Proposal for a tutorial at ETAPS'99

Verification of Parametric Systems
or
Monadic 2^{nd} Order Logic in Practice

Tiziana Margaria

Universität Dortmund, Germany
{tiziana@sunshine.cs.uni-dortmund.de}

Scope

In this tutorial we intend to present M2L(Str) as an adequate logic for modelling different classes of parametric systems, discuss how a verification environment for this kind of systems can be designed and realized, and show the fielded use of both the logic and its tools for the specification, verification, and synthesis of relevant classes of parametric systems. The detailed discussion of specific application profiles, some of which apparently out of the scope of the proposed methodology, will illustrate the remarkable modeling power of M2L(Str). This power, which marks it as a good candidate formalism for hardware/software codesign, combined with its fully automatic verification feature, indicate its high potential to enter industrial practice.

Detailed Organization of the Tutorial

The planned organization follows closely the outline of the content (see p. 3), and will be structured as follows:

1. Motivation (with examples from different application domains)
2. Basics of M2L: Syntax and Semantics, relation with automata, concept of 'regularity', decision procedure
3. Design of a family of M2L-based programming languages.
4. Example of domain-specific modelling: parametric systems.
   (a) Control structures (time-parameter)
   (b) Datapaths (structure-parameter)
   (c) Parametric sequential circuits (e.g. microprocessors): dealing at once with both control structures and parametric datapaths
   (when is it possible, which classes of systems, conditions)
   (d) Extending 'regularity'
      i. in the time dimension: modelling skewing (e.g. systolic systems)
      ii. in the structural dimension: capturing 'growing' interfaces
5. Complexity analysis: responsibilities for exponential blowups.
6. Design of an environment for M2L-based analysis and verification
7. Evaluation and Perspectives

Key learning objective

Analysis and discussion of a full design cycle: from a theory to its fielded application.

Relevance

Focussing on the theory-practice gap, its bridges and its opportunities, the tutorial directly addresses the main objective of ETAPS. The tutorial spans theory, practice and concrete application, and is therefore of potential interest to any ETAPS participant.

History

The presentation has been given before by the proposer in a variety of lengths (1-hour seminar, 6-hours short course, and in full detail during two 1-semester classes given 1996/97 at the University of Passau). The tutorial will be based on the short course material.

Duration

Default half day, possible also full day (in this case with online demo support).

Scheduling

Possibly non-overlapping with the CPN tutorial.

Audience

Target: Any ETAPS participant

Prerequisites: Basic Computer Science knowledge

Credentials of the instructor

10 years experience in the automatic verification of software and hardware systems. Leader of various projects in the area.

Outline of the Planned Content

Motivation

Generalization and reuse are two concepts underlying component-based system design. In particular, *families of components*, often given in terms of specific patterns, like n-arbiter, n-process scheduler, n-bit adder, systolic arrays of length n, etc., gain importance in modern system design. The corresponding parametric designs are intended to uniformly cover all instances of an *unbounded*, but *finite* parameter domain.

Monadic second-order logic on strings (M2L(Str)) provides a particularly attractive theory for parametric designs. It also conveniently combines two important features in a single formalism: it is an abstract specification language and an effective programming language. In fact, as a second-order predicate logic it admits high-level behavioral design capture in which one focusses on *what* is to be achieved rather than *how* this is done. Yet, in contrast to full higher-order logic, this expressiveness is not compromised by a loss of effectiveness. Every specification can be translated into an equivalent finite-state automaton, and thus is decidable and executable. In fact, it admits fully automatic verification of parametric designs, which typically seem to require proofs by induction.

Proposed more than 30 years ago by Alonzo Church as an appropriate decidable specification formalism for reasoning about sequences of bitvectors [3], it had been "forgotten" for its staggering non-elementary worst case complexity behaviour: in the worst-case the computational effort is a stack of exponentials of height proportional to the size of the formula. However, recent application to the analysis and verification of hardware systems has shown that relevant practical problems are usually far better behaved than the staggering worst-case complexity, and can be solved automatically in reasonable time. In fact, several implementations of the decision procedure already exist [7, 6], and in our experience a careful design reflecting an effort for efficiency can considerably help keeping manageable runtimes.

The tutorial will investigate M2L(Str) and its inherent structural restrictions (eg., only one parameter can be considered at a time) from the application point of view: considering a variety of case studies of increasing conceptual complexity it will illustrate that the range of M2L(Str)-based verification goes far beyond apparent hurdles, and that it well has the potential to enter industrial practice.

