Overview

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

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

Signals

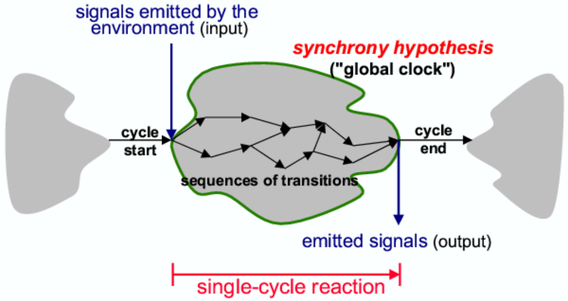
- Esterel programs/SSMs communicate through signals
- These are like wires
  - Each signal is either present or absent in each tick
  - Can't take multiple values within a tick
- Presence/absence not held between ticks
- Broadcast across the program
  - Any process can read or write a signal

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

Synchrony Hypothesis

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



G. Luettgen 2001

Synchronous Model of Computation

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

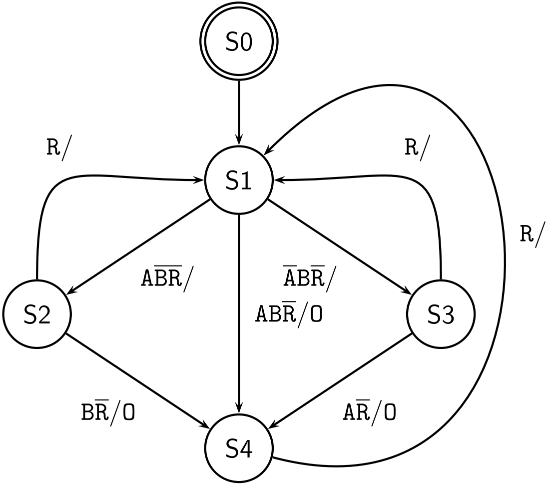
1. Computations considered to take no time (synchrony hypothesis)
2. Time is divided into discrete ticks
3. Signals are either present or absent in each tick

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

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

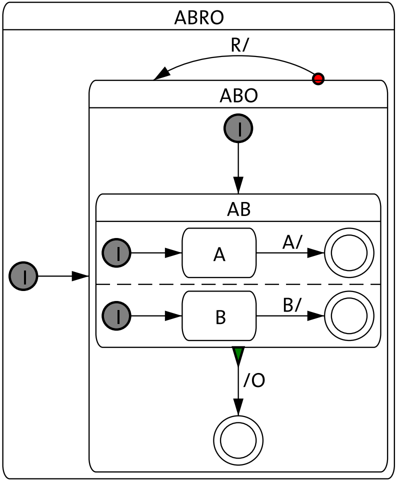
The ABRO Example—Mealy Style



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

ABRO—Safe State Machine



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

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 O` makes signal `O` present when it runs

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

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)"

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

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

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

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.

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

Basic Esterel Statements

`emit S`

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

`pause`

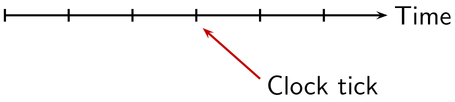
- Stop and resume after the next cycle after the pause

`present S then` *stmt1* `else` *stmt2* `end`

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

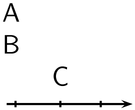
Esterel's Model of Time

- 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.



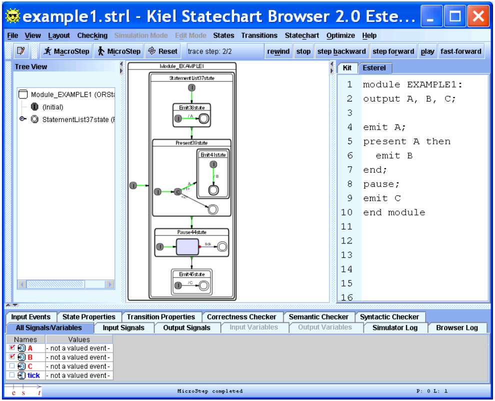
Basic Esterel Statements

```
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

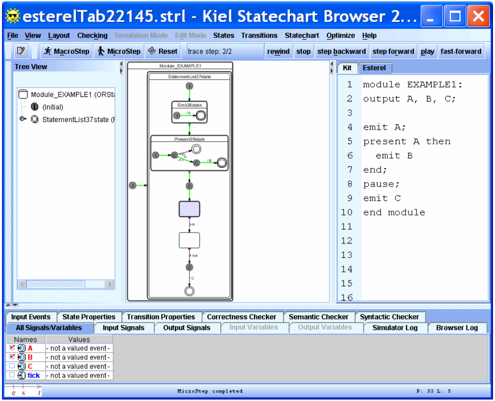
Transformation of EXAMPLE1 into SSMs



Structural translation of Esterel into SSMs

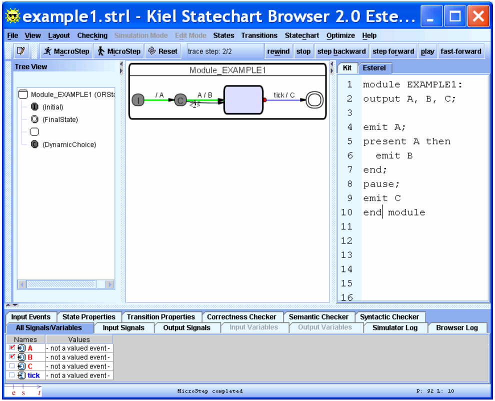
Performed with KIEL tool, www.informatik.uni-kiel.de/rtsys/kiel/

Transformation of EXAMPLE1 into SSMs



After some optimizations

Transformation of EXAMPLE1 into SSMs



Final version

Signal Coherence Rules

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

