#### Design of Embedded Systems: Models, Validation and Synthesis (EE 249)—Lecture 9

Prof. Dr. Reinhard von Hanxleden

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

2 October 2007 Last compiled: 4th October 2007, 15:09 hrs



Synchronous Languages: Esterel

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

## Overview

Introduction to Safe State Machines and Esterel

Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

Esterel Language Overview

Esterel/SSM Pragmatics

Interfacing with the Environment

Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Introduction to Esterel

- Imperative, textual, concurrent language
- Developed since early 1980s (Gérard Berry)
- Based on synchronous model of time
  - Program execution synchronized to an external clock
  - Like synchronous digital logic
  - Suits the cyclic executive approach
- Same model of computation as SyncCharts/Safe State Machines (SSMs)
- EsterelStudio generates Esterel from SSMs as intermediate code
- Currently undergoing IEEE standardization (Esterel v7)

Thanks to Stephen Edwards (http://www1.cs.columbia.edu/~sedwards/) for providing part of the following material

Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

## Introduction to Esterel

Time is divided into discrete ticks (also called cycles, steps, instants)

Two types of statements:

- Those that take "zero time" (execute and terminate in same tick, e.g., emit)
  - Correspond to Connectors in SSMs
- Those that delay for a prescribed number of ticks (e.g., await)
  - Correspond to States in SSMs

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Signals

- Esterel programs/SSMs communicate through signals
- These are like wires
  - Each signal is either present or absent in each tick
  - Can't take multiple values within a tick
- Presence/absence not held between ticks
- Broadcast across the program
  - Any process can read or write a signal

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Signals

- Status of an input signal is determined by input event, and by local emissions
- Status of local or output signal is determined per tick
  - Default status: absent
  - ▶ Must execute an "emit S" statement to set signal S present
- ▶ await A:
  - Waits for A and terminates when A occurs

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Synchrony Hypothesis

- Computations are considered to
  - take no time
  - be atomic



Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Synchronous Model of Computation

To summarize: the synchronous model of computation of SSMs/Esterel is characterized by:

- 1. Computations considered to take no time (synchrony hypothesis)
- 2. Time is divided into discrete ticks
- 3. Signals are either present or absent in each tick

Sometimes, "synchrony" refers to just the first two points (e.g., in the original Statecharts as implemented in Statemate); to explicitly include the third requirement as well, we also speak of the strict synchrony

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# The ABRO Example

- Consider the following controller specification:
  - Emit the output O as soon as both the inputs A and B have been received.
  - Reset the behavior whenever the input R is received.
- This is still a bit ambiguous; to complete:
  - If R occurs, emit nothing
  - Do nothing at initialization time
  - Input signals may be simultaneous

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

## The ABRO Example—Mealy Style



Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Write Things Once

- The disadvantage of this (flat) notation:
  - Size grows exponentially
  - A little change to the specification may incur a major change to the automaton (often ends with full rewriting)
- The answer:
  - Add hierarchy
  - More generally: Write Things Once (WTO)
- Analogy from language theory:
  - Use regular expressions to represent large (possibly infinite) sets of strings

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

#### ABRO—Safe State Machine



Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# ABRO—The Esterel Version

```
module ABRO:
input A, B, R;
output O;
loop
[ await A || await B ];
emit O
each R
end module
```

- Esterel programs built from modules
- Each module has an interface of input and output signals
- Much simpler since language includes notions of signals, waiting, and reset

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# ABRO—The Esterel Version

```
module ABRO:
input A, B, R;
output O;
loop
[ await A || await B ];
emit O
each R
end module
```

- loop ... each statement implements reset
- || runs the two awaits in parallel
- await waits for the next tick where its signal is present
- Parallel terminates when all its threads have
- emit 0 makes signal 0 present when it runs

CIAU

Signals and Synchrony The ABRO Example Write Things Once **The multiform notion of time** Uses, Advantages, Disadvantages

# The Multiform Notion of Time

- Some "classical" programming languages already include a concept of real-time
- Consider the following Ada code fragment, which signals minutes to a task B:

loop delay 60; B.Minute end

- This works in principle
- However, it is not deterministic!