Most of the considered case studies concern parametric hardware components, because their classical transparent structural decomposition in time, space, and data aspects is ideal to illustrate the power of adequate modelling in M2L(Str). However, the same principles can also be applied e.g. to parametric components for distributed systems [6]. In fact, M2L(Str) can be regarded as a uniform description language for model-based analysis of software [11] as well as hardware systems [1, 10, 8, 12] and is therefore a good candidate formalism for model-based hardware/software codesign.

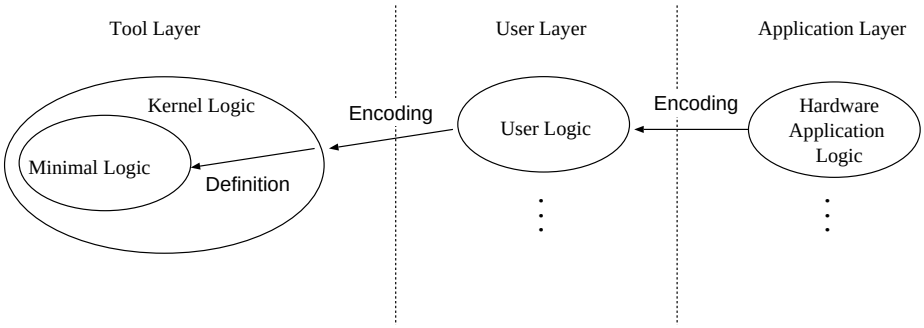

Fig. 1. Layered Logics in MOSEL

This wide range of application imposes strong demands on the tool level: different application domains come with different high-level modelling languages, have different standard patterns and require different feedback (error diagnosis). An adequate tool should take these demands into account. The tutorial will also address this aspect by sketching the MOSEL toolset, which arose directly under the pressure of these demands. In fact, the interplay between the investigation of varying application profiles and the design of the MOSEL toolset illustrates the potential of synergy between the hardware and the software community. Some of the design decision taken may appear minor at first sight, but they are, according to our experience, vital for successful technology transfer.

In the following we will first sketch the design decisions underlying the MOSEL toolset, and subsequently discuss and classify a variety of case studies in order to explore the range of the proposed fully automatic verification support.

The MOSEL Toolset: Design Decisions

MOSEL [7] is an environment for the analysis and verification in monadic second-order logic. Analysis and verification are based on model construction. Its aim is to offer a system-level open environment supporting several theories of the logic by means of a flexible set of decision procedures complemented by a variety of support components which provide input format translations, visualization, and interfaces to other logics and other analysis, verification, and synthesis tools.

The accent of the tutorial is the investigation of the conceptual application profile of M2L(Str)-based verification. Optimizations of the underlying individual decision procedures are only addressed at the side.

Modular design of a hierarchy of logics
It provides both soundness of the logics and comfort to different groups of users who 'program' in those logics. A hierarchy of logic layers, with increasingly powerful constructs, related by either direct embedding or more elaborate encodings as shown in Fig. 1, covers the following spectrum(see [7] for details):

– a reference language containing the minimal set of primitives for which the semantics is formally defined (the *minimal logic*),
– an extended language (the *kernel logic*) coinciding with the set of constructs actually implemented as primitives in the semantic decision procedure, whose design was tailored for the efficiency of the computations,
– an application-independent layer of general-purpose *user logics*, tailored for the user's comfort. They may be rather different from the kernel logic and need a compilation into the kernel logic [5],
– a domain-specific layer of *application logics*, each capturing via additional predicates and constructs the 'look and feel' of specific application domains. Here we briefly examine the domain of verification and synthesis of hardware, where we deal with families of parametric sequential circuits [8].

Modular tool design
Following a component-based design style [13], MOSEL is realized as a collection of modules which can be combined or exchanged at need. This way, it supports flexible adaptation and extension to new input or output formalisms, as well as the interchange of some of its internal components (e.g., users may exchange the BDD [2] package used in the decision procedure, or the automata minimization and determinization algorithms).

The aim is that, like in ETI [15], the best-fitting incarnations of tools may be put together at need, on an application-driven basis, from the collection of existing components, without need of programming. An example of the synergies arising from the flexibility of the free combination of tool functionalities is the use of formal methods as a means to personalize the presentation layer of an environment [9], in order to grant a *domain and user-specific* visualization of models and results.

