A Hoare Logic for Single-Input Single-Output Continuous-Time Control Systems*

Extended Abstract

Richard J. Boulton**, Ruth Hardy, Ursula Martin***

School of Computer Science
University of St Andrews
North Haugh
St Andrews
Fife
KY16 9SS
Scotland, UK
Tel: +44 1334 463252
Fax: +44 1334 463278
E-mail: um@dcs.st-and.ac.uk

Abstract. This paper presents a Hoare-style logic for reasoning about the frequency response of control systems in the continuous-time domain. Two properties, the gain (amplitude) and phase shift, of a control system are considered. These properties are for a sinusoidal input of variable frequency. The logic operates over a simplified form of block diagram, including arbitrary transfer functions, feedback loops, and summation of signals. Reasoning is compositional, i.e. properties of a system can be deduced from properties of its subsystems. A prototype tool has been implemented in a mechanised theorem prover.

* Research supported by QinetiQ and by an EPSRC studentship to the second author.
** Affiliation with the University of St Andrews ceased in June 2002.
*** Corresponding author.

1 Introduction

Many man-made dynamical systems such as cars, planes, CD-players and nuclear reactors are augmented with a control system in hardware or software. The physical system is typically refered to as the *plant*. Some plants are inherently unstable in the absence of a control system (e.g. many fighter aircraft), and the systems are often safety critical or mission critical.

Most control systems are configured as a *closed loop* (a feedback loop) in which the outputs or current behaviour of the plant are measured and subtracted from a reference input (a control value such as desired cruising speed of a car). The resulting difference is used as input to a controller which in turn produces signals to control the plant. There may also be a control component in the feedback path.

Control systems may be modelled in the continuous or discrete time domains. In the former, signals are continuously varying with time, modelled as a real number. In the latter, the signals are sampled at discrete time intervals and so the value of a signal may be discontinuous and time can be modelled using integers. Numerical modelling, simulation, and analysis of control systems are supported by computer software. For example Mathworks Simulink [4] provides a graphical representation of a control system as the standard engineers' block-diagram, which is obtained as the Laplace transform of the original dynamical system.

While there is a wealth of academic literature on the design of control systems [7], less attention has been paid to design validation, especially of software systems. In practice visual inspection of numeric plots is widely used: for example, suites of Bode and Nichols plots are used to specify and discharge design requirements for flight control [8], expressed in terms of phase and gain of an input signal.

The use of formal methods and computational logic in the analysis of control systems is of increasing importance, but has thus far largely been confined to hybrid systems and statechart-like models. The widespread use of Simulink suggests that effective formal verification techniques for block diagrams could have significant impact. In general terms one might expect to annotate points in a diagram with assertions stating what was true at that point, for example a property of phase or gain, and use a logic to reason about the assertions. Thus, for example, one might hope to replace the plotting described above with an automated analysis using computational logic.

The work described here constitutes an early step in this program. We present a Hoare logic for reasoning about assertions in block diagrams involving phase and gain. Hoare logics [3] were originally studied by Hoare, Floyd and others to give an axiomatic basis for programming, and continue to be used for a variety of applications [6]. As far as we know our work is the first to investigate Hoare-style logics for feedback systems: there has been little other formal methods work at the block-diagram level. Arthan and others developed ClawZ [1], a system that translates discrete-time models, described using Simulink [4], into formal specifications in Z. A controller implementation in an Ada-like programming language

can then be verified against these Z specifications using the Proof Power mechanised proof assistant. Similarly Mahony [5] has investigated adding feedback to the DOVE specification environment. Other authors [10] have studied reasoning about diagram langauges at a more abstract level. There has also been little work at the dynamical systems level: an exception is Tiwari's work on abstraction for dynamical systems [9].

To simplify matters we chose initially to work with continuous-time rather than discrete models, with a single input and a single output. In practice the stability, time and frequency response of control systems are often analysed: we chose the latter, which treats the amplitude and phase shift of the output signal when the system is presented with a sinusoidal input with a range of frequencies. The key observation, on which the rest of the paper is built, is that gain (amplitude) and phase are properties that behave compositionally as larger control systems are constructed from subsystems. This allows us to build a logic for properties of control systems in the style of Hoare logics.

In the rest of this paper we present gain and phase, our language Cosy for representing block diagrams, our Hoare logic, a simple worked example showing analysis of gain and phase, brief details of our implementation in a theorem proving system, and directions for further work.

