`toWhy3` Translation Script Manual

Version 0.1

Dejanira Araiza-Illan
Computer Science Department
University of Bristol
dejanira.araizaillan@bristol.ac.uk

December 2014

1 Overview

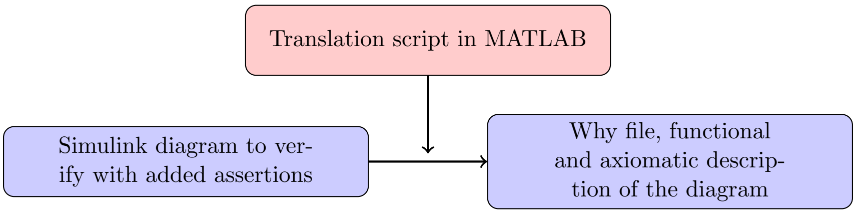
1.1 Installation

The translation script requires an installation of MATLAB (e.g., MATLAB R2013a or more recent) and Simulink to run.

To use the script, copy and paste all the provided files (`toWhy3.m`, `library_simulink.txt`, block libraries `*.mdl`) in a working directory in MATLAB's path. The translator now can be called from MATLAB's command line (see Section 1.3).

1.2 The Translation Script at a Glance

The purpose of the translation script is reading a Simulink model in `*.mdl` format, and translating it into `Why`, the logic subset of the Why3 tool language[1], for its formal verification.



1.3 Invoking the Translator

From MATLAB's command line, the script is invoked with:

```
>> toWhy3('model_name',0)
```

The first parameter is the name of the Simulink diagram to be translated, using single quotes, the second is the compilation of the model before translation to ensure data coherence (`1`), or assume data coherence is correct (`0`). The constant and initial parameters of a Simulink diagram need to be loaded into the workspace, if selecting the compilation option.

The diagram needs to be in the current MATLAB's working directory (or the path to the diagram needs to be added to MATLAB's working paths), to be accessed by the script. The `Why` file will be saved in MATLAB's current working directory.

[1]`http://why3.lri.fr/`

2 Internal Functions in Detail

2.1 Main Body

The main body of the script performs the following actions:

- Initializes all the data structures, starts the output files, reads the input libraries.
- Performs the compilation of the diagram, if required by the user.
- Calls the main functions to start the block processing and identification, and to write the Why file.

`Theories` are written for all subsystems processed internally, and for the main Simulink diagram level. The script assigns a block → Why3 theory correspondence, according to the parameters of the block (e.g., mask of the subsystem, value, operator), to link to the adequate translation of the block's functionality. Assertions in the Simulink diagram to verify through Satisfiability Modulo Theory (SMT) solvers in Why3 are added as `goal` expressions.

2.2 Function `startList`

This function identifies all the blocks and subsystems in the uppermost level of a Simulink diagram, and puts them in a linked list to be processed later.

- Parameters: None
- Return: None
- Modified linked lists: `queueB`

2.3 Function `findAnnotations`

This function identifies the assertions in the diagram, by finding `Numerical`, `Goal` or `Require` masked blocks, that contain `Assert` blocks inside. These blocks indicate the `goal` expressions to be added in the Why file. The `Numerical` blocks are ignored, as these assertions are to be verified via simulation.

- Parameters: none
- Return: none
- Modified linked lists: `oQueue`, `auxqueueB`, `requires_b`, `preconditions`, `postconditions`, `goalblock`, `goals`, `numericalblock`, `numericalcomp`

2.4 Function `getType`

This function classifies any block into four types, so that they can be processed internally or treated as atomic units.

- Parameters: handle of currently analyzed block or subsystem
- Return: an integer, 1 to 4, that indicates the type. `1`: a block from Simulink default library (treated as atomic unit), `2`: a block from our library (treated as atomic unit), `3`: subsystem to be processed internally, `4`: enabled subsystem.
- Modified linked lists: none

2.5 Function `inSimList`

This function determines if a subsystem or block is in the default Simulink library.

- Parameters: mask name of a subsystem block
- Return: an integer, 1 if the block is in the default Simulink library, 0 otherwise.
- Modified linked lists: none

2.6 Function `inMyList`

This function determines if a subsystem is part of the assertion blocks we have designed (`Numerical`, `Goal`, `Require`).

