Results on Reasoning about Updates in Transaction Logic

Anthony J. Bonner[1] and Michael Kifer[2]

[1] Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada, bonner@db.toronto.edu

[2] Department of Computer Science, SUNY at Stony Brook, Stony Brook, NY 11794, U.S.A., kifer@cs.sunysb.edu

Abstract. Transaction Logic was designed as a general logic of state change for deductive databases and logic programs. It has a model theory, a proof theory, and its Horn subset can be given a procedural interpretation. Previous work has demonstrated that the combination of declarative semantics and procedural interpretation turns the Horn subset of Transaction Logic into a powerful language for logic programming with updates [BK98,BK94,BK93,BK95]. In this paper, we focus not on the Horn subset, but on the full logic, and we explore its potential as a formalism for *reasoning* about logic programs with updates. We first develop a methodology for specifying properties of such programs, and then provide a sound inference system for reasoning about them, and conjecture a completeness result. Finally, we illustrate the power of the inference system through a series of examples of increasing difficulty.

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. In previous work, we addressed this problem by developing a general logic of state change called *Transaction Logic* (abbreviated $\mathcal{TR}$). Like classical logic, $\mathcal{TR}$ has a Horn fragment that supports logic programming; but unlike classical logic programs, programs in $\mathcal{TR}$ can *update* the database as well as evaluate queries. Previous research on $\mathcal{TR}$ has focussed on the specification and execution of logic programs in this Horn fragment [BK98,BK94,BK93,BK95]. In contrast, this paper focuses not on the Horn fragment, but on the full logic. In particular, we show that the full logic can be used to express properties of $\mathcal{TR}$ logic programs and to reason about them.

Because it is a general logic of state change, $\mathcal{TR}$ can be applied in many areas, including databases, logic programming, workflow management, and artificial intelligence. These applications, are discussed in detail in [BK95,Bon97b,DKRR98]. 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.

The results in this paper can be applied to reasoning about change in any of these areas. For instance, one can write logical formulas stating that a transaction program preserves the integrity constraints of a database, or that it can execute without aborting. One can also specify the conditions under which a transaction program will produce a given outcome. It is also possible to reason about the effects of a transaction program on a database with null values or other sources of indefinite information, such as disjunction. In addition to database transactions, $\mathcal{TR}$ can reason about actions in an AI context. For instance, one can axiomatize the elementary actions of a robot, and then reason about the effects of such actions. One can also combine simple actions into complex programs using sequential composition, pre-conditions, post-conditions, non-determinism, and subroutines, and reason about these. In AI terminology, this reasoning takes place in *open worlds*, that is, in the absence of the closed world assumption. The assumption of open worlds separates the theory of reasoning developed in this paper, from the theory of execution developed in [BK98,BK93,BK95], which is based on closed worlds.

Although the inference system developed here can deal with relatively complex properties of transaction programs, we do not claim that such reasoning goes beyond what other reasoning systems can do. However, unlike many systems, $\mathcal{TR}$ provides a declarative and operational semantics for logic programs with updates. Our purpose here is to reason about such programs. In particular, (*i*) we begin developing a methodology for reasoning about $\mathcal{TR}$ programs, (*ii*) we use $\mathcal{TR}$ itself as the basic reasoning system, and (*iii*) we illustrate the simplicity and elegance of doing so. This, we hope, will establish $\mathcal{TR}$ not only as a language for programming databases transactions, but also as a logic for reasoning about the properties of such programs.

For the interested reader, an implementation of the logic-programming fragment of $\mathcal{TR}$ is described in [Hun96], and an implementation of the reasoning system is under development. Extensions of $\mathcal{TR}$ for dealing with concurrency and communication are described in [BK96,Bon97b], and complexity results are given in [Bon97a]. Additional information on $\mathcal{TR}$, including a tutorial introduction to the implementation, and benchmark tests, is available at the Transaction Logic Web page: `www.cs.toronto.edu/~bonner/transaction-logic.html`

2 Overview of Transaction Logic

This section defines the syntax of full $\mathcal{TR}$ and of the logic-programming fragment, and describes their semantics informally and through examples. The formal semantics is presented in Section 3.

2.1 The General Logic

As in first-order classical logic, the language of $\mathcal{TR}$ includes three infinite, enumerable sets of symbols: a set $\mathcal{F}$ of function symbols, a set $\mathcal{V}$ of variables, and a set $\mathcal{P}$ of predicate symbols. Each function and predicate symbol has an associated *arity*, indicating how many arguments the symbol takes. Constants are viewed as 0-ary function symbols, and propositions are viewed as 0-ary predicate symbols. We also adopt the Prolog convention that variables begin in upper case, and predicate and function symbols begin in lower case. Function terms and atomic formulas are defined as usual in first-order logic. For brevity, atomic formulas are also referred to as *atoms*. To build complex logical formulas, $\mathcal{TR}$ extends classical logic with two logical operators: serial conjunction, denoted $\otimes$; and executional possibility, denoted $\diamond$. If ϕ and ψ are logical formulas, then so are $\phi \otimes \psi$ and $\diamond\phi$. Logical formulas in $\mathcal{TR}$ are called *transaction formulas*, and a set of transaction formulas is called a *transaction base* (since we will use them to define database transactions).

Whereas classical logic makes assertions about a static world, Transaction Logic makes assertions about a changing world. Formally, transaction formulas are interpreted on sequences of states, called *paths*. For instance, the atom $tues$ is true on paths representing Tuesdays. Likewise, the atom $withdraw(amount, account)$ is true on paths representing the withdrawal of money from a bank account. The classical connectives have a familiar interpretation. For instance, if formulas α and β are true on the same path, then the formula $\alpha \wedge \beta$ is also true on this path. Each classical connective is interpreted in terms of a single path in this fashion. In contrast, the non-classical connectives are interpreted in terms of multiple paths. If α and β are true on two adjacent paths, then $\alpha \otimes \beta$ is true on their concatenation. Also, if α is true on a path, then $\diamond\alpha$ is true on the first state of the path. Note that like the nodes of a graph, the states in Transaction Logic are not linearly ordered and can belong to many paths. Thus, the formula $\diamond\alpha$ intuitively means "*α can start now.*" These ideas are made precise in Section 3.1.

To illustrate, we give some simple examples of propositional transaction formulas and their intuitive meanings. The first two examples use classical connectives only. Their meaning in Transaction Logic is the same as in classical logic.

– A day is a Monday, a Tuesday, a Wednesday, ..., or a Sunday:

$$day \leftrightarrow mon \vee tues \vee wed \vee thurs \vee fri \vee sat \vee sun$$

– A day cannot be both a Monday and a Tuesday:

$$day \;\rightarrow\; \neg(mon \wedge tues)$$

The next examples uses serial conjunction to make statements about the order and structure of temporal phenomena.

– A weekend is a Saturday followed by a Sunday:

$$weekend \;\leftrightarrow\; sat \otimes sun$$

– Monday and Wednesday are not consecutive:

$$\neg(mon \otimes weds) \;\wedge\; \neg(weds \otimes mon)$$

– The only pairs of consecutive days are Monday-Tuesday, Tuesday-Wednesday, Wednesday-Thursday, etc.

$$\begin{aligned} day \otimes day \;\rightarrow\; & mon \otimes tues \;\vee\; tues \otimes weds \;\vee\; weds \otimes thurs \;\vee \\ & thurs \otimes fri \;\vee\; fri \otimes sat \;\vee\; sat \otimes sun \;\vee\; sun \otimes mon \end{aligned}$$

This formula orders the days of the week. The order is circular, not linear, since there is no first or last day. Eliminating the disjunct $sun \otimes mon$ would imply a linear order in which Monday is first and Sunday is last.

– A week is a sequence of seven consecutive days beginning with Sunday:

$$week \;\leftrightarrow\; sun \otimes day \otimes day \otimes day \otimes day \otimes day \otimes day$$

In this formula, each occurrence of the atom day represents a different day of the week. Also, the atom $week$ is false on any path with less than seven days. This illustrates a general property of TR: a formula can be true on a path, and false on all its subpaths.

Finally, the next two examples use the $\diamond$ modality to make statements about what is possible in the immediate future.

– February 28 is a day that can be followed by March 1:

$$feb28 \;\;\rightarrow\;\; (day \otimes \diamond mar1)$$

– February 28 and February 29 are the only days that can be followed by March 1:

$$(feb28 \vee feb29) \;\;\leftarrow\;\; (day \otimes \diamond mar1)$$

The above examples illustrate the *declarative* semantics of $\mathcal{TR}$. In addition, the Horn-like fragment of $\mathcal{TR}$ has an equivalent *procedural* semantics, just as the Horn fragment of classical logic does. With this semantics, many transaction formulas can be viewed imperatively, that is, as commands to execute actions. For instance, the atom $withdraw(amount, account)$ can be viewed as a command to withdraw an amount of money from a bank account. In this light, logical connectives become action constructors. For instance, the formula $\alpha \otimes \beta$ now means "*first do* α*, and then do* β." Likewise, the formula $\alpha \vee \beta$ means "*do* α *or do* β," and the formula $\exists X\, \alpha(X)$ means "*do* $\alpha(x)$*, for some value of* x." Even negative formulas can be treated imperatively: the formula $\neg\alpha$ simply means "*do not do* α." This interpretation of transaction formulas provides the basis for a logic programming language for database transactions, as described below.

A Specialized Theory. Transaction Logic does not distinguish different sorts of predicate symbol, just as classical logic does not. However, neither logic precludes the possibility of defining such sorts in order to develop more specialized theories. For instance, the theory of deductive databases distinguishes two sorts of predicate symbol within classical logic: base and derived. Intuitively, base predicates represent stored data, and derived predicates represent virtual data, or database views. The theory of change developed in this paper also distinguishes these two sorts of predicates. In addition, we adopt some standard terminology from deductive databases. For instance, a formula is *ground* if it has no variables, and a ground atomic formula is sometimes referred to as a *fact*. A *base fact* is a fact with a base predicate symbol, and a *derived fact* is a fact with a derived predicate symbol. A *database state* is a set (finite or infinite) of base facts. We sometimes refer to a database state simply as a *database* or a *state*.

In addition to base and derived predicates, the theory developed in this paper distinguishes a third sort of predicate, called *elementary transitions*. Intuitively, an elementary transition is an atomic update that transforms a database from one state into another. The theory does not depend on any particular set of elementary transitions, just as it does not depend on any particular set of base or derived predicates. However, to keep the presentation concrete, this paper focuses on two kinds of elementary transition. Specifically, for each base predicate, p, we introduce two new predicates, denoted $p.ins$ and $p.del$, whose arities are the same as p. Intuitively, these two elementary transitions represent the insertion and deletion of stored facts in a database.

To illustrate these ideas, and the procedural interpretation of transaction formulas, suppose that p and q are base predicate symbols of arity 1. Then, the atom $p.ins(b)$ intuitively means "*insert the fact* $p(b)$ *into the database*," and the atom $p.del(b)$ means "*delete the fact* $p(b)$ *from the database*." In addition, here are several transaction formulas and their procedural interpretations:

- $p.del(b) \otimes q.ins(p)$ means "*first delete* $p(b)$ *from the database, and then insert* $q(b)$ *into the database*."
- $p(b) \otimes p.del(b)$ means "*first check that* $p(b)$ *is in the database, and then delete* $p(b)$ *from the database*."
- $p.ins(b) \vee q.ins(a)$ means "*insert* $p(b)$ *into the database, or insert* $q(a)$ *into the database*."
- $[p.ins(a) \otimes p.ins(b)] \vee [q.del(a) \otimes q.del(b)]$ means "*do one of the following: insert* $p(a)$ *and then insert* $p(b)$*; or delete* $q(a)$ *and then delete* $q(b)$."
- $\exists X\, p.del(X)$ means "*delete* $p(x)$ *from the database, for some* x."
- $\exists X\, [q(X) \otimes p.ins(X)]$ means "*first retrieve* $q(x)$ *from the database, and then insert* $p(x)$ *into the database, for some* x."

