INTRODUCTION TO TRANSACTION LOGIC

— TUTORIAL —

Michael Kifer

University at Stony Brook
N.Y. 11794, U.S.A.

* Transaction Logic was developed jointly with Tony Bonner of University of Toronto

History

- 1991: decided to look into the theoretical foundations of logic programming with updates
- 1992: serial Transaction Logic is born
- 1994: graduates to concurrent Transaction Logic
- 1995: Transaction F-logic
- 1996: serial part of Transaction Logic implemented
- 1997: an implementation of Transaction F-logic (in Spain)
- 1998: (forthcoming) more efficient implementation of Transaction Logic

What Transaction Logic Is

- A logic designed for programming state-changing actions, executing them, and reasoning about their effects
- General logic, a conservative extension of classical predicate calculus
- Integrates declarative queries, transactional updates (abort, rollback, nested transactions), and composition thereof in one uniform, logical framework
- General Model Theory
  - Can do monotonic and non-monotonic reasoning
  - We **do not** want to commit to a particular choice of a non-monotonic theory: *Let's first understand the logic behind the phenomenon of updates!* Well-founded, stable, etc., semantics are orthogonal issues
- Proof Theory
  - Sound and complete
  - SLD-style for so-called serial-Horn programs (a generalization of the regular Horn programs)

What Transaction Logic Is (contd)

- Makes no assumption about the nature of the database states being updated. A database state can be:
  - relational databases
  - disjunctive databases
  - logic programs
  - classical first-order theories
  - non-logical entities
- Makes no assumptions about the nature of elementary updates, which can be:
  - simple tuple insertion/deletions
  - relational SQL-style bulk updates
  - updates/revisions of logical theories
  - non-logical state changes done by an algorithm
- **But**: if assumptions are made, Transaction Logic can be used to ***reason*** about the effects of actions

What Transaction Logic Is Not

- Not another theory of updates for another logical theory
  - not an attempt to explain what "update ϕ with χ" means
  - but such theories can be adapted/developed/used
- Not another variation on the theme of the situation calculus
- Not of Datalog-With-A-State-Argument variety

Why Transaction Logic?

- No acceptable logical language where **transactional updates** are integrated with **queries** *and* have a clean, logical semantics.
- No acceptable logical account for methods with **side effects** in object-oriented languages.
- No logic of action became the basis for updates in databases or logic programming.

Contrast with:
Classical logic *is* a basis for queries in logic programming and databases.

What Transaction Logic Does

Logic:

- transactional assert/retract
- methods in object-oriented DBMS
- integration of declarative and "procedural" knowledge

Transactional features:

- nested transactions
- atomicity
- isolation
- triggers
- deterministic and non-deterministic transactions
- dynamic constraints

What Transaction Logic Does (contd)

