A Tabular Expression Toolbox for Matlab/Simulink

Colin Eles* and Mark Lawford*

McMaster Centre for Software Certification
McMaster University, Hamilton, Ontario, Canada L8S 4K1
{elesc,lawford}@mcmaster.ca

Abstract. Tabular expressions have been successfully used in developing safety critical systems, however insufficient tool support has hampered their wider adoption. To address this shortfall we have developed the Tabular Expression Toolbox for Matlab/Simulink[1]. An intuitive user interface allows users to easily create, modify and check the completeness and disjointness of tabular expressions using the ATP PVS or SMT solver CVC3. The tabular expressions are translated to m-functions allowing their seamless use with Matlab's simulation and code generation.

1 Introduction

Model based design (MBD) has gained increased industrial acceptance, but successful commercial tools such as Matlab/Simulink lack formal semantics and notations that would directly support formal methods. On the other hand formal (and semi-formal) methods have not provided support tools that integrate with existing industrial software development practices and typically overburden developers with the complexity of their formal notation and user interfaces. For example, tabular expressions provide a formal method of specifying mathematical functions that are readable by domain experts, but inadequate tool support has hampered industrial adoption of tabular expressions. To address these problems we have designed a tabular expression toolbox for Matlab/Simulink. We support some of the most common types of tables and have designed the toolbox to allow easy extension to other table types.

The rest of the paper is organized as follows. In Section 2 we discuss some background on work done on formalizing Simulink and existing table tools. Section 3 presents some preliminary information on tabular expressions, and Matlab/Simulink. In Section 4 we provide details of the development and use of the toolbox. A case study using the toolbox is briefly presented in section 5.

* Supported by the Ontario Research Fund, and the National Science and Engineering Research Council of Canada.

[1] Toolbox is available from
http://www.mathworks.com/matlabcentral/
fileexchange/28812-tabular-expression-toolbox

M. Bobaru et al. (Eds.): NFM 2011, LNCS 6617, pp. 494–499, 2011.

2 Related Work

In the area of formalizing Simulink models Roy and Shankar [2] describe a tool that provides a richer type system for Simulink diagrams. Whalen *et al.* [3] have discussed an integrated approach for formally analyzing model based designs using a combination of commercial and custom tools. Tiwari [10] has proposed a method for formally analyzing Simulink and Stateflow models using push down automata.

Examples of table tools for software engineering include an Eclipse IDE plugin for designing tabular expressions using the OMDoc language to represent tables and PVS for verification purposes [4]. The SCR* toolset [5] supports a wide variety of tables and formal analysis techniques but is only available under strict licensing terms. Our work differs from these tools by providing an open source toolset for formally checking a Simulink tabular expression block, or set of tabular expression blocks from within Matlab. We do not attempt to verify the entire model. The Toolbox is the first attempt to integrate tabular methods with a widely available commercial MBD framework.

3 Preliminaries

3.1 Tabular Expression

To specify software, designers often need to describe what should be done for different equivalence classes of inputs. It has been shown that tables provide a formal yet convenient way to specify these functions [6]. We believe that tabular expressions allow for easier readability of documentation, and facilitate inspection of completeness and consistency of specified functionality. Below, a simple example is used to explain tabular expressions. For a detailed discussion of tabular expressions semantics, we refer the reader to Jin and Parnas [7].

In Fig. 1 a formal logical specification of a function appears on the left and its semantically equivalent two dimensional tabular expression as displayed by the toolbox appears on the right. For the one-dimensional function table in (1), we require the Boolean conditions in x, y (the c_i's) in the table's *predicate grid* to be complete (2) and disjoint (3). We can then return the unique value of the expression e_i from the *output grid* when c_i is true. Disjointness ensures that the specified function is deterministic, and completeness guarantees that we have considered all possible inputs, both critical properties for safety applications. Although the graphical layout of tables lends itself to ease of visual inspection of these properties, we would prefer to automate the checking of these obligations, to avoid human errors.

$$\mathrm{f}(x,y) \stackrel{\mathrm{df}}{=} \left\{ \begin{array}{|c|c|c|c|} \hline c_1 & c_2 & \ldots & c_n \\ \hline e_1 & e_2 & \ldots & e_n \\ \hline \end{array} \right. \tag{1}$$

$$\textbf{disjointness} \stackrel{\mathrm{df}}{=} i \neq j \rightarrow (c_i \wedge c_j \leftrightarrow \bot) \tag{2}$$

$$\textbf{completeness} \stackrel{\mathrm{df}}{=} (c_1 \vee c_2 \vee \ldots \vee c_n) \leftrightarrow \top \tag{3}$$

$$\mathrm{f}(x,y) \stackrel{\mathrm{df}}{=} \begin{cases} x+y & \text{if } x>1 \wedge y<0 \\ x-y & \text{if } x\leq 1 \wedge y<0 \\ x & \text{if } x>1 \wedge y=0 \\ xy & \text{if } x\leq 1 \wedge y=0 \\ y & \text{if } x>1 \wedge y>0 \\ x/y & \text{if } x\leq 1 \wedge y>0 \end{cases}$$