2 Composition of Gain and Phase

In this section we develop the aspects of control we need: see a textbook such as Ogata [7] for more details. The Laplace transform is used to transform the dynamical system representation of a continuous control system to a block diagram, essentially a directed graph with edges, corresponding to components, labelled by rational functions over the complex numbers. Assume a component is represented as a function f of time t. Then the Laplace transform maps f to a new function F that has a complex frequency value, s, as its argument:

$$F(s) = \int_0^\infty f(t)e^{-st}\,dt$$

where $f(t)$ is assumed to be zero for negative values of t. Thus in the block diagram $f(t)$ corresponds to an edge labelled F, with input s and output $F(s)$. The Laplace transforms have nice properties, e.g. the transform for two components in sequence is the product of the transforms of the components. Similarly, the output signal of a component is represented by the product of the transform of the input signal and the transform of the component.

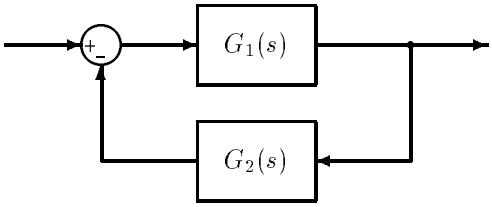
In frequency analysis, we are interested in the behaviour of sinusoidal signals. The analysis is done by substituting $j\omega$ for the variable s in the Laplace transform, where j is $\sqrt{-1}$ and ω is the (real-valued) frequency of the signal.[1] The result is a complex number which can be written in the form $re^{j\theta}$. When the transform represents a sinusoidal signal, r is its amplitude and θ is its phase.

[1] Control engineers conventionally use j rather than i and we follow that practice here.

When the transform represents a component in the control system, r is the gain (the factor by which the component increases the amplitude) and θ is the change in phase caused by the component.

Now suppose we have a control (sub)system constructed from two components in sequence, where $G_1(s)$ and $G_2(s)$ are the Laplace transforms of the components. As stated above, the Laplace transform for the combined system is given by $G_1(s)G_2(s)$. Hence, the result of substituting $j\omega$ for s is $G_1(j\omega)G_2(j\omega)$, which equals $(r_1 r_2)e^{j(\theta_1+\theta_2)}$. So, the gain of the combined system is the product of the constituent gains, and the phase shift of the combined system is the sum of the constituent phase shifts.

So, for sequencing of components, gain and phase compose in a straightforward manner. But, there are other structures to be found in the models of control systems, most notably feedback loops and summing points. Consider the following closed loop system:



The Laplace transform for the entire closed loop system is given by $C(s) = \frac{G_1(s)}{1+G_1(s)G_2(s)}$. Manipulation of complex numbers yields the following formulas for the gain and phase shift of the closed loop:[2]

$$|C(j\omega)| = \frac{r_1}{\sqrt{1 + 2r_1 r_2 \cos(\theta_1 + \theta_2) + r_1^2 r_2^2}}$$

$$\arg(C(j\omega)) = \arctan\left(\frac{\sin\theta_1 - r_1 r_2 \sin\theta_2}{\cos\theta_1 + r_1 r_2 \cos\theta_2}\right)$$

So, the gain and phase shift of a feedback loop can be expressed purely in terms of the gains and phase shifts of its subsystems, but in contrast to sequencing, in this case the gain and phase shifts become inter-dependent. As a consequence, in reasoning about frequency response, the gain and phase shift must be taken together.

Apart from feedback loops, the other significant structure in control systems is a summing point. For single-input, single-output systems, it can be assumed that the two signals to be summed ultimately come from the same source. Then the Laplace transform expressing the relationship between the output and input of this system is $C(s) = G_1(s) + G_2(s)$. Hence, the gain and phase shift are as follows:

$$|C(j\omega)| = \sqrt{r_1^2 + 2r_1 r_2 \cos(\theta_1 - \theta_2) + r_2^2}$$

[2] We assume the "two-place" arctan which is defined in $(-\pi, \pi)$, as implemented in Maple for example.

