Synchronous Reactive Languages
Lustre

Alexandre Denault
Comp-763B
Winter 2008

Outline

- What's a Synchronous Reactive Language?
- A Introduction to Lustre
- Verification, Simulation and Execution
- A Simple Example
- Esterel

Synchronous Languages

- What?
  - Time is expressed as a series of external events.
  - Events are processed at regular intervals.
  - It's like having a clock.
- Why?
  - It's easier to implement in hardware.
  - It's easier to verify.
- Concerns
  - If you want to simulate asynchronous behavior, you need to make sure your clock ticks are small enough.

Reactive Systems

- What?
  - Systems that have a relationship with their environments. (Harel)
  - Their behavior is to react to external stimuli.
- Why?
  - Because reactive systems are easier to map onto hardware.

Lustre

- First created in 1983.
- Commercially transformed to Scade in 1993 by Verilog.
- Esterel Technologies now own the rights to Scade.
- Active research is done by the Synchrone group from the Verimag research center.
- It is not a file system !

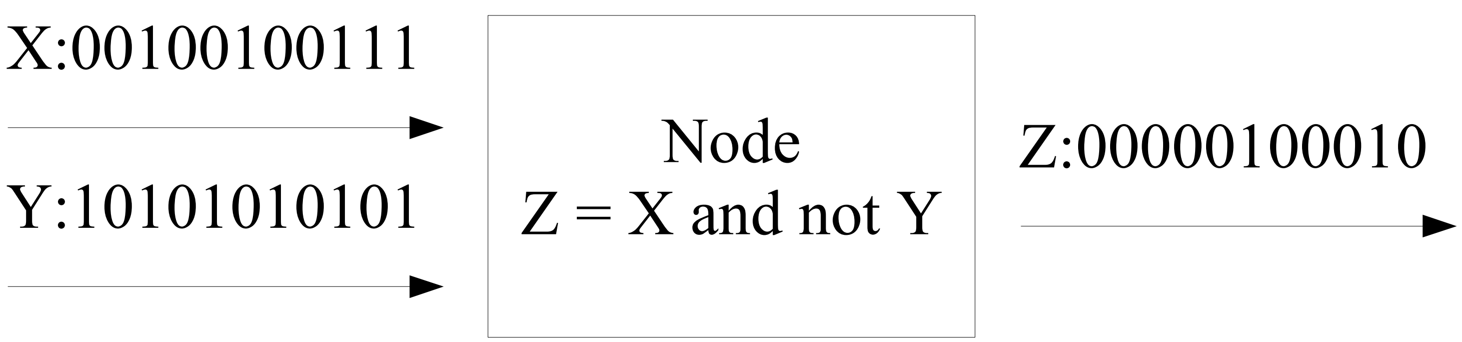
Synchrone Group Website; http://www-verimag.imag.fr/SYNCHRONE/

Application

- Cases studies on Lustre
  - Electrical load distribution on A380 aircraft
  - Gyroscope in Indian Aircraft industry
  - Ariane v launcher, experimental version
- Companies using Scade (success stories)
  - Airbus
  - Pratt & Whitney
  - CS Canada
  - Eurocopter

Deeper look at Lustre

- A control module in Lustre is called a node.
- Lustre modules operate on streams.
- Streams can be booleans or numericals.