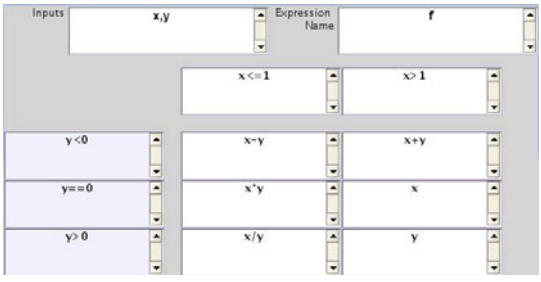

Fig. 1. A logical description of a function and its tabular expression

3.2 Matlab/Simulink

The Simulink modeling language is considered the *de facto* Model Based Design platform for industrial MBD applications [1]. A commonly cited problem with Matlab/Simulink is its lack of formal semantics. We believe that by considering a smaller "safe" subset of the Matlab language, we will be able to convincingly argue that the semantics of Matlab tables are consistent with target languages[1,3].

4 Toolbox

We desired an intuitive user interface that facilitated tabular expression creation and editing, code generation, verification, and graphical counter-example generation. Rather than building all of components for such a tool from the ground up, we have leveraged the power of existing tools, namely Matlab/Simulink, PVS and CVC3. The toolbox combines these tools, integrating them into the Simulink workflow while hiding the detailed verification steps from the end user.

The tool currently supports one and two dimensional normal function tables and also allows for sub-grids in one dimension. The toolbox generates embedded Matlab code for each table which can be saved to a Simulink block or to an M-file. Thus once created, a table can be immediately executed, integrated with other Matlab scripts and functions, or used to generate code for the target platform.

We support predicate subtyping on inputs/outputs of tables, as both CVC3 and PVS have support for predicate subtyping. The complexity of conditional and output expressions is only limited by that of the embedded Matlab language, as well as the capabilities of the backend verification languages. As complexity of expressions increases the verification time generally increases and the chance of finding a counter-example decreases.

4.1 Model

Our model of tabular expressions is similar to that presented by Jin and Parnas [7]. We identify two different constituents which are related to form the table; one or more predicate grids and an output grid. We can evaluate a grid by locating the root predicate grids, determine the true predicate cell, if the cell

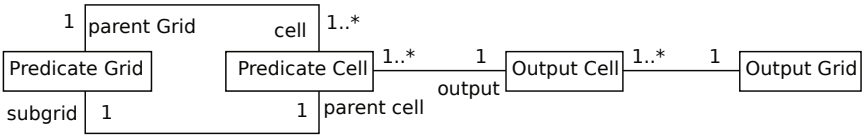

Fig. 2. Class Diagram

has a sub-grid we then evaluate that recursively, otherwise we select the output cell. A class diagram of this model is presented in Fig. 2, where the numbers on the relationships represent the multiplicity of the classes.

4.2 Table Toolbox GUI

The current version of the toolbox has been developed using the Matlab GUI API. The Tool has been integrated into Matlab/Simulink so that users do not have to leave the primary development environment to use the tool. In Fig. 3 a screenshot of the current version of the tool shows a table that has failed the disjointness check and the m-function generated from the table has been executed and viewed to assist in debugging. The table can be fixed by changing the "`||`" to "`&&`" in the second predicate. Colour coding is used to differentiate the columns that overlap for the counter example generated by CVC3. Both c_1 and c_2 are one colour, since they are true, while c_3 is a different colour to indicate it is false.

4.3 Verification and Validation

For V & V of functions we have chosen to use the theorem prover PVS [8], as well as the SMT solver CVC3 [9]. These tools offer a diverse approaches to verifying the disjointness (2) and completeness (3) conditions.

PVS has built-in support for tabular expressions, so the toolbox only needs to generate PVS for the table and then PVS generates proof obligations for (2) and (3). We make use of the random-testing functionality of PVS to attempt to find counterexamples to unprovable obligations.

We use CVC3 for the same purpose as PVS, to ensure that tables are disjoint and complete. As CVC3 does not directly support tabular expressions, we must generate the obligations (2) and (3) for CVC3 in the form of queries which are pushed onto the proof stack. CVC3 will output a counter-model of a query if it is shown to be invalid.

