UNIVERSITA' CA' FOSCARI DI VENEZIA
Dipartimento di Matematica Appl. ed Informatica
Technical Report Series on Computer Science

Rapporto di Ricerca CS98-12

Settembre 1998

A.Bossi, A.Cortesi, F.Levi (ed.)

Proceedings of the 2nd International Workshop on Verification, Model Checking, and Abstract Interpretation

Corso di Laurea in Informatica
Via Torino 155 – 30173 Mestre-Venezia

Proceedings

2nd International Workshop on Verification, Model Checking and Abstract Interpretation

A. Bossi, A. Cortesi, and F. Levi (editors)

September 19, 1998 - Pisa, Italy

Foreword

The 2nd Workshop on Verification, Model Checking and Abstract Interpretation follows the successful ILPS post-conference workshop held in Port Jefferson, NY, USA, October 1997.

Program verification aims at proving that programs meet their specifications, i.e., that the actual program behaviour coincides with the desired one. Model checking is a specific approach to the verification of temporal properties of reactive and concurrent systems, which has proven successful in the area of finite-state programs. Abstract interpretation is a method for designing and comparing semantics of programs, expressing various types of programs properties. In particular, it has been successfully used to infer run-time program properties that can be valuable to optimize programs. Clearly, among these three methods, there are similarities concerning their goals and their domains of applications. Furthermore, while much research has been performed in the area of abstract interpretation of logic programs, connections between model checking and logic programming have hardly been investigated as yet; at the same time it seems that there may be interesting directions in this area. Besides model checking of (concurrent) logic programs, one may also think of the use of specialized constraint (logic) solvers to tackle the model checking problem. The main goal of the workshop is that of enhancing cross-fertilization among these areas and in this way to clarify their relationships.

The program committee members were Krzysztof R. Apt (CWI Amsterdam), Annalisa Bossi (University Ca' Foscari di Venezia, Coordinator), Agostino Cortesi (University Ca' Foscari di Venezia), Yves Deville (Universite Catholique de Louvain), Gilberto File' (University of Padova), Gopal Gupta (New Mexico State University), Francesca Levi (University of Pisa), Jan Maluszynski (Linkoping University), Jens Palsberg (Purdue University), I.V. Ramakrishnan (SUNY Stony Brook), and David Schmidt (Kansas State University).

In response to the call for papers, 17 papers (extended abstracts) were submitted. All papers were rewiewed by at least three program committee members, and at the end 9 papers were selected for presentation.

We would like to thank all the program committee members and local organizers of the University of Pisa.

The Workshop is in conjunction with the Annual Meeting of the Compunet area: Language Design, Semantics and Verification, and it is sposored by the Italian MURST project 9701248444-044 *Tecniche formali per la specifica, l'analisi, la verifica, la sintesi e la trasformazione di sistemi software.*

Annalisa Bossi, Agostino Cortesi, and Francesca Levi

Table of Contents

Specification and Verification of Reactive Systems Using a Deductive Database *

Patrizia Asirelli and Stefania Gnesi
IEI - C.N.R., Pisa, Italy
{asirelli,gnesi}@iei.pi.cnr.it

August 28, 1998

Abstract

We present an approach to the specification and verification of concurrent systems, by means of a deductive database management system. The approach is based on the synthesis of logic formulas: starting from a temporal logic formula (ACTL logic formula), that represents the requirements of a system, a general model for such formula, is derived. From this model, all the concurrent systems satisfying the formula can be generated. Moreover, we show that this model can be used to verify when a given system, obtained elsewhere, satisfies its requirements expressed by logical specifications.

1 Introduction

The specification phase plays a fundamental role in the development of concurrent systems. This is the phase where the properties of systems are expressed and where the use of formal languages and formal methods is strongly recommended. Among formal languages, logic plays an important role, because it provides an abstract specification of concurrent systems. Indeed, different types of logics have been proposed for this purpose. In particular, modal and temporal logics, due to their ability to deal with notions such as **necessity**, **possibility** and **eventuality** have been recognized as a suitable formalism for specifying properties of concurrent systems [8]. Among them, we recall the action based version of CTL [6], ACTL[5]. The ACTL logic, being an action_based one, is more suitable than state_based logics such as CTL, to express properties of concurrent systems that are usually defined by the occurrence of actions over time. The models of ACTL formulas are Labelled Transition Systems (LTSs) which in their turn are suitable to formally specify concurrent systems [9]. The purpose of our work is the definition of an interpretation model (meta_model) for ACTL formulas that sits in between the ACTL logic itself and its usual interpretation domain, i.e. the labelled transition systems.
The meta_model will then be used for generating more detailed LTSs that represent different implementations of the system under specification or for analyzing already developed implementation of the given system as LTSs, against its formal specification. Both these functions will be performed in the realm of Logic Programming, in particular with the help of a the deductive database system Gedblog.