The corresponding concrete implementation will be discussed at an abstract and methodological level, concentrating on

– the design decisions taken in the realization of the three groups of components (*decision procedures*, *translators* between different logics, and graphical *visualization* modules),
– their consequences and tradeoffs wrt. efficiency and flexibility, and
– their interplay.

Important are issues like syntax-independence of internal representations, organization of the object-oriented implementation of the decision procedures, choice of data structures for the representation of automata (e.g. implicit vs. explicit, role and kind of BDDs), organization of the embedding and of the translations between the different logics, definition of component's interfaces and interfacing components.

Case Study-Driven Exploration of the Application Profile

In this tutorial we focus on the specification and verification of reactive systems. The M2L(Str) application-level logic allows us to capture in a common framework a wide spectrum of abstraction levels, ranging from generic architecture or protocol levels [11] to hardware-oriented register transfer and gate levels [10, 8]. In particular, both behavioural and structural description styles are supported, and from both it is possible to carry out model-based analysis, verification, and error detection. Moreover, register-transfer and gate level circuits can be automatically obtained from the models with current hardware synthesis techniques. Discussing concrete case studies, we will show:

- how to *model* behavioral descriptions in M2L(Str), using different formulations of the behavior
- how to validate the specifications, by *verifying properties* of these behavioral descriptions like e.g., consistency, equivalence, or determinism
- how to use a model-based decision procedure for this logic to *construct the minimal model*, by computing its operational semantics in form of a finite-state automaton.

The expressive power of M2L(Str) captures only one-dimensional structures (linearly or circularly arranged). This is due to the interpretation of the logic over strings, which implies that the parameterization allowed to express generalized behaviours is limited to the generic "length" of strings. Since strings may be taken to assume different meanings (in [10, 11] sampled waveforms for control circuits, in [8] the bitwidth of a datapath), a degree of freedom in the use of the logic is still left to the application designer, e.g.,

- Parametrization over *time* allows the description of the sequential behavior of a single processor in the form of difference equations (typical in control-oriented modelling).
- Parametrization over *structure* is particularly suited for VLSI implementations where the same basic cell is often instanced many times to yield a regular structure (typical in datapath-oriented modelling).

The tutorial will also develop some more elaborate parameterization schemes, which allow us to capture problems, which seem to be out of the range of the one-dimensional modelling.

Structure of the Application Domain

We initially generalised the circuit libraries of the IFIP 10.2 benchmark set to comprise

- a full library of *parametric gates*
  (n-buf, n-inv, n-and, n-or, n-xor, n-nand, n-nor, n-xnor)
- *state-holding* devices (several descriptions of D-flipflop)
- a full library of *parametric RT-level components*
  (including n-mux, n-reg, n-shiftreg, n-comp, n-adder, n-ALU,...)

The corresponding implementations are based on commercial TTL components [4].

Example	System Type			Proof Style		
Hardware						
D-ff (timing)	comb.	prop/gate	c	t	f	–
parametric ALU	comb.	beh/gate	d	s	h	homog.
controllers	seq.	beh/gate	c	t	f	–
synchr. counters	seq.	beh/RT/gate	c+d	(t)+s	h	homog.
linear systolic array	seq.	gate/gate	c+d	t+s	h	heter.
MINMAX (microproc.)	seq.	beh/RT	c+d	(t)+s	h	homog.
Software - distributed systems						
load balance controller	HW/SW			multiparadigm		

Table 1. Classification of systems and proofs

A classification of the typologies of parametric systems dealt with so far is reported in Tab. 1. Here we summarize for a few sample case studies which will be discussed in the tutorial

- the *nature* of the system under consideration,
  - combinational (i.e. stateless) and sequential (with state) systems
  - their nature as controllers, datapaths (e.g. adder), or both
  - the abstraction level of their specification and implementation (e.g gate, register-transfer)
- a compact characterization of the modelling in M2L, like the nature of the interpretation of the parameter (time or structure), and
- a compact characterization of the proof technique (proof style)
  - flat or hierarchical,
  - homogeneous or heterogeneous, i.e. whether different steps of a hierarchical proof concern parameters of the same nature (e.g. all time parameters) or not (reasoning on structure at one level and on time on another)

This discussion should provide sufficient intuition in order for the audience to feel the power the proposed methodology. E.g., being able to capture *sequential iterative systems* of any kind (uni-, bidirectional or circular), as needed for the counters and the systolic array, exceeds the range of the classical induction-based approaches.

