

Compilation of Lustre

Pascal Raymond

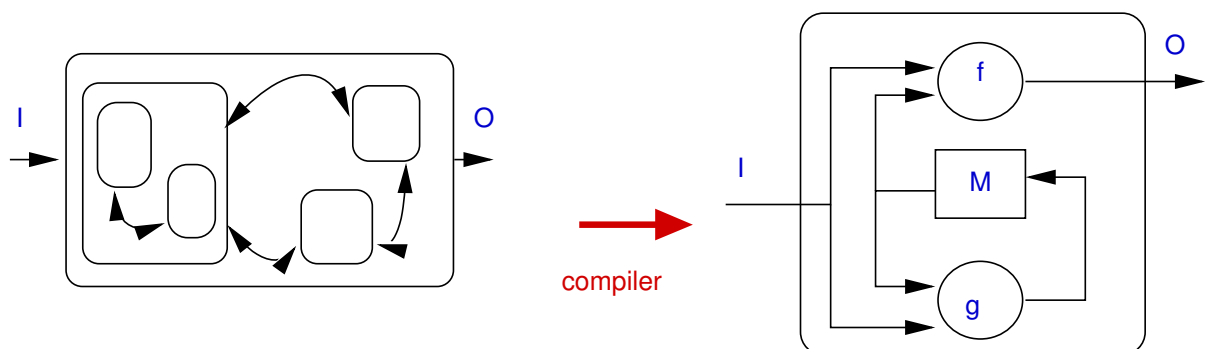
Verimag-CNRS

MOSIG - Embedded Systems

Compilation of synchronous programs _____

General problem

Transform a (hierarchic) parallel program into a (simple) sequential program.



Whole implementation of a reactive program P

```
var I, O, M;  
M := m0; proc P_step() ...;  
foreach step do  
  read(I);  
  P_step(); // combines: O := f(M, I); M := g(M, I);  
  write(O);  
end foreach
```

Job of the compiler

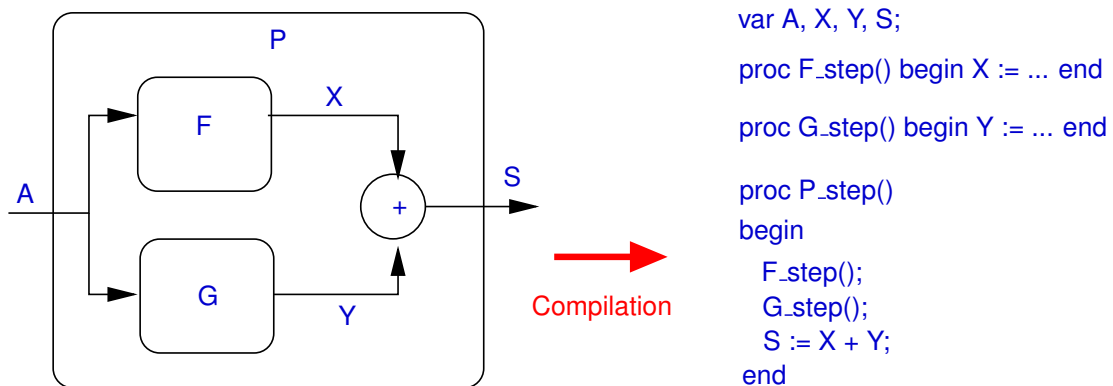
- Find the memory M and its initial value m0
- Build the core of the loop (the P_step procedure)
- As far as possible, generate efficient code

Compilation of synchronous programs _____ 2/??

Modular compilation problem _____

The "obvious" way of compiling

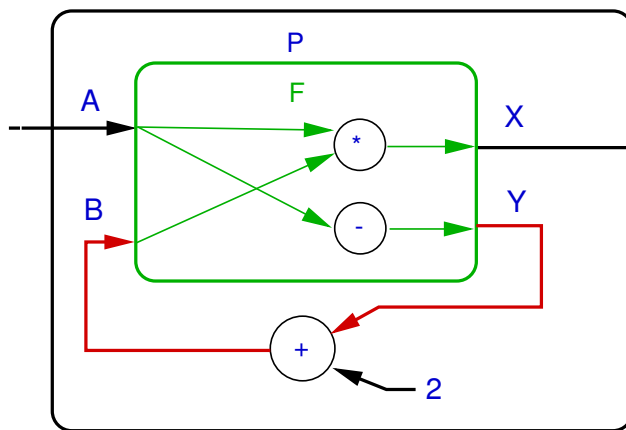
A Lustre node \rightarrow a step procedure.



