

Final report: EMR contract January-June 2002, University of St Andrews

Formal methods for Simulink: an assessment

Ursula Martin

Richard Boulton, Ruth Hardy, Tom Kelsey

University of St Andrews

July 2002

Summary

The purpose of this project was to investigate machine assisted reasoning in the context of Mathwork's Simulink, in particular in the transition from continuous to discrete models of control systems. The initial phase of the project involved an assessment of the problem and meetings with control engineers to understand what was in important engineering practice. In the second phase we selected two topics for more detailed study, which are proving very fruitful. Both are as far as we know novel, and will be developed further in collaboration with EPSRC, EU Framework 6 or industrial support.

(A) Symbolic analysis of design requirements

Design analysis of typical control applications, such as avionics, involves generation and inspection of a large number of plots, such as Nichols plots, in the continuous or discrete domain, to show that the model satisfies certain design requirements. Typically it is required that a curve avoids a certain forbidden region. We propose to use computational logic and symbolic computation techniques to automate these checks and enable more general investigations. Preliminary experiments are encouraging.

(B) Hoare logic for block diagrams

"Assertions" attached to a design or to code, such as those used in the ProofPower "compliance notation", allow one to document and reason about properties of a design from properties of components. To automate this reasoning we need to understand how properties are "modified" by components, and thus produce "verification conditions" which, if true, ensure the design has the required property. In general the logical underpinning for this approach is provided by "Hoare logic: we have developed a Hoare logic for block diagrams and implemented a prototype verification condition generator.

(C) Avenues for further research

This work has only scratched the surface of what seems to be a fertile and novel research area of great potential, currently attracting significant interest in the US also. We outline in C some avenues worthy of further investigation.

In the rest of this note we survey the background and outline our approach: detailed technical reports on (A) and (B) will follow by the end of September. Full project documentation is available on the password-protected project web-site

<http://www-theory.dcs.st-andrews.ac.uk/info/DCMCS/internal/>

We are exceptionally grateful for the funding, and much more importantly, the attention, expertise and high quality interaction with Rob Arthan (Lemma 1), John Hall (DSTL), Rick Hyde (Mathworks), Yoge Patel (Qinetiq) and Colin O'Halloran (Qinetiq) made possible by Qinetiq's sponsorship of this project.

1 Initial brief

Mathwork's Simulink tool provides a powerful high level environment for the design, analysis and simulation of systems in general and flight control systems (FCS) for specified platforms in particular. To the control system engineer the structure and content of the system representation in Simulink is intrinsic and intuitive and the design can be layered depending on the analysis level required. Over several years of industrial development this tool set is maturing into the new generation of systems engineering, representative of an advanced approach to design. However one remaining weakness is the design verification method.

Using more traditional methods the correct design implementation is achieved by the use of an acceptance test procedure (ATP) which verifies that, as far as possible, the performance requirements specification is met. The ATP approach can be carried into the Simulink environment, however this environment may allow new and novel approaches to testing. In particular verification through machine assisted reasoning is of interest.

Current implementation of verification processes use a combination of equipment qualification and bench testing of individual components through to complete systems tested against a pre-defined ATP. A control system design will have inherent static and dynamic properties based on the requirements specification. However as these properties can be interrogated in the time domain, the frequency domain and the decision domain, in principle a measure of reasonableness should be possible by a different approach. Examples of typical metrics in the frequency domain are its stability properties over the intended operating space and conditions.

A study into the feasibility of how machine assistance and reasoning might be introduced is required and what would be necessary to develop a tool to provide this capability.

2 Background

2.1 Computational logic

Computational logic means the use of computers to produce formal proofs in a given logical system. It ranges from implementations of automatic procedures, through collections of strategies for the machine to try which will not necessarily find a proof, to proof checkers for human or machine generated input. Particularly in the early systems, which lacked automation, proofs were long and hard to produce. However the results produced could be relied upon, which is why this technology became the focus of work in software verification. Advanced computational logic systems such as HOL (Gordon, Cambridge) [3], Isabelle (Paulson, Cambridge) and PVS (Rushby, SRI) [4] address both foundational issues and practical ones such as performance, search and automation, are supported by a mass of theoretical and practical work, and have been used for a variety of major proof developments. Model checking (Clarke, CMU) is a related technique for counter example generation that has also had considerable impact.