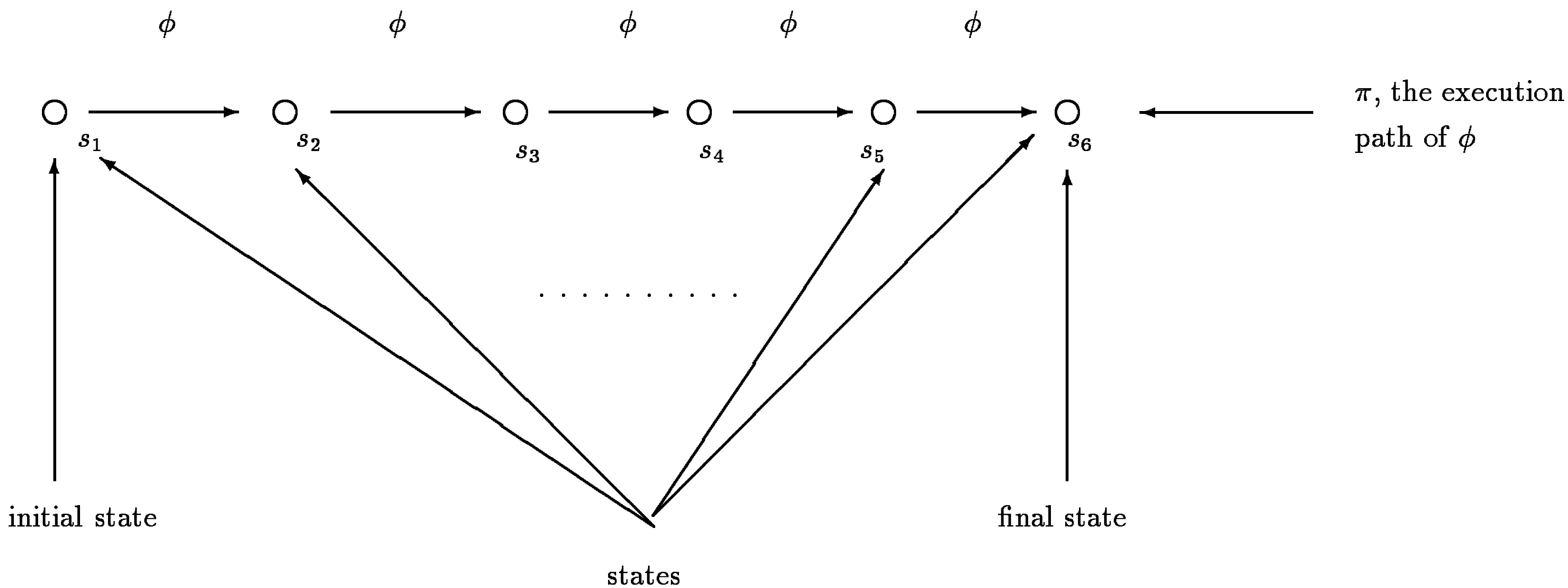
Control:

- subroutines
- serial and parallel composition of processes
- recursion, loops, conditionals
- communication and synchronization between concurrent processes

AI:

- logic for specifying and reasoning about actions
- language for specifying <u>and</u> generating plans
- frame problem:
  - not an issue for action execution
  - much smaller issue for reasoning about actions

The Whole Thing in One Slide



- Path: $\pi = \langle s_1, s_2, s_3, s_4, s_5, s_6 \rangle$
- Real world and semantics: ϕ **executes** along π $\equiv$ ϕ is **true** on π
- Proof theory: **executes** ϕ along π as it **proves** ϕ

Syntax

$\wedge, \vee, \neg$ — "classical" connectives
$\otimes, |, \odot, \Diamond$ — new connectives

- $\alpha \wedge \beta$ – execute α so that it would also be a valid execution of β. (Usually used in the context where β is a constraint on the execution of α.)
- $\alpha \vee \beta$ – execute α or execute β (*non-determinism*).
- $\neg\alpha$ – execute in any way, provided that the resulting execution is not a valid execution of α.
- $\alpha \otimes \beta$ – Execute α then execute β (*serial conjunction*).
- $\alpha \mid \beta$ – Execute α and β in parallel (*parallel conjunction*).
- $\odot\alpha$ – Execute α in *isolation* (like in the database theory).
- $\Diamond\alpha$ – Check if execution of α is possible.
- $\exists X \alpha(X)$ – Execute α for some X.

Syntax: examples

Rules:

- $a \leftarrow b$ ($\equiv a \vee \neg b$) means: one way to execute a is to execute b.
  Operationally: subroutine definition. *E.g.*,

$$a \leftarrow b \otimes (c \mid d) \otimes e$$
$$a \leftarrow f \otimes ((g \otimes h) \mid \odot(k \otimes f))$$
$$a \leftarrow \Diamond p \otimes q \otimes r$$

  Read: a is a subroutine, which can be executed in one of the following three ways:

  1. execute b, then c and d concurrently, then e; or
  2. execute f, then execute g followed by h concurrently with an isolated execution of k followed by f; or
  3. check if executing p is possible; if so, execute q then r

Constraints:

- $p \wedge (\mathtt{path} \otimes a \otimes \mathtt{path})$, where $\mathtt{path} \equiv \phi \vee \neg\phi$ – Transaction Logic's "true", means: execute p in such a way that action a is executed at some point during the process
- $p \wedge \neg(\mathtt{path} \otimes a \otimes \mathtt{path})$, means: execute p in such a way that action a is never executed in the process
- $p \wedge \neg(\mathtt{path} \otimes a \otimes \neg b \otimes \mathtt{path})$, means: execute p so that if a is executed at some point, then b is executed right after that

Overview of the Semantics

Any formula in Transaction Logic is a transaction/action/updating program/... (formulas with high degree of indeterminacy are better thought of as dynamic constraints, though).
– Formulas (*i.e.*, transactions) have ***truth values*** and ***execution paths***.
– Truth (or falsehood) is always over ***paths***, not over states.

