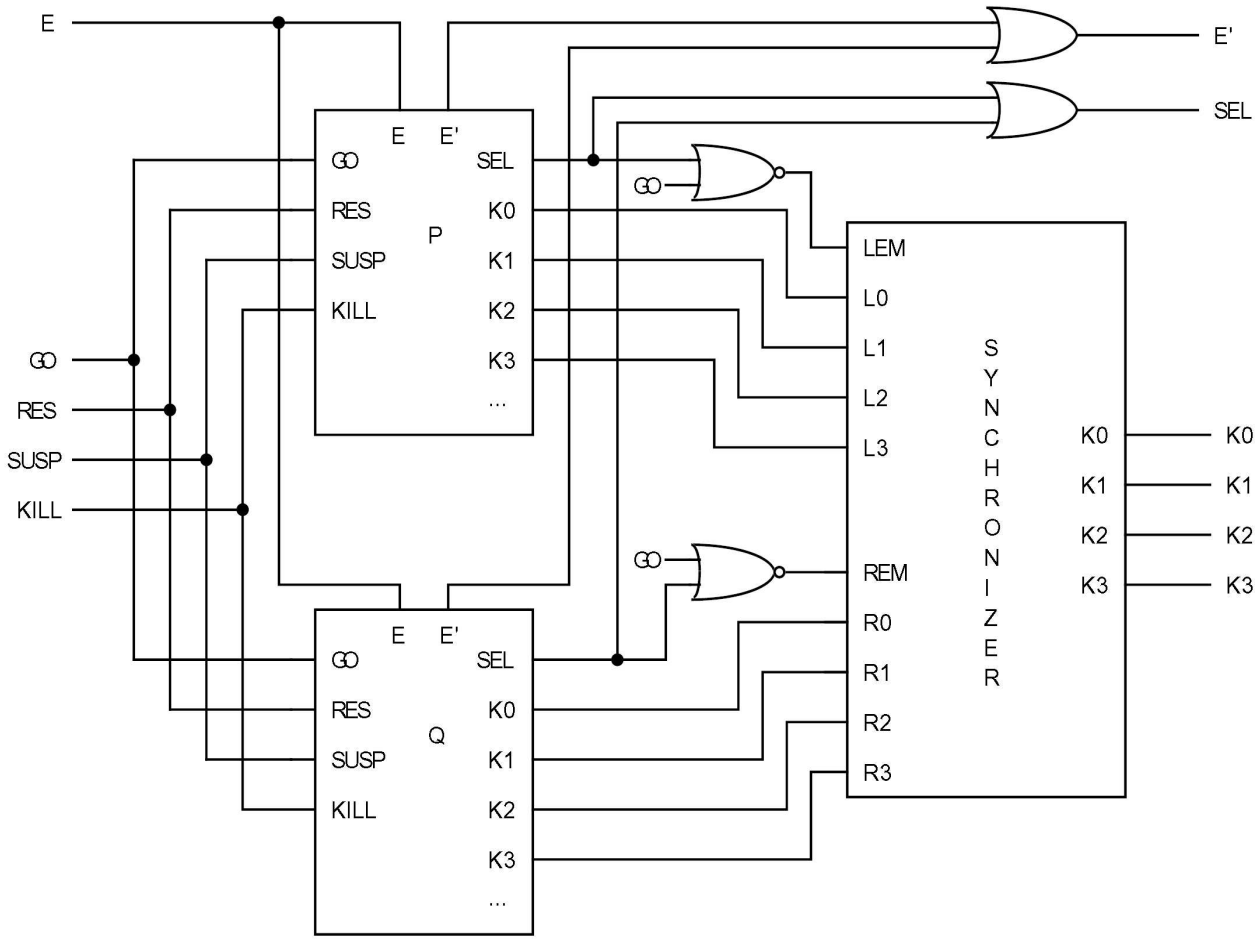


Translating the Esterel Kernel

- $p \parallel q$:

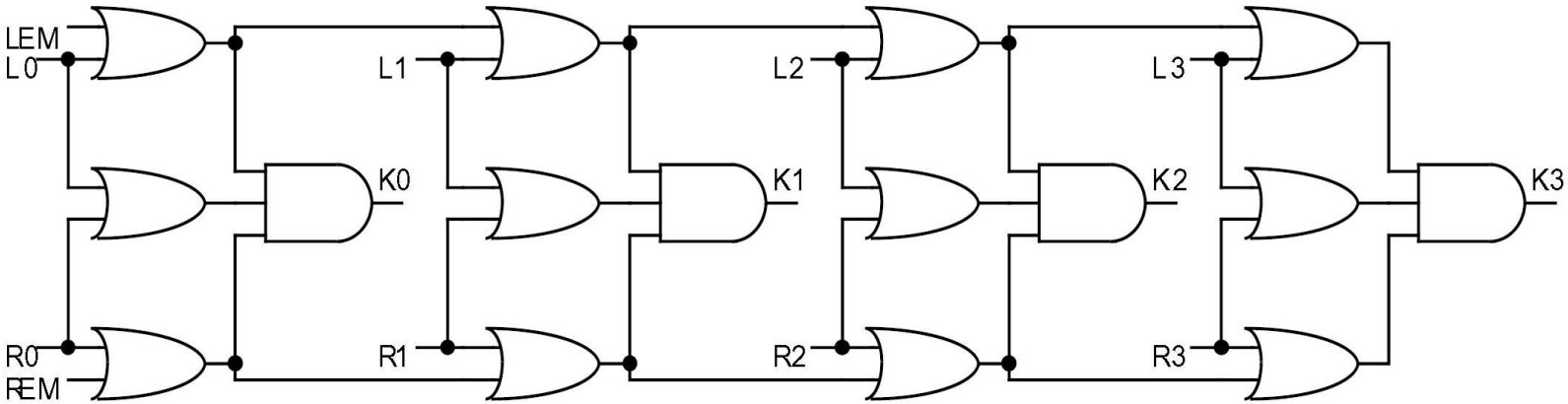
Translating the Esterel Kernel

- $p \parallel q$:



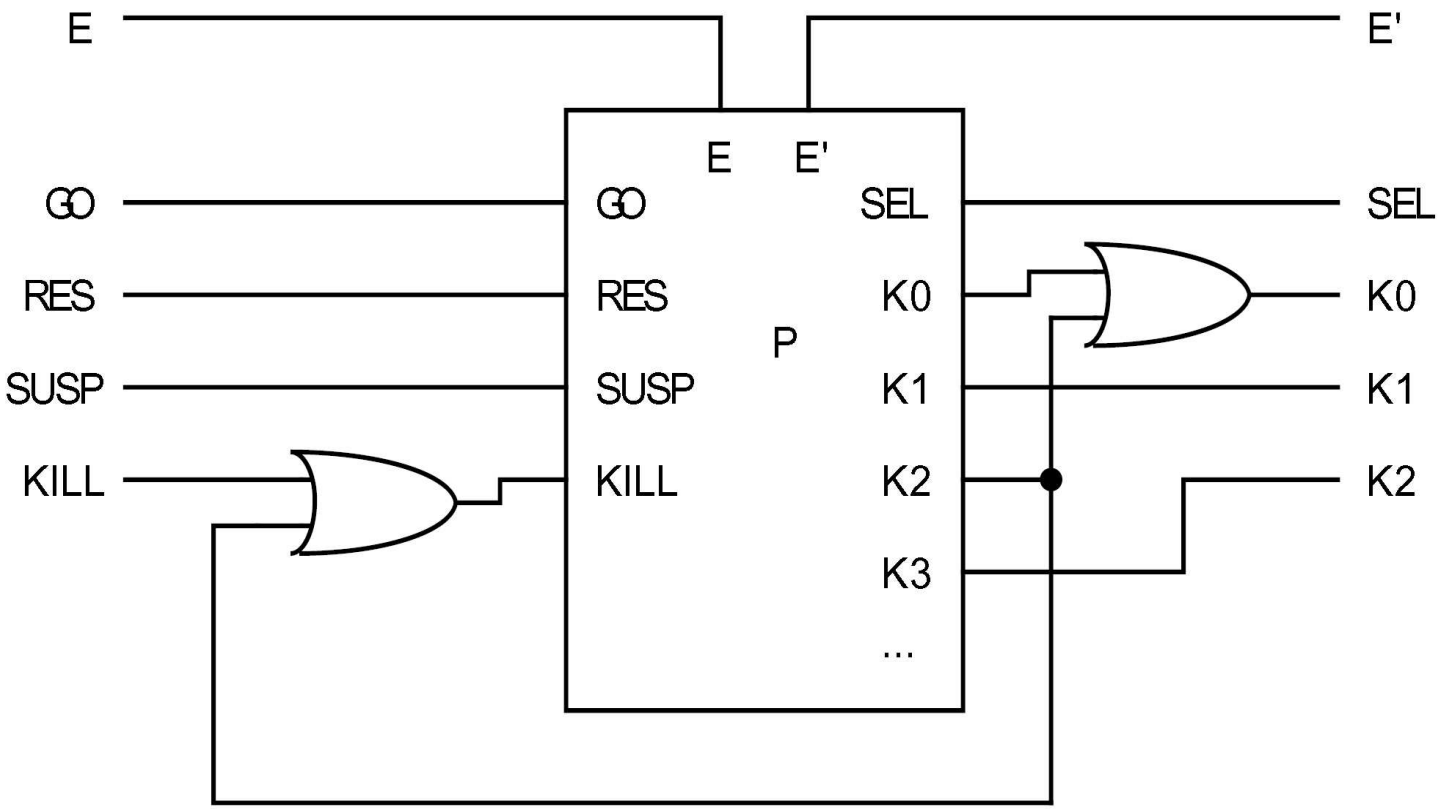
Translating the Esterel Kernel

- $p \parallel q$ (contd):
  - The synchronizer computes the maximum of the completion codes
  - Implemented with this (constructive) circuit:



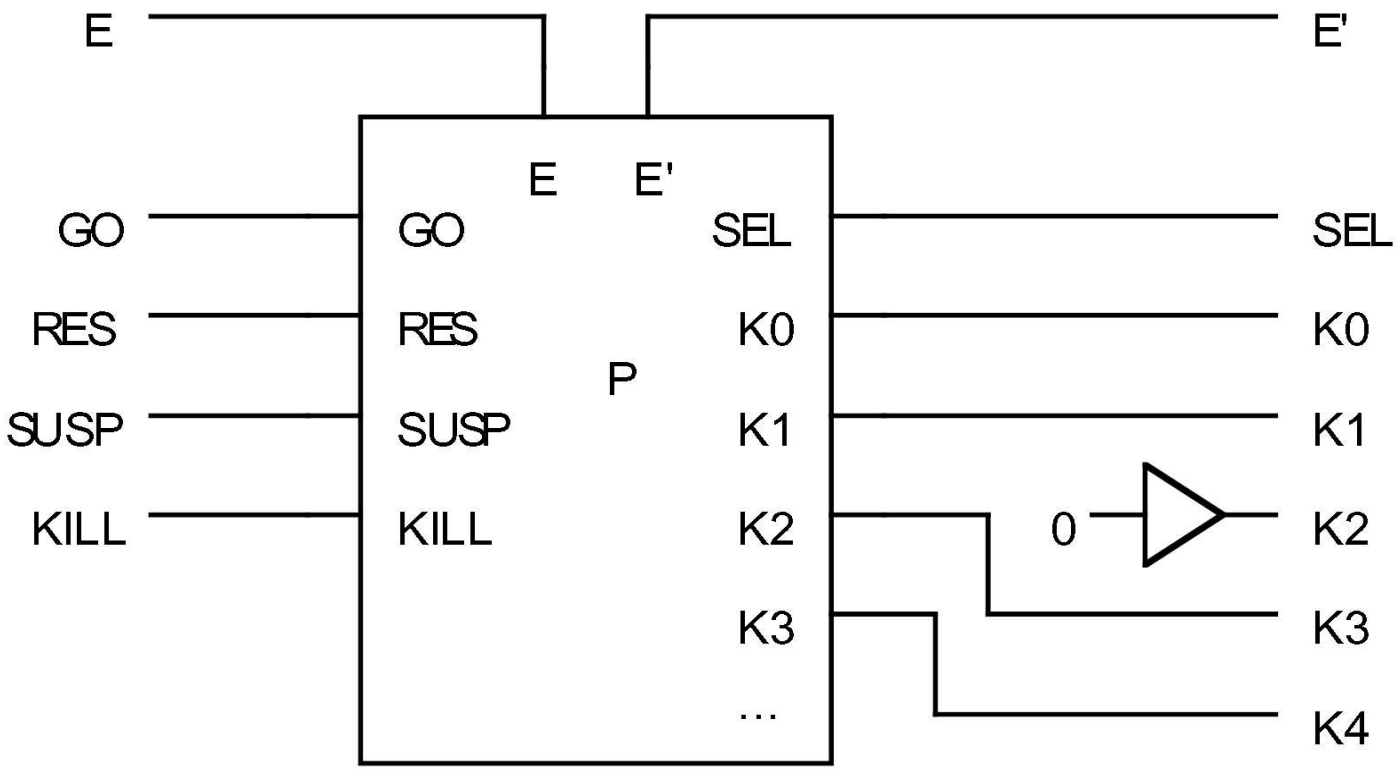
Translating the Esterel Kernel

- $\{p\}$:



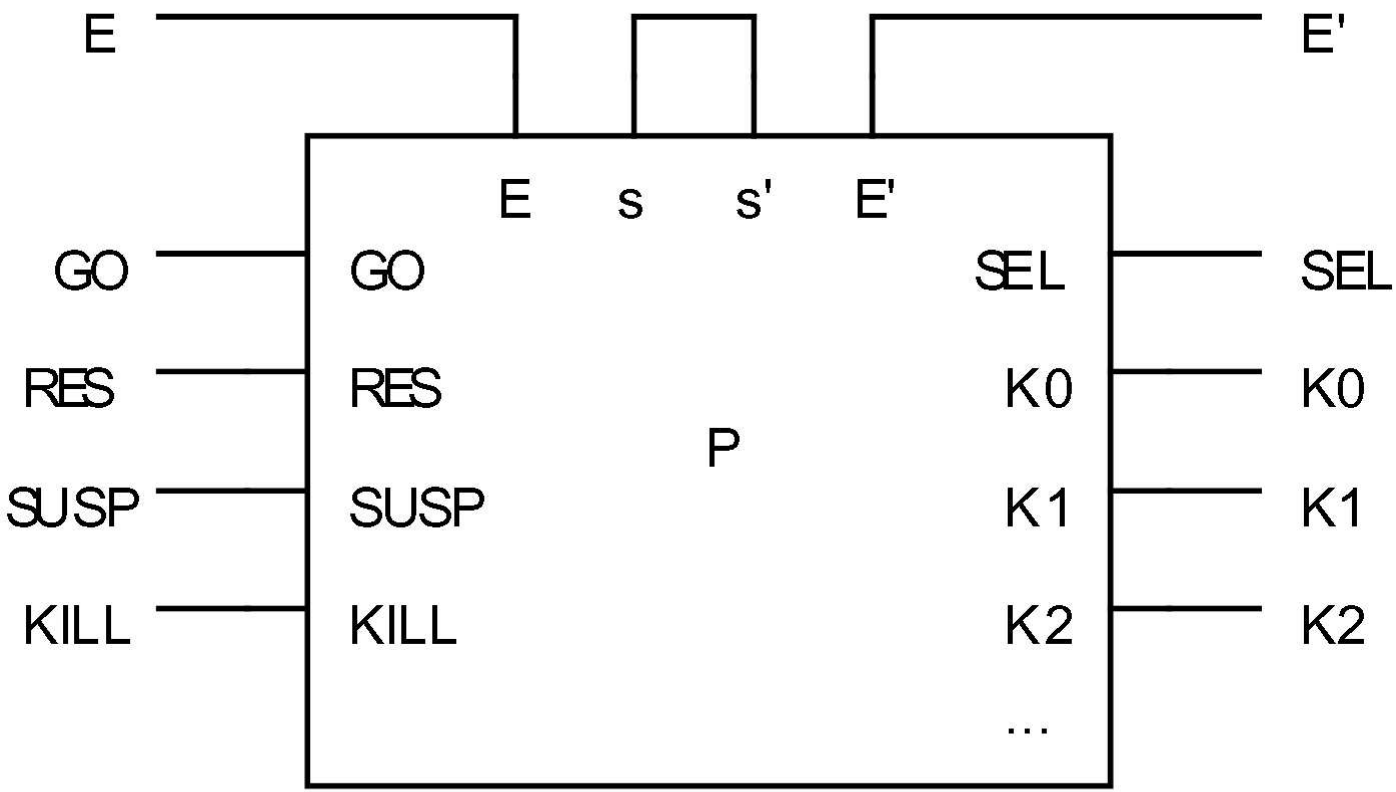
Translating the Esterel Kernel

- $\uparrow p$:



Translating the Esterel Kernel

- $p \backslash s$:



Example

```
module P2:
signal S in
  emit S;
  present O then
    present S then
      pause
    end present;
    emit O
  end present
end signal
```

Example

```
module P2:
signal S in
  emit S;
  present O then
    present S then
      pause
    end present;
    emit O
  end present
end signal
```

≡

circuit C2:

$B = \neg\, \mathrm{REG}(1)$ // *Boot*

$S = B$

$R = \mathrm{REG}(B \wedge O \wedge S)$ // *pause*

$O = (B \wedge O \wedge \neg S) \vee R$

$K0 = (B \wedge \neg O) \vee (B \wedge O \wedge \neg S) \vee R$

$K1 = B \wedge O \wedge S$

Example

```
module P2:
signal S in
  emit S;
  present O then
    present S then
      pause
    end present;
    emit O
  end present
end signal
```

≡

≡

circuit C2:

$B = \neg\ \mathrm{REG}(1)$ // *Boot*

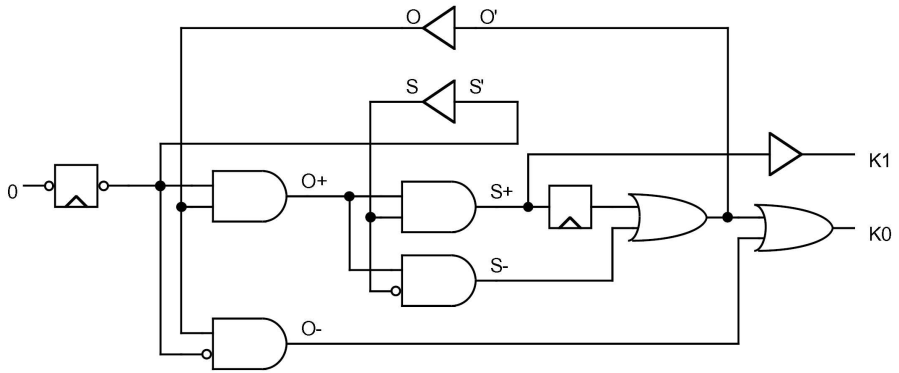
$S = B$

$R = \mathrm{REG}(B \wedge O \wedge S)$ // *pause*

$O = (B \wedge O \wedge \neg S) \vee R$

$K0 = (B \wedge \neg O) \vee (B \wedge O \wedge \neg S) \vee R$

$K1 = B \wedge O \wedge S$



To Go Further

- Gérard Berry, The Constructive Semantics of Pure Esterel, Draft book, current version 3.0, Dec. 2002, Chapters 10 and 11,
  `http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.zip`