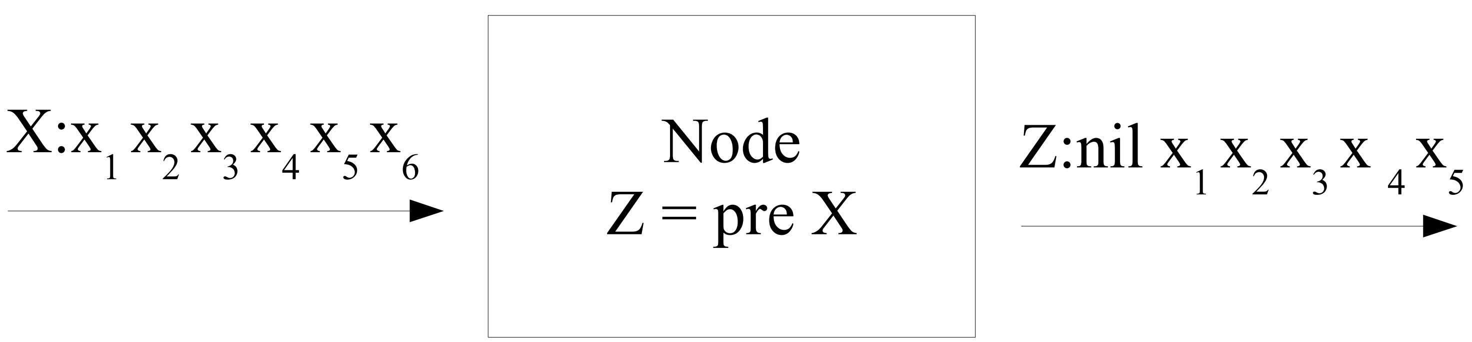
N. Halbwachs, F. Lagnier and C. Ratel.; Programming and verifying critical systems by means of the synchronous data-flow programming language Lustre.; IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems. September 1992.

Syntax Examples

- On boolean values:
  - A = X and not Y
  - A = X xor Y
  - A = if X then Y else Z
- On numerical values
  - A = X + Y
  - A = X – Y
  - A = X div Y
  - A = if (X > Y) then X else Y
  - A = if (X <> Z) then X else Y

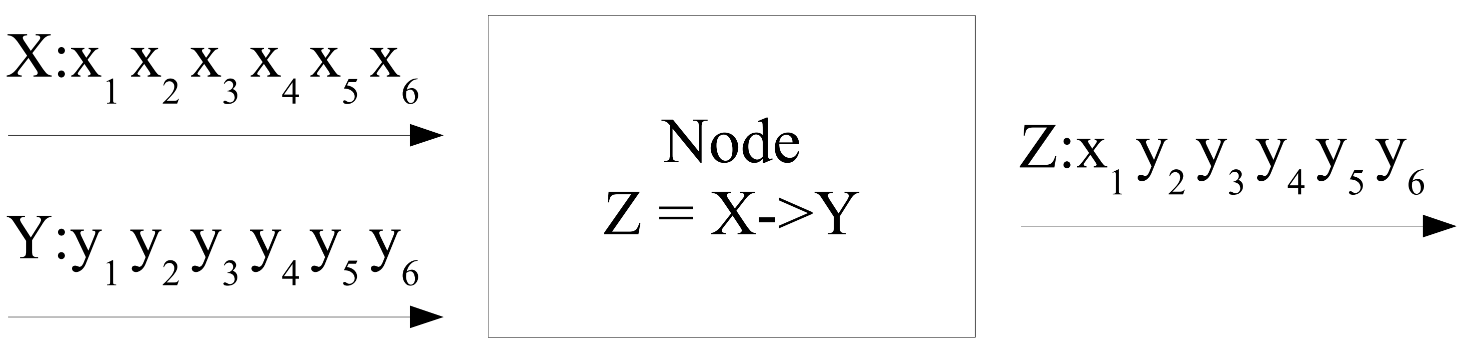
Pre Operator

- The pre operator allows us to refer the n position of a stream as the n-1 position of another stream.