In particular, starting from a formal specification of a reactive system, given by means of an ACTL formula, we have defined and implemented in Gedblog a synthesis algorithm to derive a finite meta_model M which represents the minimal LTS satisfying the formula. Then two relations have been defined and implemented again in Gedblog: i) the *satisfiability* relation, from the ACTL logic to the meta_model, since it becomes an interpretation domain

*Acknowledgments: This work is partially founded by Progetto Integrato CNR-Comitato 2 and Progetto Coordinato CNR-Comitato 12 : "Programmazione Logica (Logic Progamming)" for the ACTL formulas; ii) the *derivability* relation from the meta_model into a labelled transition system, since the former is an abstraction of the latter.

In [2] we first started our work by taking into account a subset of ACTL which dealt only with "finite formulas" that is, formulas without until operators. Afterwards, we have completed the work by extending it to handle the totality of the ACTL logic but "negation". The implementation of this extended version is under development on a new version of the Gedblog system.

2 Background

In the following we briefly introduce the technical details on the ACTL logic and the Gedblog system.

2.1 ACTL

ACTL [5] is a branching time temporal logic that is suitable for describing the behavior of systems that perform actions during their working time. In fact, ACTL embeds the idea of "evolution in time by actions" and is suitable for representing the temporal sequences of actions that characterize a system. The syntax and the informal semantics of some of the ACTL operators is shown in Table 1; the grammar in this table has the state formula symbol μ as initial production. In the table, α is a single observable action belonging to *Act*, which is the set of actions that a given system is assumed to be able to perform. An execution (path) is a (finite or not) sequence of actions. A *state* represents a time in which a single action has been completed and a new next action may be performed. It is possible that there is more than one action that the system can perform, when its execution reaches a state. Each of these actions represents the beginning of an alternative continuation of the execution. A *state formula* gives a characterization about the possible ways an execution could continue after a state has been reached, while a *path formula* states some properties of a *single* execution.
The formal semantic of ACTL formulae is given over LTSs, which describe the behavior of a system in terms of states and labelled transitions relating states.

2.2 The Gedblog System

Gedblog [3, 1] is a Deductive database management system together with some graphical features.

Our aim is to use Gedblog to build an environment where a formal specification can be graphically represented, giving the basis on which an animation of the formal specification can be started.

Gedblog supports fast prototyping of applications that can take benefit from a declarative specification style. It is based on a logic language extended with the capabilities of:

- handling separate theories, i.e. separate pieces of knowledge;
- defining and executing transactions, i.e. compound updates to the theory in objects;
- defining and verifying integrity constraints.

More precisely,Gedblog is a deductive (logic) database, that can deal with basic knowledge management functionalities (storing, retrieving, querying), and besides it is enriched with several additional features:

- Integrity Constraints and Checks, to define the data model entities must fit in;
- Transactions, to enter the operational framework.

neta_model into a labelled
.
bset of ACTL which
tors. Afterwards, we
CTL logic but "negation
nt on a new version of the

CTL logic and the Gedblog

describing the behavior of
, ACTL embeds the
g the temporal sequence
rmal semantics of some of
able has the state formula
vable action belonging to
to be able to perform. An
represents a time in which
e performed. It is possible
when its execution reaches
native continuation of the
ossible ways an execution
ula states some properties

h describe the behavior of
s.

ther with some graphic

ormal specification can be
of the formal specification

benefit from a declarative
the capabilities of

;

to the theory in objects

an deal with basic
and besides it is enriched

entities must fit in

Action formulas			
Ω	::=	*true*	"any observable action"
		false	"no observable action"
		α	"the observable action α"
		$\neg\Omega$	"any observable action different from Ω"
		$\Omega \mid \Omega'$	"either Ω or Ω'"
Ω'	::=	Ω	
State formulas			
μ	::=	*true*	"any behavior is possible."
		false	"no behavior is possible."
		$\sim \mu$	" μ is impossible"
		μ & μ'	"μ and μ'"
		$E\theta$	"there exists a possible execution in which θ"
		$A\theta$	"for each of the possible executions θ"
μ'	::=	μ	
Path formulas			
θ	::=	$G\mu$	"at any time μ"
		$F\mu$	"there is a time in which μ"
		$[\mu\{\Omega\}U\{\Omega'\}\mu']$	"at any time Ω is performed and *also* μ, *until* Ω' is performed and then μ'"
		$X\{\tau\}\mu$	"an unobservable action is immediately performed and, after that, μ"
		$X\{\Omega\}\mu$	"Ω is immediately performed and, after that, μ"