Introduction to Safe State Machines and Esterel Esterel Language Overview

Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once **The multiform notion of time** Uses, Advantages, Disadvantages

# The Multiform Notion of Time

- A design goal of synchronous languages:
  - Fully deterministic behavior
  - Applies to functionality and (logical) timing
- Approach:
  - Replace notion of physical time with notion of order
  - Only consider *simultaneity* and *precedence* of events
- Hence, physical time does not play any special role
  - Is handled like any other event from program environment
  - This is called multiform notion of time

Introduction to Safe State Machines and Esterel Esterel Language Overview

Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once **The multiform notion of time** Uses, Advantages, Disadvantages

# The Multiform Notion of Time

- Consider following requirements:
  - "The train must stop within 10 seconds"
  - "The train must stop within 100 meters"
- These are conceptually of the same nature!
- In languages where physical time plays particular role, these requirements are typically expressed completely differently
- ► In synchronous model, use similar precedence constraints:
  - "The event stop must precede the 10th (respectively, 100th) next occurrence of the event second (respectively, meter)"

Signals and Synchrony The ABRO Example Write Things Once **The multiform notion of time** Uses, Advantages, Disadvantages

# The Multiform Notion of Time

- History of system is a totally ordered sequence of logical ticks
- At each tick, an arbitrary number of events (including 0) occurs
- Event occurrences that happen at the same logical tick are considered simultaneous
- Other events are ordered as their instances of occurrences

Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Uses of SSMs/Esterel

- Wristwatch
  - Canonical example
  - Reactive, synchronous, hard real-time
- Controllers
  - Communication protocols
- Avionics
  - Fuel control system
  - Landing gear controller
  - Other user interface tasks
- Processor components (cache controller, etc.)
- General hw design

Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Advantages of SSMs/Esterel

- Model of time gives programmer precise control
- Concurrency convenient for specifying control systems
- Completely deterministic
  - ► Guaranteed: no need for locks, semaphores, etc.
- Finite-state language
  - Easy to analyze
  - Execution time predictable
  - Much easier to verify formally
- Amenable to implementation in both hardware and software

Signals and Synchrony The ABRO Example Write Things Once The multiform notion of time Uses, Advantages, Disadvantages

# Disadvantages of SSMs/Esterel

- Finite-state nature of the language limits flexibility
  - No dynamic memory allocation
  - No dynamic creation of processes
- Virtually nonexistent support for handling data (this changes in v7)
  - ▶ Must resort to some host language (e.g., C) for that
- Really suited for simple decision-dominated controllers
- Synchronous model of time can lead to overspecification
- Semantic challenges
  - Avoiding causality violations often difficult
  - Difficult to compile
- Limited number of users, tools, etc.

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Overview

#### Introduction to Safe State Machines and Esterel

#### Esterel Language Overview

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Esterel/SSM Pragmatics

#### Interfacing with the Environment

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Basic Esterel Statements

emit S

- Make signal S present in the current instant
- A signal is absent unless it is emitted

pause

Stop and resume after the next cycle after the pause

present S then stmt1 else stmt2 end

If signal S is present in the current instant, immediately run stmt1, otherwise run stmt2

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Esterel's Model of Time

- ► The standard CS model (e.g., Java's) is asynchronous
  - Threads run at their own rate
  - Synchronization is done (for example) through calls to wait() and notify()

Esterel's model of time is synchronous like that used in hardware. Threads march in lockstep to a global clock.



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### **Basic Esterel Statements**

```
module EXAMPLE1:
output A, B, C;
emit A;
present A then
  emit B
end;
pause;
emit C
end module
```



EXAMPLE1 makes signals A & B present the first instant, C present the second

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Transformation of EXAMPLE1 into SSMs



#### Structural translation of Esterel into SSMs

Performed with KIEL tool, www.informatik.uni-kiel.de/rtsys/kiel/

CAU

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Transformation of EXAMPLE1 into SSMs



#### After some optimizations

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Transformation of EXAMPLE1 into SSMs



#### Final version

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

# Signal Coherence Rules

- Each signal is only present or absent in a cycle, never both
- All writers run before any readers do

