The s2l Simulink to Lustre translator *
User Guide

December 7, 2004

1 Introduction

This document is the supporting documentation for the *s2l* tool, the translator from Simulink[1] to Lustre. For more information about the theoretical aspects of this work you should refer to [1]. Here, we describe the principal command line options, the internal architecture and some implementation details.

1.1 Tools

The s2l tool is implemented in *Java*. In addition, for parsing of the Simulink model file and the generation of the XML file it uses the *JavaCC*[2] parser tool.

For the majority of the API, *javadoc* comments have been added, so an additional hypertext documentation of the classes, methods and members is available.

Finally, to build the tool we use the *ant* tool. Building s2l generates the *s2l.jar* java archive file in the *s2l_root/build/* directory. This contains all the packages of the translator. For invocation of the tool a version of Java >= 1.4[3] should be used. The call in that case must be:

```
java_1.4 -cp <path_to_s2l.jar> <main_class> [options] [mdl_file]
```

In Unix-like systems this call is simplified using the *s2l* shell script, located under the *s2l_root/bin* directory[4]. For the rest of the documentation it is this script which is used for the invocation of the tool.

---

*VERIMAG, Centre Equation, 2, avenue de Vignate, 38610 Gières, France. Web: www-verimag.imag.fr. This work has been supported in part by the European IST project "Next TTA" under project No IST-2001-32111.

[1] Matlab and Simulink are Registered Trademarks of The MathWorks, Inc., Natick, MA.

[2] Java Compiler Compiler Version 2.0 (Parser Generator)

[3] even though previous versions will work as well, this version is better tested and is the one used for the implementation.

[4] s2l must be configured correctly.

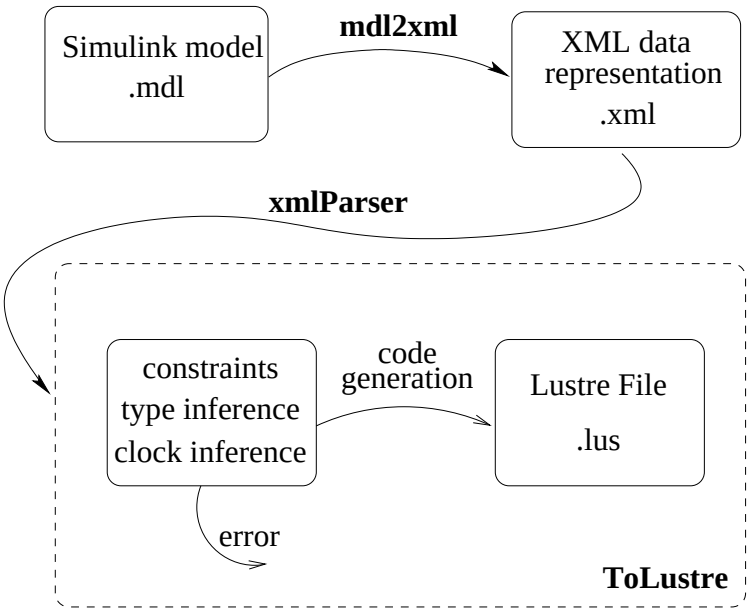

Figure 1: Architecture for the s2l project

1.2 Architecture

The architecture of the project is shown in Figure 1. The input to the parser is a Simulink model in its *.mdl* file representation. This file is parsed and all the information relevant to the translation is copied to a new file using the XML representation.

This intermediate representation is chosen for two major reasons; For every new release of Matlab/Simulink the file representation is changed, so the translator *per se* should not depend on that, but on a more stable representation. This representation is a *de-facto* standard for such data representation.

Subsequently, this XML representation is parsed and an Abstract Data Tree is created to hold all the information of the model and its elements. Finally the main translator takes that information and proceeds with all the algorithms, inferences and code generation.

2 Supported Simulink features

As described in [1] not all the Simulink models can be translated to Lustre. According to this study, we choose to translate only the discrete part of Simulink.