Modular compilation problem _____ 3/??

Problem

What about feed-back loops ?



combinational loop ?

not when inlined ! \Rightarrow

$$X = A * (2 - A)$$

- The program "is" correct (in a "parallel" world),
- but no `F_step` procedure can work!

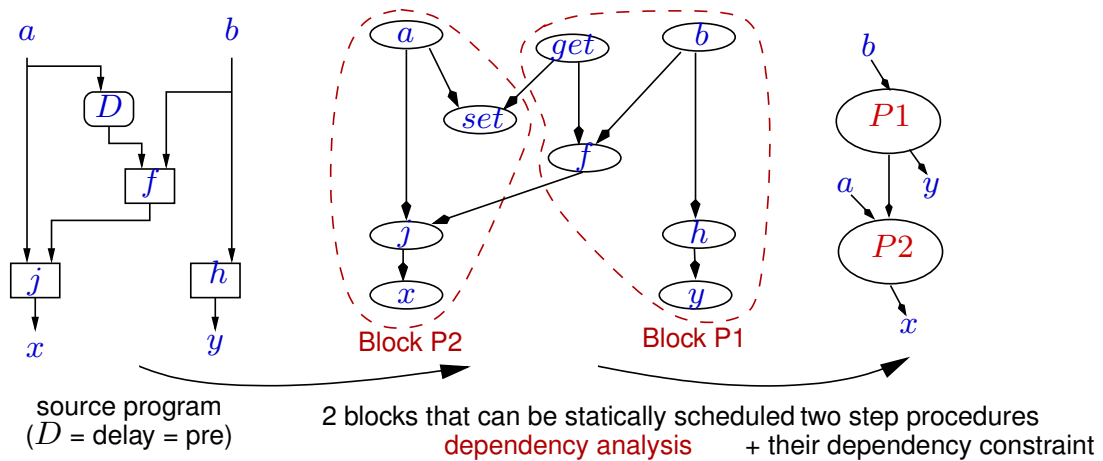
Modular compilation problem _____ 4/??

Solution(s)

- Lustre (academic): expansion (i.e. inlining) of node calls
 - \hookrightarrow Strictly compliant with the principle of substitution.
 - \hookrightarrow Forbids modular compilation.
- Scade: feedback loops (without `pre`) are forbidden.
 - \hookrightarrow Reject correct parallel programs.
 - \hookrightarrow Allow modular compilation.
 - \hookrightarrow Reasonable choice in a industrial framework.
- Compilation into ordered blocks aka Modular Static Scheduling
 - \hookrightarrow Intermediate solution
 - \hookrightarrow Split the step into a minimal set of (sequential) blocks,
 - \hookrightarrow Only expand this simplified structure.

Modular compilation problem _____ 5/??

An example of Modular Static Scheduling



- Interesting theoretical result.
- Not (yet ?) used in industry.

Modular compilation problem _____ 6/??

Compilation of Lustre _____

Example : a filtered counter

- count rising edges of X (F),
- reset with a delay (R).

```

node CptF(X, reset: bool) returns (cpt: int);
var F, R : bool;
let
  cpt = if R then 0
        else if F then pre cpt + 1
        else pre cpt;
  R = true -> pre reset;
  F = X -> (X and not pre X);
tel
    
```

Compilation of Lustre _____ 7/??

Simple loop compilation

Intuitively, do what is necessary to make definitions equivalent to assignments, i.e.:

- translate classical operators (trivial),
- replace `pre`'s and `->`'s with memory constructs,
- sequentialize according to data-dependencies (i.e. static scheduling).

