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
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?
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

The Circuit Semantics
Constructive circuits
The basic circuit translation
Translating the Esterel kernel
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 0 end
end signal
end module

≡

\[
\begin{align*}
S1 &= I \\
S2 &= \neg S1 \\
O &= S2
\end{align*}
\]

Resulting circuit is acyclic

Hence always stabilizes

Reactive and deterministic
The Circuit Semantics

Constructive circuits

The basic circuit translation

Translating the Esterel kernel

Circuit Semantics—Introduction

module P1:
input I;
output 0;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit 0 end
end signal
end module

circuit C1:
S1 = I
S2 = \neg S1
O = S2
The Circuit Semantics

Constructive circuits

The basic circuit translation

Translating the Esterel kernel

Circuit Semantics—Introduction

module P1:
input I;
output 0;
signal S1, S2 in
  present I then emit S1 end
||
  present S1 else emit S2 end
||
  present S2 then emit 0 end
end signal
end module

≡
circuit C1:
S1 = I
S2 = ¬S1
O = S2

▶ Resulting circuit is acyclic
Circuit Semantics—Introduction

module P1:
    input I;
    output 0;
    signal S1, S2 in
        present I then emit S1 end
        ||
        present S1 else emit S2 end
        ||
        present S2 then emit 0 end
    end signal
end module

≡
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 0;
present 0 else emit 0 end
end module

≡

\[ O = \neg O \Rightarrow \text{Resulting circuit never stabilizes} \Rightarrow \text{Not reactive} \]
Circuit Semantics—Introduction

module P3:
output O;
present 0 else emit 0 end
end module

≡

circuit C3:
O \equiv \neg O
Circuit Semantics—Introduction

module P3:
  output O;
  present 0 else emit 0 end
end module

≡

circuit C3:
  O ≡ ¬O

- Resulting circuit never stabilizes
- Not reactive
Circuit Semantics—Introduction

module P4:
output 0;
present 0 then emit 0 end
end module

\[ \text{Circuit } C_4: \]
\[ O = O \]

Resulting circuit can stabilize at different values

Not deterministic
Circuit Semantics—Introduction

module P4:
output 0;
present 0 then emit 0 end
end module

≡

circuit C4:
O = O
Circuit Semantics—Introduction

module P4:
output 0;
present 0 then emit 0 end
end module

≡
circuit C4:
O = O

- Resulting circuit can stabilize at different values
- Not deterministic
Circuit Semantics—Introduction

module P9:
[
    present 01 then emit 01 end
||
    present 01 then
        present 02 else emit 02 end
    end
]
Circuit Semantics—Introduction

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

≡

\[
\begin{align*}
\text{circuit C9:} \\
O_1 &= O_1 \\
O_2 &= O_1 \land \neg O_2
\end{align*}
\]
Circuit Semantics—Introduction

module P9:
[
    present 01 then emit 01 end

    present 01 then
    present 02 else emit 02 end
end
≡
circuit C9:
O1 = O1
O2 = O1 ∧ ¬O2
Circuit Semantics—Introduction

```plaintext
module P9:
[
    present 01 then emit 01 end
    ||
    present 01 then
    present 02 else emit 02 end
end
```

≡

circuit C9:

\[
\begin{align*}
O1 &= O1 \\
O2 &= O1 \land \neg O2
\end{align*}
\]

▶ Reactive and deterministic
The Circuit Semantics

Constructive circuits
The basic circuit translation
Translating the Esterel kernel

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 ∧ ¬O2

- Reactive and deterministic
- But not constructive!
Circuit Semantics—Introduction

module P12:
  present 0 then
    emit 0;
  else
    emit 0
  end

≡

\[
\text{circuit } C12: \\
O = O \lor \neg O
\]

≡

\[
\begin{array}{c}
\text{O} \\
\end{array}
\]
Circuit Semantics—Introduction

module P12:
  present 0 then
    emit 0;
  else
    emit 0
  end

⇒

≡

circuit C12:
0 = 0 ∨ ¬0

⇒

Reactive and deterministic
Circuit Semantics—Introduction

```
module P12:
  present 0 then
    emit 0;
  else
    emit 0
end
```