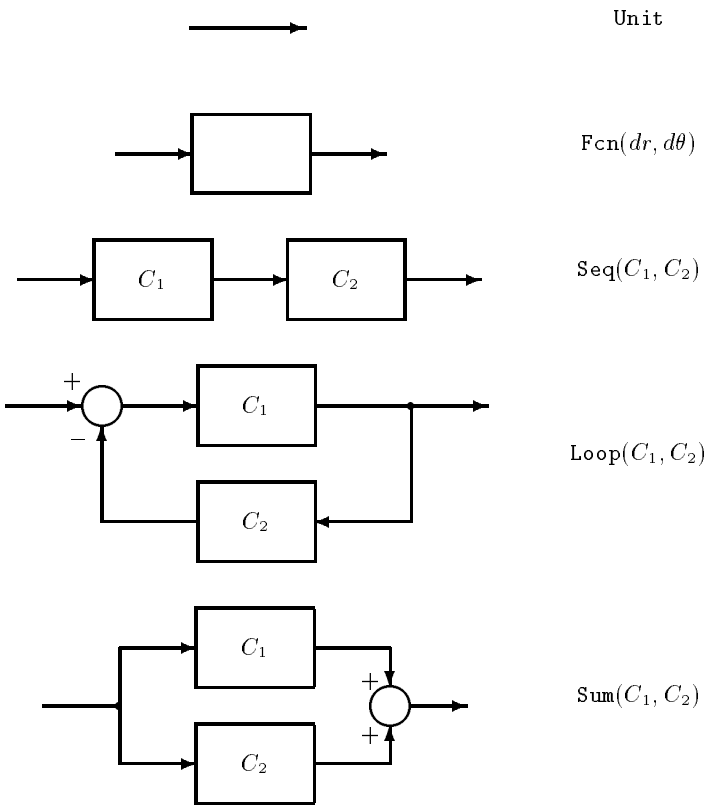


Fig. 1. The Cosy language constructs

$$\arg(C(j\omega)) = \arctan\left(\frac{r_1 \sin\theta_1 + r_2 \sin\theta_2}{r_1 \cos\theta_1 + r_2 \cos\theta_2}\right)$$

3 Cosy: A Simple Language for Control Systems

The sequencing, feedback loop, and summation constructs presented in Section 2 can represent a wide range of control systems. All the constructs have a single input and a single output and are described in terms of subsystems that also have this property. Hence, they give rise to a simple language for control systems, which we shall call Cosy. The abstract syntax of this language is given in Fig. 1, alongside the corresponding block diagrams. In addition to the compound constructs, there are two atomic forms: `Unit`, which simply represents a wire, and `Fcn`, which represents a transfer function with gain dr and phase shift $d\theta$. The names dr and $d\theta$ are used instead of r and θ to emphasize that these values

represent the *changes* in amplitude and phase caused by the transfer function. C_1 and C_2 are arbitrary subsystems.

We can characterise exactly those block diagrams that can be translated directly into Cosy, as those with a tree-structure reflecting the Cosy constructs. While an arbitrary block diagram can be an arbitrary directed graph, many examples in practice (cf [8]) seem to have this structure, perhaps because good designers build them up recursively from simpler components, and avoid intertwined loops, just as good programmers would. In fact Cosy represents a larger class of block diagrams thanks to the standard notions of block diagram equivalence [7], which allow one to eliminate certain classes of intertwined loops for example.

4 Hoare Rules for Cosy

The rules of a Hoare logic usually operate on triples consisting of a precondition (a predicate logic formula), an expression of the programming (or other) language, and a postcondition. In the Hoare logic for Cosy, an additional component is added: a pair consisting of the gain and phase shift induced by the control system represented by the Cosy expression. The gain and phase shift are required explicitly because the pre- and postconditions of some of the rules depend on them.

Notation: $\{P\}C<dr, d\theta>\{Q\}$ means component C (be it atomic or compound) causes a gain of dr and a phase shift of $d\theta$, and if property P holds at the input, then property Q holds at the output. P and Q assert properties of gain and phase. In the original Hoare logic [3], the properties were for states involving a flexible number of program variables. For an analysis of frequency response of a control system, there are two "variables" of interest: gain and phase.

In what follows, $P[r \setminus R,\ \theta \setminus \Theta]$ means the property P with references to the gain r replaced by the expression R, and references to the phase θ replaced by the expression Θ, e.g. $(r < 2)[r \setminus r * 3]$ means $r * 3 < 2$.