Strategy. We have found that by utilizing tools based on different technologies and theoretical models, we can solve a greater variety of problems. While the automatic proof strategies are often adequate for simple tables, PVS allows users to manually control proofs when required for more complicated obligations. The cost of this flexibility is a larger overhead and greater time required to prove. CVC3 as an SMT solver does not allow for guided proofs but in our experience is

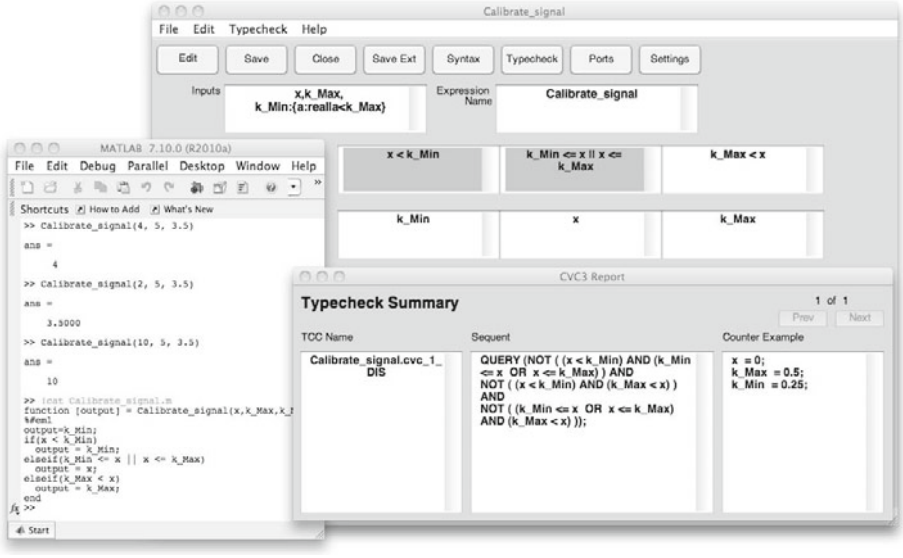

Fig. 3. Screenshot of Tabular Expression Toolbox

very fast and guarantees generation of counter-example via its constraint solving when it is applicable. For tables involving nonlinear expressions, PVS generally gives better results as CVC3 if incomplete for nonlinear constraints.

When both tools are applicable, having two diverse tools to check the tables has the potentially benefit of mitigating against an error in one of the tools. In regulated industries this has the effect of lowering the level of rigour required for qualification of a V & V tool.

5 Case Study

The table tool was used to model a power estimation module based upon requirements for the shutdown system of a nuclear power plant. This system, previously described in Wassying and Lawford [6], used tabular expressions in word documents to document the software requirements. Based upon the requirements document, the example module was implemented in Simulink by an undergraduate student.The module design used 42 different tabular expressions for the power estimation module. Upon typechecking the implemented tabular blocks in the model it was discovered that two contained typographical errors which affected the blocks functionality but would not produce syntax errors or compilation errors. Fig. 3 is an example of one of the detected errors. Errors of this nature are very easy to fix if detected immediately, and become much more difficult and expensive to detect and correct later in the development life cycle.

6 Conclusion and Future Work

Early case studies and feedback have shown that the Tabular Expression Toolbox can be a great asset in developing requirements, designs and implementations in a MBD software development process. By leveraging the power and diversity of tools such as PVS and CVC3 we achieve a greater level of assurance of table correctness. We have managed to hide the formal verification process "under the hood" leaving the designer to concentrate on the design of their system rather than having to learn two new, diverse formal system. Future goals of this project involve additional case studies; consideration of inter-block typing issues, including leveraging existing tools [2]; as well as a more detailed investigation of the consistency of the semantics of Matlab and the analysis tools employed.

References

1. Scaife, N., Sofronis, C., Caspi, P., Tripakis, S., Maraninchi, F.: Defining and translating a "safe" subset of simulink/stateflow into lustre. In: EMSOFT 2004, pp. 259–268. ACM, New York (2004)
2. Roy, P., Shankar, N.: SimCheck: An expressive type system for Simulink. In: NASA Formal Methods Symposium, pp. 149–160 (2010)
3. Whalen, M., Cofer, D., Miller, S., Krogh, B.H., Storm, W.: Integration of formal analysis into a model-based software development process. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS, vol. 4916, pp. 68–84. Springer, Heidelberg (2008)
4. Peters, D.K., Lawford, M., Trancon y Widemann, B.: An IDE for software development using tabular expressions. In: CASCON 2007, pp. 248–251. ACM, Toronto (2007)
5. Bharadwaj, R., Heitmeyer, C.: Developing high assurance avionics systems with the SCR requirements method. In: Proceedings of the 19th on Digital Avionics Systems Conferences, DASC, vol. 1, pp. 1D1/1–1D1/8 (2000)
6. Wassyng, A., Lawford, M.: Lessons learned from a successful implementation of formal methods in an industrial project. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 133–153. Springer, Heidelberg (2003)
7. Jin, Y., Parnas, D.L.: Defining the meaning of tabular mathematical expressions. Science of Computer Programming 75(11), 980–1000 (2010)
8. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: Kapur, D. (ed.) CADE 1992. LNCS (LNAI), vol. 607, pp. 748–752. Springer, Heidelberg (1992)
9. Barrett, C., Tinelli, C.: CVC3. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 298–302. Springer, Heidelberg (2007)
10. Tiwari, A.: Formal semantics and analysis methods for Simulink Stateflow models. Technical report, SRI International (2002), `http://www.csl.sri.com/~tiwari/stateflow.html`