Identify the memory

- Introduce a explicit variable for each `pre`:

```
pcpt = pre cpt;  
preset = pre reset;  
pX = pre X;
```

- Introduce a special memory

```
init = true -> false;
```

and replace each:

```
x -> y
```

with

```
if init then x else y
```

New version of the Lustre program

```
cpt = if R then 0
      else if F then pcpt + 1
      else pcpt;
R = if init then true else preset;
F = if init then X else (X and not pX);
pcpt = pre cpt;
preset = pre reset;
pX = pre X;
init = true -> false;
```

Sequentialization

Must take into account:

- Instantaneous dependences between values,
 - ↪ an (partial) order MUST exist (no combinational loop),
example: R before cpt and F before cpt
 - ↪ chose a compatible complete order (schedule),
example R, then F then cpt.
- Memorisations
 - ↪ Must be done at the end of the step, in any order.

Simple loop implementation (C-like code)

- Arithmetic and logic are translated "as is"
(ex. **and** becomes **&&**, **if..then..else** becomes **..?..:..**)
- **pre**'s are replaced with memories
- **->**'s are replaced with **init?...:...**
- Inputs/outputs are stored in global variables (for instance)

Simple loop implementation (C-like code)

```
int cpt; bool X, reset; /* I/O global vars */
int pcpt; bool pX, preset; /* non initialized memories */
bool init = true; /* the only necessary initialization */
void CptFiltre_step() {
    bool R, F; /* local vars */
    R = init ? true : preset;
    F = init ? X : (X && ! pX);
    cpt = R ? 0 : F ? pcpt + 1 : pcpt;
    pcpt = cpt; pX = X; preset = reset;
    init = false;
}
```

Optimizations

- Control structure: ? becomes **if**
- Factorize conditions
- Eliminate useless local vars

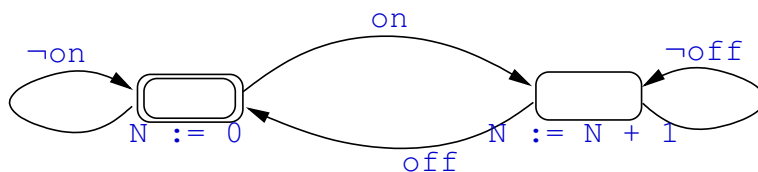
```
if (init) {  
    cpt = 0;  
    init = false;  
} else {  
    F = (X && ! pX);  
    cpt = preset ? 0 : F ? (pcpt+1) : pcpt;  
}  
pcpt = cpt; pX = X; preset = reset;
```

Compilation of Lustre _____ 14/??

Compilation into automaton _____

Idea

The following reactive automaton:



is exactly equivalent to a Lustre program:

```
node Chrono(on, off : bool) returns (N : int);  
var R : bool;  
let  
    R = false -> pre(if R then not off else on);  
    N = if R then (pre N + 1) else 0;  
tel
```

Problem: how to build the automaton from the Lustre code ?

Compilation into automaton _____ 15/??

Goal

- Automatically build an automaton equivalent to a Lustre program

How ?

- Idea: an (explicit) state \Leftrightarrow a valuation of the memory
- N.B. finite number of states \Rightarrow finite memory (e.g Boolean)

Example of CptF

- **S1** = initial state = "**init** true, all other undefined"
- simplified code : **cpt = 0**
- integer memorization: still the same
- Boolean memorization: state transition

Compilation into automaton _____ 16/??

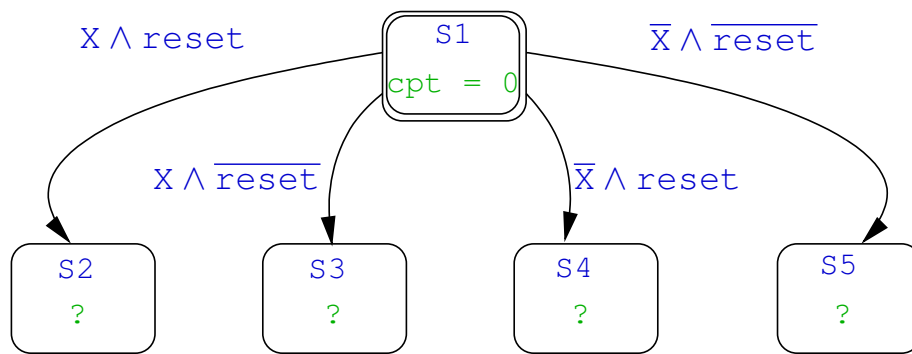
Transitions

- State **S1** (initial):
init = false; pX = X; preset = reset;

