The SCDC processor, illustrated on a Syrf case study

A. Ressouche R. de Simone

November 24, 1997

1 Introduction

Synchronous reactive languages such as *Lustre* , *Signal* and *Esterel* differ in style, and so appear as complementary in large designs. Combination of formalisms then eases application developments by allowing the designer to choice the language best suited to each part.

The data flow formalisms *Lustre* and *Signal* model reactive systems with flow network of operators acting in parallel. *Systems of dynamical equations* describe the semantic behavior of such networks. Variables and operators of these equations may belong to various types. An exchange file format DC has been defined for representation of systems of dynamical equations. It is a compilation target for *Lustre* and *Signal*.

Esterel is a control-dominated, structured imperative language. It has a "circuit" semantics, which translates a program into a constructive *boolean equation system*, including additional boolean variables to trigger external data operations. In this approach control flow and data operations are strongly separated, reflecting the nature of the language. An exchange file format SC has been defined for representation of boolean equation systems with external data operations. It is a compilation target for *Esterel*.

The difference between DC and SC formats thus relies basically on the degree of separation between (boolean) control flow and data operations (of any types), while the general principles of definitional equations, synchrony and memory registers are a common framework.

Combination of data flow and imperative formalisms may be achieved on the level of DC and SC intermediate formats. A SCDC [2] translator has been defined and implemented to embed *Esterel* modules into data flow programs. We illustrate its use on a mixed *Esterel(SC)/Lustre(DC)* version of the SAAB aircraft climatic chamber case study proposed in *Syrf*. As for today there does not exists a linker of DC files, so for technical reasons we used a simple decompiler back to *Lustre* to combine program fragments. In this example the *Esterel* code deals with mode selection and reconfiguration, while a purely combinatorial *Lustre* program synthesize the test values from actual sensor input, to guide the mode automaton.

2 The SCDC translator : short description

The SC and DC intermediate formats are structured in tables. They share most tables related to data : type table, constant table, function table, prodedure table. Although similar, the main behavioural tables consist of definitional equations, purely boolean in SC, allowing variable and operators of any declared types in DC.

Registers are also boolean latches in SC, and of any declared type in DC (called "memo" there).

In addition SC contains an *action* table, recording the data operations triggered by boolean wires in the behavioural part. The main purpose of the SCDC translator is to reintroduce the action operations into the behavioural part, in a general, multitype equation system.

The DC format allows other features which we shall not use here as target from SC.

In the last section we plan for future work for the remaining of SYRF. This concerns translation back from DC to SC (from the target subdomain, then from a larger class), and actual combination of files through linking and compositional orerations. An even further topic concerns information retrieval at source code level (back from translation) for compilation messages.

3 The SAAB Climatic chamber example

The purpose is to keep ventilation and temperature inside proper boundaries. The program consists of several regulation controlers, and a control panel to record the actual parameters and enter commands for requested values.

The system is started with an *on-off* switch. It can reach *wait, sol, work, block* modes, according to various tests in time. The truth values of tests (mainly thresholds) must be computed from successive values provided by sensors or read from the panel. The mode automaton and its behavioural switches form the control part (suitable to Esterel), while the dataflow computation of condition values from data paramaters remains purely combinational and suits better Lustre.

The desired mode automaton is shown is figure1. It relies on boolean conditions defined below. The data needed to update flow and temperature are:

- 3 fixed constants:
  1. *deltatemp*: temperature variation allowed.
  2. *solution_time*: maximal time to stay in sol mode
  3. *tgrad*: temperature limit
- and 3 variable parameters:
  1. *chamtact*: actual chamber temperature
  2. *panreq*: requested chamber temperature (computed from operator request and temperature upper and lower bounds)
  3. *con_onoff*: on-off switch

Now, the conditions useful to determine (and possibly change) the next mode are boolean combinations of the following boolean parameters:

- *In2DeltaTemp* = | chamtact - panreq| $\leq$ 2 $\star$ deltatemp
- *OutDeltaTemp* = | chamtact - panreq| $>$ deltatemp
- *TempLess5Tgrad* = true while the two last values of chamtact differ from less than 5 $\star$ tgrad
- *newpanreq* = true when a new panreq occurs
- *ten_solution_time_con_onoff*: = true each 10$\star$ solution_time of con_off