The motivation for such research has been research in reasoning itself, in the logical foundations of computer science and increasingly, in practical application. In particular, over the past five years or so, techniques based on computational logic and model checking have started to have considerable uptake in discrete digital systems design at companies such as AMD, Motorola and Intel [24]. The continuous domain is now receiving more attention: for example Harrison [11] verified floating point division on the IA 64 chip using techniques he had developed in his Cambridge PhD. NASA Langley [18] study aircraft avoidance algorithms in free-flight air-traffic control: numerical simulation and symbolic computation

missed a divide-by-zero, which was only found using PVS to do the underlying continuous mathematics correctly. This work has relied upon developing precise formulations of the necessary analysis in computational logic systems: for example to handle elementary functions (ie those built up from polynomials, *cos*, *sin*, *exp* and *log*) correctly in all circumstances, including singularities, one needs a full treatment of power series expansions. NASA Langley have a large public domain library of such material, building on the work of Gottlieb [15] when she was at St Andrews.

2.2 Symbolic computation

Computer algebra systems like Maple [1] (1 million users) incorporate a wide variety of symbolic **computation techniques**, for example for factoring polynomials or computing Grobner bases, and increasingly some numerical elements also. They can in principle do continuous mathematics "symbolically", that is to say the query "integrate $\tan(ax)$ between 0 and b " should return a symbolic answer in a and b and side conditions on a and b to indicate where it is valid. They have enjoyed some outstanding successes: for example 't Hooft and Veltman received the 1999 Nobel Prize in Physics, Veltman for using computer algebra to verify 't Hooft's results on quantum field theory.

In practice current computer algebra systems are not always reliable for continuous mathematics, as they do not handle side conditions well and many continuous mathematics problems do not have an accessible symbolic solution [7]. Our Maple-PVS system [23] addressed this problem by using the computational logic system PVS to provide additional support. Maple made calls such as `prove positive f [a,b]` to PVS, which returned `yes`, `no` (each of which could be justified by a complete rigorous argument if required), or `fail`. PVS automatically executed a domain specific high-level strategy (`plan`), which depended on a knowledge-base of definitions and pre-proved lemmas for elements of real analysis [15].

2.3 Formal methods for embedded systems

High level environments based on numeric techniques provide a powerful tool for the design, analysis and simulation of embedded software control systems in general and automotive and flight control systems in particular. For example Mathworks MATLAB/Simulink [2] provides a graphical user interface: users can build up a model of a dynamical system from primitive blocks representing mathematical functions, matrix transforms and so on, and can use the model to drive a variety of simulations. As embedded systems get larger and more complex – for example a modern car braking system may have many thousands of blocks – there is increased interest in development methods that allow design verification, direct generation of dependable code from such designs, and automatic production of the elaborate certification arguments that are required if such code is to be used in critical applications, like avionics.

Formal methods is a generic term describing symbolic and logical techniques for providing greater assurance in the design and development of software. "Assertions" attached to a design or to code allow one to document and reason about properties of a design from properties of components. To automate this reasoning we need to understand how properties are "modified" by components, and thus produce "verification conditions" which, if true, ensure the design has the required property. This requires software that can manipulate and reason about the mathematics of the design and the code that is generated. This is not a capability of numeric software, and a much wider task than that associated with symbolic computation systems like Maple [1]. Hence we rely on computational logic software.

Our partners at QinetiQ/DERA have a methodology, based on representing Simulink designs in the Z formalism (Oxford) and reasoning in ProofPower, a version of HOL [17]: assertions are documented in the

“compliance” notation. They used it in recent assurance exercise for the EuroFighter, and plan to extend the method. We know of no other work which addresses formal methods for Simulink diagrams, or addresses verification of design requirements for control systems.

However there has been much interest recently in formal methods for MATLAB Stateflow, for example under the US DARPA Mobies project, with strong input from Ford, with projects at CMU, NASA and SRI. Stateflow provides essentially a state machine modelling language within Simulink, similar to Harel’s statecharts, and allows the ready exploitation of model checking to detect flaws in designs and investigate reachability. Hybrid systems are state machines whose transitions are given by continuous conditions, and are supported by hybrid model checkers such as Henzinger’s HYTECH [16]. In earlier work for NAG Ltd we considered “light formal methods” for the computational mathematics system AXIOM and its possible extension to the NAG libraries [19,20].