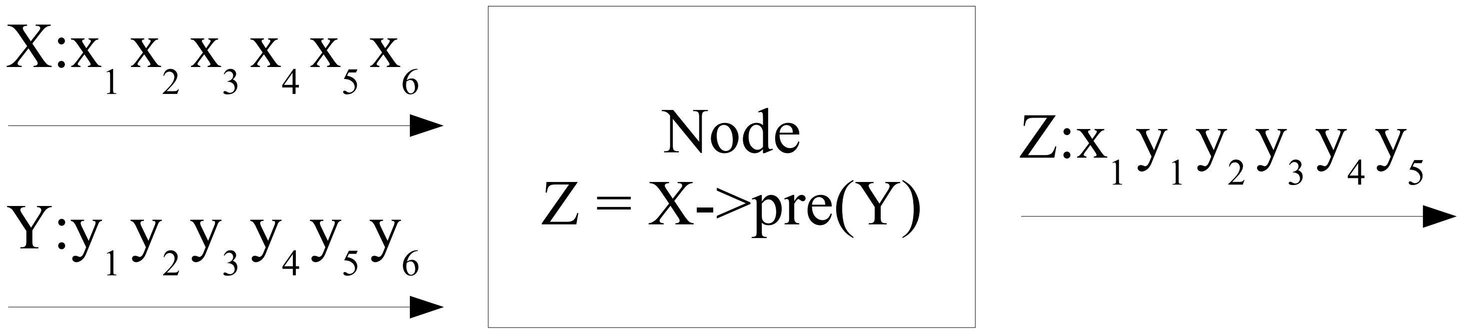
-> Operator

- The -> operator is used to initialize streams.



Pre and -> combined

- The Pre and -> operators are most useful when combined.



VSE

- **V**erification: We can run verifications on the model to insure that catastrophic situations cannot occur.
- **S**imulation: We can use a simulator to verify that the model functions correctly.
- **E**xecution: We can generate code from the model, and then integrate that code into the target platform, where it will be executed.

Verification

- We can establish safety properties for our system:
  - In a Y segment, only one car at a time should be merging in.
  - Landing gear should not retract while plane is landing.
- Verification is the action of insuring that these safety properties are never violated.
- In Lustre, safety properties are described as assertions, and must always be true.

Temporal Logic

- Describing certain assertions requires temporal logic.

  Any occurrence of a critical situation causes an alarm, which must be sustained within a five seconds delay.

- This statement could be generalized to

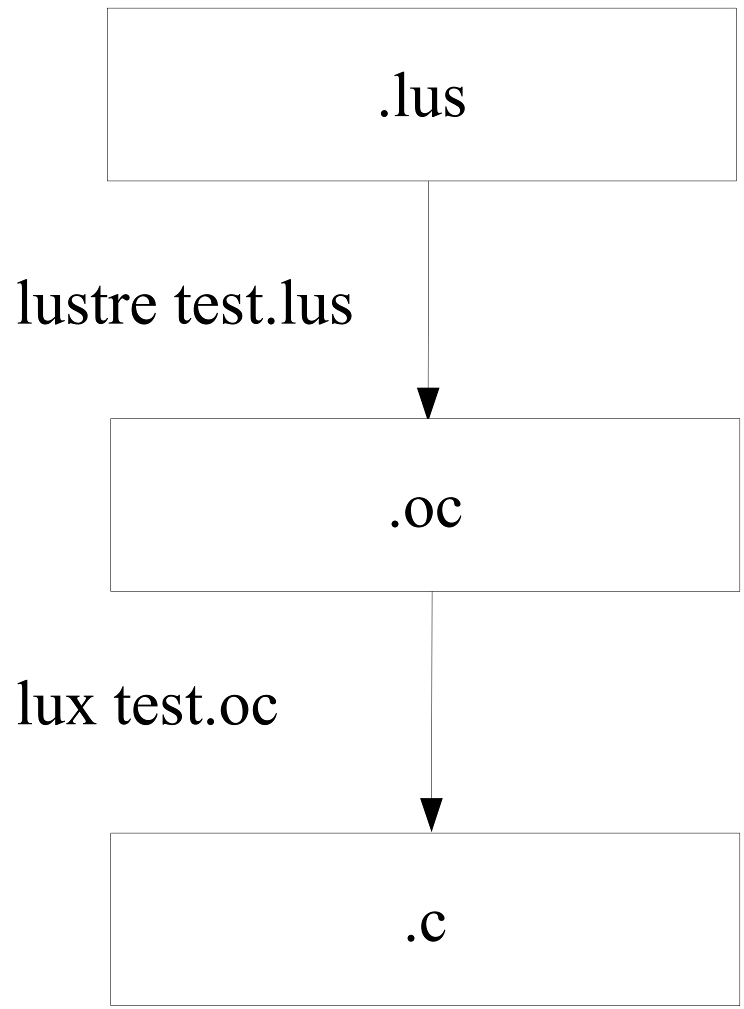
  Any occurrence of event A must cause the condition B to be true until the next occurrence of C.