#### Thus



is an erroneous program

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

# Advantage of Synchrony

- Easy to control time
- Synchronization comes for free
- Speed of actual computation nearly uncontrollable—Synchrony allows to specify function and timing independently
- Makes for deterministic concurrency
- Explicit control of "before" "after" "at the same time"

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Time Can Be Controlled Precisely

This guarantees every 60th S an M is emitted:





# The || Operator

Groups of statements separated by || run concurrently and terminate when all groups have terminated

```
[
  emit A;
  pause; emit B;
]
  pause; emit C;
  pause; emit D
];
emit E
```



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Communication Is Instantaneous

A signal emitted in a cycle is visible immediately

```
[
pause; emit A;
pause; emit A
]]
pause;
present A then
emit B end
]
```



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## **Bidirectional Communication**

Processes can communicate back and forth in the same cycle

```
[
pause; emit A;
present B then
emit C end;
pause; emit A
]]
pause;
present A then
emit B end
]
```



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Concurrency and Determinism

- Signals are the only way for concurrent processes to communicate
- Esterel does have variables, but they cannot be shared
- Signal coherence rules ensure deterministic behavior
- Language semantics clearly defines who must communicate with whom when

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

### The Await Statement

- The await statement waits for a particular cycle
- await S waits for the next cycle in which S is present

```
[
emit A;
pause;
pause; emit A
||
await A; emit B
]
```


Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## The Await Statement

- await normally waits for a cycle before beginning to check
- await immediate also checks the initial cycle

```
[
  emit A;
  pause;
  pause; emit A
]]
  await immediate A;
  emit B
]
```



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Loops

- Esterel has an infinite loop statement
- Rule: loop body cannot terminate instantly
  - Needs at least one pause, await, etc.
  - Can't do an infinite amount of work in a single cycle

loop	
emit A;	
pause;	
pause;	
emit B	
end	



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

### Loops and Synchronization

Instantaneous nature of loops plus await provide very powerful synchronization mechanisms

loop await 60 S; emit M end	



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

# Preemption

- Often want to stop doing something and start doing something else
- ► E.g., Ctrl-C in Unix: stop the currently-running program
- Esterel has many constructs for handling preemption

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## The Abort Statement

- Basic preemption mechanism
- ► General form:

abort	
statement	
when condition	

- Runs statement to completion
- If condition ever holds, abort terminates immediately.

Introduction to Safe State Machines and Esterel Esterel Language Overview Esterel/SSM Pragmatics Interfacing with the Environment Property Verification Preemption, exceptions, suspension

#### The Abort Statement



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

# Strong vs. Weak Preemption

#### Strong preemption:

- > The body does not run when the preemption condition holds
- The previous example illustrated strong preemption

#### Weak preemption:

- The body is allowed to run even when the preemption condition holds, but is terminated thereafter
- weak abort implements this in Esterel

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Strong vs. Weak Abort







emit A does run, body terminated afterwards

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Strong vs. Weak Preemption

- Important distinction
- Something cannot cause its own strong preemption

abort	
<pre>pause;</pre>	
emit A	
when A	
	]

Erroneous!

weak abort pause; emit A when A

Ok!

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Nested Preemption

```
module RUNNER
input SECOND, METER, LAP;
output ... ;
every MORNING do
 abort
   loop
     abort RUNSLOWLY when 15 SECOND;
     abort
       every STEP do
         JUMP || BREATHE
       end every
     when 100 METER:
     FULLSPEED
   each LAP
  when 2 LAP
end every
end module
```

CIAU

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## Exceptions—The Trap Statement

- Esterel provides an exception facility for weak preemption
- Interacts nicely with concurrency
- Rule: outermost trap takes precedence

### The Trap Statement



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

### Nested Traps



- Outer trap takes precedence; control transferred directly to the outer trap statement.
- emit A not allowed to run.



Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### Combining Abortion and Exceptions

```
trap HEARTATTACK in
 abort
   loop
     abort RUNSLOWLY when 15 SECOND;
     abort
       every STEP do
         JUMP || BREATHE || CHECKHEART
       end every
     when 100 METER:
     FULLSPEED
   each LAP
 when 2 LAP
handle HEARTATTACK do
 GOTOHOSPITAL
end trap
```

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

## The Suspend Statement

- Preemption (abort, trap) terminate something, but what if you want to pause it?
- Like the POSIX Ctrl-Z
- Esterel's suspend statement pauses the execution of a group of statements
- Only strong preemption: statement does not run when condition holds

Signal emission + testing, pausing Esterel's model of time Parallelism Signal awaiting, looping Preemption, exceptions, suspension

#### The Suspend Statement





People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Overview

Introduction to Safe State Machines and Esterel

#### Esterel Language Overview

#### Esterel/SSM Pragmatics

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

Interfacing with the Environment

## Property Verification

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## People Counter Example

Construct an Esterel program that counts the number of people in a room.

- People enter the room from one door with a photocell that changes from 0 to 1 when the light is interrupted, and leave from a second door with a similar photocell. These inputs may be "1" for more than one clock cycle. It is assumed that one continuous sequence of 1's corresponds to a single person passing the photocell. The two photocell inputs are called ENTER and LEAVE.
- There are two outputs: EMPTY and FULL, which are present when the room is empty and contains three people respectively.

Source: Mano, Digital Design, 1984, p. 336

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## **Overall Structure**



Conditioner detects rising edges of signal from photocell. Counter tracks number of people in the room.

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Implementing & Testing the Conditioner



<pre>module CONDITIONER: input A; output Y;</pre>
<pre>loop   await A; emit Y;   await [not A]; end</pre>
end module

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Implementing & Testing the Conditioner

```
module CONDITIONER:
input A;
output Y;
loop
  await A; emit Y;
  await [not A];
end
end module
```

```
CONDITIONER> ;
--- Output:
CONDITIONER> A; # Rising edge
--- Output: Y
CONDITIONER> A; # Doesn't generate a pulse
--- Output:
CONDITIONER> ; # Doesn't generate a pulse
--- Output:
CONDITIONER> : # Reset
--- Output:
CONDITIONER> A; # Another rising edge
--- Output: Y
CONDITIONER> :
--- Output:
CONDITIONER> A:
--- Output: Y
```

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

### Implementing & Testing the Counter: First Try



```
module COUNTER:
input ADD, SUB;
output FULL, EMPTY;
var count := 0 : integer in
 loop
   present ADD then
     if count < 3 then
       count := count + 1
   end end;
   present SUB then
     if count > 0 then
       count := count - 1
   end end;
   if count = 0 then emit EMPTY end:
   if count = 3 then emit FULL end:
   pause
 end
end
end module
```

CAU

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Implementing & Testing the Counter: First Try



COUNTER> : --- Output: EMPTY COUNTER> ADD SUB: --- Output: EMPTY COUNTER> ADD: --- Output: COUNTER> SUB: --- Output: EMPTY COUNTER> ADD: --- Output: COUNTER> ADD; --- Output: COUNTER> ADD: --- Output: FULL COUNTER> ADD SUB: --- Output: # Oops!

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

#### Implementing & Testing the Counter: Second Try



```
module COUNTER:
input ADD, SUB;
output FULL, EMPTY;
var c := 0 : integer in
  loop
   present ADD then
     present SUB else
       if c < 3 then c := c + 1 end end
   else
     present SUB then
       if c > 0 then c := c - 1 end end
   end:
   if c = 0 then emit EMPTY end:
   if c = 3 then emit FULL end;
   pause
  end
end
end module
```

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Implementing & Testing the Counter: Second Try



COUNTER> ; --- Output: EMPTY COUNTER> ADD SUB; --- Output: EMPTY COUNTER> ADD SUB; --- Output: EMPTY COUNTER> ADD; --- Output: COUNTER> ADD: --- Output: COUNTER> ADD: --- Output: FULL COUNTER> ADD SUB: --- Output: FULL # Working COUNTER> ADD SUB: --- Output: FULL COUNTER> SUB: --- Output: COUNTER> SUB: --- Output: COUNTER> SUB; --- Output: EMPTY COUNTER> SUB; --- Output: EMPTY

CIAU

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Assembling the People Counter



```
module PEOPLECOUNTER:
```

```
input ENTER, LEAVE;
output EMPTY, FULL;
```

```
signal ADD, SUB in
  run CONDITIONER[signal ENTER / A, ADD / Y]
||
  run CONDITIONER[signal LEAVE / A, SUB / Y]
||
  run COUNTER
end
end module
```

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

# Vending Machine Example

Design a vending machine controller that dispenses gum once.