- A **path** is a sequence of states.
- Transaction ϕ being **true** on path $\pi = \langle s_1, s_2, s_3, ..., s_n\rangle$ means:

  ϕ can execute at state s_1, changing it to state s_2, ..., to s_n, terminating at s_n.

  $\Rightarrow$ ***Truth*** over a path $\equiv$ ***execution*** over that path.
- There is more to it with parallel execution. Basic idea: execution happens over multi-paths — paths with "pauses"; other transactions can execute during those pauses.

Queries are transactions that execute over 1-paths (length-1; have the form $\langle s\rangle$).
$\Rightarrow$ queries are transactions that do not change state.

- When execution is restricted to 1-paths, Transaction Logic reduces to classical logic
- The three conjunctions, $\wedge$, $\otimes$, $|$, then all reduce to the classical $\wedge$,
- but they are distinct notions over n-paths ($n > 1$).

Examples of Execution

- Let $\phi_1 \quad = \quad a.del \otimes b.ins \otimes d \otimes c.ins$
  ($a.del$, $b.ins$, d, etc., are propositions for now, will explain later)

ϕ_1 started at state $\{d, a\}$ can pass through states $\{d\}$, $\{d, b\}$; verifies that d is true at the latter state; then goes to state $\{d, b, c\}$, and terminates.
$\Rightarrow \quad \phi_1$ is true over path $\pi = \langle \{d, a\}, \{d\}, \{d, b\}, \{d, b, c\} \rangle$.

- Let $\phi_2 = \phi_1 \otimes e$

Works like ϕ_1, but at the end (at $\{d, b, c\}$) checks if e is true.
Finds out that e is false, so π is not an execution path of ϕ_2.
$\Rightarrow \phi_2$ is false over π.
(In fact, it happens to be false over every path that starts at $\{d, a\}$ in some model.)

What is the nature of states?
And what are these strange-looking symbols: $a.del$, $b.ins$, etc.?

States

- Can think of the states as sets of atoms.
- Or formulas.
- But this is inadequate, in general:

  $p \leftarrow q$ means one thing in classical semantics, another in logic programming. Throw in the stable-model vs. well-founded semantics, add some spice (disjunctive programs, stationary semantics), and you get the idea.

- Transaction logic ***isolates*** the details of state semantics from the rest through data oracles:
  - A **data oracle** is simply a mapping
    $\mathcal{O}^d : States \longrightarrow Sets\ of\ First\text{-}order\ Formulas$
  - $\mathcal{O}^d(s)$ tells the logic what's true at state s.

Elementary Updates

- The strange-looking $a.del$, $b.ins$, etc., are just some ordinary propositions that happen to denote elementary updates (merely our notational convention).
- The semantics of elementary updates is specified via transition oracles.
- Transaction Logic is parameterized by data oracles and transition oracles.
- Each incarnation of the logic has its own data oracle (determines the set of allowed states and their semantics) and transition oracle (determines the set of allowed elementary transitions).
- The rest of the logic is independent of this choice: once the oracles are specified, the machinery cranks up and begins to run.

Transition Oracles

- *Transition oracles* are mappings of the form:

$$\mathcal{O}^t \;:\; States \times States \longrightarrow SetsOfGroundAtoms$$

- $b \in \mathcal{O}^t(\mathbf{D}_1, \mathbf{D}_2)$ means, executing b causes state transition from state $\mathbf{D}_1$ to $\mathbf{D}_2$.