We only translate the discrete-time part of a Simulink model. Concretely, this means blocks of the "*Discrete*" library (such as "Unit-delay", "Zero-order

hold", "Discrete filter" and "Discrete transfer function"), generic mathematical operators such as sum, gain, logical and relational operators, other useful operators such as switches, and, finally, subsystems or triggered subsystems.

These limitations, visible to the user, are explained in the following 4 points;

(1) Since Simulink semantics depends on the simulation method, we restrict the translation only to one method, namely, "*solver: fixed-step, discrete*" and "*mode: auto*". This means that Simulink models must be saved AND simulated correctly under the above simulation method before feeding them to s2l. If the controller to be translated is part of a bigger Simulink model containing both discrete and continuous part, we suggest that you copy/paste it to a new model, correct the parameters and simulate the new one. If Simulink does not give an error then you can use this model with the translator.

(2) We do not translate *S-functions* or *Matlab functions*. Such functions are often helpful. On the other hand, they can also create side-effects, which is something to be avoided and contrary to the "functional programming" spirit of Lustre.

(3) As the Simulink models to be translated are, in principle, controllers embedded in larger models containing both discrete and continuous parts, we assume that for every input of the model to be translated (i.e., every input of the controller) the sampling time is explicitly specified. This also helps the user to see the boundaries of the discrete and the continuous parts in the original model.

(4) In accordance with the first goal, we want the Lustre program produced by the translator to type check if and only if the original Simulink model "type checks" (i.e., is not rejected by Simulink because of type errors). However, the behavior of the type checking mechanism of Simulink depends on the simulation method and the "*Boolean logic signals*" flag (BLS). Thus, apart from the simulation method which must be set as mentioned in (1), we also assume that BLS is on. When set, BLS insists that inputs and outputs of logical blocks (and, or, not) be of type *boolean*. Not only this is good modeling and programming practice, it also makes type inference more precise and simplifies the verification of the translated Simulink models using Lustre-based model-checking tools. We also set the "*algebraic loop*" detection mechanism of Simulink to the strictest degree, which rejects models with such loops. These loops correspond to cyclic data dependencies in the same instant in Lustre. The Lustre compiler rejects such programs.

It should also be noted that Simulink is a product evolving in time. This evolution has an impact on the semantics of the tool. For instance, earlier versions of Simulink had weaker type-checking rules. We have developed and tested our translation method and tool with Simulink 4.1 (Matlab 6 release 12).

The translator groups blocks into three categories; "non-translatable", "known blocks" and the rest.

The first category contains blocks that we do not need to translate but are commonly found in Simulink models. This includes the "Probe" and "Scope" blocks. These blocks should not have outputs and also not affect the type or the Sample Time of any signal in the system.

known blocks
Sources Inport, Constant, DiscretePulseGenerator, Random, DataTypeConversion *Sinks* Outport, Ground, Terminator *Discrete* UnitDelay, DiscreteTransferFcn, DiscreteFilter, Zero-OrderHold *Math Operations* Sum, Product, Gain, CombinatorialLogic, LogicalOperator, RelationalOperator *Signal Routing* Mux, Demux, BusCreator , BusSelector, Switch *Signal Attributes* DataTypeConversion *Ports & Subsystems* Subsystem, Trigger, Enable *various categories* Saturation, Lookup2D

non-translatable
Sinks Scope, Probe, Display

Figure 2: Blocks recognized by s2l.

In the second category we have the blocks for which we know the exact specifications such as number of inputs and outputs, how they behave according to type and clock inference and how they are translated into Lustre code.

The last category contains the rest of the blocks. These blocks behave as "neutral" with respect to clock and type, meaning that they do not change anything between inputs and outputs. So during the type and clock inference phase no special manipulation is done. As for the code generation, since they are unknown blocks for s2l they are translated as external nodes.

