Abstract: We propose an extension of classical predicate calculus, called \textit{Transaction Logic}, which provides a logical foundation for the phenomenon of state changes in logic programs and databases. Transaction Logic comes with a natural model theory and a sound and complete proof theory. The proof theory not only verifies programs, but also executes them, which makes this logic an ideal tool for declarative programming of database transactions and state-modifying logic programs. The semantics of Transaction Logic leads naturally to features whose amalgamation in a single logic has proved elusive in the past. These features include hypothetical and committed updates, dynamic constraints on transaction execution, non-determinism, and bulk updates. Finally, Transaction Logic holds promise as a logical model of hitherto non-logical phenomena, including so-called \textit{procedural knowledge} in AI, and the behavior of object-oriented databases, especially methods with side effects. This paper presents the semantics of Transaction Logic and a sound and complete SLD-style proof theory for a Horn-like subset of the logic.
5.1 INTRODUCTION

Updates are a crucial component of any database programming language. Even the simplest database transactions, such as withdrawal from a bank account, require updates. Unfortunately, updates are not accounted for by the classical Horn semantics of logic programs and deductive databases, which limits their usefulness in real-world applications. As a short-term practical solution, logic programming languages have resorted to handling updates with ad hoc operators without a logical semantics. To address this problem, this paper provides the theoretical foundations for logic programming with updates. This is accomplished in three ways: (i) we develop a general logic of state change, called Transaction Logic, including a natural model theory; (ii) we show that this logic has a "Horn" fragment, with both a procedural and a declarative semantics; and (iii) we show that programs in the logic can be executed by an SLD-style proof procedure in the logic-programming tradition. The result is a rule-based language with a purely logical semantics (and a sound-and-complete proof theory) in which users can program and execute database transactions. Moreover, in the absence of updates, this language reduces to classical Horn logic. It therefore represents a conservative extension of the logic programming paradigm.

Transaction Logic (or $\mathcal{TR}$ for short) is a general logic of state change that accounts for database updates and transactions and for important related phenomena, such as the order of update operations, transaction abort and rollback, savepoints, and dynamic constraints [BK95; Bon97c]. $\mathcal{TR}$ has applications in many areas, including databases, logic programming, workflow management, and artificial intelligence. These applications, both practical and theoretical, are discussed in detail in [BK95; Bon97c]. For instance, in logic programming, $\mathcal{TR}$ provides a clean, logical alternative to the `assert` and `retract` operators of Prolog. In relational databases, $\mathcal{TR}$ provides a logical language for programming transactions, for updating database views, and for specifying active rules. In object-oriented databases, $\mathcal{TR}$ can be combined with object-oriented logics, such as F-logic [KLW95], to provide a logical account of `methods`—procedures hidden inside objects that manipulate these objects' internal states [Kif95]. In AI, $\mathcal{TR}$ suggests a logical account of procedural knowledge and planning, and of subjunctive queries and counterfactuals.

Other Logics. On the surface, there would seem to be many other logics available for specifying database transactions, since many logics reason about updates or about the related phenomena of time and action. However, despite a plethora of action logics, researchers continue to complain that there is no clear declarative semantics for updates either in databases or in logic program-