The following abbreviations are defined for the gain and phase shift of loops and summations:

$$lr(dr_1, d\theta_1, dr_2, d\theta_2) = \frac{dr_1}{\sqrt{dr_1^2 dr_2^2 + 2dr_1 dr_2 \cos(d\theta_1 + d\theta_2) + 1}}$$

$$lt(dr_1, d\theta_1, dr_2, d\theta_2) = \arctan\left(\frac{\sin d\theta_1 - dr_1 dr_2 \sin d\theta_2}{\cos d\theta_1 + dr_1 dr_2 \cos d\theta_2}\right)$$

$$sr(dr_1, d\theta_1, dr_2, d\theta_2) = \sqrt{dr_1^2 + 2dr_1 dr_2 \cos(d\theta_1 - d\theta_2) + dr_2^2}$$

$$st(dr_1, d\theta_1, dr_2, d\theta_2) = \arctan\left(\frac{dr_1 \sin d\theta_1 + dr_2 \sin d\theta_2}{dr_1 \cos d\theta_1 + dr_2 \cos d\theta_2}\right)$$

Fig. 2 presents an axiom or rule for each of the Cosy language constructs plus some additional logical rules and a special rule for all components.

In the Unit Axiom, `Unit` has no effect on the signal, so the postcondition is the same as the precondition. The gain is 1, and the phase shift is 0. The

The Unit Axiom

$$\frac{}{\vdash \{P\}\texttt{Unit}\langle 1, 0\rangle\{P\}}$$

The (Transfer) Function Axiom

$$\frac{}{\vdash \{P[r \setminus r * dr,\ \theta \setminus \theta + d\theta]\}\texttt{Fcn}(dr, d\theta)\langle dr, d\theta\rangle\{P\}}$$

The Sequencing Rule

$$\frac{\vdash \{P\}C_1\langle dr_1, d\theta_1\rangle\{Q\} \qquad \vdash \{Q\}C_2\langle dr_2, d\theta_2\rangle\{R\}}{\vdash \{P\}\texttt{Seq}(C_1, C_2)\langle dr_1 * dr_2, d\theta_1 + d\theta_2\rangle\{R\}}$$

The Generalised Loop Rule

$$\frac{\vdash \{P\}C_1\langle dr_1, d\theta_1\rangle\{Q\} \qquad \vdash \{R\}C_2\langle dr_2, d\theta_2\rangle\{S\}}{\begin{array}{l}\vdash \{(P \wedge R)[r \setminus r * lr(dr_1, d\theta_1, dr_2, d\theta_2),\ \theta \setminus \theta + lt(dr_1, d\theta_1, dr_2, d\theta_2)]\} \\ \texttt{Loop}(C_1, C_2)\langle lr(dr_1, d\theta_1, dr_2, d\theta_2), lt(dr_1, d\theta_1, dr_2, d\theta_2)\rangle \\ \{Q[r \setminus r * dr_1,\ \theta \setminus \theta + d\theta_1] \ \wedge\ S[r \setminus r * dr_2,\ \theta \setminus \theta + d\theta_2]\}\end{array}}$$

The Generalised Sum Rule

$$\frac{\vdash \{P\}C_1\langle dr_1, d\theta_1\rangle\{Q\} \qquad \vdash \{R\}C_2\langle dr_2, d\theta_2\rangle\{S\}}{\begin{array}{l}\vdash \{(P \wedge R)[r \setminus r * sr(dr_1, d\theta_1, dr_2, d\theta_2),\ \theta \setminus \theta + st(dr_1, d\theta_1, dr_2, d\theta_2)]\} \\ \texttt{Sum}(C_1, C_2)\langle sr(dr_1, d\theta_1, dr_2, d\theta_2), st(dr_1, d\theta_1, dr_2, d\theta_2)\rangle \\ \{Q[r \setminus r * dr_1,\ \theta \setminus \theta + d\theta_1] \ \wedge\ S[r \setminus r * dr_2,\ \theta \setminus \theta + d\theta_2]\}\end{array}}$$

Precondition Strengthening

$$\frac{\vdash P' \Rightarrow P \qquad \vdash \{P\}C\langle dr, d\theta\rangle\{Q\}}{\vdash \{P'\}C\langle dr, d\theta\rangle\{Q\}}$$