2.2 Transaction Logic Programming

Like classical logic, Transaction Logic has a Horn fragment, called *serial-Horn* $\mathcal{TR}$, in which logical formulas can be interpreted as programs. As in classical Horn logic, the procedural semantics of serial-Horn $\mathcal{TR}$ is based on an

SLD-style proof procedure that executes programs by proving theorems, in the logic programming tradition. However, unlike classical logic programming, the proof procedure for serial-Horn $\mathcal{TR}$ does more than just evaluate queries: it also *updates* the database. These updates are handled in a completely logical manner, and are an essential part of the proof theory of $\mathcal{TR}$. This section defines the syntax of serial-Horn $\mathcal{TR}$, and illustrates its use in programming database transactions and robot actions. A detailed development can be found in [BK98,BK94,BK93,BK95].

Serial-Horn $\mathcal{TR}$ is based on the idea of *serial goals*. A serial goal is a transaction formula of the form $a_1 \otimes a_2 \otimes ... \otimes a_n$, where each a_i is an atomic formula and $n \geq 0$. A *serial-Horn rule* has the form $b \leftarrow a_1 \otimes a_2 \otimes ... \otimes a_n$, where the body, $a_1 \otimes a_2 \otimes ... \otimes a_n$, is a serial goal and the head, b, is an atomic formula. In addition, the theory of change developed in this paper requires that b have a derived predicate symbol. Finally, a serial-Horn transaction base is simply a set of serial-Horn rules. Observe that a serial-Horn transaction base can be transformed into a classical Horn rulebase by replacing each occurrence of $\otimes$ by $\wedge$. This transformation changes the serial-Horn rule $b \leftarrow a_1 \otimes ... \otimes a_n$ into the classical Horn rule $b \leftarrow a_1 \wedge ... \wedge a_n$. In the absence of updates, this transformation is a logical equivalence. Classical Horn logic is thus a special case of serial-Horn $\mathcal{TR}$.

Specifying and executing logic programs in serial-Horn $\mathcal{TR}$ is similar to using Prolog. To specify programs, the user writes a set of serial-Horn rules (the transaction base). These rules define database transactions, including queries, updates, or a combination of both. To execute programs, the user submits a logical formula to a theorem-proving system, which also acts as a run-time system. This system executes transactions, updates the database, and generates query answers, all as a result of proving theorems. As in Prolog, rules are evaluated in a top-down fashion, rule premises are evaluated from left to right, and if a rule premise fails, then the program defined by the rule also fails.

Unlike Prolog, $\mathcal{TR}$ provides a basic property of database transactions, namely *atomicity*: the appearance that a program either executes to completion or not at all. Atomicity allows a complex program to be treated as an atomic or elementary operation. Unfortunately, Prolog lacks this basic transactional feature. Consequently, state-changing procedures in Prolog are the most difficult to understand, debug and maintain, and their semantics is heavily dependent on rule order. Moreover, even if the operational semantics of Prolog did support atomicity (as in $\mathcal{TR}$), a logical semantics would still be needed to account for it and for updates. This semantics is provided by the model theory of $\mathcal{TR}$. Operationally, when a $\mathcal{TR}$ program fails, the theorem prover aborts the program's execution, and rolls back *both* the program state and the database state to the last choice point (or save-point), from whence execution resumes. In this way, the theorem prover guarantees atomicity by implementing other transactional features, such as abort, rollback, and save-points [Bon97b]. These features are all supported by the implementation of serial-Horn $\mathcal{TR}$ described in [Hun96].

Example 1. **(Financial Transactions)** Suppose the balance of a bank account is given by the base predicate $balance(Acct, Amt)$. The transaction base below defines four derived predicates, each representing a transaction program: $change(Acct, Bal_1, Bal_2)$ changes the balance of account $Acct$ from Bal_1 to Bal_2; $withdraw(Amt, Acct)$ withdraws an amount from an account; $deposit(Amt, Acct)$ deposits an amount into an account; and $transfer(Amt, Acct_1, Acct_2)$ transfers an amount from account $Acct_1$ to account $Acct_2$.

$$
\begin{aligned}
&transfer(Amt, Acct_1, Acct_2) \leftarrow withdraw(Amt, Acct_1) \otimes deposit(Amt, Acct_2)\\
&withdraw(Amt, Acct) \leftarrow balance(Acct, Bal)\\
&\qquad\qquad\qquad\qquad\qquad \otimes Bal > Amt \otimes change(Acct, Bal, Bal - Amt)\\
&deposit(Amt, Acct) \leftarrow balance(Acct, Bal) \otimes change(Acct, Bal, Bal + Amt)\\
&change(Acct, Bal_1, Bal_2) \leftarrow balance.del(Acct, Bal_1) \otimes balance.ins(Acct, Bal_2)
\end{aligned}
$$

These rules can be interpreted in a typical logic-programming style. The first rule says: to transfer an amount, Amt, from $Acct_1$ to $Acct_2$, first withdraw Amt from $Acct_1$; and then, if the withdrawal succeeds, deposit Amt in $Acct_2$. Likewise, the second rule says, to withdraw Amt from an account $Acct$, first retrieve the balance of the account; then check that the account will not be overdrawn by the transaction; then, if all is well, change the balance from Bal to $Bal - Amt$. The last rule changes the balance of an account by deleting the old balance and then inserting the new one. Observe that the *transfer* transaction fails (aborts) if either of the subtransactions *withdraw* or *deposit* fails. Likewise, the *withdraw* transaction fails if the query $balance(Acct, Bal)$ fails (*e.g.*, if $Acct$ is not a valid account) or if the test $Bal > Amt$ fails (*i.e.*, if the balance is not big enough).

Example 2. **(Non-deterministic, Recursive Robot Actions)** The transaction base below simulates the movements of a robot arm in a world of toy blocks. States of this world are defined in terms of three base predicates: $on(x, y)$, which says that block x is on top of block y; $isclear(x)$, which says that nothing is on top of block x; and $wider(x, y)$, which says that x is strictly wider than y. The rules define four derived predicates representing actions that change the state of the world.

$$
\begin{aligned}
&stack(N, X) \leftarrow N > 0 \otimes move(Y, X) \otimes stack(N-1, Y)\\
&stack(0, X) \leftarrow\\
&move(X, Y) \leftarrow pickup(X) \otimes putdown(X, Y)\\
&pickup(X) \leftarrow isclear(X) \otimes on(X, Y) \otimes on.del(X, Y) \otimes isclear.ins(Y)\\
&putdown(X, Y) \leftarrow wider(Y, X) \otimes isclear(Y)\\
&\qquad\qquad\qquad\qquad \otimes on.ins(X, Y) \otimes isclear.del(Y)
\end{aligned}
\tag{1}
$$

The actions $pickup(X)$ and $putdown(X, Y)$ mean, respectively: "*pick up block* X" and "*put down block* X *on top of block* Y*, where* Y *must be wider than*

X." The effects of both actions are specified in terms of elementary inserts and deletes to database relations. The remaining rules combine simple actions into more complex ones. For instance, $move(X, Y)$ means, "move block X to the top of block Y," and $stack(N, X)$ means, "stack N arbitrary blocks on top of block X."

The actions *pickup* and *putdown* are deterministic, since each set of argument bindings specifies only one robot action.[1] In contrast, the action *stack* is *non-deterministic*. To perform this action, the theorem prover searches the database for blocks that can be stacked (represented by variable Y). If, at any step, several such blocks can be placed on top of the stack, the system arbitrarily chooses one of them.

3 Herbrand Semantics

As described earlier, $\mathcal{TR}$ provides a general framework for combining elementary database operations into complex transactions. This framework treats database states and elementary operations abstractly: a state can be any abstract data type and an elementary operation can be any mapping from states to states. The theory of change developed in this paper instantiates this framework in a particular way, by assuming a particular set of states, and a particular set of elementary operations: a state in this paper is a set of base facts, and an elementary operation may insert a base fact into a state or delete a base fact. All of the results developed for $\mathcal{TR}$ in general are valid for any instantiation of the framework, and in particular, for the theory of change developed here.

In [BK98,BK94,BK93,BK95], a general model-theoretic semantics for $\mathcal{TR}$ is developed, along with an equivalent Herbrand-style semantics. In addition, for the serial-Horn fragment of $\mathcal{TR}$, the Herbrand semantics can be formulated as a least-fixpoint theory, in the logic programming tradition. This theory shows that every serial-Horn transaction base has a unique minimal model. This model is a central feature of our theory of reasoning, and provides an essential link with the theory of execution developed in [BK98,BK93,BK95]. This section reviews the basic ideas, and presents a specialized version of the Herbrand semantics for the particular theory of change developed in this paper.

As we shall see, Transaction Logic is an extension of first-order classical logic. The notions of classical entailment and classical satisfaction therefore arise frequently in this paper. We use the symbol $\models^c$ to denote both notions, where the distinction between satisfaction and entailment will be clear from context.

3.1 Model Theory

In the Herbrand semantics of $\mathcal{TR}$, the domain and interpretation of function symbols are determined by the language of the logic. The Herbrand universe $\mathcal{U}$ is the set of all ground first-order terms that can be constructed from the

[1] Assuming that only one block can be on top of another block.

function symbols in the language; the Herbrand base $\mathcal{B}$ is a set of all ground atomic formulas in the language; and a classical Herbrand structure is any subset of $\mathcal{B}$. Notice that the Herbrand universe and Herbrand base are infinite, fixed, and do not depend on the transaction base (or the database). Intuitively, this means that the Herbrand universe does not represent a closed domain. As a consequence, unlike classical logic, the Herbrand semantics of $\mathcal{TR}$ is equivalent to the general semantics (as developed in [BK98,BK93,BK95]).[2] A central feature in the semantics of $\mathcal{TR}$ is the notion of a *path* and the associated operation of *splitting* a path into two subpaths.

Definition 1. (Paths and Splits) *A* path *of length k, or a* k-path*, is any finite sequence of states, $\pi = \langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_k \rangle$, where $k \geq 1$. A* split *of π is any pair of subpaths, π_1 and π_2, such that $\pi_1 = \langle \mathbf{D}_1 \ \ldots \ \mathbf{D}_i \rangle$ and $\pi_2 = \langle \mathbf{D}_i \ \ldots \ \mathbf{D}_k \rangle$ for some i ($1 \leq i \leq k$). In this case, we write $\pi = \pi_1 \circ \pi_2$.*

Logical formulas in $\mathcal{TR}$ are evaluated on paths. To formalize this idea, we define the basic semantic structures of $\mathcal{TR}$ to be *mappings* from paths to classical Herbrand structures (Definition 2). Such a mapping specifies what atoms are true on what paths. Building on this, we then define what formulas are true on what paths (Definition 3). Finally, we develop the standard logical notions of model, entailment and validity for $\mathcal{TR}$.

To simplify the technical development, we augment classical Herbrand structures with an additional, abstract structure, denoted $\top$. We define every first-order formula to be true on $\top$. This abstract structure is a technical device that allows the semantic structures in $\mathcal{TR}$ to be defined as *total* mappings. Without $\top$, *partial* mappings would be needed to capture the right semantics. Although the resulting logic would be equivalent, partial mappings would significantly complicate the technical development.

