# Predicate diagrams as Basis for the Verification of Reactive Systems

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität
München

vorgelegt von
Cecilia E. Nugraheni
aus Surakarta, Indonesien

München, 9. Januar 2004

Gedruckt mit Unterstützung des Deutschen Akademischen Austauschdienstes

Erstgutachter: Prof. Dr. Fred Kröger
Zweitgutachter: Dr. Stephan Merz
Tag der mündlichen Prüfung: 13. Februar 20

Informatik

Cecilia E. Nugraheni

# Predicate diagrams as Basis for the Verification of Reactive Systems

*Typoskript-Edition*

HIERONYMUS
MÜNCHEN

Dissertation an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilans-Universität München



ISBN 3-89791-332-1

Als Typoskript gedruckt



Tonerbasierter Digitaldruck und binden: Herbert Hieronymus, München
Printed in Germany

**This is for you, Father.**

Like the name you gave me, this work is *esti nugraheni.**

* *esti nugraheni = this is really a gift of God.*

# Abstract

This thesis proposes a diagram-based formalism for verifying temporal properties of reactive systems. Diagrams integrate deductive and algorithmic verification techniques for the verification of finite and infinite-state systems, thus combining the expressive power and flexibility of deduction with the automation provided by algorithmic methods.

Our formal framework for the specification and verification of reactive systems includes the Generalized Temporal Logic of Actions (TLA*) from MERZ for both mathematical modeling reactive systems and specifying temporal properties to be verified. As verification method we adopt a class of diagrams, the so-called *predicate diagrams* from CANSELL et al.

We show that the concept of predicate diagrams can be used to verify not only *discrete systems*, but also some more complex classes of reactive systems such as *real-time systems* and *parameterized systems*. We define two variants of predicate diagrams, namely *timed predicate diagrams* and *parameterized predicate diagrams*, which can be used to verify real-time and parameterized systems.

We prove the completeness of predicate diagrams and study an approach for the generation of predicate diagrams. We develop prototype tools that can be used for supporting the generation of diagrams semi-automatically.

# Zusammenfassung

In dieser Arbeit schlagen wir einen diagramm-basierten Formalismus für die Verifikation reaktiver Systeme vor. Diagramme integrieren die deduktiven und algorithmischen Techniken zur Verifikation endlicher und unendlicher Systeme, dadurch kombinieren sie die Ausdrucksstärke und die Flexibilität von Deduktion mit der von algoritmischen Methoden unterstützten Automatisierung.

Unser Ansatz für Spezifikation und Verifikation reaktiver Systeme schließt die *Generalized Temporal Logic of Actions* (TLA*) von MERZ ein, die für die mathematische Modellierung sowohl reaktiver Systeme als auch ihrer Eigenschaften benutzt wird. Als Methode zur Verifikation wenden wir Prädikatendiagramme von CANSELL et al. an.

Wir zeigen, daß das Konzept von Prädikatendiagrammen verwendet werden kann, um nicht nur diskrete Systeme zu verifizieren, sondern auch kompliziertere Klassen von reaktiven Systemen wie Realzeitsysteme und parametrisierte Systeme. Wir definieren zwei Varianten von Prädikatendiagrammen, nämlich gezeitete Prädikatendiagramme und parametrisierte Prädikatendiagramme, die benutzt werden können, um die Realzeit- und parametrisierten Systeme zu verifizieren.

Die Vollständigkeit der Prädikatendiagramme wird nachgewiesen und ein Ansatz für die Generierung von Prädikatendiagrammen wird studiert. Wir entwickeln prototypische Werkzeuge, die die semi-automatische Generierung von Diagrammen unterstützen.

# Contents

# List of Figures

# Chapter 1

# Introduction

The dependency on its own technological achievements by modern society is getting more and more. Powerful computer systems, which are the backbones of almost conceivable technology today, are in the development or in the implementation stages. The complexity of these systems is growing ceaselessly. Most of today's computing systems are characterized by an ongoing interaction with their environments. This interaction occurs in various forms such as the transmission of data over a communication network to another machine, interaction with a human user, or the exchange of information with the sensors and actuators of an embedded control system. Such systems are called *reactive*, in contrast to *transformational* systems that compute an output from a given input.