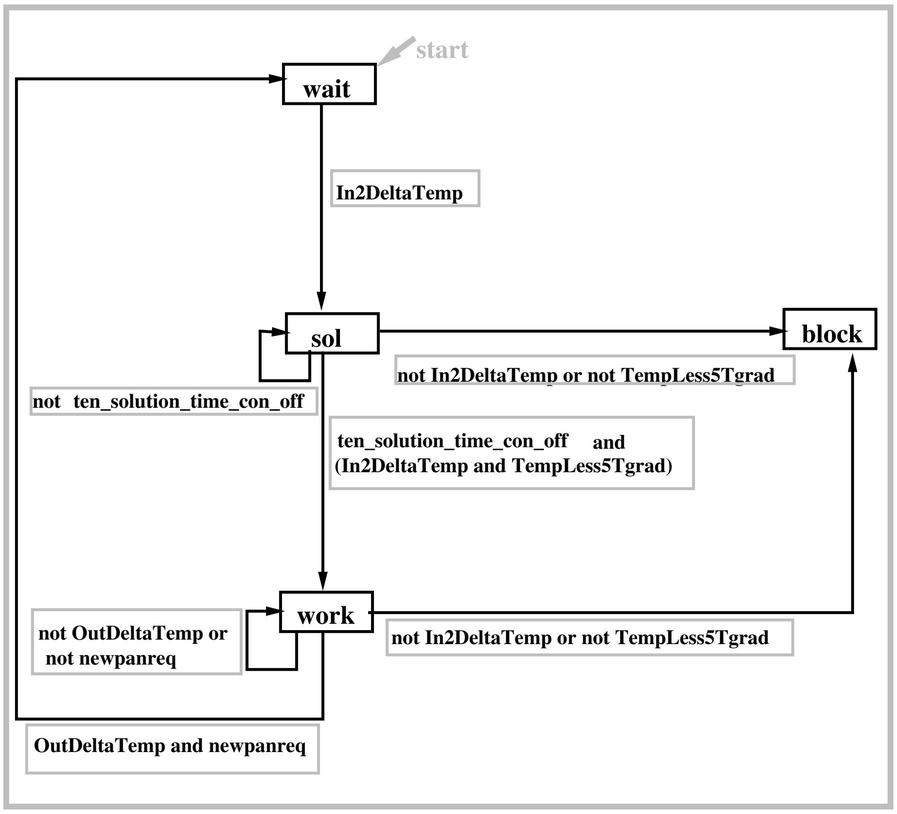

Figure 1: climchamber modes computation

3.1 Results (implementation)

The Esterel program implementing the mode automaton (on boolean condition events) is provided in annex. It can be compiled into an explicit finite state machine, and visualised using AUTOGRAPH [1] as in figure 2.

The Lustre program computing these conditions is also displayed in annex.

We produced the single compound program in the following steps: first the Esterel description was compiled in SC format, then translated into DC using our translator. Then the DC program was decompiled into a subset of Lustre, and included as a (sub)node to Lustre, together with the original Lustre part. In the future linking should be available at DC level.

4 Future work

Of course the case study was rather small, but typical of a large class of applications, where combinatorial data-flow operations compute threshold conditions to switch modes of operations. When the modes becomes more complex, structural programming proves useful.

The actual translation needs to be complemented in several ways, to secure true combination of formalisms :

- *reverse translation* The translation back from DC to SC appears to be more complex, since now every data operations split into a data operation and a causal dependency affecting control.