Postcondition Weakening

$$\frac{\vdash \{P\}C\langle dr, d\theta\rangle\{Q\} \qquad \vdash Q \Rightarrow Q'}{\vdash \{P\}C\langle dr, d\theta\rangle\{Q'\}}$$

The Shift Right Rule

$$\frac{\vdash r' \neq 0 \qquad \vdash \{P[r \setminus r * r',\ \theta \setminus \theta + \theta']\}C\langle dr, d\theta\rangle\{Q\}}{\vdash \{P\}C\langle dr, d\theta\rangle\{Q[r \setminus r/r',\ \theta \setminus \theta - \theta']\}}$$

The Shift Left Rule

$$\frac{\vdash r' \neq 0 \qquad \vdash \{P\}C\langle dr, d\theta\rangle\{Q[r \setminus r * r',\ \theta \setminus \theta + \theta']\}}{\vdash \{P[r \setminus r/r',\ \theta \setminus \theta - \theta']\}C\langle dr, d\theta\rangle\{Q\}}$$

The Component Rule

$$\frac{\vdash \{Q\}C\langle dr, d\theta\rangle\{R\}}{\vdash \{P[r \setminus r * dr,\ \theta \setminus \theta + d\theta]\}C\langle dr, d\theta\rangle\{P\}}$$

Fig. 2. Axioms and rules of the Hoare logic for control systems

axiom for transfer functions is analogous to the Assignment Axiom for imperative programming languages. If P is true at the input having had the variables r and θ (gain and phase) replaced by the values they have at the output, then P is true at the output. The value of r at the output is equal to the product of the value of r at the input with the function's gain dr. Similarly, the value of θ at the output is the sum of θ at the input and the function's phase shift $d\theta$.

The sequencing rule simply states that gain and phase compose by multiplication and addition when two components are joined in sequence. Note the connection between the postcondition of C_1 and the precondition of C_2.

For loops and summations, there are many variations the rules could take. The rules given here are quite general, but are arguably less natural for proofs than some of the other possibilities. Note that the substitutions have been normalised to avoid division and subtraction.

The logic features the traditional Hoare rules for strengthening a precondition and weakening a postcondition, but it also has two other logical rules. Arbitrary amounts of gain and phase shift can be moved between the precondition and postcondition and vice versa. There are two rules for this, one for each direction. The only restriction is that the gain factor moved between the two conditions must be non-zero to avoid division-by-zero problems.

The transfer function axiom can be generalised to cover any system. If the gain and phase shift of the system are known, the system can be treated in the same way as an atomic transfer function. The Component Rule achieves that. Observe that only the values of dr and $d\theta$ are required from the hypothesis; the precondition and postcondition are thrown away.

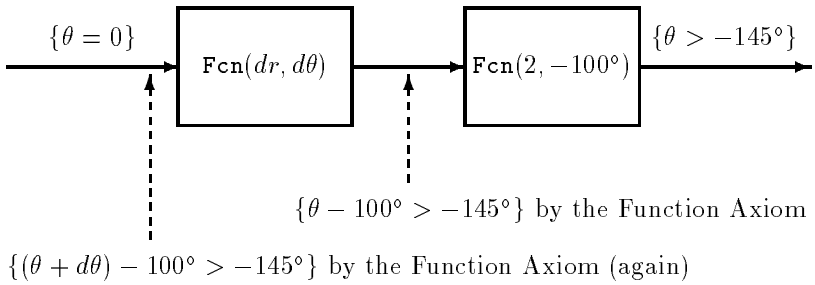
Following Gordon's approach [2], the logic has been mechanised in the HOL98 theorem proving system, allowing goal-directed reasoning, machine assistance in the details of the proof, and automatic generation of verification conditions (VCs). VCs are the logical formulas that must ultimately be proved in a proof within the Hoare logic. The VCs themselves are pure predicate logic formulas, that is, they do not involve the constructs of Cosy.

