Annex 7.3.A
Transformations with the "big L approach"

Simin Nadjm-Tehrani
Dept. of Computer & Information Science, Linköping University,
S-581 83 Linköping, Sweden, e-mail: simin@ida.liu.se
tel: +46-13-282411, fax: +46-13-282666

1 Introduction

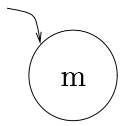
Proving bounded response properties of a synchronous controller relies on two factors:

1. How many "steps" it takes to reach a particular discrete state.

2. How variations in the duration of the step affects the real-time response.

Of course, the first question can not be answered by considering the controller alone. The environment (plant) behaviour is a major part of that. In this annex we describe the transformations needed before theroem proving can be applied on the closed loop system for confirming the answer to the first question. This is done on "unfolded" models of the system, which means that the transition function for the closed loop system has to be specified in terms of the langauge of a theorem prover. In the case of NP-Tools, this is propositional logic and integer arithmetic.

The second question is typically assumed to have a well-defined answer. The software engineer assumes that the 'right' period will be delivered by the control engineer. In the Saab case study we see that this is not necessarily to be taken for granted (see section 2.1). Here, the physical modelling performed in the early stages of the case study (see Annex 7.1A) are reconsidered for determining a reasonable step size, in turn affecting the number of 'unfoldings' mentioned above.



$$\dot{x} = au_1x + bu_2 + cu_1u_3$$

Figure 1: The continuous model of the plant based on physical assumptions. Here x is the chamber temperature, u_1 the airflow as an input to the temperature equation, u_2 the controller signal based on voltage, and u_3 the temperature of incoming air.

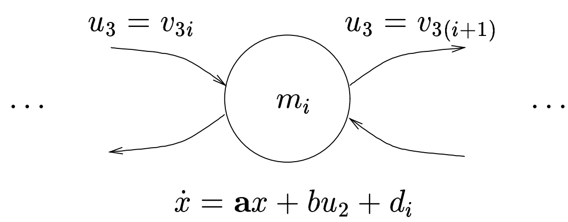


Figure 2: The multi-mode model of the plant, giving the chamber temperature in terms of a continuous control signal in each mode.

2 Transformations on the plant model

In this annex, we show how restrictions of the plant model are used, via a number of transformations, to derive a flat Lustre model for the closed loop system. The approach is illustrated with the climatic chamber case study described in Annex 7.1.A (to deliverable 7.1) in the SYRF project.

A model in differential and algebraic equations (DAE) does not directly lend itself for plugging into the closed loop model in the theorem prover. First, the plant model is "simplified" by restricting *s*ome inputs to piece-wise constant signals – replacing the non-linear continuous dynamics with discrete modes and linear dynamics in each mode. Then, using the knowledge that the remaining input signals are control signals issued by a synchronous (and sampled) controller, each linear differential equation in each mode is transformed to difference equations (Lustre programs with real variables).

Finally, using the same scheme applied for compilation of mode-automata till flat Lustre, obtain a multi-mode Lustre model of the plant. This model is now equivalent to the original model modulo restrictions on (physical) input signals. One last step is to approximate the variables in the Lustre program from reals to integer.

The simplest model of the plant for the climatic chamber can be considered as a single mode system with one differential equation for the chamber temperature. This model is depicted in Figure 1.

Now, using two of the assumptions presented in deliverable 7.2, this model can be reduced to a multi-mode hybrid transition system whereby the state in each mode is dependent on the incoming control signal and previous state only. The change in state is moreover linear in the incoming control signal. This system is depicted in Figure 2.

In order to obtain the relevant constants in the various modes of the model we have utilised the information in the simulation models of the system.

Using the analysis capabilities in a tool like Matlab or MatrixX, we can go from simulation models to discrete time versions of the plant, for different sampling intervals. This makes it possible to write a Lustre node which is parametrised with the sampling interval. It is conceivable that, in the future, a flat Lustre program can be automatically obtained by translators from mode-automata. In the third year of the project we experimented with a manual translation.