Figure 2 shows the blocks actually treated by the tool with respect to the Simulink library (in italics) that they belong.

```
Simulink to Lustre  May2004 -- integration of sf charts
Usage: s2l <OPTIONS>... <FILE.mdl>
Translates a simulink model to Lustre

where options can be
-p,--pollux         Generates Lustre code for the Pollux Lustre compiler
-d,--debug          Provides debug information to standard output
-m                  Generates a MainNode that folds the entire model
-mp, --monoperiodic All the flows in the lustre program will have
                    the same clock
-o <fileName>       The lustre output file name
--sf-lustre-file    The Lustre file of the SF
-f,--fullnames      The names of nodes and certan variables correspond
                    to their path
--varnames          Output the correspondance of the variable names in
                    the lustre file
-xml                Translates the model to XML and exits.
-v,--version        Prints the version of the program
--help              Display this help and exit

report bugs at      Christos.Sofronis@imag.fr
```

Figure 3: The help page of the s2l tool

3 Command line invocation

The s2l script is called with only one argument; the Simulink model to be translated. For instance for a model named "engine" the call is:

```
s2l engine.mdl
```

This will invoke the translator with the default behavior. There are a number of options to call s2l with additional behavior. Calling *s2l*, with the *--help* option will give the help page with the tool's version and those options. The result of invoking `s2l --help` is shown in Figure 3. These options are discussed in more detail here:

- **-p, --pollux**. There are certain cases where pollux's semantics differ from reluc's, i.e. pollux will not accept a node that has zero input arguments. For this case, if the translator is called using the **-p** switch it will introduce a boolean argument to every zero-argument node and in the node call it will pass **true** as an argument to the node. Care is taken also that the argument in the call of the node is sampled correctly.

- **-d, --debug**. This is a flag used to send verbose messages to standard output for each translation step. In addition, it generates two files in the current directory with information about the model. These files are "allVars" containing all the variables along with their type and clock information and "results" with all the blocks with their main attributes.

- **-m**. In the Simulink model, various inputs and outputs may be sampled at different rates (have different sample times). This will produce Lustre code with input variables having different *clock* information (assigned using the *when* mechanism). Sometimes code produced in this way is more difficult to compile to the target language. Therefore we generate a wrapper node whose inputs and outputs are the same as the current root node, but all sampled to the same clock, which is the fastest one. Within this code the appropriate clocks are generated, the signals are sampled and the root node is appropriately called.

- **-mp, --monoperiodic**. One of the requirements of the tool is that all the inputs explicitly have their sample time declared, for the Clock Inference algorithm to compute the clock information of every variable used in the Lustre program. It may be the case that throughout the model there is only one sample time and thus we don't want to declare input sample times. For such a case we must use this switch and the Clock Inference will assign every variable of the model to the basic clock. *Be careful using this option because no check is done if any sample time exists in the model, which may result in an incorrect Lustre file.*

- **-o** must be followed by a filename with a *.lus* extension. This will be the resulting Lustre file.

- **--sf-lustre-file**. This option must be followed by the filename of the Lustre program that contains the translated Stateflow chart(s) of the model.

- **-f,--fullnames**. During translation we try to keep the names of the variables to reflect to the corresponding signals in the Simulink model. However, a Simulink system may contain subsystems which contain more subsystems and so on. This option prefixes to the name of all variables the names of the parent systems concatenated with underscores. Deeply-nested subsystems would result in very long names so by default this feature is turned off.

- **--varnames**. Using this option the tool will print to the standard output the names of the variables per node and the signal in the Simulink model that they correspond to.

- **-xml**. As shown earlier the first step of the translation is to extract the useful information out of the Simulink model file and generate an XML file. Normally, this file is deleted after being used. This option will instruct the tool to generate the XML file and then quit, without deleting it.

- **-v,--version**. Prints to the standard output the version of the program.

- **--help**. This option displays the help message and exits. Also this help message is displayed in case an error occurs in the options.

References

[1] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis. Translating discrete-time Simulink to Lustre. In *Embedded Software (EMSOFT'03)*, volume 2855 of *LNCS*. Springer, 2003. 1, 2