- This requires knowledge of the futur, which Lustre does not.
- However, this can be rewritten as

  Any time A has occurred in the past, either B has been continuously true, or C has occurred at least once, since the last occurrence of A.

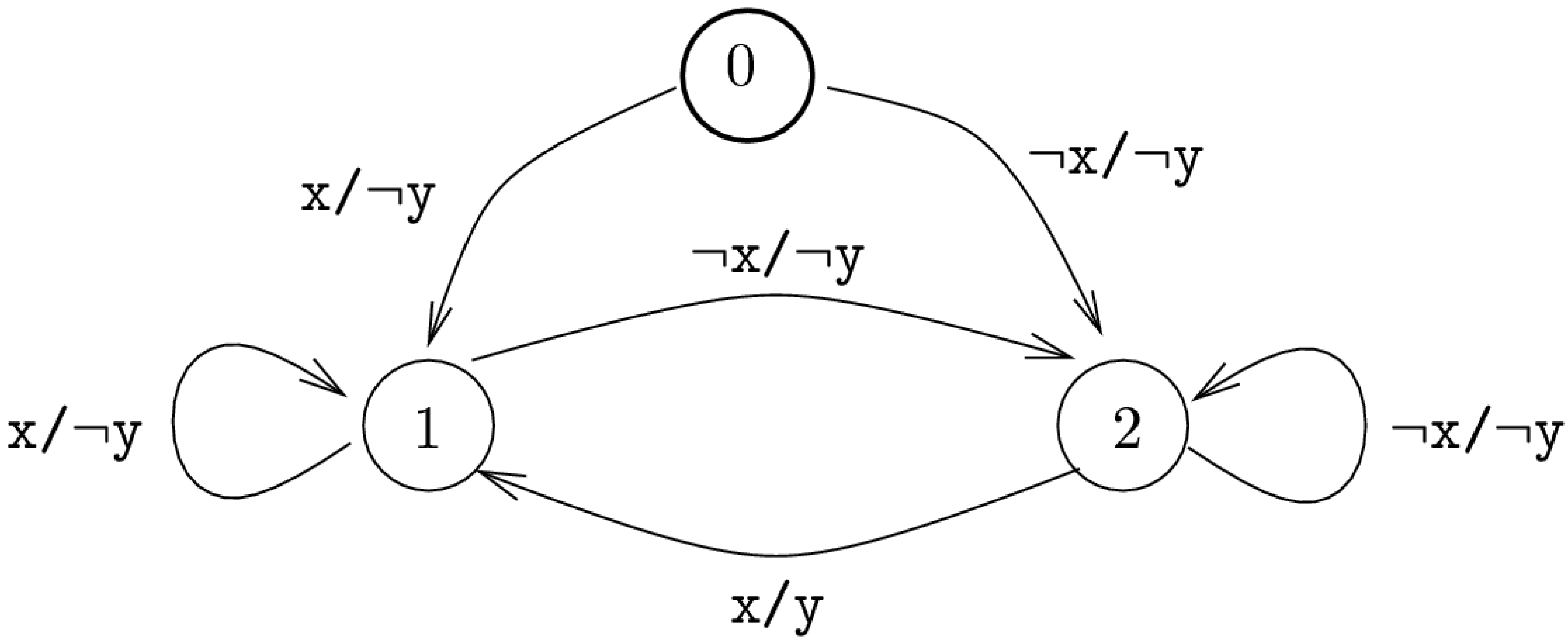
Various Forms of Lustre

- Lustre: .lus
  - This contains the Lustre nodes in their textual format.
- Lustre-Esterel Common Format: .oc
  - The Lustre application gets reduced to a FSA.
  - It can then be optimized.
- C source code: .c
  - The FSN gets transformed into C source code.