3 Our approach

Our initial challenge from Qinetiq was to understand how computational logic and formal methods could aid in assurance arguments when passing from a design in the continuous domain to an implementation in the digital (discrete) domain. Both are represented by block diagrams, the former corresponding to s-transforms and the latter to z-transforms. We realised that in essence the problem concerned adding assertions to the continuous and discrete models, and showing that the two assertions were equivalent. Thus we considered the general question of designing an assertion language and logic for block diagrams, which led to (B), and as an immediate application we looked at what assertions were used in practice, which led to (A).

(A) Symbolic analysis of design requirements

Design verification of typical control applications, such as avionics, involves generation and inspection of a large number of plots, such as Nichols plots, in the continuous or discrete domain, to show that the model satisfies certain design requirements. For example the GARTEUR reference model [6, pages 43-47] contains a number of requirements of the form “show that the Nichols plot of a given continuous transfer function avoids a certain region”. A good design will have the curve as close to the region as possible, or within specified tolerances. In current practice these requirements are addressed by inspecting the output of a numeric plot. The method is recognised as capable of improvement: for example analysing a non-linear control law requires linearising at sample points and doing a Nichols plot at each one, with the danger of missing aberrant behaviour through poor choice of sample points. Likewise analysing a parameterised design requires testing for instances of the parameter.

We investigated the use of computational logic and symbolic computation to automate these checks and enable more general investigations. Our approach was based in the following simple idea. Tests based on the Nichols plot concern showing that a given curve $y = f(x)$ does not cross a given line $y = g(x)$ in an interval $[a,b]$. The following theorem provides a way of testing this symbolically: there are clearly a great many other variations and extensions based on the same idea.

Theorem 1 Given a line $y = g(x)$ and a curve $y = f(x)$ defined in an interval $[a,b]$ satisfying

- (i) $f(x)$ is continuous and twice differentiable in $[a,b]$
 - (ii) $f(x)$ is monotonic increasing and concave in $[a,b]$, ie $f'(x) > 0$ and $f''(x) < 0$ in $[a,b]$.
 - (iii) $g(b) > f(b)$
 - (iv) $g'(b) < f'(b)$
-

then $f(x) < g(x)$ throughout the region $[a,b]$. (*)

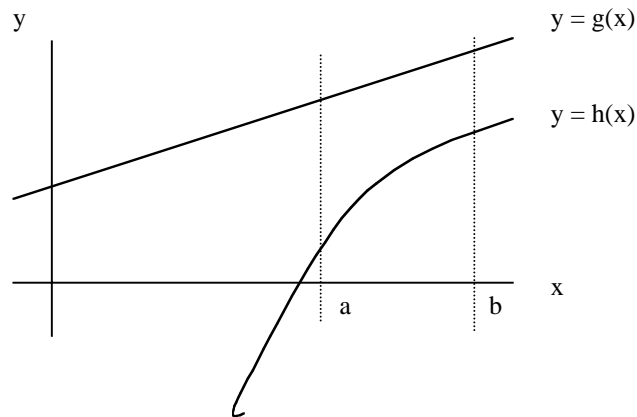


Figure 1: Notation for Theorem 1

Thus to test if (*) holds for a particular f and g we need

- to work out the verification conditions (i)-(iv) above: we do this using Maple to compute and simplify derivatives
- to verify that conditions (i)-(iv) hold: we do this using PVS to test continuity and inequalities

We emphasise that this test is symbolic and shows that the condition (*) holds for **every** point in the region $[a,b]$, not just for a set of sample points.

In the case of the Nichols plot of a transfer function $G(s)$ which is a rational function (ie a quotient of two polynomials), the curve $y = f(x)$ will be given parametrically by

$$x = \text{argument}(G(i\omega))$$

$$y = 20 \ln(G(i\omega))/\ln(10)$$

and its derivatives $f'(x)$, $f''(x)$ will be quotients of polynomials in \cos , \sin and x .

Maple can readily provide the support that we need to compute derivatives, simplify and so on: PVS provides much of what we need already and the necessary material is under constant development, especially by our partners at NASA Langley who need similar support for elementary analysis for their work on air traffic control algorithms. We note that we cannot hope for a fully automatic test, as the question of whether the curve $g(x)-f(x)$ is positive in the interval $[a,b]$ is a version of the zero-constant problem and hence undecidable. There is much theoretical work on techniques for analysing more general questions of this kind, for example in the field of quantifier elimination or semi-algebraic sets.