Depending on the values of x and $reset$, 4 next states:

- $x \wedge reset \rightarrow S2 \equiv \overline{init} \wedge pX \wedge preset$
- $x \wedge \overline{reset} \rightarrow S3 \equiv \overline{init} \wedge pX \wedge \overline{preset}$
- $\bar{x} \wedge reset \rightarrow S4 \equiv \overline{init} \wedge \overline{pX} \wedge preset$
- $\bar{x} \wedge \overline{reset} \rightarrow S5 \equiv \overline{init} \wedge \overline{pX} \wedge \overline{preset}$

Compilation into automaton _____ 17/??



- Code of the other states:

→ $S2 \rightarrow \text{cpt} = 0$

→ $S3 \rightarrow \text{cpt} = \text{pcpt}$

→ $S4 \rightarrow \text{cpt} = 0$

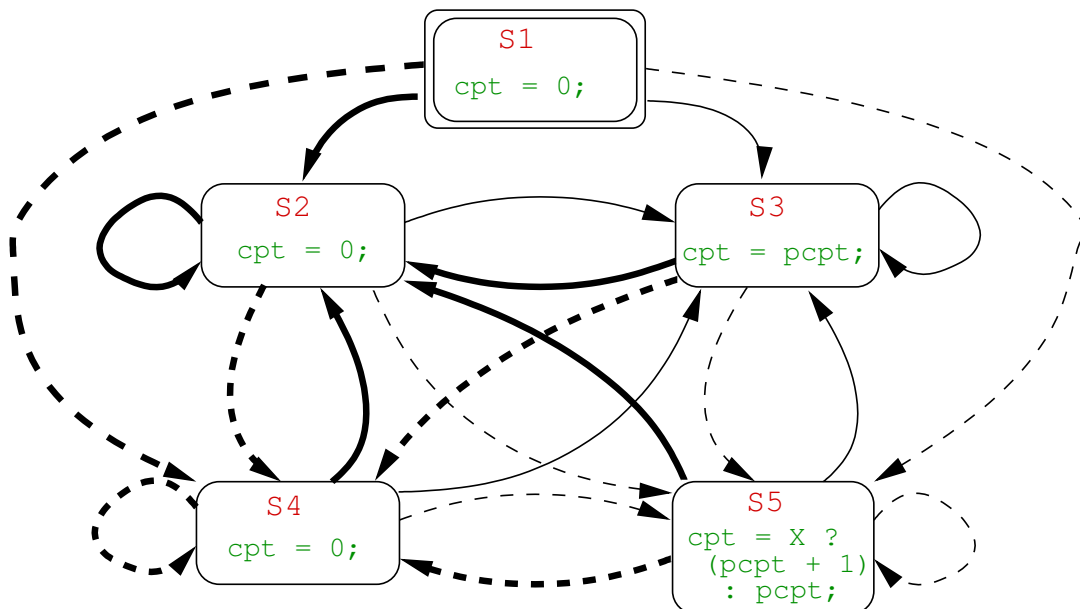
→ $S5 \rightarrow F = X, \text{cpt} = X ? (\text{pcpt} + 1) : \text{pcpt}$

- Transitions of the other states:

→ same than S1 (only depend on inputs)

Compilation into automaton _____ 18/??

Finally ...



⇒ problem: size!

Compilation into automaton _____ 19/??