Definition 2. (Herbrand Path Structures) *A* Herbrand path structure *is a mapping $\mathbf{M}$ that assigns a classical Herbrand structure (or $\top$) to every path. This mapping is subject to the following restriction, for all states $\mathbf{D}$ and $\mathbf{D}_i$, and every base fact p:*

1. $\mathbf{M}(\langle \mathbf{D} \rangle) \models^c \mathbf{D}$
2. $\mathbf{M}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle) \models^c p.ins$ *if* $\mathbf{D}_2 = \mathbf{D}_1 \cup \{p\}$
3. $\mathbf{M}(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle) \models^c p.del$ *if* $\mathbf{D}_2 = \mathbf{D}_1 - \{p\}$

Note $p.ins$ and $p.del$ are defined to be non-strict operations of insertion and deletion. That is, in a procedural interpretation, $p.ins$ may execute on a database that already has p, in which case the database does not change. Likewise for $p.del$. In addition, because this paper considers only Herbrand path structures, we shall

[2] With the above universe, the Herbrand semantics becomes equivalent to the general semantics even in classical logic. This is a consequence of the Skolem-Löwenheim Theorem.

often omit the adjective "Herbrand." Therefore, in the sequel, the term "path structure" should be taken to mean "Herbrand path structure."

A path structure, specifies what ground atoms are true on what paths. Intuitively, each atom that is true on a path represents a property of the path.[3] In arbitrary path structures, these atoms are independent of one another. In particular, the atoms assigned to path π are independent of the atoms assigned to subpaths of π. Intuitively, this means that we know nothing about how the various properties of the various paths are related to each other. Such knowledge, when it exists, is encoded as transaction formulas, as in the examples of Section 2. These formulas constrain the assignment of atoms to paths, forcing the atoms on one path to be related in precise ways to atoms on other paths.

As in classical logic, in order to define the truth value of quantified formulas and of open formulas, it is convenient to introduce variable assignments. A *variable assignment* is a mapping $\nu : \mathcal{V} \longrightarrow \mathcal{U}$, which takes a variable as input and returns a Herbrand term as output. We extend the mapping from variables to terms in the usual way, *i.e.*, $\nu(f(t_1, \ldots, t_n)) = f(\nu(t_1), \ldots, \nu(t_n))$. The mapping can be extended to atomic formulas in a similar fashion.

Definition 3. (Satisfaction) *Let* $\mathbf{M}$ *be a Herbrand path structure, let* π *be a path, and let* ν *be a variable assignment. If* $\mathbf{M}(\pi) = \top$ *then* $\mathbf{M}, \pi \models \phi$ *for every transaction formula,* ϕ*; otherwise,*

1. **Base Case:** $\mathbf{M}, \pi \models_\nu p$ *if and only if* $\nu(p) \in \mathbf{M}(\pi)$, *for any atomic formula* p.
2. **Negation:** $\mathbf{M}, \pi \models_\nu \neg\phi$ *if and only if it is not the case that* $\mathbf{M}, \pi \models_\nu \phi$.
3. **"Classical" Conjunction:** $\mathbf{M}, \pi \models_\nu \phi \wedge \psi$ *if and only if* $\mathbf{M}, \pi \models_\nu \phi$ *and* $\mathbf{M}, \pi \models_\nu \psi$.
4. **Serial Conjunction:** $\mathbf{M}, \pi \models_\nu \phi \otimes \psi$ *if and only if* $\mathbf{M}, \pi_1 \models_\nu \phi$ *and* $\mathbf{M}, \pi_2 \models_\nu \psi$ *for* some *split* $\pi_1 \circ \pi_2$ *of path* π.
5. **Universal Quantification:** $\mathbf{M}, \pi \models_\nu (\forall X)\phi$ *if and only if* $\mathbf{M}, \pi \models_\mu \phi$ *for* every *variable assignment* μ *that agrees with* ν *everywhere except on* X.
6. **Executional Possibility:** $\mathbf{M}, \pi \models_\nu \Diamond\phi$ *if and only if* π *is a 1-path (i.e.,* $\pi = \langle \mathbf{D} \rangle$*, for some state* $\mathbf{D}$*) and* $\mathbf{M}, \pi \circ \pi' \models_\nu \phi$ *for some path* π'.

As in classical logic, the variable assignment can be omitted for *sentences*, *i.e.*, for formulas with no free variables. From now on, we will deal only with sentences, unless explicitly stated otherwise. If $\mathbf{M}, \pi \models \phi$, then we say that sentence ϕ is *satisfied* (or is *true*) on path π in structure $\mathbf{M}$.

Definition 4. (Models) *A path structure,* $\mathbf{M}$*, is a* model *of a transaction formula* ϕ *if* $\mathbf{M}, \pi \models \phi$ *for every path* π*. In this case, we write* $\mathbf{M} \models \phi$*. A path structure is a model of a set of formulas if it is a model of every formula in the set.*

[3] For instance, in the examples of Section 2.1, if the atoms $tues$ and $feb.28$ are true on a path, then the path represents a Monday and it represents February 28.

Definition 5. (Logical Entailment) *Let ϕ and ψ be two transaction formulas. Then ϕ* entails *ψ if every model of ϕ is also a model of ψ. In this case, we write $\phi \models \psi$.*

Entailment in Transaction Logic is an extension of entailment in classical logic, as the following result shows.

Lemma 1 (Relationship to Classical Logic). *If α and β are classical first-order formulas, then $\alpha \models \beta$ if and only if $\alpha \models^c \beta$.*

Definition 6. (Abbreviations)

1. **"Classical" disjunction:** $\phi \vee \psi$ *means* $\neg(\neg\phi \wedge \neg\psi)$
2. **Existential quantification:** $\exists X\, \phi$ *means* $\neg\, \forall X\, \neg\phi$
3. **"Classical" implication:** $\phi \leftarrow \psi$ *means* $\phi \vee \neg\psi$
4. **"Classical" equivalence:** $\phi \leftrightarrow \psi$ *means* $(\phi \leftarrow \psi) \wedge (\psi \leftarrow \phi)$
5. **Path:** *The propositional constant* `path` *is defined as* $\phi \vee \neg\phi$
6. **States:** *The propositional constant* `state` *is defined as* $\Diamond($`path`$)$, *and* $[\phi]$ *is an abbreviation for* $\phi \wedge$ `state`

The first five items in Definition 6 give standard abbreviations in classical logic,[4] while the last item gives abbreviations unique to Transaction Logic. Intuitively, `state` is a formula that is true only on states, that is, only on paths of length 1. This allows us to reason about the properties of states and the effects that transactions have on states. In particular, the formula $\phi \wedge$ `state` is true on states that satisfy formula ϕ. The combination $\phi \wedge$ `state`, which tests the properties of individual states, is used so frequently in reasoning about the effects of transactions that we abbreviate it as $[\phi]$.

Definition 7. (Validity) *A transaction formula, ϕ, is* valid *if every path structure is a model of ϕ. In this case, we write $\models \phi$.*

Example 3. **(Validity)** The following formulas are valid, for all transaction formulas α, β, γ:

$$(\alpha \vee \beta) \otimes \gamma \;\leftrightarrow\; (\alpha \otimes \gamma) \vee (\beta \otimes \gamma) \qquad\qquad \alpha \;\leftrightarrow\; \alpha \otimes \texttt{state}$$

$$[\alpha \otimes \beta] \;\leftrightarrow\; [\alpha \wedge \beta] \qquad\qquad \alpha \;\leftrightarrow\; (\Diamond\alpha) \otimes \alpha$$

$$\Diamond(\alpha \vee \beta) \;\leftrightarrow\; (\Diamond\alpha) \vee (\Diamond\beta) \qquad\qquad \Diamond(\alpha \otimes \beta) \;\leftrightarrow\; \Diamond(\alpha \otimes \Diamond\beta)$$

$$(\alpha \wedge \beta) \otimes \gamma \;\rightarrow\; \alpha \otimes \gamma \wedge \beta \otimes \gamma \qquad\qquad \Diamond(\alpha \wedge \beta) \;\rightarrow\; \Diamond\alpha \wedge \Diamond\beta$$

[4] While these abbreviations are standard, the connectives involved are not, which is why the word "classical" is put in quotation marks. Indeed, even though the connectives $\leftarrow$, $\wedge$, and $\vee$ have a familiar look, they all are defined over paths, not states, so they are all but standard. However, we use classical notation for them because their definitions are very similar to the definitions for the corresponding classical connectives.

Note that, in general, $\wedge$ does not distribute through $\otimes$ and $\Diamond$, so the last two formulas do not remain valid if the direction of implication is reversed. For instance, suppose that $\alpha = a.ins$, $\beta = a.ins \otimes b.ins$, and $\gamma = b.ins \vee \texttt{state}$. Then, the formula $(\alpha \otimes \gamma) \wedge (\beta \otimes \gamma)$ is true on the path $\langle\{\}, \{a\}, \{a, b\}\rangle$ in all path structures, but the formula $(\alpha \wedge \beta) \otimes \gamma$ is not.

Lemma 2 (Partial Deduction Theorem). *If $\alpha \to \beta$ is valid, then α logically entails β; that is, if $\models \alpha \to \beta$ then $\alpha \models \beta$, for all transaction formulas, α, β.*

The next example shows that the converse of Lemma 2 is not always true.

Example 4. **(Validity vs. Entailment)** The following expressions are entailments, for all transaction formulas α, β, γ:

$$\alpha \to \beta \;\models\; \alpha \otimes \gamma \to \beta \otimes \gamma \qquad\qquad \alpha \to \beta \;\models\; \Diamond\alpha \to \Diamond\beta$$

However, the following formulas are *not* always valid:

$$(\alpha \to \beta) \;\to\; (\alpha \otimes \gamma \to \beta \otimes \gamma) \qquad\qquad (\alpha \to \beta) \;\to\; (\Diamond\alpha \to \Diamond\beta)$$

For instance, consider the left-hand formula, with $\alpha = \texttt{state}$ and $\beta = \gamma = b.ins$. Let $\mathbf{M}$ be any model for which $b.ins$ is true only on paths of the form $\langle \mathbf{D}_1, \mathbf{D}_2\rangle$ where $\mathbf{D}_2 = \mathbf{D}_1 \cup \{b\}$. Then, for any such path,

1. $\mathbf{M}, \langle \mathbf{D}_1, \mathbf{D}_2\rangle \models b.ins$ — by Definitions 2 and 3
2. $\mathbf{M}, \langle \mathbf{D}_1, \mathbf{D}_2\rangle \models \texttt{state} \to b.ins$ — by the semantics of $\to$

In addition,

3. $\mathbf{M}, \langle \mathbf{D}_1\rangle \models \texttt{state}$ — since $\texttt{state}$ is true on every 1-path
4. $\mathbf{M}, \langle \mathbf{D}_1, \mathbf{D}_2\rangle \models \texttt{state} \otimes b.ins$ — by 1 and 3 and the semantics of $\otimes$
5. $\mathbf{M}, \langle \mathbf{D}_1, \mathbf{D}_2\rangle \not\models b.ins \otimes b.ins$ — as $b.ins \otimes b.ins$ is true only on 3-paths
6. $\mathbf{M}, \langle \mathbf{D}_1, \mathbf{D}_2\rangle \not\models \texttt{state} \otimes b.ins \to b.ins \otimes b.ins$ — using 4, 5 and the semantics of $\to$.