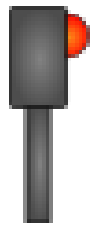


Finite State Automaton

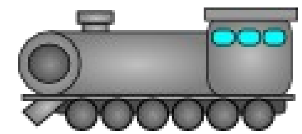
```
node EDGE(x: bool) returns (y: bool);
   Y = false -> X and not pre(X)
```



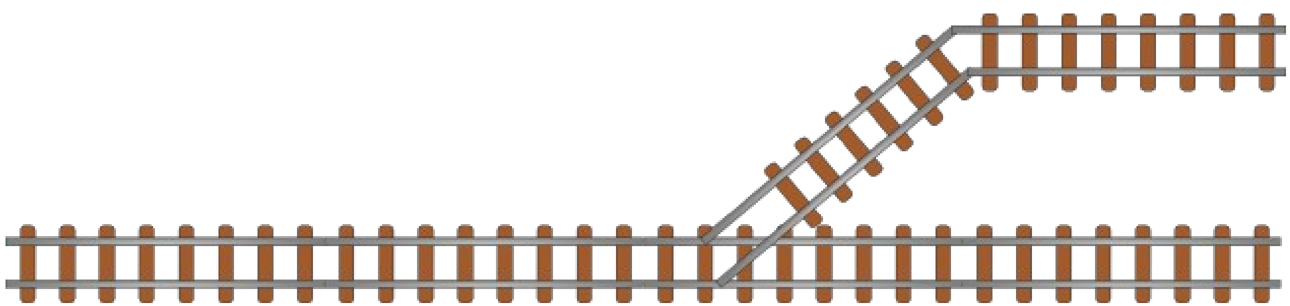
Simple Example



Signal Light

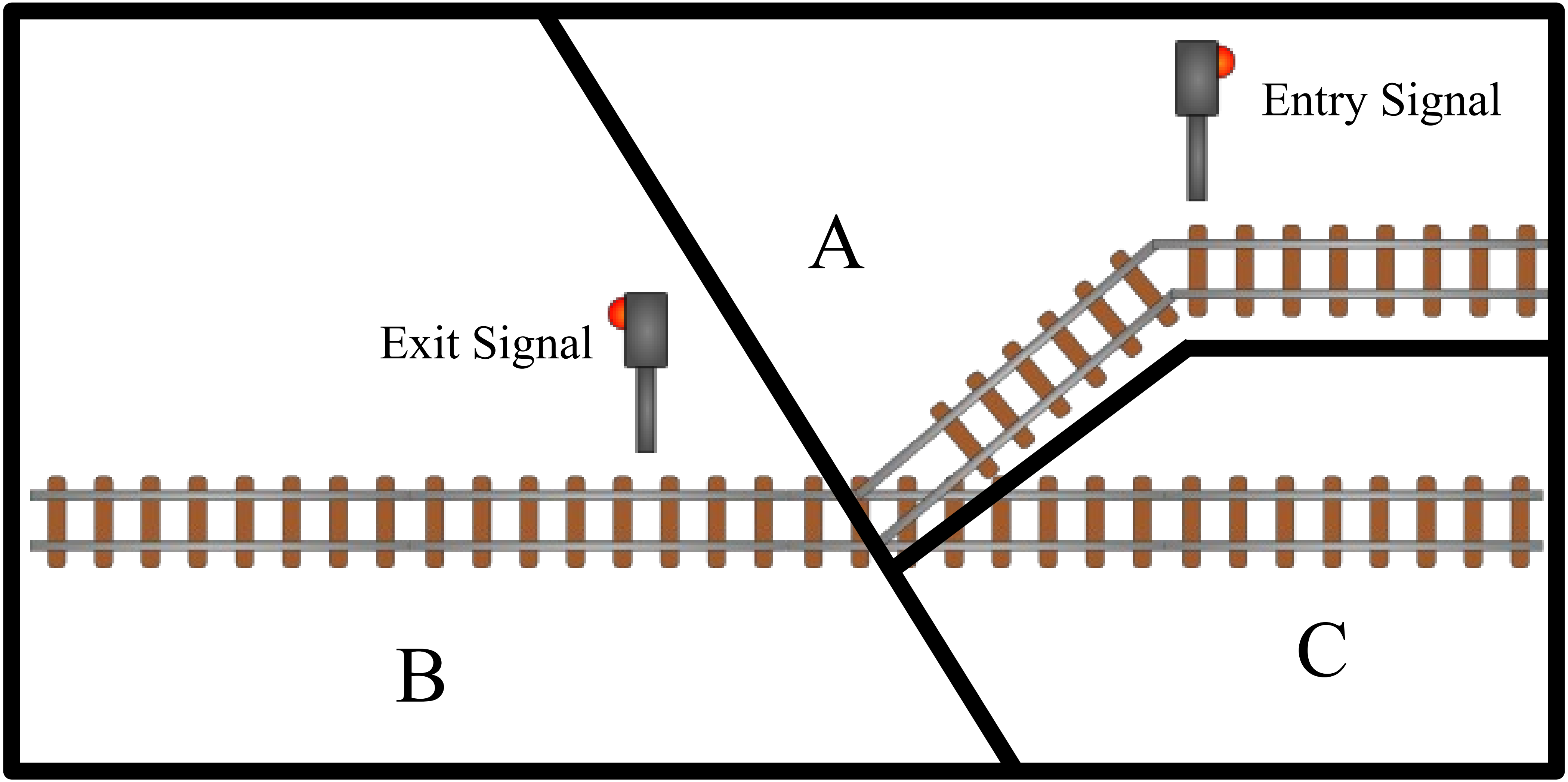


Train



Train Track

Train Track



Ins and Outs

- Ins
  - on_A : Is there a train on track segment A?
  - on_B : Is there a train on track segment B?
  - on_C : Is there a train on track segment C?
  - ack_AB : Is track in the AB position?
  - ack_BC : Is track in the BC position?
- Outs
  - grant_access : Should access light be green?
  - grant_exit : Should exit light be green?
  - do_AB : Should track switch in the AB position?
  - do_BC : Should track switch in the BC position?

Lustre Model

```
node UMS(on_A,on_B,on_C,ack_AB,ack_BC: bool)
     returns (grant_access,grant_exit,
      do_AB,do_BC: bool);
var empty_section, only_on_B: bool;
let
   grant_access = empty_section and ack_AB;
   grant_exit = only_on_B and ack_BC;
   do_AB = not ack_AB and empty_section;
   do_BC = not ack_BC and only_on_B;
   empty_section = not(on_A or on_B or on_C);
   only_on_B = on_B and not(on_A or on_C);
tel
```