So far we have prototyped this idea on Theorem 1 and another similar test, and performed the necessary computations in Maple and PVS. Our next steps are to refine this idea to

- develop a more comprehensive suite of tests
- extend the support available in Maple and PVS.

The ultimate goal is to

- provide automated support for analysis of Nichols plots in many common cases
- extend our work to non-linear or parameterised designs
- provide robust support in MATLAB through calls to Maple and PVS

(B) Assertions for block diagrams

“Assertions” attached to a design or to code allow one to document and reason about properties of a design from properties of components. They are widely used in formal and semi-formal analyses of programs, for example, the SPARK Ada “compliance notation” is used in a number of critical systems applications. To automate this reasoning we need to understand how properties are “modified” by components, and thus produce “verification conditions” which, if true, ensure the design has the required property. One approach to the logical underpinning for this idea is provided by “Hoare logic.”

Block diagrams are the standard representations of the transfer functions used in control engineering, representing the s or z transform of a continuous or discrete control system. They are the representation used in Simulink. We have developed a preliminary version of a Hoare logic for block diagrams and implemented a prototype verification condition generator.

B1 Intuitive explanation

As a very simple example, the phase shift of a system S consisting of a controller and plant in sequence is the sum of the phase shift of the controller and the phase shift of the plant.

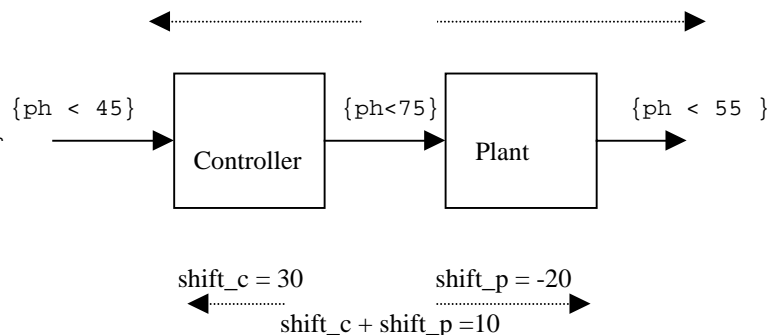


Figure 2: Addition of phase shift

We can add “assertions” about the phase to the diagram, in the form of the expressions $\{ph < 45\}$, $\{ph < 75\}$, $\{ph < 55\}$ which describe properties of the system which are true at the point at which they are attached. In words, these express that if the phase is at most 45 before the signal reaches the controller, then it will be at most 55 when it exits the plant. The condition $\{ph < 45\}$ is called a “pre-condition” and the condition $\{ph < 55\}$ a “post-condition” for the system S .


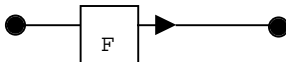
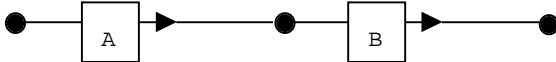
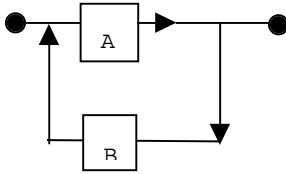
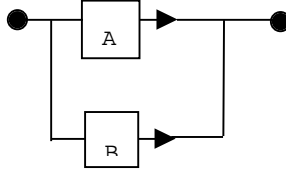
The three assertions about p are not independent. If we know that $\{ph < 45\}$ initially, and that $shift_c = 30$, $shift_p = -20$, we can work forwards and deduce that $\{ph < 55\}$ at the end. We use the general observation about block diagrams that the phase shift of a system consisting of two blocks in sequence is the sum of the phase shift of two blocks. Thus the phase shift across the whole diagram is $shift_c + shift_p = 10$. This general observation is an example of an “inference rule”.

Suppose on the other hand we know the values of $shift_c$, $shift_p$ and want to work backwards, and work out the conditions on the input that will make the statement $\{ph > 55\}$ true at the end. Then we work backwards through the diagram, again using our inference rule, and discover that we need $\{ph > 45\}$ to be true initially. The condition $\{ph > 45\}$ is called a “verification condition” for the postcondition $\{ph > 55\}$. To automate this kind of reasoning we need to explain how assertions are “modified” by components, and thus produce “verification conditions” which, if true, ensure the design has the required property. The logical underpinning for this approach is provided by “Hoare logic” which gives the “inference rules” that describe how properties are “modified” by components. Since Hoare’s original paper [8] there has been much work both on both the logical foundations and practical application of such logics for different kinds of systems: a recent example is Nipkow’s work on Hoare logics for Java.