Thus, from 2 and 6, the following formula is *not* true on path $\langle \mathbf{D}_1, \mathbf{D}_2\rangle$ of model $\mathbf{M}$:

$$(\texttt{state} \to b.ins) \;\to\; (\texttt{state} \otimes b.ins \to b.ins \otimes b.ins)$$

Hence, the formula is not valid.

This example shows that the deduction theorem does not always hold in $\mathcal{TR}$. However, since the deduction theorem holds for first-order classical logic, it follows from Lemma 1 that it also holds in $\mathcal{TR}$ for the special case of first-order formulas.[5]

[5] It is possible to define a stronger notion of entailment, $|\!\models$, for which the deduction theorem holds: $\alpha \;|\!\models \beta$ iff for every $\mathbf{M}$ and π, if $\mathbf{M}, \pi \models \alpha$ then $\mathbf{M}, \pi \models \beta$. However, this notion of entailment is not appropriate for the theory of reasoning developed in this paper.

Corollary 1 (Deduction Theorem for First-Order Formulas). *If α and β are first-order formulas, then $\alpha \models \beta$ if and only if $\models \alpha \rightarrow \beta$.*

Sections 4, 5 and 6 show that logical entailment allows us to express properties of $\mathcal{TR}$ logic programs and to reason about them.

3.2 Executional Entailment

In addition to logical entailment, $\mathcal{TR}$ supports another form of entailment, called *executional entailment.* While logical entailment allows us to *reason* about $\mathcal{TR}$ programs, executional entailment allows us to *execute* them. In particular, executional entailment provides the theoretical foundation for an SLD-style proof procedure for serial-Horn $\mathcal{TR}$, a procedure that executes logic programs as it proves theorems. However, unlike the proof procedures for classical logic programs, this procedure does more than just evaluate queries: it also *updates* the database. These results are described in detail elsewhere [BK98,Bon97b,BK93,BK95]. This section reviews the main ideas.

Definition 8. (Executional Entailment) *Let* $\mathbf{P}$ *be a set of transaction formulas (a transaction base), let* ϕ *be a transaction formula, and let* $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n$ *be a sequence of database states. Then, the following statement*

$$\mathbf{P}, \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \models \phi \tag{2}$$

is true if $\mathbf{M}, \langle \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \rangle \models \phi$ *for every model* $\mathbf{M}$ *of* $\mathbf{P}$*. We also define*

$$\mathbf{P}, \mathbf{D}_0 \text{---} \models \phi \tag{3}$$

to be true if there is a database sequence $\mathbf{D}_1, \ldots, \mathbf{D}_n$ *that makes (2) true.*

Formally, statement (2) says that every model of $\mathbf{P}$ satisfies ϕ on the path $\langle \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \rangle$. However, it can also be interpreted procedurally in terms of program execution. From this perspective, ϕ is a program invocation, and $\mathbf{P}$ is a set of program and subroutine definitions. Statement (2) then says that a successful execution of ϕ can change the database from state $\mathbf{D}_0$ to $\mathbf{D}_1 \ldots$ to $\mathbf{D}_n$.

Normally, users who want to execute transactions know only the initial database state $\mathbf{D}_0$; they do not know the execution path in advance, and they just want to reach a final state of the execution. To account for this situation, the version of entailment in statement (3) allows us to omit the intermediate and the final database states. Intuitively, statement (3) says that transaction ϕ can execute successfully starting from database $\mathbf{D}_0$. When the context is clear, we simply say that transaction ϕ *succeeds.* Likewise, when statement (3) is not true, we say that transaction ϕ *fails.* In [BK98,BK93,BK95], we develop a logical inference system that allows us to *compute* a database sequence $\mathbf{D}_1, \ldots, \mathbf{D}_n$ that satisfies statement (2) whenever a transaction succeeds.

Example 5. **(Executional Entailment)** Suppose $\mathbf{P}$ contains the following rules:

$$q \leftarrow r \qquad q \leftarrow s \qquad r \leftarrow a.ins \otimes b.ins \qquad s \leftarrow a.del \otimes b.del$$

Then, the following statements are all true:

$$\begin{array}{ll} \mathbf{P}, \{\,\}, \{a\}, \{a,b\} \models a.ins \otimes b.ins & \mathbf{P}, \{a,b\}, \{b\}, \{\,\} \models a.del \otimes b.del \\ \mathbf{P}, \{\,\}, \{a\}, \{a,b\} \models r & \mathbf{P}, \{a,b\}, \{b\}, \{\,\} \models s \\ \mathbf{P}, \{\,\}, \{a\}, \{a,b\} \models q & \mathbf{P}, \{a,b\}, \{b\}, \{\,\} \models q \end{array}$$

$$\mathbf{P}, \{\,\}, \{a\}, \{a,b\}, \{b\}, \{\,\} \models r \otimes s$$
$$\mathbf{P}, \{\,\}, \{a\}, \{a,b\}, \{b\}, \{\,\} \models q \otimes q$$

Hence, the following statements are true as well:

$$\begin{array}{ll} \mathbf{P}, \{\,\}\text{---} \models a.ins \otimes b.ins & \mathbf{P}, \{a,b\}\text{---} \models a.del \otimes b.del \\ \mathbf{P}, \{\,\}\text{---} \models r & \mathbf{P}, \{a,b\}\text{---} \models s \\ \mathbf{P}, \{\,\}\text{---} \models q & \mathbf{P}, \{a,b\}\text{---} \models q \\ \mathbf{P}, \{\,\}\text{---} \models r \otimes s & \mathbf{P}, \{\,\}\text{---} \models q \otimes q \end{array}$$

Executional entailment is an extension of classical entailment, as the following lemma shows.

Lemma 3 (Relationship to Classical Logic). *If α is a first-order formula, and $\mathbf{P}$ is a set of first-order formulas, then $\mathbf{P}, \mathbf{D} \models \alpha$ if and only if $\mathbf{P} \cup \mathbf{D} \models^c \alpha$.*

3.3 Minimal Models

Like classical logic programs, the Herbrand semantics of serial-Horn $\mathcal{TR}$ can be formulated as a fixpoint theory. One of the results of this theory is that a serial-Horn transaction base has a unique minimal model. This model provides an essential link between the theory of reasoning developed in this paper and the theory of execution developed in [BK98,BK93,BK95]. In particular, when reasoning about a $\mathcal{TR}$ program, we shall be reasoning about formulas that are true in the minimal model, since these formulas tell us a great deal about how a program executes. This section provides the necessary background on minimal models in serial-Horn $\mathcal{TR}$.

Definition 9. (Ordered Structures)

- *If σ_1 and σ_2 are classical Herbrand structures (or $\top$), then $\sigma_1 \leq \sigma_2$ if $\sigma_1 \subseteq \sigma_2$ or $\sigma_2 = \top$.*
- *If $\mathbf{M}_1$ and $\mathbf{M}_2$ are two Herbrand path structures, then $\mathbf{M}_1 \leq \mathbf{M}_2$ if $\mathbf{M}_1(\pi) \leq \mathbf{M}_2(\pi)$ for every path, π.*

This ordering on path structures allows us to develop a fixpoint theory for serial-Horn $\mathcal{TR}$ that is analogous to the fixpoint theory of classical Horn logic. The following theorem is a basic result of this theory.

Theorem 1 (Unique Minimal Model). *If* $\mathbf{P}$ *is a set of serial-Horn rules, then* $\mathbf{P}$ *has a unique minimal Herbrand model,* $\mathbf{M_P}$*. That is,* $\mathbf{M_P}$ *is a Herbrand model of* $\mathbf{P}$*, and* $\mathbf{M_P} \leq \mathbf{M}$ *for all other Herbrand models,* $\mathbf{M}$*, of* $\mathbf{P}$*.*

Another basic result is that for serial-Horn rules, executional entailment is equivalent to satisfaction in the minimal model. Intuitively, this means that reasoning about truth in the minimal model is equivalent to reasoning about program execution. Our theory of reasoning in $\mathcal{TR}$ is based on this idea.

Theorem 2. *If* $\mathbf{P}$ *is a set of serial-Horn rules, and* α *is a serial goal, then*

$$\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \models \alpha \quad \textit{if and only if} \quad \mathbf{M_P}, \langle \mathbf{D}_1, ..., \mathbf{D}_n \rangle \models \alpha$$

4 Expressing Properties of Update Programs

This section shows how to express properties of $\mathcal{TR}$ programs as transaction formulas. If these formulas are true in the minimal model of a transaction base, then they describe the possible executions of a transaction program. We shall make this idea precise after first illustrating how to represent program properties as transaction formulas.

In this paper, we shall be concerned with two kinds of properties: (i) under what conditions can a program execute successfully, and (ii), if a program executes successfully, what effect does it have on the database. We shall refer to these as *success* and *effect* properties, respectively. Examples of success properties include the following:

- Can program α always succeed?
- If $p \vee q$ is true, can program α succeed?
- If the database satisfies its integrity constraints, can transaction α succeed?

Examples of effect properties include the following:

- Is p always true after program α executes?
- If $p \vee q$ is true before program α executes, is $r \vee s$ true afterwards?
- Does every execution of transaction α preserve the database integrity constraints?

Properties like these can be expressed by transaction formulas of a very specific form:

Success properties: $[condition] \rightarrow \Diamond\alpha$
Effect properties: $[condition_1] \otimes \alpha \rightarrow \alpha \otimes [condition_2]$

These formulas have a simple interpretation. The success property says, "if *condition* is true, then α can execute successfully." The effect property says, "If $condition_1$ is true before α executes, then $condition_2$ is true after α executes." Here, each condition is an arbitrary formula of classical first-order logic, which the underlying database state must satisfy. For example, the three success properties listed above are expressed as follows:

$$\texttt{state} \rightarrow \Diamond\alpha \qquad [p \vee q] \rightarrow \Diamond\alpha \qquad [\mathcal{C}] \rightarrow \Diamond\alpha$$

where $\mathcal{C}$ a first-order formula representing the database integrity constraints. Likewise, the three effect properties listed above are expressed as follows:

$$\texttt{state} \otimes \alpha \rightarrow \alpha \otimes [p] \qquad [p \vee q] \otimes \alpha \rightarrow \alpha \otimes [r \vee s] \qquad [\mathcal{C}] \otimes \alpha \rightarrow \alpha \otimes [\mathcal{C}]$$

In general, we can (and will) also use formulas in which the direction of the implication sign is reversed, *i.e.*, in which $\rightarrow$ is replaced by $\leftarrow$.

The reverse implication allows reasoning about conditions that must have been true before transaction execution. For instance, $[condition] \leftarrow \Diamond\alpha$ means that if α can execute then *condition* must have been true in the initial state of the execution. Likewise, $[condition_1] \otimes \alpha \leftarrow \alpha \otimes [condition_2]$ means that if $condition_2$ is true in the final state of an execution of α, then $condition_1$ must have been true when this execution started.

The examples above illustrate informally that transaction formulas can be used to express properties of $\mathcal{TR}$ programs. We now make this idea precise. Recall that $\mathcal{TR}$ programs are defined by serial-Horn transaction bases.

Definition 10. (Properties) *Let* $\mathbf{P}$ *be a serial-Horn transaction base, and let* ψ *be a transaction formula. Then,* ψ *is a property of* $\mathbf{P}$ *if it is true in the minimal model, that is, if* $\mathbf{M_P} \models \psi$.

The following two theorems establish a precise relationship between the properties of a transaction base, $\mathbf{P}$, and the way in which transactions defined by $\mathbf{P}$ execute. This connects the theory of reasoning developed in this paper with the theory of execution developed in [BK98,BK93,BK95]. The minimal model semantics makes this link possible.