2.1 Choice of T

The above simulation models were even used to derive a valid samplig interval given the dynamics of the system. Two example simulations are presented in Figure 3. These models showed, for example, that the sampling interval implicit in the Lustre program was unnecessarily short. This could be deduced by the use of counters before determining a mode change in the controller, which indicated sampling at 10Hz – whereas 0.01 Hz to 1 Hz would be more reasonable in this case.

This exercise illustrates an observation which is worth emphasising. First, the answer to question 2 also affects the models used to confirm the answer in question 1. Secondly, choosing a conservatively short sampling interval is not always a good idea (contrary to what some software engineers assume). As well as well-known adverse effects on the achieved stability in the system (detectable by control engineers), too short a sampling interval means a larger number of steps to use in the bounded response proof, and too large a number of unfoldings.

In our attempts at proving the bounded response property we had to unfold the system transition function for 30 steps (performed automatically by the tool Lucifer). Had it been te original step length, the system would have to be unfolded 30000 times! The change in the length of the step meant that we had to modify part of the controller code in which the basic clock of the system was calculated. It would seem that having this information at the time of coding is beneficial – at least, for making the places where such modifications are later made clearly labelled.

3 Transformation on the controller model

The Lustre code for the climatic chamber was thus processed by the tool Lucifer[1] in order to get an NP-Tools model. This process identified a number of points for which Lucifer can be further improved, though most of the issues involved are at the level of syntax, and are already dealt with in the current preprocessor of the Lustre compiler. These points are listed here in order to facilitate the decision: is it more efficient to directly change the implementation of Lucifer, or is integration of Lucifer with the Lustre compiler preprocessor a better way of accepting larger subsets of Lustre programs?

The following list describes the types of syntactical rejections when processing the Lustre program by Lucifer. Just in case it is not obvious, we also present the way the original Lustre program was changed in order to get it through Lucifer:

Input arguments as tuples

Tuples were not accepted as input arguments. We had to create new identifiers instead. An example being:

```
node panelin (on_off: bool; opinp: [int,int])
```

The same problem arose when using tuples on the left hand side of an assignment. Interestingly enough, this problem arose non-deterministically. That is, there were cases

[1] Version released April 99.

when similar usage went through the semantic check. This is an example of a case for which the semantic sheck failed:

```
(controlsignals, chambonoff, panop) =
                 controller(on_off,opinp1,opinp2,tchamb,airflow);
```

We had to change the type of the first output of controller from tuple to two scalars (as well as a similar change in another node within the controller). The result which went through was:

```
(voltage,flowcontrolsignal, chambonoff, panop) =
                 controller(on_off,opinp1,opinp2,tchamb,airflow);
```

Parameter passing

Same identifier was not accepted as a formal parameter and a constant. We had to do some variable renamings to get over this. For example:

```
node modeselect (time: int; pantreq: int; chamtact: int; solution_time: int;
                  tgrad: int; deltatemp: int; cononoff1: bool)
```

where `deltatemp` and `tgrad` are constants declared at top level.

Lustre feature not supported

There was one standard construct in Lustre which did not go through the compiler, namely the `current` operation. For example:

```
 flowtime = 0 -> if trans then
                    current((volumedm when trans) div (secflow when trans))
```

This construct was, however, accepted when rewritten as:

```
 flowtime = 0 -> if trans then
    current((volumedm div secflow) when trans)
```

Array types

The rejection of arrays was perhaps the most inconvenient of all rejections.

```
temp : int^6;
```

had to be changed with:

```
temp0,temp1,temp2,temp3,temp4,temp5:int;
```

Type mismatches

This was a consequence of all previous types of anomalies, and arose in several parameter passings or assignments. In other words, when the program code was only partially changed to deal with the above types of rejections, certain remaining type missmatches showed up.