In this tutorial: States are relational databases (sets of atoms).
State transitions can be of only these kinds:

- *Insert:* $p.ins(t_1, ..., t_n) \in \mathcal{O}^t(\mathbf{D}_1, \mathbf{D}_2)$ iff $\mathbf{D}_2 = \mathbf{D}_1 \cup \{p(t_1, ..., t_n)\}$.
- *Delete:* $p.del(t_1, ..., t_n) \in \mathcal{O}^t(\mathbf{D}_1, \mathbf{D}_2)$ iff $\mathbf{D}_2 = \mathbf{D}_1 - \{p(t_1, ..., t_n)\}$.

Can have more complex elementary updates: theory revision/update a la Katsuno-Mendelzon, rule insertion/deletion to/from logic programs, stack operations, etc.

A Database Example: Financial Transactions

$$transfer(Amt, Acct1, Acct2) \leftarrow withdraw(Amt, Acct1) \mid deposit(Amt, Acct2)$$

$$withdraw(Amt, Acct) \leftarrow \odot\, (balance(Acct, Bal) \otimes Bal \geq Amt \\ \otimes changeBalance(Acct, Bal, Bal - Amt))$$

$$deposit(Amt, Acct) \leftarrow \odot\, (balance(Acct, Bal) \otimes changeBalance(Acct, Bal, Bal + Amt))$$

$$changeBalance(Acct, Bal1, Bal2) \leftarrow balance.del(Acct, Bal1) \\ \otimes balance.ins(Acct, Bal2)$$

- All variables are implicitly universally quantified (as usual in LP).

Query:

$$?-\ transfer(Fee, Client, Broker) \mid transfer(Cost, Client, Seller)$$

- Note: Prolog will **not** execute correctly anything analogous to this (because actions in Prolog lack transactional features).

Semantics — Path Structures

A ***path structure*** is a creature that assigns ordinary first-order semantic structures to paths (more precisely, multi-paths, but we will not press this issue here):

$$\mathbf{M} : \mathit{Paths} \longrightarrow \mathit{FirstOrderSemanticStructures}.$$

Two conditions tie in the oracles:

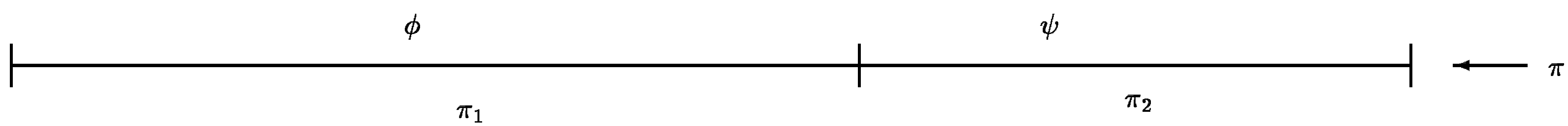
- *Data oracle compliance:* if $\mathbf{D}$ is a state, ϕ is a first-order formula, and $\mathcal{O}^d(\mathbf{D}) \models^c \phi$ ($\models^c$ means classical logical entailment), then $M(\langle \mathbf{D} \rangle) \models^c \phi$.
- *Transition oracle compliance:* If $\mathcal{O}^t(\mathbf{D}_1, \mathbf{D}_2) \models^c \psi$ then $M(\langle \mathbf{D}_1, \mathbf{D}_2 \rangle) \models^c \psi$.

Omitting some gory details:

1. **Base Case:** $\mathbf{M}, \pi \models p(t_1, \ldots, t_n)$ iff $\mathbf{M}(\pi) \models^c p(t_1, \ldots, t_n)$, for any atomic formula $p(t_1, \ldots, t_n)$.
   (Read: $p(t_1, \ldots, t_n)$ is a query or a transaction invocation;
   π is its execution path)
2. **Negation:** $\mathbf{M}, \pi \models \neg\phi$ iff $\mathbf{not}(\mathbf{M}, \pi \models \phi)$.
   (Read: cannot execute ϕ along the path π.)