```
present A else
  emit A
end
```

is an erroneous program

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"

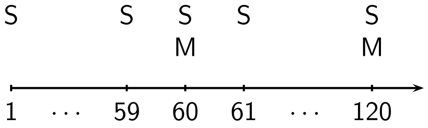
Time Can Be Controlled Precisely

This guarantees every 60th S an M is emitted:

```
every 60 S do
  emit M
end
```

`every` invokes its body every 60th `S`

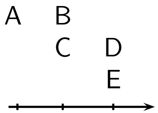
`emit` takes no time (cycles)



The || Operator

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

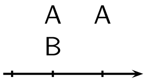
```
[
  emit A;
  pause; emit B;
||
  pause; emit C;
  pause; emit D
];
emit E
```



Communication Is Instantaneous

A signal emitted in a cycle is visible immediately

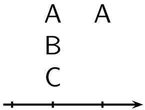
```
[
  pause; emit A;
  pause; emit A
||
  pause;
  present A then
    emit B end
]
```



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
]
```



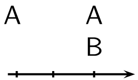
Concurrency and Determinism

- 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

The Await Statement

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

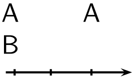
```
[
  emit A;
  pause;
  pause; emit A
||
  await A; emit B
]
```



The Await Statement

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

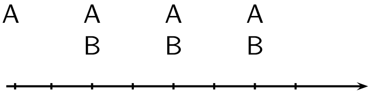
```
[
  emit A;
  pause;
  pause; emit A
||
  await immediate A;
  emit B
]
```



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

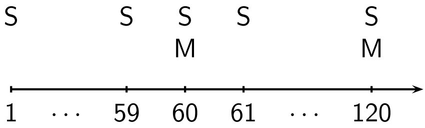
```
loop
  emit A;
  pause;
  pause;
  emit B
end
```



Loops and Synchronization

Instantaneous nature of loops plus `await` provide very powerful synchronization mechanisms

```
loop
  await 60 S;
  emit M
end
```



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

The Abort Statement

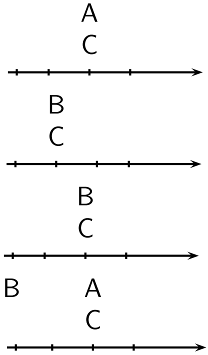
- Basic preemption mechanism
- General form:

```
abort
  statement
when condition
```

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

The Abort Statement

```
abort
  pause;
  pause;
  emit A
when B;
emit C
```



Normal Termination

Aborted termination

Aborted termination; emit A preempted

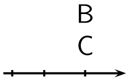
Normal Termination
B not checked
in first cycle
(like await)

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

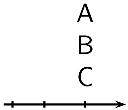
Strong vs. Weak Abort

```
abort
  pause;
  pause;
  emit A;
  pause
when B;
emit C
```



`emit A` not allowed to run

```
weak abort
  pause;
  pause;
  emit A;
  pause
when B;
emit C
```



`emit A` does run, body terminated afterwards

Strong vs. Weak Preemption

- Important distinction
- Something cannot cause its own strong preemption

```
abort
  pause;
  emit A
when A
```

Erroneous!

```
weak abort
  pause;
  emit A
when A
```

Ok!

Nested Preemption

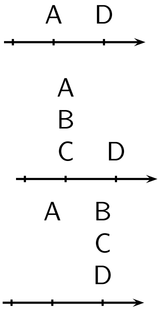
```
module RUNNER
input SECOND, METER, LAP;
output ... ;

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
```

Exceptions—The Trap Statement

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

The Trap Statement

```
trap T in
[
  pause;
  emit A;
  pause;
  exit T
||
  await B;
  emit C
]
end trap;
emit D
```



Normal termination from first process

`emit C` also runs

Second process allowed to run even though first process has exited

Nested Traps

```
trap T1 in
  trap T2 in
  [
    exit T1
    ||
    exit T2
  ]
  end;
  emit A
end;
emit B
```

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

Combining Abortion and Exceptions

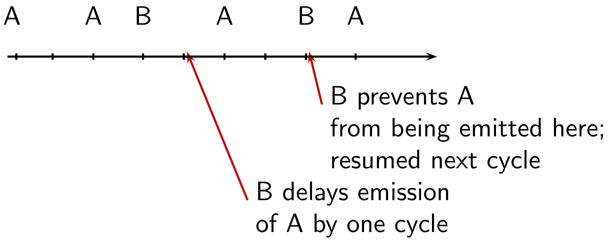
```
trap HEARTATTACK in
  abort
    loop
      abort RUNSLOWLY when 15 SECOND;
      abort
        every STEP do
          JUMP || BREATHE || CHECKHEART
        end every
      when 100 METER;
      FULLSPEED
    each LAP
  when 2 LAP
handle HEARTATTACK do
  GOTOHOSPITAL
end trap
```

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

The Suspend Statement

```
suspend
  loop
    emit A;
    pause;
    pause
  end
when B
```



Overview

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

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.

