Transaction Logic Programming*

(or, A Logic of Procedural and Declarative Knowledge)

Anthony J. Bonner†
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A4, Canada
bonner@db.toronto.edu

Michael Kifer‡
Department of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11790, U.S.A.
kifer@cs.sunysb.edu

Abstract

An extension of predicate logic, called *Transaction Logic*, is proposed, which accounts in a clean and declarative fashion for the phenomenon of state changes in logic programs and databases. Transaction Logic has a natural model theory and a sound and complete proof theory, but unlike many other logics, it allows users to *program* transactions. This is possible because, like classical logic, Transaction Logic has a "Horn" version which has a procedural as well as a declarative semantics. In addition, the semantics leads naturally to features whose amalgamation in a single logic has proved elusive in the past. These features include both hypothetical *and* committed updates, dynamic constraints on transaction execution, nondeterminism, and bulk updates. Finally, Transaction Logic holds promise as a logical model of hitherto non-logical phenomena, including so-called *procedural knowledge* in AI, *active databases*, and the *behavior* of object-oriented databases, especially methods with side effects. Apart from the applications to Databases and Logic Programming, we also discuss applications to a number of AI problems, such as planning, temporal specifications, and the frame problem.

Technical Report CSRI-323
March 1995

(a substantial revision of Report CSRI-270 of April 1992)

Computer Systems Research Institute
University of Toronto

---

*Available in the file *csri-technical-reports/270/report.ps* by anonymous ftp to *csri.toronto.edu*.

†Work supported in part by an Operating Grant from the Natural Sciences and Engineering Research Council of Canada and by a Connaught Grant from the University of Toronto.

‡Supported in part by NSF grant CCR-9102159 and a grant from New York Science and Technology Foundation. Work done during sabbatical year at the University of Toronto. Support of Computer Systems Research Institute of University of Toronto is gratefully acknowledged.

Contents

1 Introduction

We introduce a novel logic, called Transaction Logic (abbreviated $\mathcal{TR}$), that accounts in a clean, declarative fashion for the phenomenon of updating arbitrary logical theories, most notably, databases and logic programs. Unlike most logics of action, $\mathcal{TR}$ is a declarative formalism for specifying and executing procedures (if the user will permit the use of that oxymoron), procedures that update and permanently change a database, a logic program or, more generally, a logical theory. As a special case, transactions can be defined as logic programs. This is possible because, like classical logic, $\mathcal{TR}$ has a "Horn" version that has *both* a procedural and a declarative semantics, as well as an efficient SLD-style proof procedure. This paper presents the model theory and proof theory of $\mathcal{TR}$, and develops many of its applications.

$\mathcal{TR}$ was designed with several application in mind, especially in databases, logic programming, and AI. It was therefore developed as a general logic, so that it could solve a wide range of update-related problems. Individual applications were then carved out of different fragments of the logic. These applications, both practical and theoretical, are discussed throughout the paper, especially in Section 6, where many are developed in detail. We outline several of them here.

1. $\mathcal{TR}$ provides a logical account for many update-related phenomena. For instance, in logic programming, $\mathcal{TR}$ provides a logical treatment of the assert and retract operators in Prolog. This treatment effectively extends the theory of logic programming to include updates as well as queries. In object-oriented databases, $\mathcal{TR}$ can be combined with object-oriented logics, such as F-logic [51], to provide a logical account of *methods*—procedures hidden inside objects that manipulate these objects' internal states. Thus, while F-logic covers the structural aspect of object-oriented databases, its combination with $\mathcal{TR}$ would account for the behavioral aspect as well. Applications of $\mathcal{TR}$ to so called *active databases* are described in [23]. In AI, $\mathcal{TR}$ suggests a logical account of planning and design. STRIPS-like actions,[1] for instance, as well as many aspects of hierarchical and non-linear planning are easily expressed in $\mathcal{TR}$. Although there have been previous attempts to give these phenomena declarative semantics, until now there has been no unifying *logical* framework that accounts for them all.

2. Since $\mathcal{TR}$ is a full-fledged logic, it is more flexible and expressive than procedural systems in specifying transactions. Like procedural languages, $\mathcal{TR}$ is a language for combining simple actions into complex ones; but in $\mathcal{TR}$, actions can be combined in a greater variety of ways. In procedural languages, sequential composition is the only combinator, whereas in $\mathcal{TR}$, each logical operator combines actions in its own way. The result is that in $\mathcal{TR}$, one can specify transactions at many levels of detail, from the procedural to the declarative. At one extreme, the user may spell out an exact sequence of operations in excruciating detail. At the other extreme, he may specify loose constraints that the transaction must satisfy. In general, sequences and constraints can be arbitrarily mixed, and in this way, procedural and declarative knowledge are seamlessly integrated.

3. Because of its generality, $\mathcal{TR}$ supports a wide range of functionality in several areas. This functionality includes database queries and views; unification and rule-base inference; transaction and subroutine definition; deterministic and non-deterministic actions; static and dynamic

[1] STRIPS was an early AI planning system that simulated the actions of a robot arm [33].