Table 1: Some of the ACTL operators

Gedblog can manage logical theories that consist of different kinds of clauses:
Facts, Rules, Integrity Constraints, Checks and Transactions.
By means of the system-defined predicate *theory*, it is possible to perform inclusion among theories. In this way, given a *starting* theory Th, the associated Gedblog theory can be defined as the set-theoretic *union* of all the theories in the inclusion tree rooted in Th.

Gedblog was implemented in IC-Prolog and included a graphic specification language, i.e. an Input/Output graphic model (declarative, based on prototypes), to define graphics and interactions with graphic object.This version of the system was obtained by integrating the features of Motif and X11 in the Gedblog theories. GEDBLOG has been ported into Sicstus Prolog. More information on GEDBLOG, a demo version of the system Jedblog, (GEDBLOG with a Java interface) and a demo example can be found at: http://rep1.iei.pi.cnr.it/projects/GEDB/.

3 A logical approach to reactive systems specification and verification

When the specification of a concurrent system is given by means of a set of ACTL formulas, we associate to the conjunction of the set of formulas a finite *meta_model* M. A *meta_model* is an extension of the notion of an LTS in the sense that it subsumes all possible LTSs that are models of an ACTL formula. More precisely:

Def.1: A *Meta_model* M is a structure (MS,MD,S_0,V_s,T), where MS is a set of states, MD $\subseteq$ MS $\times$ MS is a set of transitions, $S_0 \in$ MS is the initial state, V_s is a set of constraints on states, represented as first order logic formulas and T $\subseteq$ MS $\times t$ is a finite set of pairs that associate to states of MS the "flag t", denoting the absence of further constraints over them.

States and transitions of the *meta_model* are called *meta_states* and *meta_transitions*.

The above definition of *meta_model* was introduced in [2] to derive a finite representation for each formula of "finite" ACTL. We here give an extension of such notion to deal with the full ACTL.

Def.2: A *Meta_model_OR* M is a structure (MS,MS',MD,MD',SO,V_s,T), where MS is a set of *meta_states*, MS' is a set of *meta_states_OR*, MD $\subseteq$ MS $\times$ MS, MD' $\subseteq$ MS' $\times$ MS $\cup$ MS' $\times$ MS', $SO \in$ MS $\cup$ MS' is the initial state, V_s is a set of constraints on *meta_states* and *meta_states_OR*, T $\subseteq$ MS $\times \{t\}$.

The above definitions of *meta_model* and *meta_model_OR* can be translated within Gedbolg by means of its logical theories handling capability as a set of states, labelled transitions and states constraints. In the following, we simply refer to *meta_models* indicating both structures defined above.

The satisfiability relation of ACTL formulas over *meta_models* has been defined and implemented. The meta_model associated to an ACTL formula μ is then derived incrementally from an initial state using a defined derivability relation. At each step a new *meta_model* is obtained by expanding the previous one according to the representation defined for the sub_formula μ_i of μ that is being analyzed. Moreover, during the generation of a *meta_model* some constraints, V_s, are inserted. They provide the properties that every LTS has to satisfy to be a model of the concurrent system under specification. The meta_model can then be used to obtain more refined models, all satisfying the initial specification, hence giving an animation of the possible implementations of the concurrent systems.

The approach can be divided into three steps:

i) **Synthesis Phase**: Given an ACTL formula, its *meta_model* is generated by Gedblog. More precisely, given the specification of a concurrent systems, in terms of ACTL formulas the associated *meta_model* is built using the Gedblog transaction mechanism.

ii) **Animation Phase**: The obtained *meta_model* becomes a new input for Gedblog to generate more detailed LTSs, each one satisfying the initial ACTL formula. That is, starting from the *meta_model* we build one or more LTSs, among the admissible ones, each one representing a correct implementation of the system.
The LTS are built according to the following steps:

   1. From S_0 only one state, q_0, can be derived that represents the initial state of the LTS under construction.
   2. Each state of the LTS is derived from a *meta_state* S_i of the given *meta_model*, this means that each *meta_state* can be splitted into a set of states, each one sharing its constraints.
   3. Transitions can be traced between a pair of states of the LTS under construction according to the constraints associated to the related *meta_model*.

iii) **Deductive Model Checking**: The previous phases mechanism allows also the analysis of an LTS to be supported. This means that we shall exploit all the deductive capabilities of Gedblog, by defining a logic program where the definition of "correctness" for an LTS is given. This correctness definition can then be used, to verify the admissibility of other LTS models (generated by means of other tools) with respect

to an ACTL formula. The *meta_model* associated to the formula is used to analyze if the structure of the LTS agrees with the constraints of the *meta_model*. In this case we can say that the LTS is a model of such formula.
To realize this phase we use rules and constraints, besides transactions.