The fully automatic treatment of relevant classes of parametric circuits offered by the M2L(Str) logic is a central feature for the practicability of the method in an industrial environment: only push-button techniques are in fact widely acceptable by system designers. Moreover, user interaction must be possible completely within the application level. We therefore envisage a hierarchy of application level formalisms the syntax of which may coincide with decidable subsets of several widespread high-level specification formalisms.

References

1. D. Basin, N. Klarlund: *Hardware verification using monadic second-order logic*, Proc. CAV '95, Liège (B), July 1995, LNCS N. 939, Springer Verlag, pp. 31-41.
2. R.E. Bryant: "*Graph-based algorithms for boolean function manipulation*," IEEE Trans. Computing, vol. C-35(8), August 1986, pp. 677-691.
3. A. Church: "*Logic, arithmetic and automata*," Proc. Int. Congr. Math., Almqvist and Wiksells, Uppsala 1963, pp. 23-35.
4. *Databook of Analog and Synchronous Components*, Fairchild - 1993.
5. C. Gsottberger: *The Application Layer of the* MOSEL *Toolkit*, Master Thesis, Fakultät für Mathematik und Informatik, Universität Passau (D), Dec. 1997.
6. J. Henriksen, J. Jensen, M. Jørgensen N. Klarlund, R. Paige, T. Rauhe, A. Sandholm: "*Mona: Monadic second-order logic in practice*," Proc. of TACAS'95, Århus (DK), May 1995, LNCS 1019, Springer Verlag, pp. 89-110.
7. P. Kelb, T. Margaria, M. Mendler, C. Gsottberger: "MOSEL*: A Flexible Toolset for Monadic Second-Order Logic*," TACAS'97, Enschede (NL), April 1997, LNCS, Springer Verlag, to appear.
8. T. Margaria: *Fully Automatic Verification and Error Detection for Parameterized Iterative Sequential Circuits*, Proc. TACAS'96, Passau (D), March 1996, LNCS 1055, Springer Verlag, pp. 258-277.
9. T. Margaria, V. Braun: *Formal Methods and Customized Visualization: A Fruitful Symbiosis*, Proc. VISUAL'98, Int. Worksh. on *Visual Issues for Formal Methods*, satellite to ETAPS'98, März 1998, Lissabon (P), in "Services and Visualization: Towards User-Friendly Design', LNCS 1385, Springer Verlag, pp.190-207.
10. T. Margaria, M. Mendler: *Automatic treatment of sequential circuits in second-order monadic logic*, 4th GI/ITG/GME Worksh. on Methoden des Entwurfs und der Verifikation digitaler Systeme, Kreischa (D), March 1996, pp. 21-30, Shaker Verlag.
11. T. Margaria, M. Mendler: *Model-based automatic synthesis and analysis in second-order monadic logic*, R. Cleaveland and D. Jackson, eds., Proc. 1st ACM SIGPLAN Workshop on Automated Analysis of Software, Paris, January, 1997, pp.99–112.
12. T. Margaria, M. Mendler, C. Gsottberger: *Modelling and Verification of Unbounded Length Systolic Arrays in Monadic Second Order Logic*, Infinity'98 - Int. Workshop on Infinite State Systems, Aalborg (DK), 18 Juli 1998. (Proc. available as Techn. Rep. TU München, Juli 1998.)
13. T. Margaria, B. Steffen: *Coarse-grain Component Based Software Development: The* METAFrame *Approach*, Proc. STJA'97, Smalltalk und Java in Industrie und Ausbildung, 10.-11. September 1997, Erfurt (D), ISBN 3-00-001828-X, pp.29-34.
14. T. Margaria, B. Steffen: *Backtracking-free Design Planning by Automatic Synthesis in METAFrame* Proc. FASE'98, Int. Conf. on Fundamental Aspects of Software Engineering, Lissabon, Apr. 1998, LNCS 1382, pp.188-204, Springer Verlag.
15. B. Steffen, T. Margaria, V. Braun: *The Electronic Tool Integration Platform: Concepts and Design*, STTT, Int. Journal on *Software Tools for Technology Transfer*, Vol.1 (1/2), 1997, Springer Verlag, pp. 9-30. See also `http://eti.cs.uni-dortmund.de`.