- Two inputs, N and D, are present when a nickel and dime have been inserted.
  - $\mathbf{N} = \mathbf{0}$



A single output, GUM, should be present for a single cycle when the machine has been given fifteen cents.



No change is returned.

Source: Katz, Contemporary Logic Design, 1994, p. 389

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Vending Machine Solution—Variable Encoding



```
module VENDING:
input N, D;
output GUM;
loop
var m := 0 : integer in
 trap WAIT in
  1000
   present N then m := m + 5; end;
   present D then m := m + 10; end;
   if m >= 15 then exit WAIT end;
   pause
   end loop
  end trap;
  emit GUM; pause
 end
end
end module
```

CIAU

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Alternative Solution—State Encoding



loop
await
case immediate N do await
case N do await
case N do nothing
case immediate D do nothing
end
case immediate D do nothing
end
case immediate D do await
case immediate N do nothing
case D do nothing
end
end;
emit GUM; pause
end

CIAU

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

# Tail Lights Example

Construct an Esterel program that controls the turn signals of a 1965 Ford Thunderbird.



Source: Wakerly, Digital Design Principles & Practices, 2ed, 1994, p. 550



People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Tail Light Behavior











EE 249

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

# Tail Lights

- There are three inputs, which initiate the sequences: LEFT, RIGHT, and HAZ
- Six outputs: LA, LB, LC, RA, RB, and RC
- The flashing sequence is



People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

# A Single Tail Light



```
module LIGHTS:
output A, B, C;
loop
  emit A; pause;
  emit A; emit B; pause;
  emit A; emit B; emit C; pause;
  pause
  end
end module
```

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

#### The T-Bird Controller: Esterel Version

```
input LEFT, RIGHT, HAZ;
output LA, LB, LC, RA, RB, RC;
loop
  await
   case immediate HAZ do
     abort
       run LIGHTS[signal LA/A, LB/B, LC/C]
     11
       run LIGHTS[signal RA/A, RB/B, RC/C]
     when [not HAZ]
   case immediate LEFT do
     abort run LIGHTS[signal LA/A, LB/B, LC/C]
     when [not LEFT]
   case immediate RIGHT do
     abort run Lights[signal RA/A, RB/B, RC/C]
     when [not RIGHT]
  end
 end
```

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

### The T-Bird Controller: SSM Version





People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## Comments on the T-Bird

- This solution uses Esterel's innate ability to control the execution of processes, producing succinct easy-to-understand source but a somewhat larger executable.
- An alternative: Use signals to control the execution of two processes, one for the left lights, one for the right.
- A challenge: Synchronizing hazards.
- Most communication signals can be either level- or edge-sensitive.
- Control can be done explicitly, or implicitly through signals.
People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

#### Traffic-Light Controller Example



Control a traffic light at the intersection of a busy highway and a farm road.

Source: Mead and Conway, Introduction to VLSI

Systems, 1980, p. 85.

- Normally, the highway light is green
- If a sensor detects a car on the farm road:
  - The highway light turns yellow then red.
  - The farm road light then turns green until there are no cars or after a long timeout.
  - Then, the farm road light turns yellow then red, and the highway light returns to green.
- Inputs: The car sensor C, a short timeout signal S, and a long timeout signal L.
- Outputs: A timer start signal R, and the colors of the highway and farm road lights HG, HY, HR, FG, FY, and FR.