For example let us consider the following ACTL formula:

$$\phi \;=\; EX_a\ (AX_{a\ \vee\ b}tt\ \&\ \ EX_b tt)\ or\ EX_a(AGAX_{a\ \vee\ b}EX_b tt)$$

Applying the procedure described in the above phases we will generate for ϕ the associated meta_model and a couple of LTSs that are both models for ϕ, as depicted in Figure 1.

4 Conclusions and related work

In this paper we have presented an environment to support the formal specification and verification of concurrent systems. The environment has been realized implementing a synthesis algorithm that allows the construction of a finite meta_model for each formula of a subset of the ACTL logic. The implementation is under development within the deductive database management system, Gedblog.

Our main purpose was to use a deductive database, that a deductive repository where constraints, modularization and guidance from the user are easy to handle.

To the author knowledge there are at least three other approaches to be mentioned related to the issues of specification and verification of concurrent system in the logic programming framework. [11, 10, 7]

In [11] a model checker has been defined by means of an extension of the classical tableau-based model checking procedures using deductive methods. There, starting with a general skeleton of the product graph between the system's reachable-state graph and the temporal tableau for the negation of the formula to be checked, the proof is carried on refining the graph until or a counterexample has been found or the impossibility of such a counterexample is shown.

The second [10] work uses the XSB tabled logic programming system for implementing efficient local model checking. In that paper they present an approach to a model checker for CCS-like value passing languages and the alternation-free fragment of the modal mu-calculus.

Finally the third [7] develops a general framework for the specification and verification of real-time system using Constraint Logic Programming (CLP) and the notion of timed automata.

References

[1] P. Asirelli, D. Di Grande, P. Inverardi and F. Nicodemi: **Graphics by a logical Database Management System"** *Journal of Visual Languages and Computing (1994),5,365-388.*

[2] P. Asirelli, S. Gnesi, M.C. Rossi: **A Deductive Database Support to the Specification of Concurrent Systems** *SOFSEM'96 , Lecture Notes in Computer Science, n. 1175, Springer-Verlag, (1996).*

[3] P. Asirelli, P. Inverardi, D. Aquilino, D. Apuzzo, G. Bottone, M.C. Rossi: **Gedblog Reference Manual.** *Revised Version: Nota interna B4-18; Aprile 1995.*

[4] R. De Nicola, A. Fantechi, S. Gnesi, G. Ristori: **An action-based framework for verifying logical and behavioural properties of concurrent system** *Computer Networks and ISDN Systems,25, (7), pp. 761-778, (1993).*