Remarks on the size

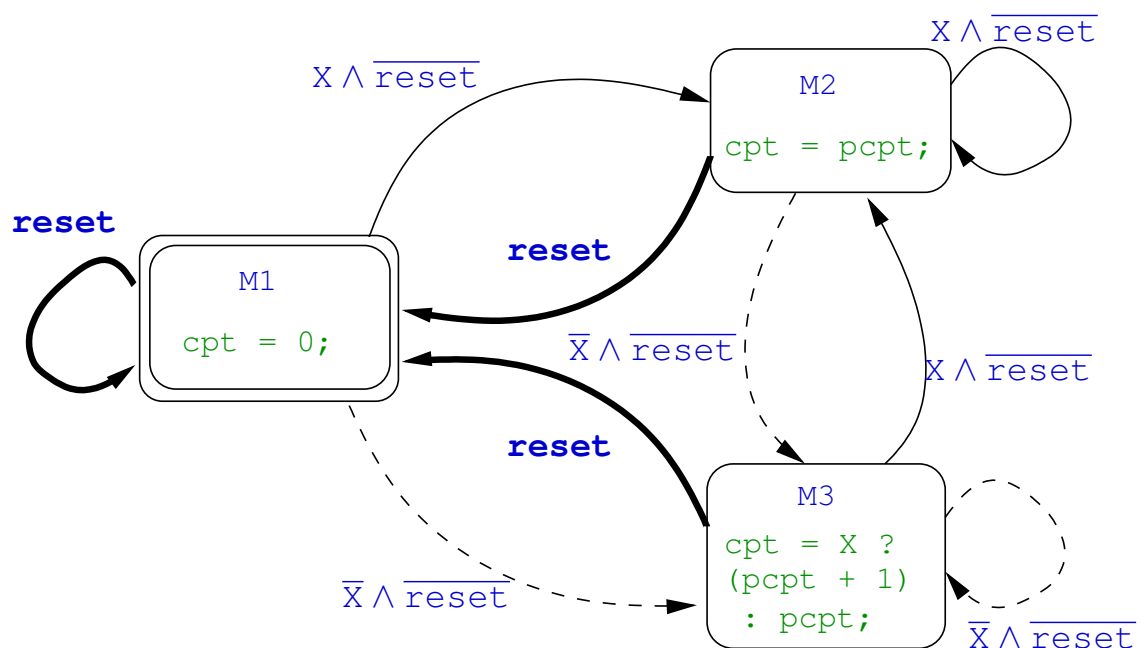
- n memories \Leftrightarrow (worst case) 2^n states, 2^{2n} transitions
 \Rightarrow *Combinatorial explosion*

But not always:

- Unreachable states
 \hookrightarrow Example : $(\text{pre } X, \text{pre}(X \text{ or } Y)) \Rightarrow$ “only” 3 states
 \hookrightarrow Counter-example : CptF !
- State equivalence
 \hookrightarrow Example CptF : $S1, S2$ et $S4$ “are doing the same thing”
 \Rightarrow *Importance of producing a minimal automaton*

Compilation into automaton _____ 20/??

Minimal automaton of CptF



Compilation into automaton _____ 21/??

Implementation en C

With a switch (for instance):

```
typedef enum {M1, M2, M3} TState;
TState state = M1;
void CptFiltre_step() {
    switch (state) {
        case M1:  cpt = 0; break;
        case M2:  cpt = pcpt; break;
        case M3:  cpt = X? (pcpt + 1) : pcpt; break;
    }
    pcpt = cpt;
    if (reset) state = M1;
    else if (X) state = M2;
    else state = M3;
}
```

Compilation into automaton _____ 22/??

Simple loop or automaton ? _____

Automate

- Optimal in computation time
- Possible huge size

Simple loop

- Less slower
- Linear size

⇒ Only “reasonable” solution fram an industrial point of view

Automaton, what for?

- Not reasonable for code generation, but ...
- Precious for *reasoning* about programs, i.e. for validation.

Simple loop or automaton ? _____ 23/??

The stopwatch

```
node Stopwatch(on_off, reset, freeze: bool)
returns (time: int);
var running, freezed: bool; cpt: int;
let
  running = Switch(on_off, on_off);
  freezed = Switch(
    freeze and running,
    freeze or on_off);
  cpt = Count(reset and not running, running);
  time = if freezed then (0 -> pre time) else cpt;
tel
```

- expand, identify memory, sequentialize ...
- automaton ...