Esterel

Jon Beeler

CS 156

4/11/00

Embedded Systems (Reactive)

- React to events from the environment
- Input driven
- Often used in critical situations
  - Need clear definition formalisms and formal verification tools
- Real-time systems and control automata
- Deterministic - Same input -> Same output

Synchronous Languages

- Esterel, Lustre, Signal
- Classical languages can not express control theory algorithms
- Synchronous languages are too simple for complex systems
  - Must interact with other languages
- Create the control-dominated (part of) reactive programs

History of Esterel

- Created - early 80s in France
- Language developed together with mathematical semantics.
- Can generate C code, hardware specifications, or finite automata

Language Basics

- boolean, integer, float, double,and string
  - Other types constructed in host language
- Functions: list of args, one return type
- Procedures: list of reference args, list of value args, and return type
- Variables: local with explicit scope

Signals

- Signals instantly broadcast through program
  - Pure: a presence status, either present or absent
  - Value: status + a value of some type
- Declared as either input or output
- `output <name>[:=<init>][:<type>];`
- `output CurrentTime:= Noon : Time;`

Sensors and Output

- Sensors are valued signals without presence information
- `sensor <name> : <type>;`
- Signal or Sensor Output: `emit S;` or `sustain S;`

Relations

- `relation Sig1 # Sig2;`
  - *Sig1* and *Sig2* are incompatible.
- `relation Sig1 => Sig3;`
  - *Sig1* and *Sig3* are synchronous (*Sig1* can be present only if *Sig3* is also present)

Flow Control

- Sequencing Statements: `p; q`
- Concurrency: `p || q`
- Looping:

```
loop                repeat e times
    p                   p
end loop            end repeat
```

- Present Signal Test:
  - present S then `p else q end present`

Temporal Statements

- `await [number] <signal>`
- `abort p when <signal>`
- `loop p each <signal>` or `every <signal> do p`
- `suspend p when <signal>`

Traps

```
trap T, U in
   p
handle T do
   q
handle U do
   r
end trap
```

Modules

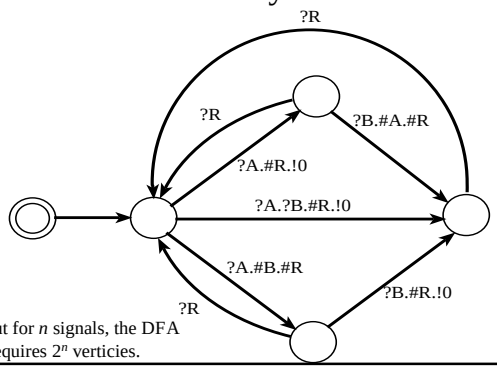
```
module name :
   interface declaration
   statements
end module
```

- can insert a module into another module using: `run <module>`
  `[ type <var_name> \ <local_name>]`

ABR0 Example

- Specification ABR0:
  - Emit an output 0 as soon as two inputs A and B have occurred.
  - Reset this behavior each time the input R occurs.

ABR0 Mealy Machine



But for *n* signals, the DFA Requires 2^n verticies.

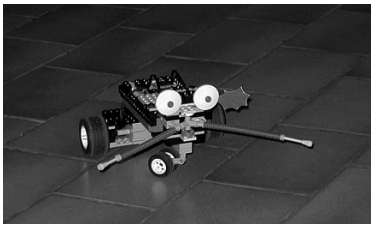
ABR0 in Esterel

```
Module ABR0
input A, B, R;
output 0;
loop
   [ await A || await B ];
   emit 0
each R
end module
```

Another Example: Speedometer

```
module SPEED:
input Centimeter, Second;
relation Centimeter # Second;
output Speed : integer;
loop
     var Distance := 0 : integer in
          abort
               every Centimeter do
                    Distance := Distance + 1
               end every
          when Second do
               emit Speed(Distance)
          end abort
     end var
end loop
end module
```

Applications: Lego Robots



Esterel can be used to generate C code to be used in LegOS to control Lego robots.

```
module lego1 :
input TOUCH_1, TOUCH_3;
output  MOTOR_A_SPEED(integer), MOTOR_C_SPEED(integer),
        MOTOR_A_DIR(integer), MOTOR_C_DIR(integer), CPUTS(string);
relation TOUCH_1 # TOUCH_3;
constant MOTOR_FWD, MOTOR_REV, MAX_SPEED : integer;
var t : integer in
    emit MOTOR_A_SPEED(MAX_SPEED/2); emit MOTOR_C_SPEED(MAX_SPEED/2);
    loop
        emit MOTOR_A_DIR(MOTOR_FWD); emit MOTOR_C_DIR(MOTOR_FWD);
        emit CPUTS("fwd");
        await [TOUCH_1 or TOUCH_3];
        present TOUCH_1   then   t:=1;
                          else   t:=3;
        end present;
        emit MOTOR_A_DIR(MOTOR_REV);  emit MOTOR_C_DIR(MOTOR_REV);
        emit CPUTS("rev");
        await 100 tick;
        if t=1 then
            emit MOTOR_A_DIR(MOTOR_FWD); emit CPUTS("right");
        else
            emit MOTOR_C_DIR(MOTOR_FWD); emit CPUTS("left");
        end if;
        await 100 tick;
    end loop
end var.
```

Real Application: Validating DSPs at TI

- Largest part of DSP development is validation
- Esterel used to create a software model of DSPs
- Esterel has tools to analyze state coverage
  - Ideal for testing all cases
- Can automatically generate test patterns to test missing states.