People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

## The Traffic Light Controller (SSM)



People Counter Example Vending Machine Example Tail Lights Example Traffic-Light Controller Example

#### The Traffic Light Controller (Esterel) module TIMER: input R, SEC; module TLC: output L, S; input C, SEC; output HG, HY, HR, 1000 FG. FY. FR: weak abort await 3 SEC: signal S, L, S in run TIMER sustain S run FSM await 5 SEC: end sustain L end module when R; end end module

```
module FSM:
input C, L, S;
output R, HG, HY, HR,
FG, FY, FR;
```

```
loop
emit HG; emit FR;
emit R;
await [C and L];
emit HY; emit R;
await S;
emit HR; emit FG;
emit R;
await [(not C) or L];
emit FY; emit R;
await S;
end
end module
```

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

## Overview

Introduction to Safe State Machines and Esterel

Esterel Language Overview

Esterel/SSM Pragmatics

Interfacing with the Environment Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

#### **Property Verification**

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

## Interfacing with the Environment

- At some point, our reactive system must control real-world entities
- There are usually different options for the interface—differing in
  - Ease of use
  - Ease of making mistakes!
- Example: External device that can be ON or OFF

Options:

- 1. Single pure signal
- 2. Two pure signals
- 3. Boolean valued signal

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

# Valued Signals

- Beside the status present or absent, a signal can have an additional value.
- Valued signals are declared with a certain type
- output S: integer declares an output signal of type integer
- emit S(15) makes signal S present and assigns it the value 15
- Value of signal S can be tested by ?S
- The value is persistent across logical ticks
- ► To preserve determinism, only one signal value per tick allowed

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

# Valued Signals

#### Single valued signal:

Only one statement can emit signal per instant

#### Combined valued signal:

- Multiple emitters allowed
- Indicated with combine keyword
- Are combined with (commutative and associative) binary operator
- boolean: combination function can be and or or
- integer, float, double: can use + or \*

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

## Variables

- ... are assignable objects with name and type
- ... similar rules as for signals (regarding placement, scoping)
- Value is undefined until first assignment

```
var
X : double,
Count := ? Distance : integer,
Deadline : Time
in
P
end var
```

- Must declare type individually for each variable
  - var X, Y integer is incorrect!

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals Option 3: Boolean valued signal

# Different Modes of Motor Control

- **Option 1**: Single pure signal
  - Motor is running in every instant which has the MOTOR signal present

Pro:

Minimal number of signals

Con:

- High number of signal emissions (signal is emitted in every instant where the motor is on)—may be unnecessary run-time overhead
- Somewhat heavy/unintuitive representation

input BUMPER; output MOTOR; abort sustain MOTOR when BUMPER

Esterel Signal Types Option 1: Single pure signal **Option 2: Two pure signals** Option 3: Boolean valued signal

# Different Modes of Motor Control

Option 2: Two pure signals

- Motor is switched on with signal MOTOR\_ON present
- Motor is switched off with signal MOTOR\_OFF present
- If neither MOTOR\_ON or MOTOR\_OFF is present, motor keeps its previous state Pro:
  - Signal emissions truly indicate significant change of external state
  - Simple representation in Esterel

Con:

- No way to control inconsistent outputs
- No memory cannot check in retrospect which signal was emitted

input BUMPER; output MOTOR\_ON, MOTOR\_OFF;

emit MOTOR\_ON; await BUMPER; emit MOTOR\_OFF;

Esterel Signal Types Option 1: Single pure signal **Option 2: Two pure signals** Option 3: Boolean valued signal

## Inconsistent Outputs

- Problem with MOTOR\_ON and MOTOR\_OFF: undefined behavior with both signals present
- Can address this at host-language level
- Can (and should) also address this at Esterel-level:

```
present BUMPER else
emit MOTOR_ON;
await BUMPER
end present;
emit MOTOR_OFF
||
await immediate MOTOR_ON and MOTOR_OFF;
exit INTERNAL_ERROR
```

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals **Option 3: Boolean valued signal** 

## Valued Signal for Motor Control

**Option 3:** Boolean valued signal

- Merge pure signals MOTOR\_ON and MOTOR\_OFF into one valued signal MOTOR
- Motor is switched on if every emit-statement in that instant emits true
- Here: In case of conflicting outputs, motor stays switched off

input BUMPER; output MOTOR combine BOOLEAN with and;

emit MOTOR(true); await immediate BUMPER; emit MOTOR(false);