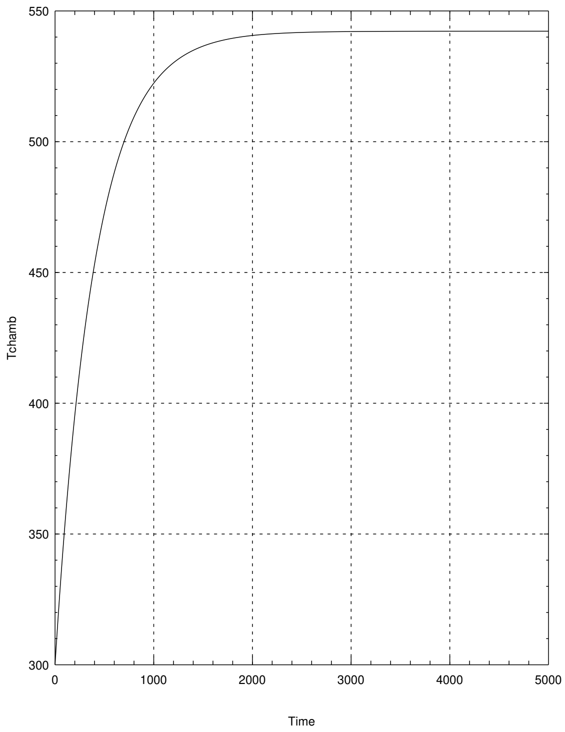
4 The final result

The program eventually accepted by Lucifer was transformed to 25 NP-Tools files corresponding to one for each node in the controller (some several A4 pages long). A positive aspect of the translated model is that it preserves the structure. Infact, in the case of Lustre which is a textual language, it provides an overview. A negative aspect of the translated model was that each node is described in terms of an interface (block with in and out pins) and a text window in which the logical representation of the transition function is presented (using formulas). This textual representation is not readable for large nodes and infact does not easily fit the standard window size – even for the example considered which is still a demo example. Thus the original structure and size of the Lustre nodes is crucial for readability of the transformed models. These models are, however, not considered to be primarily checked by inspection, why this is not a major weakness. However, some pretty-printing would not do any harm.

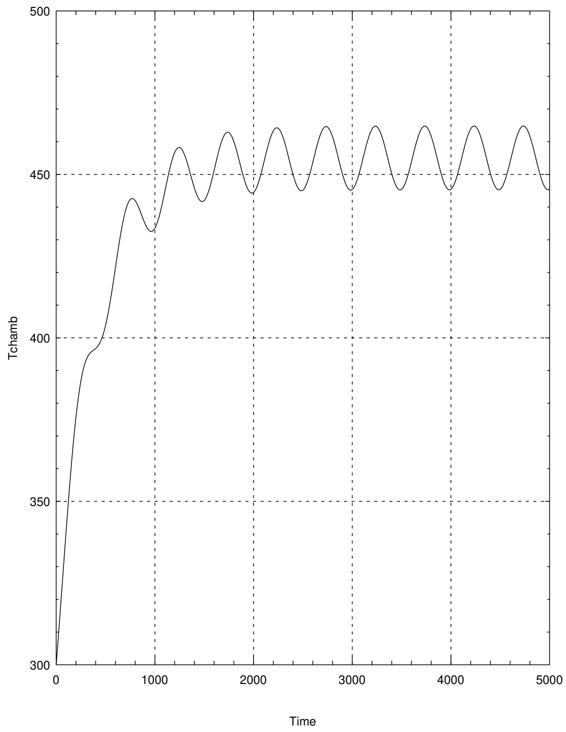
The major benefit of Lucifer is, however, that the Lustre program can be translated and unfolded at the same time. So inductions which require a base step involving more than one initial step can be performed on the unfolded models.

While proving the bounded response property, the analysis messages resulted in discovering a unit inconsistency between the plant and controller subsystems. That is, the air flow unit the programmer of the controller had assumed to be used was not consistent with the measure of flow we had used in the plant equation (according to standard thermodynamics textbooks). This is a typical case of underspecification at the requirements stage leading to trivial errors but fatal failures.

This inconsistency could be resolved by adding a conversion equation in the system description. By the time the deliverables had to be written the validity of the main inductive assertions had not been proved yet. However, several base steps were proved (within minutes) before the inductive proof was interrupted. It might be the case that adjusting prover parameters plays a role in getting to shorter proofs.



(a) With constant incoming temperature



(b) With sinusidal incoming temperature

Figure 3: Two simulations of the non-linear plant model based on constant airflow and different incoming temperatures.