≡

circuit C12:

\[ O = O \lor \neg O \]

≡

- Reactive and deterministic
- **Meaning**: If it stabilizes, there is only one possible value for each wire’s voltage
Circuit Semantics—Introduction

\[
\text{module } P12: \\
present 0 \text{ then} \\
\hspace{1em} \text{emit } 0; \\
\hspace{1em} \text{else} \\
\hspace{2em} \text{emit } 0 \\
\text{end} \\
\equiv \\
\text{circuit } C12: \\
O = O \lor \neg O
\]

- 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 Diagram]

- 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

- Consider following delay assignment:

- Circuit is reactive and deterministic (Newtonian model)
Circuit Semantics—Introduction

- Consider following delay assignment:

![Circuit diagram]

- Circuit is reactive and deterministic (Newtonian model)
- **But**: Circuit never stabilizes (Vibration model)
Circuit Semantics—Introduction

- Consider following delay assignment:

![Circuit Diagram]

- 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 02 then emit 01 end
  else
    present 01 then emit 02 end
  end

≡

circuit C13:
  O1 = I \land O2
  O2 = \neg I \land O1

▶ Reactive and deterministic
▶ Cyclic, yet always stabilizes
▶ Hence: Electrical stabilization does not require acyclicity
▶ In fact: Electrical stabilization equivalent to constructiveness
Circuit Semantics—Introduction

module P13:
  present I then
    present 02 then emit 01 end
  else
    present 01 then emit 02 end
  end
Circuit Semantics—Introduction

module P13:
  present I then
    present 02 then emit 01 end
  else
    present 01 then emit 02 end
  end

≡

circuit C13:
  O1 = I ∧ O2
  O2 = ¬I ∧ O1
Circuit Semantics—Introduction

module P13:
  present I then
    present 02 then emit 01 end
  else
    present 01 then emit 02 end
  end

≡

circuit C13:
  O1 = I ∧ O2
  O2 = ¬I ∧ O1

► Reactive and deterministic
Circuit Semantics—Introduction

module P13:
present I then
  present 02 then emit 01 end
else
  present 01 then emit 02 end
end

≡
circuit C13:
O1 = I ∧ O2
O2 = ¬I ∧ O1

- Reactive and deterministic
- Cyclic, yet always stabilizes
Circuit Semantics—Introduction

module P13:
  present I then
    present 02 then emit 01 end
  else
    present 01 then emit 02 end
end

≡
circuit C13:
  O1 = I \land O2
  O2 = \neg I \land O1

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

module P13:
present I then
  present 02 then emit 01 end
else
  present 01 then emit 02 end
end

≡

circuit C13:
\[
O_1 = I \land O_2 \\
O_2 = \neg I \land O_1
\]

▸ 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

- Allow insertion of arbitrary delays
- Registers:
  - $\text{reg}(X) = 0 \rightarrow \text{pre}(X)$
Constructive Circuits

Constructive Boolean (intuitionistic) logic:
- Evaluate equations with constant folding rules

There is no law of excluded middle (x or not x → 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 0's corresponds to Cannot-analysis
Constructive Circuits

Constructive Boolean (intuitionistic) logic:

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

- 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
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
  - $\text{not } 0 \rightarrow 1$
  - $\text{not } 1 \rightarrow 0$
  - $1 \text{ or } x \rightarrow 1$
  - $x \text{ or } 1 \rightarrow 1$
  - $0 \text{ or } 0 \rightarrow 0$
  - $0 \text{ and } x \rightarrow 0$
  - $x \text{ and } 0 \rightarrow 0$
  - $1 \text{ and } 1 \rightarrow 1$

- There is no law of excluded middle ($x \text{ or } \text{not } x \rightarrow 1$)!
- 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 0's corresponds to Cannot-analysis
Constructive Circuits

Constructive Boolean (intuitionistic) logic:

- Evaluate equations with constant folding rules
  - not 0 → 1
  - not 1 → 0
  - 1 or x → 1
  - x or 1 → 1
  - 0 or 0 → 0
  - 0 and x → 0
  - x and 0 → 0
  - 1 and 1 → 1