Considering our dependency on these systems, it is clear that they should be correct. Reactive systems are most often composed of several communicating concurrent processes. The inherent complexity of concurrency and communication makes the discovery of design errors a difficult task. Not only may there be mistakes in the *calculations* such system perform (as in transformational systems), but there is also the possibility of *synchronization* failures (such as deadlocks, starvation, unexpected message reception etc.). One of the most challenging problems facing today's software engineers and computer scientists is therefore to find ways and establish techniques in order to reduce the number of errors in reactive systems.

This thesis presents a methodology for the formal analysis of some classes of reactive systems.

## 1.1 Classification of reactive systems

Reactive systems are commonly classified as *discrete, real-time* and *hybrid* [77]:

- A discrete system only represents the qualitative aspect of time, that is the order of events, but does not measure the time elapsed between these events. The behavior is fully described by the discrete events.
- A real-time system captures the metric aspects of time; discrete events may have time stamps.
- In hybrid systems, we allow the inclusion of variables that evolve continuously over time between discrete events. The evolution of the continuous variables is described separately from the discrete events, usually by differential equations.

The behavior of a reactive system can be characterized in different ways, for example by the stream of outputs produced by the system, or by the actions taken by the system. In this thesis, a reactive system is characterized by the sequence of states, which are interpretations of the variables traversed by the system. We call a system *finite state* if this set is finite, and otherwise *infinite state*. Of course all real-time and hybrid systems are infinite-state due to the presence of real-valued clock and continuous variables.

Reactive systems usually consist of a collection of processes running parallel in the systems. *Parallel systems* can be classified as *interleaving* and *non-interleaving* systems. An interleaving system is a system in which each step can be attributed to exactly one process. A non-interleaving system allows steps that represent simultaneous operations of two or more different processes. When a parallel system consists of a collection of identical processes, it is categorized as a *parameterized system.*

## 1.2 Formal specification and verification

Formal methods are a collection of notations and techniques for describing and analyzing systems. These methods are *formal* in the sense that they are based on some mathematical theories, such as logic, automata or graph theory. They are aimed at enhancing the quality of systems. Formal specification techniques introduce a precise and unambiguous description of the properties of systems. This is useful in eliminating misunderstanding and can be used further for debugging systems. Formal analysis techniques can

be used to verify that a system satisfies its specification or to systematically seek for cases where it fails to do so.

A formal framework for the specification and verification of reactive systems should include at least the following parts [78, 88]:

- a *mathematical model* of reactive systems,
- a *requirement specification* or *property* languages and
- a *verification method.*

**Model** The large majority of frameworks that include a verification method use transition systems as their computational model of reactive systems. This is certainly due to the simplicity of this model. A transition system is essentially a graph, where the nodes represent system's states and the edges represent the atomic transitions between these states. Concurrency is modeled by non-deterministic interleaving of atomic actions. Another important ingredient in the description of transition systems for modeling reactive systems are fairness and liveness conditions that require some actions to be eventually taken or some state to be eventually reached. These conditions help to balance the local non-determinism of transition systems of choosing among actions that are permitted at a given source state and lead to different possible target states.

**Property** In order to reason about the behavior of reactive systems, *temporal logic* was proposed as a convenient specification or property language. Temporal logics are extensions of classical (propositional and/or first-order) logics, incorporating a model of the flow of time, either as metric constraints or via a suitable semantics. Temporal logics are often classified according to whether time is assumed to have a *linear* or a *branching* structure.

**Verification** Verification of reactive systems consists of establishing whether a reactive systems satisfies its specification, that is, whether all possible behaviors of the system are included in the property specified. For finite-state systems and a restricted class of infinite-state systems, verification of temporal-logic properties is decidable: algorithms can be devised that determine in a finite number of steps whether a system satisfies its specification. For the vast majority of infinite-state systems no such algorithms exist; the problem is undecidable. In this case, verification relies on human interaction and heuristics.

The use of temporal logics for the specification and verification of reactive systems goes back to PNUELI's seminal paper on temporal logic [94]. Formulas of temporal logic are interpreted over runs of transition systems and can thus express properties of reactive systems. Many useful properties of reactive systems can be expressed in temporal logics, including safety properties ("nothing bad happens") and liveness properties ("something good happens").