Theorem 3 (Success Properties). *Let* $\mathbf{P}$ *be a serial-Horn transaction base, let* α *be a serial goal, and let* ϕ *be a first-order formula made from base predicate symbols. Then, the following two statements are equivalent:*

– $\mathbf{P}$ *has property* $[\phi] \rightarrow \Diamond\alpha$
– *For every database state* $\mathbf{D}$, *if* $\mathbf{D} \models^c \phi$ *then* $\mathbf{P}, \mathbf{D}\text{---} \models \alpha$

Suppose that $\mathbf{P}$ has the property $[\phi] \rightarrow \Diamond\alpha$. Theorem 3 says that if ϕ is true in the current database state, then α can execute successfully. A similar theorem holds if $\rightarrow$ is replaced by $\leftarrow$.

Theorem 4 (Effect Properties). *Let* $\mathbf{P}$ *be a serial-Horn transaction base, let* α *be a serial goal, and let* ϕ *and* ψ *be first-order formulas made from base predicate symbols. Then, the following two statements are equivalent:*

– $\mathbf{P}$ *has property* $[\phi] \otimes \alpha \ \rightarrow \ \alpha \otimes [\psi]$
– *For every sequence of database states* $\mathbf{D}_1, ..., \mathbf{D}_n$, *if* $\mathbf{D}_1 \models^c \phi$ *and* $\mathbf{P}, \mathbf{D}_1, ..., \mathbf{D}_n \models \alpha$ *then* $\mathbf{D}_n \models \psi$

Suppose that $\mathbf{P}$ has the property $[\phi] \otimes \alpha \ \rightarrow \ \alpha \otimes [\psi]$. Theorem 4 says that whenever α executes, if the initial state satisfies ϕ, then the final state satisfies ψ. A similar theorem holds if $\rightarrow$ is replaced by $\leftarrow$.

4.1 Examples

This section gives several examples of transaction formulas that express properties of $\mathcal{TR}$ programs. Each example centers on a particular issue, such as elementary operations, sequential composition, pre-conditions, non-determinism, and subroutines. The final example suggests the richness of reasoning when a program combines several of these issues. Each example also shows how the meaning of transaction formulas can be expressed in terms of executional entailment. To simplify the exposition, the examples use propositional $\mathcal{TR}$, though examples of predicate $\mathcal{TR}$ are easy to generate.

Example 6. **(Elementary Operations)** If b and c are *distinct* base facts, then the formulas below are true in the minimal model of every transaction base.

$$c.ins \ \rightarrow \ c.ins \otimes [c] \qquad [b] \otimes c.ins \ \leftrightarrow \ c.ins \otimes [b] \qquad \texttt{state} \ \leftrightarrow \ \Diamond c.ins$$

The first formula says that $c.ins$ makes c true (as one would expect). The syntax can be read as follows: if $c.ins$ is executed, then after execution, c is true. The second formula says that b is unaffected by $c.ins$, *i.e.*, b is true before an execution of $c.ins$ if and only if b is true after execution. This is an example of a frame axiom [MH69,Rei91] in our setting. The third formula says that it is always possible to execute $c.ins$.

It follows from Theorems 4 and 3 that the following statements are also true, for any transaction base $\mathbf{P}$, any pair of distinct base facts b and c, and all database states, $\mathbf{D}_1$, $\mathbf{D}_2$:

– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models c.ins$ then $c \in \mathbf{D}_2$
– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models c.ins$ then $b \in \mathbf{D}_1$ *iff* $b \in \mathbf{D}_2$
– $\mathbf{P}, \mathbf{D}_1 \text{---} \models c.ins$

Example 7. **(Sequential Composition)** If a, b and c are distinct base facts, then the formulas below are true in the minimal model of every transaction base. They state some basic properties of the sequential transaction $b.ins \otimes c.del$, which first inserts b into the database, and then deletes c.

$$b.ins \otimes c.del \rightarrow b.ins \otimes c.del \otimes [b \wedge \neg c]$$
$$[a] \otimes b.ins \otimes c.del \leftrightarrow b.ins \otimes c.del \otimes [a]$$
$$\texttt{state} \leftrightarrow \Diamond(b.ins \otimes c.del)$$

The first formula says that the transaction makes b true and c false. The syntax can be read as follows: if the transaction is executed, then after execution, $b \wedge \neg c$ is true. The second formula says that the transaction does not affect a, *i.e.*, a is true before execution if and only if it is true after execution. The third formula says that it is always possible to execute the transaction.

It follows from Theorems 4 and 3 that the following executional entailments are also true, for any transaction base $\mathbf{P}$, any triple of distinct base facts a, b, c, and all databases $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_3$:

- If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3 \models b.ins \otimes c.del$ then $b \in \mathbf{D}_3$ and $c \notin \mathbf{D}_3$
- If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3 \models b.ins \otimes c.del$ then $a \in \mathbf{D}_1$ *iff* $a \in \mathbf{D}_3$
- $\mathbf{P}, \mathbf{D}_1 \text{---} \models b.ins \otimes c.del$

Example 8. **(Pre-conditions)** If b and c are distinct base facts, then the formulas below are true in the minimal model of every transaction base. They state some basic properties of the transaction $c \otimes c.del$, which deletes c from the database, but fails if c is not in the database to begin with.

$$(c \otimes c.del) \rightarrow (c \otimes c.del) \otimes [\neg c]$$

$$[b] \otimes (c \otimes c.del) \leftrightarrow (c \otimes c.del) \otimes [b]$$

$$[c] \leftrightarrow \Diamond(c \otimes c.del)$$

The first formula says that the transaction makes c false. The second formula says that the transaction does not affect b. The third formula says that the transaction is possible if and only if c is in the database.

It follows from Theorems 4 and 3 that the following executional entailments are also true, for any transaction base $\mathbf{P}$ and any pair of databases $\mathbf{D}_1$ and $\mathbf{D}_2$:

- If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models c \otimes c.del$ then $c \notin \mathbf{D}_2$
- If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models c \otimes c.del$ then $b \in \mathbf{D}_1$ *iff* $b \in \mathbf{D}_2$
- $\mathbf{P}, \mathbf{D}_1 \text{---} \models c \otimes c.del$ if and only if $c \in \mathbf{D}_1$

Example 9. **(Non-Determinism)** If a, b and c are distinct base facts, then the formulas below are true in the minimal model of every transaction base. They state some basic properties of the non-deterministic transaction $b.ins \vee c.del$, which either inserts b into the database or deletes c.

$$(b.ins \vee c.del) \rightarrow (b.ins \vee c.del) \otimes [b \vee \neg c]$$

$$[a] \otimes (b.ins \vee c.del) \leftrightarrow (b.ins \vee c.del) \otimes [a]$$

$$\mathtt{state} \leftrightarrow \Diamond(b.ins \vee c.del)$$

The first formula says that the transaction makes b true or c false. The second formula says that the transaction does not affect a. The third formula says that the transaction is always possible.

It follows from Theorems 4 and 3 that the following executional entailments are also true, for any transaction base $\mathbf{P}$ and any pair of states $\mathbf{D}_1$ and $\mathbf{D}_2$:

– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models b.ins \vee c.del$ then $b \in \mathbf{D}_2$ or $c \notin \mathbf{D}_2$
– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2 \models b.ins \vee c.del$ then $a \in \mathbf{D}_1$ *iff* $a \in \mathbf{D}_2$
– $\mathbf{P}, \mathbf{D}_1, \text{---} \models b.ins \vee c.del$

Example 10. **(Subroutines)** Suppose that transaction base **P** consists of exactly one rule:

$$q \leftarrow b.ins \otimes c.del$$

Intuitively, this rule defines a subroutine, where q is the subroutine name. The rule says that one way to execute q is to execute the transaction $b.ins \otimes c.del$. Because of the minimal model semantics, this is the *only* way to execute q. This is reflected in the following formula, which is true in the minimal model of **P**:

$$q \leftrightarrow b.ins \otimes c.del$$

Because of this equivalence, q has all the properties of $b.ins \otimes c.del$ described in Example 7. In particular, the following formulas are true in the minimal model of **P**, where a is a base fact distinct from b and c:

$$q \rightarrow q \otimes [b \wedge \neg c] \qquad [a] \otimes q \leftrightarrow q \otimes [a] \qquad \texttt{state} \leftrightarrow \Diamond q$$

These formulas say that q makes b true and c false, that q does not affect the value of a, and that q is always possible.

It follows from Theorems 4 and 3 that the following executional entailments are also true, for all states $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_3$:

– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3 \models q$ then $b \in \mathbf{D}_3$ and $c \notin \mathbf{D}_3$
– If $\mathbf{P}, \mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3 \models q$ then $a \in \mathbf{D}_1$ *iff* $a \in \mathbf{D}_3$
– $\mathbf{P}, \mathbf{D}_1 \text{---} \models q$

Example 11. **(Potpourri)** Suppose that transaction base **P** consists of exactly two rules:

$$q \leftarrow a \otimes c.ins \qquad\qquad q \leftarrow b \otimes d.del$$

Intuitively, these rules define a non-deterministic subroutine, where q is the subroutine name. The first rule says that one way to execute q is to execute the transaction $a \otimes c.ins$. The second rule says that another way to execute q is to execute the transaction $b \otimes d.del$. Because of the minimal model semantics, these are the *only* ways to execute q. This is reflected in the following formula, which is true in the minimal model of **P**:

$$q \leftrightarrow (a \otimes c.ins \ \vee \ b \otimes d.del)$$

This equivalence says that q amounts to a non-deterministic choice between the two transactions $a \otimes c.ins$ and $b \otimes d.del$. Consequently, the following formulas are also true in the model, where e is a base fact distinct from c and d:

$$q \rightarrow q \otimes [c \vee \neg d] \qquad [e] \otimes q \leftrightarrow q \otimes [e] \qquad [a \vee b] \leftrightarrow \Diamond q$$

The first formula says that q makes c true or d false, the second formula says that q does not affect the value of e, and the third formula says that q is possible if and only if a or b is true.

In general, if we know nothing about the database state when q begins executing, then q can execute in two possible ways (or it can fail to execute at all). However, knowledge about the initial database state can reduce the possibilities. For instance, if a is false in the initial state, then $a \otimes c.ins$ cannot execute. Thus, any execution of q must cause an execution of $b \otimes d.del$, so d must be false in the final state. This is represented by the following formulas, which are both true in the minimal model of $\mathbf{P}$:

$$[\neg a] \otimes q \ \rightarrow \ b \otimes d.del \qquad\qquad [\neg a] \otimes q \ \rightarrow \ q \otimes [\neg d]$$

Likewise, if b is false in the initial state, then $b \otimes d.del$ cannot execute. Thus, any execution of q must cause an execution of $a \otimes c.ins$, so c must be true in the final state. This situation is represented by the following formulas, which are both true in the minimal model of $\mathbf{P}$:

$$[\neg b] \otimes q \ \rightarrow \ a \otimes c.ins \qquad\qquad [\neg b] \otimes q \ \rightarrow \ q \otimes [c]$$

These formulas express a forward mode of reasoning, from initial state to final state. However, we can also reason backwards, from final state to initial state, since knowledge about the final state can also tell us something about how q executed. For instance, if c is false in the final state, then $a \otimes c.ins$ could not have executed, so $b \otimes d.del$ must have executed, so b must have been true in the initial state. This is represented by the following formulas, which are both true in the minimal model of $\mathbf{P}$:

$$b \otimes d.del \ \leftarrow \ q \otimes [\neg c] \qquad\qquad [b] \otimes q \ \leftarrow \ q \otimes [\neg c]$$

Likewise, if d is true in the final state, then $b \otimes d.del$ could not have executed, so $a \otimes c.ins$ must have executed, so a must have been true in the initial state. This state of affairs is represented by the following formulas, which are both true in the minimal model of $\mathbf{P}$:

$$a \otimes c.ins \ \leftarrow \ q \otimes [d] \qquad\qquad [a] \otimes q \ \leftarrow \ q \otimes [d]$$

5 An Inference System

This section develops an inference system for reasoning about transaction formulas. To simplify matters, the system presented here is intentionally limited in two ways: (i) it reasons about the effects of transactions, but not about when transactions can succeed; and (ii) it reasons about ground formulas only, not about formulas with variables. Both of these restrictions can be lifted without difficulty, and are the subject of a forthcoming work.