  `N` =  `D` = 

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

  `GUM` = 

- No change is returned.

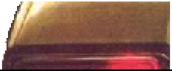
Source: Katz, *Contemporary Logic Design*, 1994, p. 389

Tail Lights Example

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

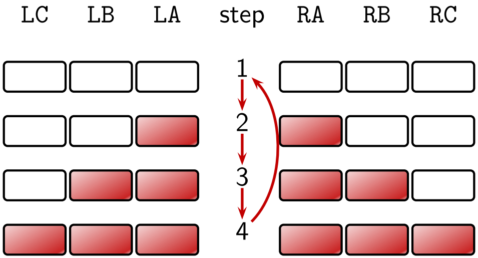
Source: Wakerly, *Digital Design Principles & Practices*, 2ed, 1994, p. 550

Tail Light Behavior

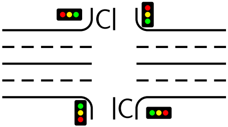


Tail Lights

- 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



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`.

Overview

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

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

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 $*$

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!

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
```

Different Modes of Motor Control

```
input BUMPER;
output MOTOR_ON,
         MOTOR_OFF;

emit MOTOR_ON;
await BUMPER;
emit MOTOR_OFF;
```

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

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
```

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);
```

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

Events vs. State

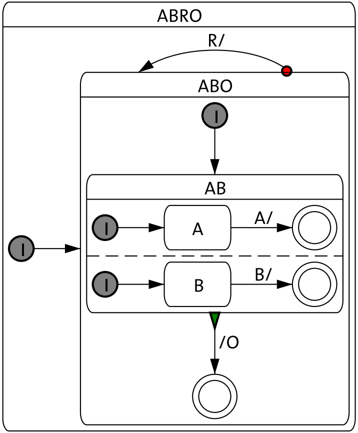
- Excessive signal emissions
  - make the behavior difficult to understand
  - cause overhead if fed to the external environment
- State:
  - "Robot is turning left"
  - "Motor is on"
  - Esterel:
    - waiting for some signal
    - terminated thread
    - value of valued signal
- Event:
  - Change of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

Overview

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

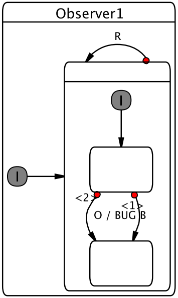
Example: ABRO



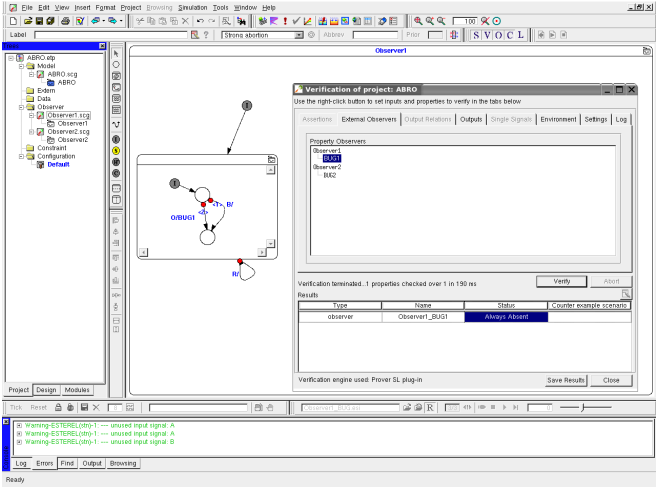
Property P1:

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

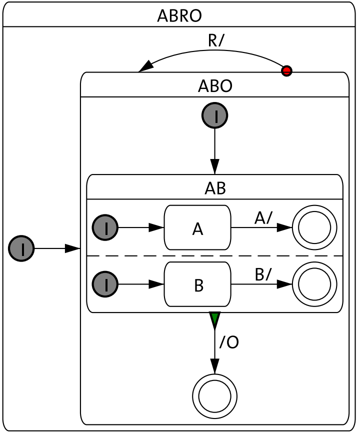
Observer for P1:



Screenshot of Esterel-Studio Verifier



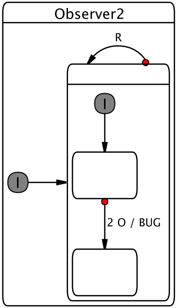
Example: ABRO



Property P2:

O is never emitted twice since the last occurrence of R

Observer for P2:



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")

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/`