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals **Option 3: Boolean valued signal** 

#### Valued Signal for Motor Control Option 3 contd. Pro:

- Again only one signal for motor control
- Explicit control of behavior for inconsistent outputs
- Valued signal has memory—can be polled in later instances, after emission
- Easy extension to finer speed control

Con:

- Inconsistent outputs are handled deterministically—but are not any more detected and made explicit
- For certain classes of analyses/formal methods that we may wish to apply, valued signals are more difficult to handle than pure signals

Esterel Signal Types Option 1: Single pure signal Option 2: Two pure signals **Option 3: Boolean valued signal** 

### Events vs. State

- Excessive signal emissions
  - make the behavior difficult to understand
  - cause overhead if fed to the external environment
- State:
  - "Robot is turning left"
  - "Motor is on"
  - Esterel:
    - waiting for some signal
    - terminated thread
    - value of valued signal
- Event:
  - Change of State
  - "Turn motor on"
  - Esterel:
    - emit pure signal
    - change value of signal

Introduction Example: ABRO

#### Overview

Introduction to Safe State Machines and Esterel

- Esterel Language Overview
- Esterel/SSM Pragmatics
- Interfacing with the Environment

#### Property Verification Introduction

Example: ABRO

Introduction Example: ABRO

# Property Verification

- One advantage of formal foundation of synchronous model: Ability to formally verify certain properties
- Can conveniently specify properties using observers, using the familiar SSM/Esterel formalism
- Observers scan for
  - Always type properties (must always be fulfilled)
  - Never type properties (should never occur)
- Verifier, based on model checking, is included in Esterel Studio

Introduction Example: ABRO

### Example: ABRO



#### Property P1:

*D* cannot be emitted if *B* has not been received since the last occurrence of *R* 

#### Observer for P1:



Introduction Example: ABRO

#### Screenshot of Esterel-Studio Verifier



Example: ABRO

## Example: ABRO



#### **Property P2:**

 $\Pi$  is never emitted twice since the last occurrence of R

Observer for P2:



Introduction Example: ABRO

# Summary I

- Classical real-time languages include specific notions of physical time—however, they do not achieve complete determinism this way
- Synchronous languages replace notion of physical time with notion of order, considering only simultaneity and precedence of events—this is the multiform notion of time
- The Write-Things-Once principle aids to make representations compact, and to ease modifications. For state machines, WTO is achieved by adding hierarchy

# Summary II

- Esterel is an imperative, control-oriented synchronous language
- Synchronous model of time, as used by SSMs
  - Time divided into sequence of discrete ticks
  - Instructions either run and terminate in the same tick or explicitly in later ticks
- Idea of signals and broadcast
  - "Variables" that take exactly one value each tick and don't persist
  - Coherence rule: all writers run before any readers

# Summary III

- Esterel and SSMs are high-level descriptions—however, there are still several options to express the same behavior
- May e. g.alternatively use state encoding or variable encoding to memorize control state across logical ticks
- Can use macro facility to modularize description
- Care should be taken to select a suitable interface with the environment—single pure signals, two pure signals, or Boolean valued signal
- The formally founded semantics of Esterel allows to perform formal verification (for more, should attend additional class, e. g. "Verification of Concurrent Programs")

Introduction Example: ABRO

## To Go Further

- Nicolas Halbwachs, Synchronous programming of reactive systems, a tutorial and commented bibliography, *Tenth International Conference on Computer-Aided Verification*, CAV'98 Vancouver (B.C.), LNCS 1427, Springer Verlag, June 1998, http: //www-verimag.imag.fr/~halbwach/cav98tutorial.html
- Gérard Berry, The Foundations of Esterel, Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, *Foundations of Computing Series*, 2000, ftp: //ftp.esterel.org/esterel/pub/papers/foundations.ps
- Esterel Web, http://www-sop.inria.fr/esterel.org/
- Home page of Esterel Technologies, http://www.esterel-technologies.com/v3/