The inference system consists of a collection of axioms and inference rules. Each axiom is a transaction formula, and each inference rule consists of several transaction formulas separated by a horizontal line. An inference rule has the following simple interpretation: if the formulas above the line can be derived, then the formulas below the line can also be derived. Based on the axioms, the inference rules derive additional transaction formulas. Section 6 shows how to use the inference system to reason about transactions.

As in modal logic, reasoning in $\mathcal{TR}$ is an extension of reasoning in classical logic. Thus, the inference system below starts with the axioms and inference rules for classical logic, and augments them with a small number of axioms and inference rules specific to $\mathcal{TR}$.

Definition 11. (Inference System $\mathcal{T}$) *$\mathcal{T}$ is the system of axioms and inference rules below, where α, β, and γ stand for arbitrary transaction formulas.*

Axioms:

Classical Axioms: *Every valid formula of first-order classical logic, where parameters representing first-order formulas now represent transaction formulas, is an axiom in $\mathcal{T}$.*

Distributivity:

$$\gamma \otimes (\alpha \vee \beta) \;\leftrightarrow\; (\gamma \otimes \alpha) \vee (\gamma \otimes \beta) \qquad\qquad (\alpha \vee \beta) \otimes \gamma \;\leftrightarrow\; (\alpha \otimes \gamma) \vee (\beta \otimes \gamma)$$

Auxiliary Axioms:

1. $\quad \mathtt{state} \otimes \alpha \;\leftrightarrow\; \alpha \qquad\qquad \alpha \;\leftrightarrow\; \alpha \otimes \mathtt{state}$

2. $\quad \neg\mathtt{path} \otimes \alpha \;\leftrightarrow\; \neg\mathtt{path} \qquad\qquad \neg\mathtt{path} \;\leftrightarrow\; \alpha \otimes \neg\mathtt{path}$

3. $\quad [\alpha \wedge \beta] \;\leftrightarrow\; [\alpha] \otimes [\beta]$

Inference Rules:

Classical Rules: *Every inference rule of first-order classical logic, where parameters representing first-order formulas now represent transaction formulas, is an inference rule of $\mathcal{T}$.*

Attachment:

$$\frac{\alpha \;\rightarrow\; \beta}{\gamma \otimes \alpha \;\rightarrow\; \gamma \otimes \beta} \qquad\qquad \frac{\alpha \;\rightarrow\; \beta}{\alpha \otimes \gamma \;\rightarrow\; \beta \otimes \gamma}$$

Because inference system $\mathcal{T}$ extends classical inference, we can immediately write down a number of additional axioms and inference rules that are included in $\mathcal{T}$. For example, since $\alpha \vee \neg\alpha$ is a valid formula of classical logic, for all first-order formulas α, it is an axiom of system $\mathcal{T}$, for all transaction formulas α. Likewise, since the following are inference rules of classical logic, for all first-order formulas α, β, γ, they are also inference rules of system $\mathcal{T}$, for all transaction formulas α, β, γ:

$$\text{Transitivity}: \; \frac{\alpha \rightarrow \beta, \quad \beta \rightarrow \gamma}{\alpha \rightarrow \gamma} \qquad\qquad \text{Disjunction}: \; \frac{\alpha_1 \rightarrow \beta_1, \quad \alpha_2 \rightarrow \beta_2}{(\alpha_1 \vee \alpha_2) \rightarrow (\beta_1 \vee \beta_2)}$$

The following rule, which is used extensively in this paper, is also a rule of classical inference, and thus of system $\mathcal{T}$:

$$\text{Substitution}: \quad \frac{\gamma, \quad \alpha \leftrightarrow \beta}{\gamma\{\alpha/\beta\}}$$

where $\gamma\{\alpha/\beta\}$ is the result of replacing some occurrence of α by β in γ.

Theorem 5 (Soundness). *Every transaction formula derivable in system $\mathcal{T}$ is valid. More generally, if α is a set of transaction formulas, and β can be derived from α and $\mathcal{T}$, then $\alpha \models \beta$.*

5.1 Derived Rules and Formulas

From the axioms and inference rules in system $\mathcal{T}$, we can derive numerous other rules and formulas. Many of these will be used in Section 6 to derive properties of transactions. All the derived formulas and rules in this section concern the pre-conditions and post-conditions of transactions, that is, formulas of the form $[\phi]$. The first two lemmas tell us when such conditions can be eliminated or generalized. The remaining lemmas tell us how to derive conditions involving arbitrary first-order formulas, and how to reason about non-deterministic transactions. The examples give proofs for special cases of the lemmas.

Lemma 4 (Eliminating Vacuous Conditions). *The following formulas can be derived in system $\mathcal{T}$, for any transaction formula α:*

$$[\texttt{path}] \otimes \alpha \;\leftrightarrow\; \alpha \qquad\qquad \alpha \;\leftrightarrow\; \alpha \otimes [\texttt{path}] \qquad\qquad [\texttt{path}] \;\leftrightarrow\; \texttt{state}$$

Proof. We derive only the right- and left-most formulas, since the middle formula has a similar derivation. Observe that $\models^c ((\psi \vee \neg\psi) \wedge \phi) \leftrightarrow \phi$ for all first-order formulas, ϕ. Hence,

1.	$((\psi \vee \neg\psi) \wedge \phi) \;\leftrightarrow\; \phi$	for all $\mathcal{TR}$ formulas, ϕ, by Classical Axioms
2.	$(\texttt{path} \wedge \texttt{state}) \;\leftrightarrow\; \texttt{state}$	using $\phi = \texttt{state}$ and the definition of path
3.	$[\texttt{path}] \;\leftrightarrow\; \texttt{state}$	by Definition 6
4.	$\texttt{state} \otimes \alpha \;\leftrightarrow\; \alpha$	Auxiliary Axiom 1
5.	$[\texttt{path}] \otimes \alpha \;\leftrightarrow\; \alpha$	by 3, 4 and Substitution.

Note that Clause 3 above proves the right-most formula in this lemma.

Lemma 5 (Implied Conditions). *If ϕ and ψ are first-order formulas, and $\phi \models^c \psi$, then $[\phi] \rightarrow [\psi]$ can be derived in system $\mathcal{T}$.*

Proof. If $\phi \models^c \psi$, then $\models^c \phi \rightarrow \psi$ by the deduction theorem for classical logic, so $\models^c (\phi \wedge \delta) \rightarrow (\psi \wedge \delta)$ for all first-order formulas, δ. Hence,

1.	$(\phi \wedge \delta) \rightarrow (\psi \wedge \delta)$	for all $\mathcal{TR}$ formulas, δ, by Classical Axioms
2.	$(\phi \wedge \texttt{state}) \rightarrow (\psi \wedge \texttt{state})$	using $\delta = \texttt{state}$
3.	$[\phi] \rightarrow [\psi]$	by Definition 6.

Example 12. **(Combining Conditions)** The following inference rule can be derived in system $\mathcal{T}$, for all transaction formulas, α, ϕ_i, ψ_i:

$$\frac{\begin{array}{c}[\phi_1] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_1] \\ [\phi_2] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_2]\end{array}}{[\phi_1 \wedge \phi_2] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_1 \wedge \psi_2]}$$

To show this, we first derive a related rule:

1.	$[\phi_2] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_2]$	by Hypothesis
2.	$[\phi_1] \otimes [\phi_2] \otimes \alpha \;\rightarrow\; [\phi_1] \otimes \alpha \otimes [\psi_2]$	by Attachment
3.	$[\phi_1] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_1]$	by Hypothesis
4.	$[\phi_1] \otimes \alpha \otimes [\psi_2] \;\rightarrow\; \alpha \otimes [\psi_1] \otimes [\psi_2]$	by Attachment
5.	$[\phi_1] \otimes [\phi_2] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_1] \otimes [\psi_2]$	by 2, 4 and Transitivity

Using equivalences, we transform line 5 into the final form:

6.	$[\phi_1 \wedge \phi_2] \;\leftrightarrow\; [\phi_1] \otimes [\phi_2]$	Auxiliary Axiom 3
7.	$[\psi_1 \wedge \psi_2] \;\leftrightarrow\; [\psi_1] \otimes [\psi_2]$	Auxiliary Axiom 3
8.	$[\phi_1 \wedge \phi_2] \otimes \alpha \;\rightarrow\; \alpha \otimes [\psi_1 \wedge \psi_2]$	by 5, 6, 7 and Substitution.

The following lemma generalizes Example 12. It allows us to reason about the effects of an update program under arbitrary first-order conditions. This is done by combining simple conditions into complex conditions. In a database context, this allows us to reason about the effects of transactions on first-order integrity constraints. In Artificial Intelligence, it allows us to reason about so-called open worlds.

Lemma 6 (Building Complex Conditions). *The following inference rules can be derived in system $\mathcal{T}$, where α, ϕ_i and ψ_i are transaction formulas:*

$$(a)\quad \frac{[\phi_i] \otimes \alpha \rightarrow \alpha \otimes [\psi_i], \quad i = 1, ..., k}{\begin{array}{c}[\wedge\phi_i] \otimes \alpha \rightarrow \alpha \otimes [\wedge\psi_i] \\ [\vee\phi_i] \otimes \alpha \rightarrow \alpha \otimes [\vee\psi_i]\end{array}} \qquad (b)\quad \frac{[\phi_i] \otimes \alpha \leftarrow \alpha \otimes [\psi_i], \quad i = 1, ..., k}{\begin{array}{c}[\wedge\phi_i] \otimes \alpha \leftarrow \alpha \otimes [\wedge\psi_i] \\ [\vee\phi_i] \otimes \alpha \leftarrow \alpha \otimes [\vee\psi_i]\end{array}}$$

$$(c)\quad \frac{[\phi] \otimes \alpha \rightarrow \alpha \otimes [\psi]}{[\neg\phi] \otimes \alpha \leftarrow \alpha \otimes [\neg\psi]} \qquad (d)\quad \frac{[\phi] \otimes \alpha \leftarrow \alpha \otimes [\psi]}{[\neg\phi] \otimes \alpha \rightarrow \alpha \otimes [\neg\psi]}$$

The following example derives an inference rule that is used frequently in reasoning about non-deterministic transactions. The rule infers sufficient pre-conditions for $\alpha_1 \vee \alpha_2$ from sufficient pre-conditions for α_1 and α_2.