## 1.3 Verification techniques

There are basically two approaches to verification of reactive systems: the algorithmic approach on one hand and the deductive approach on the other hand. When verifying temporal properties of reactive systems, algorithm methods are used when the problem is decidable and deductive methods are employed otherwise.

The most popular algorithmic verification method is *model checking*, independently proposed by CLARKE & EMERSON [27, 28] and QUEILLE & SIFAKIS [95]. A complete state graph of the system is built and specialized methods are used to check whether all paths through this graph conform to some properties. A *counterexample* is found whenever a path that does not satisfy a temporal property is encountered. Although this method is fully automatic for finite-state systems, it suffers from the so-called *state-explosion* problem. The size of the state space is typically exponential in the number of components, and therefore the class of systems that can be handled by this method is limited. State space reduction techniques such as symbolic representations [25, 82, 22], symmetry [29, 43] and partial order reductions [52, 92, 105] have yielded good results but the state spaces that can be handled in this manner are still quite modest.

*Automata-theoretic* verification methods [108, 66] are closely related to model checking, in which both the system and the property are represented by $\omega$-automata (automata on infinite words) and automata-theoretic methods are used to establish language inclusion.

On the other end of the spectrum we find deductive verification methods based on theorem proving; these methods are extension of the proof methods originally established by FLOYD [47] and HOARE [56] for sequential systems. They typically reduce the proof of a temporal property to a set of proofs of first-order verification conditions, which can then be dealt by standard theorem provers. Deductive methods are very powerful and generally applicable to infinite-state systems, but suffer from the high level of user interaction

required to complete a proof.

While it is clear that any way out of this impasse must rely on a combination of theorem proving and model checking, specific methodologies are needed to make such a combination work with a reasonable degree of automation.

## 1.4 Abstraction

An attractive method for proving a temporal property $\varphi$ for a reactive system $\mathcal{S}$ is to find a simpler abstract system $\mathcal{A}$ such that if $\mathcal{A}$ satisfies $\varphi$ then $\mathcal{S}$ satisfies $\varphi$ as well. In particular, if $\mathcal{A}$ is finite-state, the validity of $\varphi$ for $\mathcal{A}$ can be established automatically using a model checker, which may not have been possible for $\mathcal{S}$ due to an infinite or overly large state-space.

Thus, abstraction is a key methodology in combining deductive and algorithmic techniques. Abstraction can be used to reduce problems to model-checkable form, where deductive tools are used to construct valid abstract descriptions or to justify that a given abstraction is valid.

There is much work on the theoretical foundations of reactive system abstraction [31, 37, 74, 36, 53, 33, 75, 40], usually based on the ideas of abstract interpretation [34].

Most abstractions weakly preserve temporal properties: if a property holds for the abstract systems, then a corresponding property will hold for the concrete one. However, the converse will not be true: not all properties satisfied by the concrete system will hold at the abstract level. Thus, only positive results transfer from the abstract to the concrete level. This means, in particular, that abstract counter-examples will not always correspond to concrete ones.

## 1.5 Diagram-based verification

The deductive approach for verifying temporal properties of reactive systems is based on *verification rules*, which reduce the system validity of a temporal property to the general validity of a set of first-order *verification conditions*. While this methodology is complete, relative to the underlying first-order reasoning, the proofs do not always reflect an intuitive understanding of the system and its specification; without this intuition, the proofs can be difficult to construct.

The need for a more intuitive approach to verification leads to the use of *diagram-based formalisms*. Usually, these diagrams are graphs whose vertices

are labeled with first-order formulas, representing sets of system states, and whose edges represent possible system transitions. This approach combines some of the advantages of deductive and algorithmic verification: the process is goal-directed and incremental, and can handle infinite-state systems.

Some features shared by these formalisms are [19, 26]:

- Diagrams (or sequences of diagrams) are formal proof objects, which succinctly represent a set of verification conditions that replaces a combination of textual verification rules.
- The graphical nature of diagrams makes them easier to construct and understand than text-based proofs and specifications.
- Diagrams can describe and verify infinite-state systems using a finite and often compact representation.
- Diagrams can be viewed as the abstract representation of the systems being considered.
- The construction of a diagram can be incremental, starting from a high-level outline and then filling in details as necessary.
- The verification conditions are *local* to the diagram; failed verification conditions point to missing edges or nodes, or possible bugs in the system. The necessary global properties of diagrams can be proved algorithmically.
- Besides their use as a formal basis for verification, diagrams can also serve as support for explaining how systems are working and for documenting them.