- There is no law of excluded middle (x or not x → 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 → 1
  - not 1 → 0
  - 1 or x → 1
  - x or 1 → 1
  - 0 or 0 → 0
  - 0 and x → 0
  - x and 0 → 0
  - 1 and 1 → 1

- There is no law of excluded middle (x or not x → 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 → 1
  - not 1 → 0
  - 1 or x → 1
  - x or 1 → 1
  - 0 or 0 → 0
  - 0 and x → 0
  - x and 0 → 0
  - 1 and 1 → 1

- There is no law of excluded middle (x or not x → 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

Basic circuit translation does not address schizophrenia (see later)
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 \( \neg( \text{GO} \lor (\text{RES} \land \text{SEL})) \)
The Global Environment

![Diagram of a circuit with inputs I and O, registers GO and RES, and outputs DONE, K0, K1, and KILL. The diagram shows the logic for selecting between GO and RES, and the conditions for setting DONE to 1.]
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 $k \neq 1$:

  ![Circuit Diagram](image)

  - GO → K0 (code 0)
  - GO → K2 (code 2)
Translating the Esterel Kernel

- **Completion, with** $k \neq 1$:
  
  ![Diagram of circuit for completion](image)

- $k = 1$ (pause):
  
  ![Diagram of circuit for pause](image)
Translating the Esterel Kernel

- Completion, with $k \neq 1$:

- $k = 1$ (pause):
Translating the Esterel Kernel

\[ \neg s : \]

\[ \text{GO} \quad s' \quad \text{K0} \]
Translating the Esterel Kernel

\[ p; q: \]
Translating the Esterel Kernel

▶ \( s?p, q: \)
Translating the Esterel Kernel

- $s \? p, q$: 

![Circuit Diagram]
Translating the Esterel Kernel

$\Rightarrow s \supset p$: 
Translating the Esterel Kernel

$p*$:
Translating the Esterel Kernel

\( p^* : \)

The Circuit Semantics
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

\[\{p\} : \]

![Circuit Diagram]
Translating the Esterel Kernel

\[ \uparrow p: \]

\[ E \rightarrow \text{GO} \rightarrow \text{RES} \rightarrow \text{SUSP} \rightarrow \text{KILL} \rightarrow E \]

\[ E' \rightarrow \text{GO} \rightarrow \text{RES} \rightarrow \text{SUSP} \rightarrow \text{KILL} \rightarrow E' \]

\[ \text{SEL} \rightarrow K0 \rightarrow K1 \rightarrow K2 \rightarrow K3 \rightarrow K4 \]

\[ \text{0} \]
Translating the Esterel Kernel

▶ p\(s:

\[
\begin{array}{cccc}
E & \quad & s & \quad & E' \\
GO & \quad & GO & \quad & SEL \\
RES & \quad & RES & \quad & K0 \\
SUSP & \quad & SUSP & \quad & K1 \\
KILL & \quad & KILL & \quad & K2 \\
& & & \quad & \ldots \\
\end{array}
\]
Example

```
module P2:
signal S in
    emit S;
    present 0 then
        present S then
            pause
        end present;
    emit 0
end present
end signal
```
Example

module P2:
  signal S in
    emit S;
    present 0 then
      present S then
        pause
      end present;
    emit 0
  end present
end signal

circuit C2:
B = \neg \text{REG}(1) \quad // \text{Boot}
S = B
R = \text{REG}(B \land O \land S) \quad // \text{pause}
O = (B \land O \land \neg S) \lor R
K0 = (B \land \neg O) \lor (B \land O \land \neg S) \lor R
K1 = B \land O \land S
Example

module P2:
signal S in
  emit S;
  present 0 then
    present S then
      pause
    end present;
  emit 0
end present
end signal

circuit C2:
B = ¬ REG(1)     // Boot
S = B
R = REG(B ∧ O ∧ S)     // pause
O = (B ∧ O ∧ ¬S) ∨ R
K0 = (B ∧ ¬O) ∨ (B ∧ O ∧ ¬S) ∨ R
K1 = B ∧ O ∧ S
To Go Further