5 A sample proof

As a very simple example, suppose there are two components in sequence, a controller and a plant, and the phase shift of the combined system is required to be greater than $-145°$, as follows:

$$\{\theta = 0\}\texttt{Seq}(\texttt{Fcn}(dr, d\theta), \texttt{Fcn}(2, -100°))<\ldots>\{\theta > -145°\}$$

The gain and phase shift of the controller have been left as symbolic entities dr and $d\theta$. The aim is to determine constraints on them using the Hoare logic. The proof, using the function axiom and sequencing rule, can be illustrated like this:



The sequencing rule is used to combine the two theorems obtained by means of the function axiom. The result is:

$$\begin{array}{l}\vdash \{(\theta + d\theta) - 100° > -145°\} \\ \quad \mathtt{Seq}(\mathtt{Fcn}(dr, d\theta), \mathtt{Fcn}(2, -100°)) < dr * 2, d\theta - 100° > \\ \quad \{\theta > -145°\}\end{array}$$

Now, if it can proved that $(\theta = 0) \Rightarrow ((\theta + d\theta) - 100° > -145°)$, precondition strengthening can be used to obtain the required result. The condition is true if and only if $d\theta > -45°$, which is the required constraint on the controller.

In this example, the constraint on $d\theta$ could have been determined from the compound gain and phase shift $< dr * 2, d\theta - 100° >$, but for large examples, the expressions for the overall gain and phase shift may become too complex to be practical. Compositional rules with pre- and postconditions permit reasoning at the level of subcomponents, which may be tractable even when reasoning about the whole system monolithically is not.

6 Conclusions and Future Work

This paper has proposed a framework for formally reasoning about the frequency response of continuous-time control systems. The framework takes the form of a Hoare-style logic, based around expressions for the gain and phase shift of control-system structures in terms of the gain and phase shift of substructures.

The framework can be used to derive expressions for the gain and phase of a control system. This goes beyond usual control engineering practice because it can be used for symbolic values as readily as for numeric values. Furthermore, using a Hoare-style logic offers more than calculation (numeric or symbolic): Constraints (typically inequalities) can be derived. This can be easier than seeking exact values for gain and phase. The approach can also be used in reverse to obtain constraints on the parameters of a controller so as to achieve specified behaviour of the overall system. Constraints may be derived for a particular frequency, a range of frequencies, or for all frequencies. Unlike many frequency response analysis methods, this approach is not limited to constraints that can be expressed graphically. And all this is done within the rigors of a formal proof system.

A fuller development for our work would require a more detailed treatment of dynamical systems, Laplace transforms, block diagrams, an appropriate assertion language, and their associated analytical and logical subtleties: hence this paper is a first step. We believe our logic to be sound with respect to the semantics of block diagrams corresponding to the Laplace transform.

There are many directions for further research: extending our techniques to a wider class of arguments than phase and gain; developing the constraint approach we defined above; extending the class of block diagrams we can handle (for example by extending our logic to handle intertwined loops); development of a suitable assertion language; adaptation to multiple-input/output (MIMO), state-space or discrete-time systems; and finally effective automation as a component of standard toolsets such as Simulink.

Acknowledgements: We are indebted to Manuela Bujorianu, John Hall, Rick Hyde, and Yoge Patel for their insights into control engineering, and to Rob Arthan, Tom Kelsey, and Colin O'Halloran for helpful discussions.

References

1. R. Arthan, P. Caseley, C. O'Halloran, and A. Smith. ClawZ: Control laws in Z. In *Proc. 3rd IEEE International Conference on Formal Engineering Methods (ICFEM 2000)*, York, September 2000.
2. 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.
3. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 583, October 1969.
4. The MathWorks. Simulink. `http://www.mathworks.com/products/simulink/`
5. B. Mahony. The DOVE approach to the design of complex dynamic processes. In *Proc. of the First International Workshop on Formalising Continuous Mathematics*, NASA conference publication NASA/CP-2002-211736, pages 167–187, August 2002.
6. Tobias Nipkow Hoare Logics in Isabelle/HOL. In *Proof and System-Reliability*, pages 341-367, Kluwer, 2002
7. K. Ogata. *Modern Control Engineering*. Prentice-Hall, third edition, 1997.
8. R. W. Pratt, editor. *Flight Control Systems: Practical Issues in Design and Implementation*, volume 57 of *IEE Control Engineering Series*. The Institution of Electrical Engineers, 2000.
9. Ashish Tiwari and Gaurav Khanna Series of abstractions for hybrid automata. In *Proc 5th International Workshop on Hybrid Systems: Computation and Control* HSCC 2002, LNCS 2289, Springer 2002
10. C Gurr and K Tourlas Towards the principled design of software engineering diagrams. In *Proc. 22nd International Conference on Software Engineering* pages 509-520, ACM Press 2000.