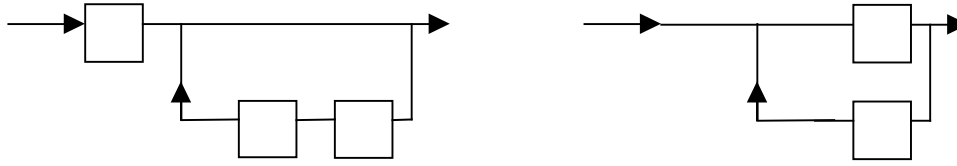
B 2 Our approach

We outline our approach to developing a Hoare logic for a block diagrams. We emphasise that this is preliminary work.

(i) **Structure of block diagrams** To apply our Hoare logic to generate global verification conditions we need to recurse over the structure eof the block diagram, by analogy with the way we recurse over the tree structure of programs. Block diagrams are not tree structures as they allow interlaced loops, so our first step is to reduce them to an equivalent “tree-structured” block diagram in terms of a set of primitives. We use the primitives below, *unit*, *function*, *sequence*, *loop* and *sum*, where F is a primitive function and A, B stand for any “tree-structured” block diagram.

Unit	
Function	
Sequence(A,B)	
Loop(A,B)	
Sum(A,B)	

We conjecture that any block diagram can be reduced to a “tree-structured” block diagram (ii) To reduce the block diagram to the form we need we use standard equivalence rules on block diagrams [9]: for example these two diagrams are equivalent.



Seq (A, Loop (Unit, Sequence (B, A)))

Seq(Unit, Loop (A, B))

Figure 4: Sample block diagram equivalences

Repeated use of this and similar equivalence rules allows us to reduce the diagram to a tree structure of the required form.

There is much more that could be done here in understanding the algebraic structure of block diagrams. Plotkin [11] has suggested that these block diagrams form a trace monoidal category [12], currently the subject of much interest as a model of linear logic.

(ii) **Language for stating assertions** In our pilot work an assertion is a predicate $P(x, y, \dots)$ where $x, y \dots$ are state variables. In our preliminary examples we assume that we are working with transfer functions of the form $G(z)$, where z is a complex number, and x, y are the gain and phase (modulus and argument) of the complex number $G(j\omega)$. We chose this x and y as they are widely used in engineering applications and they appear to enjoy compositional properties, that is the gain and phase of a block diagram can be computed in terms of the gain and phase of its components, whereas other properties, such as stability, do not. The design of a general purpose assertion language for block diagrams is a significant future research problem.

(iii) **Inference rules** Having reduced the diagram to “tree structured” form we provide inference rules in the Hoare style for each primitive. For example, the general observation we made above, about the phase being additive when blocks were composed, becomes the rule

$$\frac{\vdash \{P\} C1 <dr1, dph1> \{Q\} \quad \vdash \{Q\} C2 <dr2, dph2> \{R\}}{\vdash \{P\} \text{Seq}(C1, C2) <dr1 * dr2, dph1 + dph2> \{R\}}$$

where $<dr1, dph1>, <dr2, dph2>$ represent the change of gain and phase across the blocks $C1, C2$ and $\text{Seq}(C1, C2)$ respectively.

(iv) **Verification condition generation**

We use a standard technique for verification condition generation by working backwards through the tree: see [13]. We have implemented a prototype in HOL, code available from the web page. We have used it to verify formally the derivation explained in the informal example above.

B 3 Further work

As far as we are aware this is the first attempt to provide a Hoare logic style framework for reasoning about block diagrams. There are numerous avenues for further research:

(i) develop a fuller analysis of the structure and semantics of block diagrams in both the continuous and discrete cases, and use this to develop an assertion language and framework for reasoning both locally and globally. The approach of regarding block diagrams as trace monoidal categories, and developing a suitable fragment of the mu-calculus to represent the assertions, seems especially promising.

(ii) investigate approaches that would extend to arbitrary blocks, the particular those described by pre/post-conditions only, as occurs, for example, when importing blocks and using light formal methods approaches to document interfaces.