Example 13. **(Non-Deterministic Transactions)** The following inference rule can be derived in system $\mathcal{T}$, where α_i, β_i and ϕ_i are transaction formulas:

$$\frac{\begin{array}{c}[\phi_1] \otimes \alpha_1 \;\rightarrow\; \beta_1 \\ [\phi_2] \otimes \alpha_2 \;\rightarrow\; \beta_2\end{array}}{[\phi_1 \wedge \phi_2] \otimes (\alpha_1 \vee \alpha_2) \;\rightarrow\; (\beta_1 \vee \beta_2)}$$

Here is the derivation:

1.	$[\phi_1] \otimes \alpha_1 \ \rightarrow\ \beta_1$	by Hypothesis
2.	$[\phi_1 \wedge \phi_2] \ \rightarrow\ [\phi_1]$	by Lemma 5, since $\phi_1 \wedge \phi_2 \models^c \phi_1$
3.	$[\phi_1 \wedge \phi_2] \otimes \alpha_1 \ \rightarrow\ [\phi_1] \otimes \alpha_1$	by Attachment
4.	$[\phi_1 \wedge \phi_2] \otimes \alpha_1 \ \rightarrow\ \beta_1$	by 1, 3 and Transitivity
5.	$[\phi_1 \wedge \phi_2] \otimes \alpha_2 \ \rightarrow\ \beta_2$	Likewise.

Combining lines 4 and 5 gives the result:

6. $[\phi_1 \wedge \phi_2] \otimes \alpha_1 \ \vee\ [\phi_1 \wedge \phi_2] \otimes \alpha_2$
$\rightarrow\ \beta_1 \vee \beta_2$ by 4, 5, and the rule of Disjunction

7. $[\phi_1 \wedge \phi_2] \otimes \alpha_1 \ \vee\ [\phi_1 \wedge \phi_2] \otimes \alpha_2$
$\leftrightarrow\ [\phi_1 \wedge \phi_2] \otimes (\alpha_1 \vee \alpha_2)$ by Distributivity

8. $[\phi_1 \wedge \phi_2] \otimes (\alpha_1 \vee \alpha_2) \ \rightarrow\ (\beta_1 \vee \beta_2)$ by 6, 7 and Substitution.

The following lemma generalizes Example 13 to include both pre-conditions and post-conditions, whether necessary or sufficient. The lemma allows us to infer the effects of non-deterministic transactions from the effects of their sub-transactions.

Lemma 7 (Conditions for Non-Determinism). *The following inference rules can be derived in system $\mathcal{T}$, where α_i, β_i and ϕ_i are transaction formulas:*

$$(a)\quad \frac{[\phi_i] \otimes \alpha_i \ \rightarrow\ \beta_i, \quad i = 1, ..., k}{[\wedge_i \phi_i] \otimes (\vee_i \alpha_i) \ \rightarrow\ (\vee_i \beta_i)} \qquad (b)\quad \frac{\beta_i \ \rightarrow\ \alpha_i \otimes [\phi_i], \quad i = 1, ..., k}{(\vee_i \beta_i) \ \rightarrow\ (\vee_i \alpha_i) \otimes [\vee_i \phi_i]}$$

$$(c)\quad \frac{[\phi_i] \otimes \alpha_i \ \leftarrow\ \beta_i, \quad i = 1, ..., k}{[\vee_i \phi_i] \otimes (\vee_i \alpha_i) \ \leftarrow\ (\vee_i \beta_i)} \qquad (d)\quad \frac{\beta_i \ \leftarrow\ \alpha_i \otimes [\phi_i], \quad i = 1, ..., k}{(\vee_i \beta_i) \ \leftarrow\ (\vee_i \alpha_i) \otimes [\wedge_i \phi_i]}$$

In general, when a disjunctive transaction such as $\alpha \vee \beta$ is executed, either α or β can execute, where the choice is non-deterministic. The lemma below gives conditions under which the choice becomes deterministic, *e.g.*, conditions under which β is guaranteed to execute. The idea is simple: suppose that a necessary pre-condition for α is false when $\alpha \vee \beta$ begins executing; then α cannot execute, so β must execute. A dual rule for post-conditions is also given.

Lemma 8 (Loss of Choice). *The following inference rules can be derived in system $\mathcal{T}$, where α, β and ϕ are transaction formulas:*

$$\frac{[\phi] \otimes \alpha \ \leftarrow\ \alpha}{[\neg\phi] \otimes (\alpha \vee \beta) \ \rightarrow\ \beta} \qquad\qquad \frac{\alpha \ \rightarrow\ \alpha \otimes [\phi]}{\beta \ \leftarrow\ (\alpha \vee \beta) \otimes [\neg\phi]}$$

6 Proving Properties of Update Programs

Inference system $\mathcal{T}$ developed in the last section provides a core of general axioms and inference rules for reasoning about update programs. However, it lacks knowledge about specific updates. In particular, it knows nothing about elementary database operations, such as $q.ins$ and $q.del$, which insert and delete facts from the database. It also knows nothing about the transaction base, which defines named procedures, such as subroutines and database views. This section provides the missing knowledge. In particular, we augment system $\mathcal{T}$ with axioms that describe elementary operations and transaction bases. The resulting inference system, called $\mathcal{T}^{\mathbf{P}}$, can reason about the effects of user-defined update programs. This section defines $\mathcal{T}^{\mathbf{P}}$ and illustrates how to use it to prove properties of programs.

Definition 12. (Inference System $\mathcal{T}^{\mathbf{P}}$) *If* $\mathbf{P}$ *is a serial-Horn transaction base, then* $\mathcal{T}^{\mathbf{P}}$ *consists of the axioms and inference rules in system* $\mathcal{T}$ *plus the axioms below.*

Elementary Operations: *If* b *and* c *are distinct base facts, then*

Effect Axioms:	$c.ins \leftrightarrow c.ins \otimes [c]$	$c.del \leftrightarrow c.del \otimes [\neg c]$
Frame Axioms:	$[b] \otimes c.ins \leftrightarrow c.ins \otimes [b]$	$[b] \otimes c.del \leftrightarrow c.del \otimes [b]$
Query Axiom:	$b \leftrightarrow [b]$	

Completion Axioms: *If* q *is a defined atom, then*

$q \leftrightarrow (\phi_1 \vee \phi_2 \vee \cdots \vee \phi_n)$ *if* $\mathbf{P}$ *has exactly* n *rules of the form* $q \leftarrow \phi_i$,

$q \leftrightarrow \neg\mathtt{path}$ *if* $\mathbf{P}$ *has no rules defining* q.

Intuitively, the effect axioms say that inserting c makes c true, and deleting c makes c false. The frame axioms say that inserting or deleting c does not affect b. The query axiom says that b is a query. The completion axiom is the analogue in $\mathcal{TR}$ of the Clark completion in classical logic programs [Cla78].

Theorem 6 (Soundness). *Every transaction formula derivable in system* $\mathcal{T}^{\mathbf{P}}$ *is a property of* $\mathbf{P}$*, i.e., it is true in the minimal model of* $\mathbf{P}$.

It is well-known that no inference system can be complete for proving properties of general programs, since such reasoning is not even semi-decidable. An interesting question, however, is for what class of transactions and properties does a completeness result hold for $\mathcal{T}^{\mathbf{P}}$? Although we do not have an answer to this question at the moment, we conjecture that $\mathcal{T}^{\mathbf{P}}$ is complete for proving success and effect properties (defined in Section 4) of programs defined by serial-Horn transaction bases (defined in Section 2.2) as long as (i) the transaction base is ground and non-recursive, and (ii) the properties are boolean combinations of base facts. In addition, we conjecture that $\mathcal{T}^{\mathbf{P}}$ can easily be extended to (i) transaction bases with variables, and (ii) properties with quantifiers and variables.

6.1 Examples

This section illustrates how to use inference system $\mathcal{T}^{\mathbf{P}}$ to reason about update programs. The examples derive many of the properties expressed in Section 4, including programs involving sequential composition, pre-conditions, non-determinism, and subroutines. The examples have also been chosen to illustrate the various axioms and inference rules of $\mathcal{T}^{\mathbf{P}}$. Most derivations are presented in full detail, in order to illustrate the proper use of the system.

Example 14. **(Basic Properties of Sequential Transactions)** We derive the following properties of the sequential transaction $b.ins \otimes c.del$, where a, b and c are distinct base facts. These properties are similar to those expressed in Example 7.

$$b.ins \otimes c.del \rightarrow b.ins \otimes c.del \otimes [\neg c]$$
$$b.ins \otimes c.del \rightarrow b.ins \otimes c.del \otimes [b]$$
$$[a] \otimes b.ins \otimes c.del \leftrightarrow b.ins \otimes c.del \otimes [a]$$

First, we infer that $b.ins \otimes c.del$ makes c false:

1.	$c.del \ \rightarrow \ c.del \otimes [\neg c]$	Effect Axiom
2.	$b.ins \otimes c.del \ \rightarrow \ b.ins \otimes c.del \otimes [\neg c]$	by Attachment.

Second, we infer that $b.ins \otimes c.del$ makes b true:

1.	$b.ins \ \rightarrow \ b.ins \otimes [b]$	Effect Axiom
2.	$b.ins \otimes c.del \ \rightarrow \ b.ins \otimes [b] \otimes c.del$	by Attachment
3.	$[b] \otimes c.del \ \leftrightarrow \ c.del \otimes [b]$	Frame Axiom
4.	$b.ins \otimes c.del \ \rightarrow \ b.ins \otimes c.del \otimes [b]$	by 2, 3 and Substitution.

Finally, we show that $b.ins \otimes c.del$ preserves the value of a:

1.	$[a] \otimes b.ins \otimes c.del \ \leftrightarrow \ [a] \otimes b.ins \otimes c.del$	Classical Axiom $\alpha \leftrightarrow \alpha$
2.	$[a] \otimes b.ins \ \leftrightarrow \ b.ins \otimes [a]$	Frame Axiom
3.	$[a] \otimes b.ins \otimes c.del \ \leftrightarrow \ b.ins \otimes [a] \otimes c.del$	by 1, 2 and Substitution
4.	$[a] \otimes c.del \ \leftrightarrow \ c.del \otimes [a]$	Frame Axiom
5.	$[a] \otimes b.ins \otimes c.del \ \leftrightarrow \ b.ins \otimes c.del \otimes [a]$	by 3, 4 and Substitution.

By combining the basic properties derived in Example 14, we can derive more-complex properties, as the next example shows.

Example 15. **(Complex Properties of Sequential Transactions)** We derive the following property of the sequential transaction $b.ins \otimes c.del$:

$$[a] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [a \wedge b \wedge \neg c]$$

First, we put the three properties from Example 14 into the form required by Lemma 6:

1.	$[\mathtt{path}] \otimes b.ins \;\leftrightarrow\; b.ins$	by Lemma 4
2.	$b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [b]$	by Example 14
3.	$[\mathtt{path}] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [b]$	by 1, 2 and Substitution
4.	$b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [\neg c]$	by Example 14
5.	$[\mathtt{path}] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [\neg c]$	by 1, 4 and Substitution
6.	$[a] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [a]$	by Example 14.

Next, we combine lines 3, 5 and 6, and simplify the result.

7.	$[\mathtt{path} \wedge \mathtt{path}] \otimes b.ins \otimes c.del$ $\quad\rightarrow\; b.ins \otimes c.del \otimes [b \wedge \neg c]$	by 3, 5, Lemma 6
8.	$\mathtt{path} \wedge \mathtt{path} \;\leftrightarrow\; \mathtt{path}$	Classical Axiom
9.	$[\mathtt{path}] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [b \wedge \neg c]$	by 7, 8, Substitution
10.	$[a \wedge \mathtt{path}] \otimes b.ins \otimes c.del$ $\quad\rightarrow\; b.ins \otimes c.del \otimes [a \wedge b \wedge \neg c]$	by 6, 9, Lemma 6
11.	$a \wedge \mathtt{path} \;\leftrightarrow\; a$	Classical Axiom
12.	$[a] \otimes b.ins \otimes c.del \;\rightarrow\; b.ins \otimes c.del \otimes [a \wedge b \wedge \neg c]$	by 10, 11, Substitution.