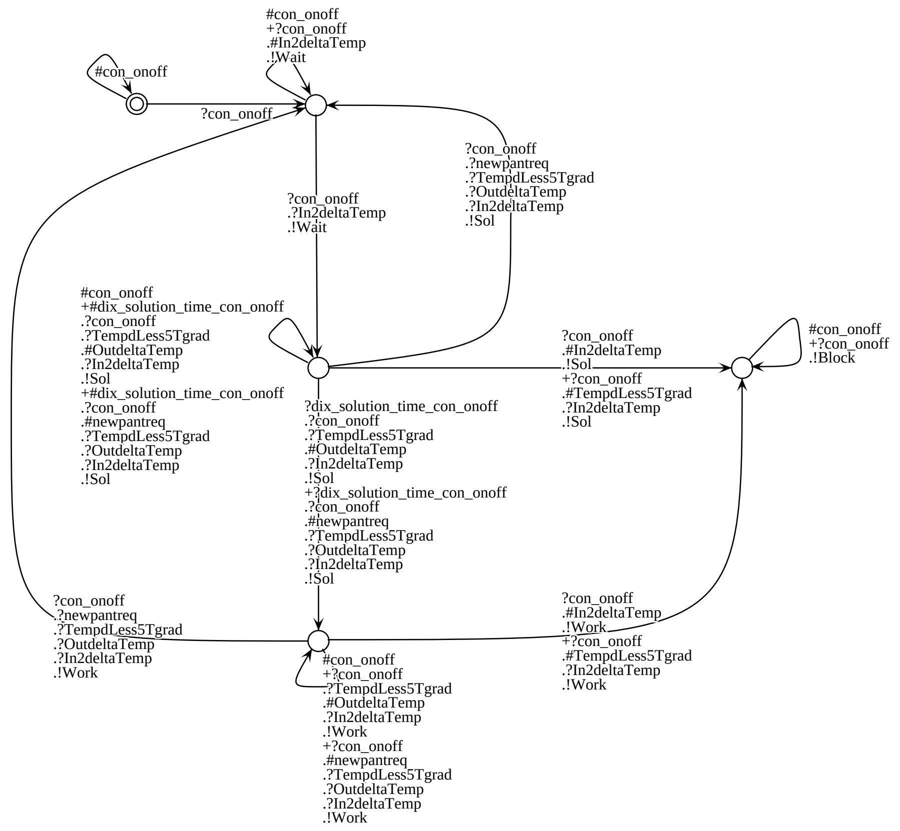

Figure 2: esterel part behavior

- *optimisation*: purely boolean equation systems correspond to *netlists* in hardware terminology, and there is a rich body of optimisation techniques borrowed from electronic CAD systems which canalready be applied on (single) SC files. There is very little in comparison for DC-like formalisms.

- *compositionality*: linking programs from different origins into SC or DC requires to enhance the intermediate formats to allow contexts descriptions, with a "hole" for missing subparts; in equation systems this often simply means concatenating equations, in a parallel product, with possibly variable renaming. An important point here is that components could be replaced with optimised forms, with identical behaviours.

- *verification*: model-checking is current practice in boolean equation systems, where only finite domains are concerned, and symbolic representation techniques such as BDD apply. But in real cases often the non-finite domains (integers, rationals) have an impact on reachable state spaces. How to reduce this or best deal with it is still a matter of research, while this is certainly no unique global solution to the problem. Less expressivity means greater automation and decision procedures.

- *source recovery*: simulators, compilers, optimisers, and verifiers all need to provide user with informations in terms of source programs, across translations. This requires book-keeping in earlier phases.

Some of these topics shall be goals for the rest of *Syrf*'s duration, starting with the reverse translation from DC to SC.

5 Annex

```
module compute_mode:

input con_onoff, dix_solution_time_con_onoff;
output Wait, Sol, Work, Block;  %modes

input  In2deltaTemp, OutdeltaTemp, TempdLess5Tgrad, newpantreq;

suspend
[
   trap block in
      await tick;
      loop                      %wait state
 weak abort
    sustain Wait
 when immediate In2deltaTemp;
 weak abort            %sol state
    weak abort
       weak abort
  await tick;
  sustain Sol
       when dix_solution_time_con_onoff
                                       %or is it 60 to turn sec to min
   do await tick;   %work state
      sustain Work
       end weak abort
    when [not (In2deltaTemp and TempdLess5Tgrad)]
do exit block
    end weak abort
 when [OutdeltaTemp and newpantreq]
     do  await tick
 end weak abort
      end loop
   handle block do
      await tick; %block state
      sustain Block

   end trap
]
when immediate [not con_onoff]
end module



include "mode.strl.lus"

const
  deltatemp     = 50;
  tgrad         = 40;
  solution_time = 1;
```

```
node modeselect (con_onoff: bool; pantreq: int; chamtact: int)
returns (Wait, Sol, Work, Block,
        In2deltaTemp, OutdeltaTemp, TempLess5Tgrad: bool);

     var temp: int^6;
         newpantreq : bool;
         ret, halt : bool;

let
     newpantreq = false -> if (pre(pantreq) = pantreq) then false
           else true;
     (temp[0..4], temp[5]) =
       (chamtact^5, chamtact)->(pre(temp[1..5]), chamtact);
     TempLess5Tgrad = false-> if (temp[5]-temp[0] < 5*tgrad) then true
                              else false;
     In2deltaTemp = true ->
                    if ((pantreq-chamtact <= 2*deltatemp) and
                        (chamtact-pantreq <= 2*deltatemp)) then true
                    else false;
     OutdeltaTemp = false ->
                    if ((pantreq-chamtact > deltatemp) or
                        (pantreq-chamtact < deltatemp)) then true
                    else false;
     (Wait,Sol,Work,Block,ret,halt) =
        compute_mode(con_onoff, timer(con_onoff),
                     In2deltaTemp, OutdeltaTemp,
                     TempLess5Tgrad, newpantreq);

tel -- modeselect

node timer (con_onoff : bool)
returns (dix_solution_time_con_onoff : bool);

var n : int;
let
     n = 0 ->
         if con_onoff then
           if (pre(n)+1=10*solution_time) then 0 else pre(n)+1
         else pre(n);
    dix_solution_time_con_onoff = false ->
                                  con_onoff and (n=10*solution_time);
tel --timer
```

References

[1] *Meije verification tools* http://www.inria.fr/meije/verification

[2] A. Ressouche *The Scdc processor* Internal Meije report