(iii) develop automated support for our work. Having identified a suitable logic this would mean building an implementation in a suitable theorem prover. Much of the work necessary has been done by Arthan and others in Clawz: it should be easy to extend their Z-representation and translation into Proof Power to our framework. More demanding will be providing the necessary theorem proving support – for example for complex numbers – in our theorem prover of choice.

(C) Further avenues of research

This section attempts to summarise the possible avenues of research identified so far in the project. It does not consider what resources would be required to conduct the research, e.g. theory support in the chosen theorem prover.

C 1 Summary of Interesting Properties of Dynamic Systems

Standard analyses of certain properties are an important part of the design verification process for embedded control systems. The properties of interest appear to include:

- Stability (in various forms), analysed using a step-function input on the closed-loop transfer function.
- Frequency response: gain (amplitude) and phase shift, analysed using a sinusoidal input: both open-loop and closed-loop response are of interest.
- Handling properties: analysed through rise time, damping, and steady-state behaviour
- In the analysis of frequency response, a number of graphical representations are used including Bode diagrams (two separate plots, one of gain against frequency, the other of phase against frequency), and Nichols charts (gain against phase as frequency varies). The shape and path of these graphs are used by engineers to determine the response of the system being modelled.

Of these, analyses of handling properties via Nichols plots were identified as the most useful for further investigation.

C 2 Connections Between the Continuous and Discrete Models

In broad terms our plan for investigating the correspondence between a continuous design and its discrete implementation is to devise suitable assertions which hold about each, and then show they are equivalent.

There are other avenues:

- Investigate the relationship between continuous-time and discrete-time Nichols plots. This might arise as a consequence of trying to prove bounds on both continuous-time and discrete-time plots.
 - Find out what modifications the engineers make to their continuous-time models so that they more closely match the corresponding discrete-time model. Is phase shifting to compensate for the zero-order hold one of the modifications (or the modification) that the engineers make? Is there any interesting relationship that we could prove something about?
 - Investigate the relationship between stability conditions for the two time-domains. We have looked at this already, but there are lots of variations to investigate, e.g. we could compare the continuous-time model with the discrete-time model after the former has been modified as discussed above.
-

C 3 Composability

The transfer function of a full real-world system will typically be too large for formal analysis to be practical if the function is treated as a monolithic entity. We are therefore interested in finding properties that can be determined by composing properties of the individual components that make up the system, whether continuous or discrete. For example, it appears that the phase shift of a system consisting of a controller and plant in sequence is the sum of the phase shift of the controller and the phase shift of the plant.

Our first notion, explored in (B) above, was to build something akin to a Floyd-Hoare logic [8, 14] for this purpose. There is a wide range of degrees of formalisation possible here, ranging from extracting verification conditions using something like ClawZ [17], through to "embedding" the calculus (or Floyd-Hoare logic) in the logic of a theorem prover [13] and proving that the properties of the calculus follow from a formalisation of the control theory (Laplace transforms, etc.).

Other possible research along these lines is as follows.

- Consideration of larger (open-loop) systems composed of several transfer functions. Gains should multiply, and phases add. E.g. a constraint on phase shift for a full system could be decomposed into constraints on the phase shift for the components.
- Extend the above to systems involving (closed) loops. Initial experiments suggest that general formulas for the gain and phase of a closed-loop system can be obtained in terms of the gain and phase of the corresponding open-loop system, but the independence between the gain and phase is lost. Hence, gain and phase would have to be treated together in any calculus. I.e. it would not be possible to have one calculus for gain, and another for phase.
- Determine bounds on curves by composing (probably adding) gain and phase values arising out of the factors of the numerator and denominator of the transfer function. It is not obvious that this is possible.
- Investigate how (if at all) composability works in the discrete-time domain, and repeat the above. This is arguably more industrially relevant than the continuous-time version.

C 4 Integration with Simulink

Ultimately, we want to offer formal analysis of the control system models to the control engineers and verification engineers. This might take the form of "batch processing" similar to the way ClawZ operates, using Simulink model files as the communication medium between tools. Alternatively, an interactive facility might be provided that adds new functions to Matlab. Two possibilities are:

- Use ClawZ with a new library to translate Simulink model files into Z. This could work well with the composability research outlined above. It could range from producing Z specifications for particular properties, to using ClawZ to translate the Simulink model into a calculus/logic/language embedded in a theorem prover.
 - Alternatively, inter-process communication could be established between Matlab, ProofPower, and Maple, for example using an interface such as Prosper [21]. This would allow new ways of operating. For example, for properties that can be proved automatically, Simulink might be extended with an on-the-fly (as opposed to batch/off-line) proof facility. The idea would be to put some formal verification in the hands of the control engineers, without them necessarily realising that formal proof was being used.
-

C 6 Advanced Topics

Below is a summary of other avenues of research that we gleaned from discussions with control engineers but which we consider to be topics for consideration only after substantial research on simpler problems.