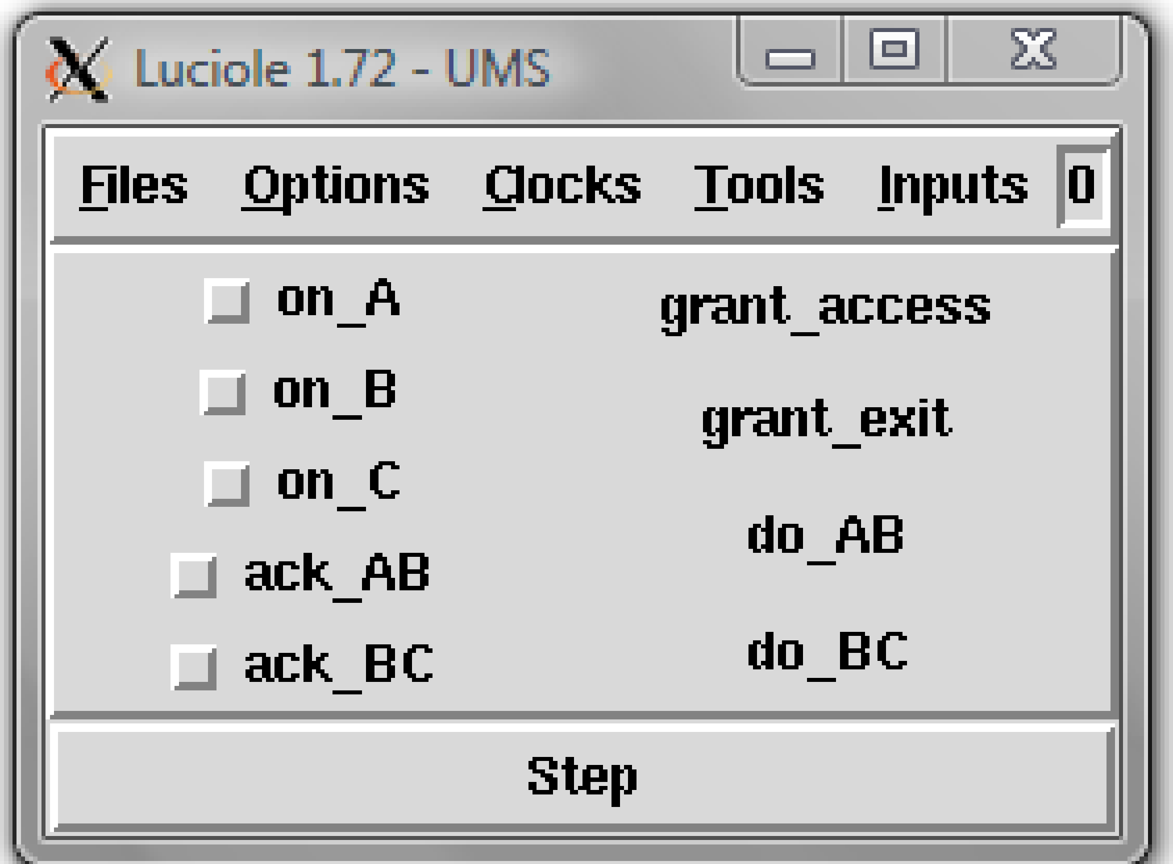
Verification Statements

```
node UMS_verif(on_A,on_B,on_C,
       ack_AB,ack_BC: bool)
     returns(property: bool);
var
   grant_access,grant_exit: bool;
   do_AB,do_BC: bool;
   no_collision,exclusive_req: bool;
   no_derail_AB,no_derail_BC: bool;
   empty_section, only_on_B: bool;
let
   empty_section = not(on_A or on_B or on_C);
   only_on_B = on_B and not(on_A or on_C);

   -- ASSERTIONS
   assert not(ack_AB and ack_BC);
   assert true ->
    always_from_to(ack_AB,ack_AB,do_BC);
   assert true ->
    always_from_to(ack_BC,ack_BC,do_AB);
   assert empty_section -> true;
   assert true -> implies(edge(not
    empty_section), pre grant_access);
   assert true -> implies(edge(on_C), pre
    grant_exit);
   assert true -> implies(edge(not on_A),on_B);
   assert true -> implies(edge(not on_B), on_A
    or on_C);
   -- UMS CALL
   (grant_access,grant_exit,do_AB,do_BC) =
    UMS(on_A,on_B,on_C,ack_AB,ack_BC);
   -- PROPERTIES
   no_collision =
    implies(grant_access,empty_section);
   exclusive_req = not(do_AB and do_BC);
   no_derail_AB = always_from_to(ack_AB,
    grant_access, only_on_B);
   no_derail_BC = always_from_to(ack_BC,
    grant_exit, empty_section);
   property = no_collision and exclusive_req and
    no_derail_AB and no_derail_BC;

tel
```

Simulation


Luciole 1.72 - UMS
Files
Options
Clocks
Tools
Inputs
0
on_A
on_B
on_C
ack_AB
ack_BC
grant_access
grant_exit
do_AB
do_BC
Step

Implementation

Esterel

- First created in 1982
- Commercially transformed in 1999
- Last open version of Esterel is 5.92, and was released in 2000
- A commercial version of Esterel is available from Esterel Technologies
- The Open Esterel tools were mostly developed by the Inria research center.

Gérard Berry;The Foundations of Esterel; To appear in Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, editors, MIT Press, 1998

Esterel Website: http://www-sop.inria.fr/esterel.org/files/

Lustre vs Esterel

Lustre

- Declarative Language
  - Describes **what** is to be computed.

```
Node ABRO(A,B,R: bool)
     returns (O: bool);
let
   O = R and (A or B);
tel
```

Esterel

- Imperative Language
  - Describes **how** this is to be computer.

```
module ABRO:
input A, B, R;
output O;

loop
   [ await A || await B ];
  emit O
each R

end module
```

Conclusion

- Lustre is a Synchronous Reactive Language
- It operates on streams (input and output)
- Models in Lustre can be Verified, Simulated and Execute
- I can draw cool trains
- Similarities between Lustre and Esterel