- Parameters: mask name of a subsystem block
- Return: an integer, 1 if the block is part of our blocks, 0 otherwise.
- Modified linked lists: none

2.7 Function `pred_and_suc`

This function identifies the blocks connected to the inputs (predecessors) or outputs (successors) of a block, and puts them into a data structure.

- Parameters: handle of currently analyzed block
- Return: arrays `pchildren` and `pparents` with predecessor and successor blocks (connected to inputs and outputs of currently analyzed block)
- Modified linked lists: none

2.8 Function `bypassSs`

This function to connect inner blocks of subsystems to outer blocks, bypassing `Inport` and `Outport` blocks.

- Parameters: handle of currently analyzed block
- Return: arrays `bchildren` and `bparents` with predecessor and successor blocks (connected to inputs and outputs of currently analyzed block)
- Modified linked lists: none

2.9 Function `extractSs`

This function extracts the inner blocks of subsystem and adds them to a linked list, to be translated into a `theory`.

- Parameters: handle of currently analyzed subsystem
- Return: none
- Modified linked lists: `inports`, `outports`, `enables`, `oQueue`, `auxBlocks`, `connectBlock`

3 Function `createMainTheory`

This function assembles the `theory` of the uppermost Simulink diagram level in the Why file. All the signals are added first as `functions`, and then all the blocks' `theories` are added (linked to the ones in a library by the `clone` directive).

- Parameters: identifier of the opened `Why` file and name of the Simulink diagram
- Return: none
- Modified linked lists: none

4 Function `createSubsTheory`

This function assembles `theories` in the Why file, for all the subsystems and enabled subsystems processed internally.

- Parameters: identifier of the opened `Why` file and handle of the subsystem
- Return: none
- Modified linked lists: none

5 Function `addGoals`

This function adds the verification `goals` into the Why file, following the linked lists of annotation blocks processed previously into linked lists.

- Parameters: identifier of the opened `Why` file
- Return: none
- Modified linked lists: none

6 Function `get_the_data`

This function finds particular block parameters, for an accurate translation of the block's functionality. These parameters are required by the `library_simulink.txt` file, when linking a block to its Why `theory`.

- Parameters: string with the requested parameter about a Simulink block, and handle of the currently analyzed block
- Return: string to be written into the `Why` file, according to the block parameters
- Modified linked lists: none

7 Currently Supported Blocks and Their Specific Parameters

The supported blocks and specific parameters are contained in the file `library_simulink.txt`. For example, the `Sum` block can perform addition or subtraction according to the signs of its inputs. Thus, its functionality must be linked to an addition or subtraction block `theory`, respectively, after the analysis of the inputs.

SUPPORTED BLOCK TYPE	SPECIFIC PARAMETER
Product	–
UnitDelay	–
Sum	`sum_type`
Constant	–
Delay	–
From	–
Quadratic	–
Is_equal_scalar	–
Gain	`gain_value` (for scalar gains only)
Is_pos_def	–
Compare To Zero	`sign_type_ctz`
Transpose	–
Concatenate	`VH`
m0	–
RelationalOperator	`sign_type_rel`
MinMax	`minormax`

8 Adding Support for More Simulink Blocks

These aspects need to be modified when adding a new block, from Simulink default library:

- A `theory` needs to be provided in Why, describing the functionality of a block, from its inputs and outputs.
- Modification of the `library_simulink.txt` file, to link to the corresponding Why `theory`.
- Modification of the `get_the_data` function, to add particular parameters that need to be extracted when processing the block, to link to its accurate functionality (e.g., options processing the inputs, data types, number of inputs-outputs).

References

[1] D. Araiza-Illan, K. Eder, and A. Richards. *Formal Verification of Control Systems' Properties with Theorem Proving*. Proc. UKACC CONTROL, pp. 244– 249, Loughborough, UK, 2014.

[2] D. Araiza-Illan, K. Eder, and A. Richards. *Verification of Control Systems Implemented in Simulink with Assertion Checks and Theorem Proving: A Case Study*. Submitted to ECC 2015.

[3] F. Bobot, J. Filliâtre, C. Marché, G. Melquiond, and A. Paskevich. *The Why3 Platform*. March 2013.

[4] P. Roy, and N. Shankar. *SimCheck: a Contract Type System for Simulink*. Innovations in Systems and Software Engineering 7: 73-83, 2011.