The transaction $b.ins \otimes c.del$, used in the examples above, contains database updates but no queries. To derive properties of transactions with queries, we sometimes need Lemma 9, below. Intuitively, this lemma says that if b is a base fact acting as a yes/no query, then it gives rise to the pre-condition $[b]$. This allows us to derive transaction pre-conditions from many of the queries in a transaction program, as shown in Example 16, below.

Lemma 9 (Deriving Pre-conditions from Queries). *If b is a base fact, then the formula $b \leftrightarrow [b] \otimes b$ can be derived in system $\mathcal{T}^{\mathbf{P}}$.*

Proof.

1.	$b \;\leftrightarrow\; [b]$	Query Axiom
2.	$b \;\leftrightarrow\; b \wedge b$	Classical Axiom
3.	$b \;\leftrightarrow\; [b \wedge b]$	by 1, 2 and Substitution
4.	$[b \wedge b] \;\leftrightarrow\; [b] \otimes [b]$	Auxiliary Axiom 3
5.	$b \;\leftrightarrow\; [b] \otimes [b]$	by 3, 4 and Substitution
6.	$b \;\leftrightarrow\; [b] \otimes b$	by 1, 5 and Substitution.

Example 16. **(Non-Determinism and Pre-conditions)** Consider a transaction that chooses non-deterministically between two sub-transactions, $a \otimes c.ins$ and $b \otimes d.del$. We derive the following two properties of this transaction:

$$(a \otimes c.ins \vee b \otimes d.del) \rightarrow (a \otimes c.ins \vee b \otimes d.del) \otimes [c \vee \neg d]$$

$$[\neg a] \otimes (a \otimes c.ins \vee b \otimes d.del) \rightarrow (a \otimes c.ins \vee b \otimes d.del) \otimes [\neg d]$$

The first property says that at the end of transaction execution, c is true or d is false. The second property says that if a is false at the start of execution, then d is false at the end. Here is the derivation of the first property:

1.	$c.ins \rightarrow c.ins \otimes [c]$	Effect Axiom
2.	$a \otimes c.ins \rightarrow a \otimes c.ins \otimes [c]$	by Attachment
3.	$d.del \rightarrow d.del \otimes [\neg d]$	Effect Axiom
4.	$b \otimes d.del \rightarrow b \otimes d.del \otimes [\neg d]$	by Attachment.

Applying Lemma 7(b) to lines 2 and 4 (using $a \otimes c.ins$ for α_1 and β_1, and using $b \otimes d.del$ for α_2 and β_2) gives the first property.

To derive the second property, we first use Lemma 9 to infer that $[a]$ is a pre-condition for the sub-transaction $a \otimes c.ins$.

5.	$[a] \otimes a \leftrightarrow a$	by Lemma 9
6.	$[a] \otimes a \otimes c.ins \leftrightarrow a \otimes c.ins$	by Attachment
7.	$[a] \otimes (a \otimes c.ins) \leftarrow (a \otimes c.ins)$	by Definition 6.

Next, we invoke Lemma 8 to remove the disjunct $a \otimes c.del$ from the transaction:

8.	$[\neg a] \otimes (a \otimes c.ins \vee b \otimes d.del) \rightarrow b \otimes d.del$	by 7 and Lemma 8
9.	$d.del \leftrightarrow d.del \otimes [\neg d]$	Effect Axiom
10.	$[\neg a] \otimes (a \otimes c.ins \vee b \otimes d.del) \rightarrow b \otimes d.del \otimes [\neg d]$	by 8, 9, Substitution.

Finally, we invoke classical inference to put the disjunct $a \otimes c.del$ back into the transaction:

11.	$b \otimes d.del \rightarrow (a \otimes c.ins \vee b \otimes d.del)$	Classical Axiom
12.	$b \otimes d.del \otimes [\neg d] \rightarrow (a \otimes c.ins \vee b \otimes d.del) \otimes [\neg d]$	by Attachment.

Applying Transitivity to lines 10 and 12 gives the second property.

Example 17. **(Subroutines)** As in Example 11, suppose that transaction base $\mathbf{P}$ consists of the following two rules, where a and b are base facts:

$$q \leftarrow a \otimes c.ins \qquad\qquad q \leftarrow b \otimes d.del$$

Then, the following two properties can be derived in system $\mathcal{T}^{\mathbf{P}}$:

$$q \rightarrow q \otimes [c \vee \neg d] \qquad\qquad [\neg a] \otimes q \rightarrow q \otimes [\neg d]$$

These properties follow immediately from the Completion Axiom and the results of Example 16. Here is the derivation of the first property:

1. $q \leftrightarrow (a \otimes c.ins \vee b \otimes d.del)$ Completion Axiom
2. $(a \otimes c.ins \vee b \otimes d.del$
   $\rightarrow (a \otimes c.ins \vee b \otimes d.del) \otimes [c \vee \neg d]$ by Example 16
3. $q \rightarrow q \otimes [c \vee \neg d]$ by 1, 2 and Substitution.

Here is the derivation of the second property:

4. $[\neg a] \otimes (a \otimes c.ins \vee b \otimes d.del)$
   $\rightarrow (a \otimes c.ins \vee b \otimes d.del) \otimes [\neg d]$ by Example 16
5. $[\neg a] \otimes q \rightarrow q \otimes [\neg d]$ by 1, 4 and Substitution.

7 Conclusions

Transaction Logic was originally introduced as a logical language for specifying and executing transactions that query and update database states. However, it quickly became apparent that being a general logic for expressing change-related phenomena, Transaction Logic has applications in areas outside databases and logic programming. For instance, the logic can be used for such typical AI problems as plan generation [BK95]. Recently, it has also been applied to workflow analysis [DKRR98].

Although many other formalisms have been developed for reasoning about action and change, they all represent significant deviations from the logic-programming paradigm. Dynamic Logic [Har79], Temporal Logic [Pnu77], and the situation calculus [McC63] are three examples. These formalisms were not intended for logic programming, and they are a world apart from Prolog with assert and retract. It may be possible to model Prolog updates using these formalisms (perhaps through a complex encoding), but the resulting semantics would do great violence to the traditional, classical semantics of logic programs.

Transaction Logic demonstrates that this is unnecessary. In $\mathcal{TR}$, one can specify, execute and reason about logic programs with updates, and remain firmly within the tradition of logic programming. In fact, the theory of classical logic programs is a special case of the theory of Transaction Logic programs: when updates are removed, the Horn fragment of Transaction Logic reduces to the Horn fragment of classical logic.

In previous works, we developed the semantics of Transaction Logic, and showed how to specify and execute Transaction Logic programs [BK98,BK94,BK93,BK95]. Syntactically, these programs look very much like Prolog programs with assert and retract. Semantically, they also behave like Prolog *except* that updates are treated as database transactions, so they are rolled back whenever a program fails. This one difference leads to a simple and natural model theory for logic programs with destructive updates. It also improves upon Prolog with assert and retract in several ways. For instance, (i)

the semantics of updating programs is purely logical; (*ii*) the semantics does not depend on rule order; (*iii*) programs are easier to understand, debug and maintain; and (*iv*) it extends the logic programming paradigm to a wide range of database applications, in which updates are transactional.

In this paper, we took a first step in showing how to reason about $\mathcal{TR}$ programs. Again, this development is in the logic-programming tradition. For instance, unlike most other formalisms for reasoning about programs, we do *not* require two separate languages: a language for specifying programs, and a meta language for specifying properties of programs. Instead, we use Transaction Logic for *both* purposes: the Horn fragment is used to specify logic programs, and the full logic is used to express properties of these programs. This is just like using full classical logic to express properties of classical logic programs.

In addition to *expressing* properties of $\mathcal{TR}$ programs, we also presented a sound inference system for *proving* these properties. We showed how to use the system to reason about the feasibility of transaction execution, and about the conditions that must hold before and after such executions. While this inference system is sound for recursively defined transactions, it is most useful for non-recursive transactions. It is not hard to see that there can be no complete inference system for reasoning about recursive programs, but we believe that such a system can be developed for the non-recursive case. We are currently investigating this possibility.

At the same time, it is clear that much work needs to be done to bring the theory of reasoning in $\mathcal{TR}$ to the level of the more established theories mentioned above. Inductive reasoning could be used to handle certain kinds of recursion, the set of properties reasoned about should be expanded, and work is underway to develop a resolution-based proof procedure for reasoning about transactions.

Finally, an extensive comparison of $\mathcal{TR}$ with other formalisms appears in [BK98,BK95]. The interested reader is referred to these discussions.

Acknowledgments. We thank Rodney Topor and the anonymous referees for their valuable comments. The first author was partially supported by a research grant from the Natural Sciences and Engineering Research Council of Canada (NSERC). The second author was partially supported by the NSF grant IRI-9404629.

References

[BK93] A.J. Bonner and M. Kifer. Transaction logic programming. In *Intl. Conference on Logic Programming*, pages 257–282, Budapest, Hungary, June 1993. MIT Press.

[BK94] A.J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133:205–265, October 1994.

[BK95] A.J. Bonner and M. Kifer. Transaction logic programming (or a logic of declarative and procedural knowledge). Technical Report CSRI-323, University of Toronto, November 1995. `http://www.cs.toronto.edu/~bonner/transaction-logic.html`.

[BK96] A.J. Bonner and M. Kifer. Concurrency and communication in transaction logic. In *Joint Intl. Conference and Symposium on Logic Programming*, pages 142–156, Bonn, Germany, September 1996. MIT Press.

[BK98] A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.

[Bon97a] A.J. Bonner. The power of cooperating transactions. Manuscript, 1997.

[Bon97b] A.J. Bonner. Transaction Datalog: a compositional language for transaction programming. In *Proceedings of the International Workshop on Database Programming Languages*, Estes Park, Colorado, August 1997. Springer Verlag.

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 292–322. Plenum Press, 1978.

[DKRR98] H. Davulcu, M. Kifer, C.R. Ramakrishnan, and I.V. Ramakrishnan. Logic based modeling and analysis of workflows. In *ACM Symposium on Principles of Database Systems*, June 1998.

[Har79] D. Harel. *First-Order Dynamic Logic*, volume 68 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[Hun96] Samuel Y.K. Hung. Implementation and Performance of Transaction Logic in Prolog. Master's thesis, Department of Computer Science, University of Toronto, 1996. `http://www.cs.toronto.edu/~bonner/transaction-logic.html`.

[Kif95] M. Kifer. Deductive and object-oriented data languages: A quest for integration. In *Intl. Conference on Deductive and Object-Oriented Databases*, volume 1013 of *Lecture Notes in Computer Science*, pages 187–212, Singapore, December 1995. Springer-Verlag. Keynote address at the 3d Intl. Conference on Deductive and Object-Oriented databases.

[KLW95] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, pages 741–843, July 1995.

[McC63] J. McCarthy. Situations, actions, and clausal laws, memo 2. Stanford Artificial Intelligence Project, 1963.

[MH69] J.M. McCarthy and P.J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969. Reprinted in *Readings in Artificial Intelligence*, 1981, Tioga Publ. Co.

[Pnu77] A. Pnueli. A temporal logic of programs. In *Intl. Conference on Foundations of Computer Science*, pages 46–57, October 1977.

[Rei91] R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifschitz, editor, *Aritifial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarty*, pages 359–380. Academic Press, 1991.