- Dealing with models of flexible aircraft?
- Dealing with non-linearity (variation with amplitude).
- Multi-input/multi-output systems with cross-coupling.
- Allowing mass and position of centre-of-gravity to vary within bounds. This might involve proving that a system remains stable (or has the desired handling properties) within certain (symbolic) bounds.
- Assertions about the response during transitions between flight modes, especially when mode switching is frequent.
- Investigate assertions about the effects of moving from sub-sonic to supersonic flight, and vice versa.

References

- [1] Maple 6, www.maplesoft.com
 - [2] MATLAB, www.mathworks.com
 - [3] M J C Gordon & T F Melham, Introduction to HOL, Cambridge University Press 1993
 - [4] J Rushby et al, Formal Verification for Fault Tolerant Architectures: Prolegomena to the Design of PVS, IEEE Trans Software Eng, 21(1995) 107-125
 - [5] Ursula Martin, Towards formal methods for mathematical modeling, Proceedings 5th NASA Langley Workshop on Formal Methods 2000, NASA Press 2000, <http://shemesh.larc.nasa.gov/fm/Lfm2000/Proc/>
 - [6] Garter consortium, Action Group FM(AG08). Robust flight control design challenge problem formulation and manual: the high incidence research model (HIRM). Technical Report TP-088-4, version 3, Group for Aeronautical Research and Technology in Europe (GARTEUR), gartersecretary@inta.es, April 1997
 - [7] E Kaltofen, Challenges of symbolic computation, my favorite open problems, J Symbolic Comp 29 (2000) 891-919
 - [8] C A R. Hoare. An axiomatic basis for computer programming. Communications of the ACM, 12(10):576-580, 583, October 1969.
 - [9] M Ogata, Modern Control Engineering, Prentice Hall, 1999
 - [10] Personal communication, 42 bus stop, Copenhagen, 2002
 - [11] John Harrison, Formal Verification of IA-64 Division Algorithms, TPHOLS 2000, LNCS 1869, Springer 2000
-

-
- [12] S Abramsky, E Haghverdi and P Scott, Geometry of interaction and linear combinatory algebras, Mathematical structures in computer science, to appear.
- [13] M J C Gordon. Mechanizing programming logics in higher order logic. In G. Birtwistle and P. A. Subrahmanyam, editors, Current Trends in Hardware Verification and Automated Theorem Proving, pages 387-439. Springer-Verlag, 1989.
- [14] R. W. Floyd. Assigning meanings to programs. In Proceedings of the American Mathematical Society Symposia in Applied Mathematics, volume 19, pages 19-32, 1967.
- [15] H Gottliebsen, Transcendental functions and continuity checking in PVS, TPHOLS 2000, LNCS 1869, Springer 2000
- [16] Bruce H. Krogh, Approximating Hybrid System Dynamics for Analysis and Control, HSCC 1999, LNCS 1569, Springer, 1999
- [17] R Arthan, C O'Halloran et al, ClawZ: Control Laws in Z, ICFEM 2000, IEEE Press 2000
- [18] Victor Carreno and Cesar Munoz, Aircraft Trajectory Modeling and Alerting Algorithm Verification, TPHOLS 2000, LNCS 1869, Springer 2000
- [19] Martin Dunstan et al, Lightweight formal methods for computer algebra systems, ISSAC'98, ACM Press, 1998
- [20] Martin Dunstan et al, Formal Methods for Extensions to CAS, FM'99, LNCS 1709, Springer 1999
- [22] L Dennis et al, The PROSPER Toolkit, TACAS 6, LNCS 1785, Springer 2000
- [23] A Adams, M Dunstan, H Gottliebsen, T Kelsey, U Martin and S Owre, Computer Algebra meets Automated Reasoning: Integrating Maple and PVS. in TPHOLS 2001, LNCSxxxx, Springer Verlag,
- [24] IEEE Spectrum, April 2002
- <http://www.spectrum.ieee.org/WEBONLY/publicfeature/apr02/ecar.html>
-