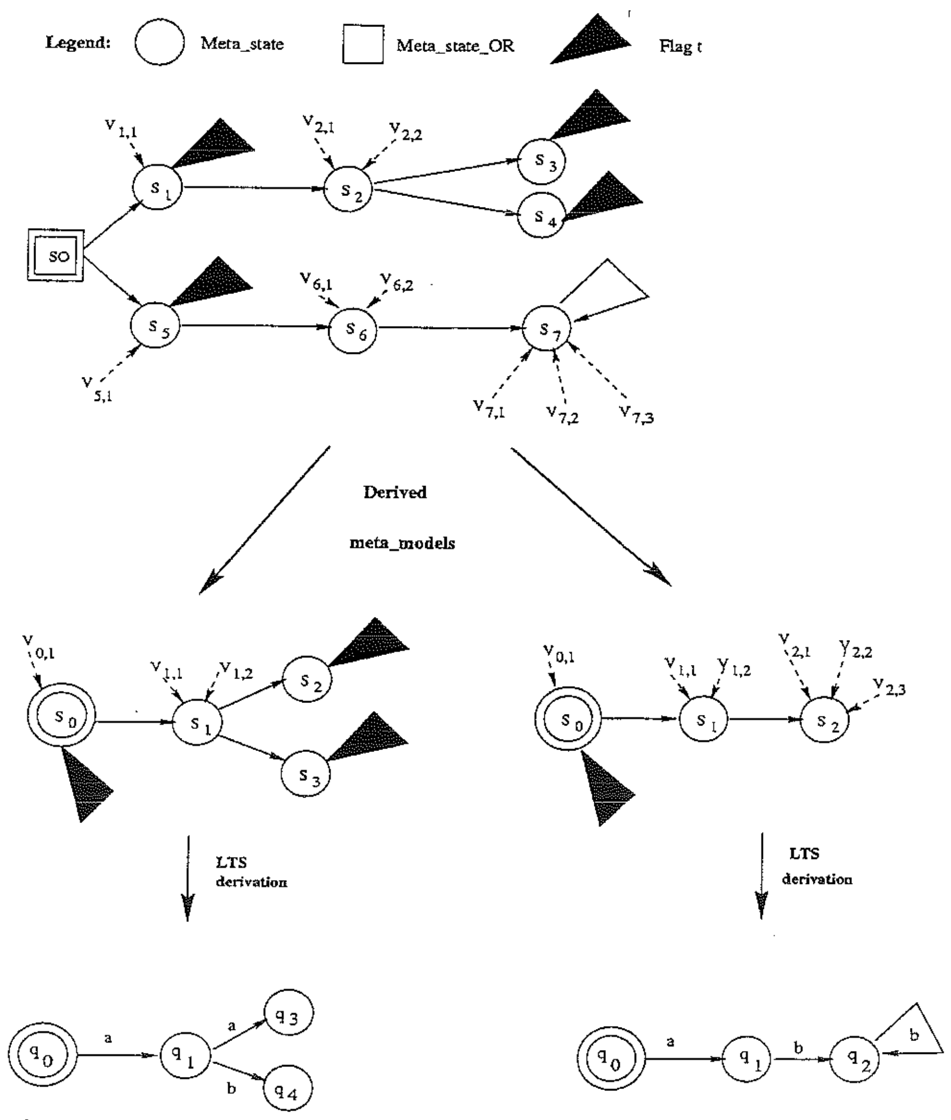
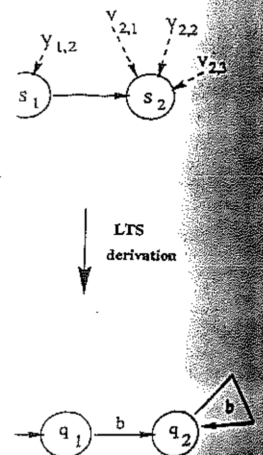


Figure 1 From Logical Specification to Implementations



[5] De Nicola, R. and Vaandrager, F. W. **Action versus State based Logics for Transition Systems. Proceedings Ecole de Printemps on Semantics of Concurrency.** *Lecture Notes in Computer Science*,**469**, Springer-Verlag, 407-419, (1990).

[6] Emerson, E. A. and Halpern, J. Y. **"Sometimes" and "Not Never" Revisited: on Branching Time versus Linear Time Temporal Logic.** *Journal of ACM*, **33** **(1)**, 151-178, (1986).

[7] G. Gupta and E. Pontelli **A Constraint Based Approach for Specification and Verification of Real-time Systems**, *IEEE Real Time Systems Symposium*, IEEE Computer Society, (1997).

[8] Manna, Z. and Pnueli, A. **The Anchored Version of the Temporal Framework, in Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency**, *Lecture Notes in Computer Science*,**354**, Springer-Verlag, 201-284, (1989).

[9] R. Milner: **Communication and Concurrency**, *Prentice Hall, (1989).*

[10] Y.S. Ramakrishna et al. **Efficient Model Checking using Tabled Resolution**, *Lecture Notes in Computer Science, n. 1254, Springer-Verlag, 143-154,(1997).*

[11] H.B. Simpa, T. E. Uribe, Z. Manna, **Deductive Model Checking**, *Lecture Notes in Computer Science, n. 1102, Springer-Verlag, 208-219, (1996).*

[12] M. C. Rossi: **Sistema logico detuttivo per il supporto allo svilutto di sistemi ed all'analisi di dati telemetrici** *Tesi di Laurea, Scienze dell'informazione, Università di Pisa, Febbraio 1996.*

[13] A. Fantechi, S. Gnesi, G. Ristori, M. Carenini, M. Vanocchi, P. Moreschini: **Assisting Requirement Formalization by Means of Natural Language Translation***Formal Methods in System Design, 4, 243-263 (1994)*