There are some work using graphs to visualize and structure temporal proof, for example the diagram from OWICKI and LAMPORT [90], *proof charts* from COUSOT [35], *Predicate-action diagrams* proposed by LAMPORT [70], *verification diagrams* from MANNA & PNUELI [79], *generalized verification diagram* from SIPMA [99] and *predicate diagrams* from CANSELL et al. [26].

## 1.6 Scope of the thesis

In this work, three classes of reactive systems are considered: discrete systems, real-time systems and parameterized systems. We will use the Generalized Temporal Logic of Action (TLA*) from MERZ [83], which is a variant

of Temporal Logic of Action (TLA) from LAMPORT [69], to formalize our methodology. Our choice is due to its completeness; in the sense that it provides all the components of a formal framework for the specification and verification of reactive systems as mentioned in Section 1.2. Like in TLA, in TLA* there is no distinction between systems and properties, both are represented as formulas. It also provides proof systems that can be used to prove that a specification satisfies a desired property. This verification process is reduced to the proof that the specification implies the property.

In this thesis, we will follow the diagram-based verification techniques. The verification will be done by means of predicate diagrams from CANSELL et al. It is already shown that this diagram is suitable to TLA formalism [26].

The main goal of this thesis can be stated in the following questions:

1. How can reactive systems be represented in TLA*?
2. How can predicate diagrams be used to verify discrete systems?
3. What about the completeness of predicate diagrams, i.e. for any specification and any formula of the temporal propositional logic, if the specification implies the formula, can the implication be proven by a suitable predicate diagram?
4. How far can predicate diagrams be used to verify some other classes of reactive systems, in particular the more complex systems than discrete systems such as real-time systems and parameterized systems?
5. Is it possible to generate or to construct predicate diagrams automatically?

## 1.7 Chapter outlined

In Chapter 2 mathematical preliminaries are introduced, including set notations, strings and languages, graphs and classical logics, in order to establish the terminology and notational used in this book.

Chapter 3 is addressed to the properties of reactive systems. First, we give the very general definition of properties as arbitrary subsets of infinite states. Second, we give the definition of the syntax and semantics of TLA* introduced by MERZ [83]. We also present the general form of specification we will use in this sequel and introduce the writing style for writing specifications in the next sections.

In the following chapter, Chapter 4, we will consider a class of finite automata over infinite words, called Muller automata [86], including the formal definition and an algorithm for translating pTLA* formulas to a Muller automaton. We also briefly describe timed automata, which will be used in the verification of real-time systems.

Chapter 5 deals with the verification of discrete systems. We define the formula that will be used to represent the specification of discrete systems. We then present predicate diagrams, including the definition and the steps to be done in order to verify discrete systems using the diagrams. As illustration, we take the Bakery Algorithm. We show that predicate diagram is complete, i.e. for any specification and any formula of the temporal propositional logic, if the specification implies the formula, then there exists a suitable predicate diagram that can be used to prove the implication.

Chapter 6 is devoted to the specification and verification of real-time systems. First, we give the standard formula for real-time specification we use in this book. Second, we define a variant of predicate diagrams, which we call *timed predicate diagrams* that can be used to verify real-time systems. As illustration, we take the FISCHER's protocol problem.

The verification of parameterized systems will be considered in the following chapter. After defining the specification of parameterized systems, we explain how can predicate diagrams be used to verify the properties related to whole processes in the protocol. As a motivated example we take the Tickets protocol. We then define a variant of predicate diagrams called *parameterized predicate diagrams* that can be used to verify the property of a single process in the Tickets protocol.

In Chapter 8 the method for automatically generation of diagrams will be studied. It is started by briefly describing the concept of abstract interpretation and the algorithm of the diagrams generation. Two tools that have been developed in this work will be presented. Then it will be shown how these tools can be used in the generation of diagrams for the case studies presented in the previous chapters, namely the Bakery algorithm and the Ticket protocol.

The final chapter concludes the thesis with a review of its goals and their achievements and an outlook on future research.