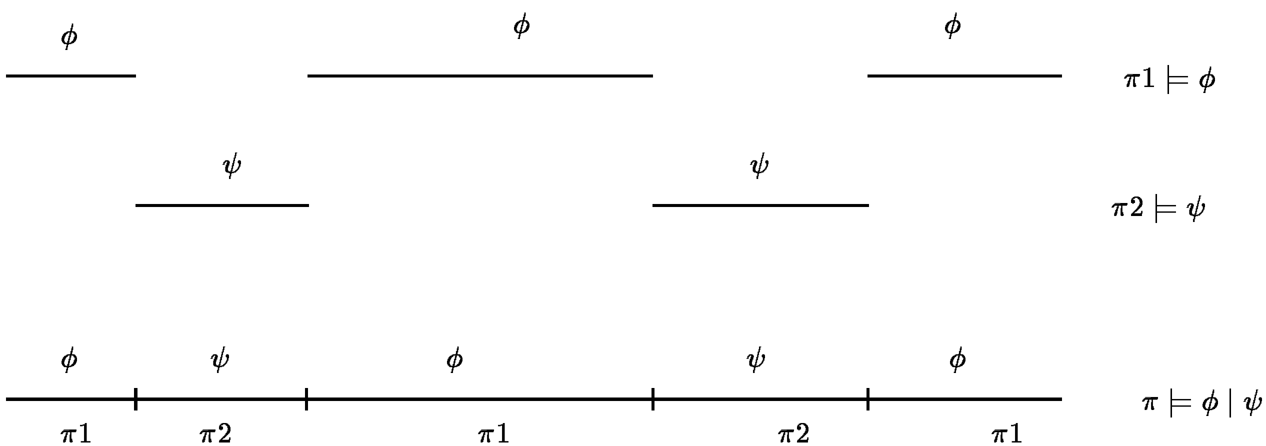
Semantics — Path Structures (contd.)

3. **"Classical" Conjunction:** $\mathbf{M}, \pi \models \phi \wedge \psi$ iff $\mathbf{M}, \pi \models \phi$ and $\mathbf{M}, \pi \models \psi$. (Read: can exec ϕ and ψ along the same path—dynamic constraints.)
4. **Serial Conjunction:** $\mathbf{M}, \pi \models \phi \otimes \psi$ iff $\mathbf{M}, \pi_1 \models \phi$ and $\mathbf{M}, \pi_2 \models \psi$ for *some* paths π_1, π_2 such that $\pi = \pi_1 \circ \pi_2$. (Read: do ϕ then ψ.)



Semantics — Path Structures (contd.)

5. **Concurrent Conjunction:** $\mathbf{M}, \pi \models \phi \mid \psi$ iff $\mathbf{M}, \pi_1 \models \phi$ and $\mathbf{M}, \pi_2 \models \psi$ for *some* paths π_1, π_2 such that $\pi \in \pi_1 \| \pi_2$.
   (Read: do ϕ and ψ concurrently.)



6. **Possibility:** $M, \langle s_1 \rangle \models \Diamond\phi$ iff there is a path $\pi = \langle s_1, ..., s_n \rangle$ such that $M, \pi \models \phi$.
   Note: $\Diamond\phi$ is always a query (is true at states, even if ϕ executes over a sequence of states longer than 1).

- Will not properly define $\odot$, $|$, $\exists$ in this tutorial (so read!)

Semantics – Example

So, why is $\phi_1 = a.del \otimes b.ins \otimes d \otimes c.ins$ true (executes) over the path $\pi = \langle\{d,a\},\{d\},\{d,b\},\{d,b,c\}\rangle$?

Let M be a path structure. By the definitions of our oracle and path structures:

- $\mathcal{O}^t(\{d,a\},\{d\}) \models a.del$, hence $M, \langle\{d,a\},\{d\}\rangle \models a.del$
  $$\{d,a\} \;\text{—}\; a.del \longrightarrow \{d\}$$

- $\mathcal{O}^t(\{d\},\{d,b\}) \models b.ins$, hence $M, \langle\{d\},\{d,b\}\rangle \models b.ins$
  $$\{d\} \;\text{—}\; b.ins \longrightarrow \{d,b\}$$

- $\mathcal{O}^d(\{d,b\}) \models d$, hence $M, \langle\{d,b\}\rangle \models d$

- $\mathcal{O}^t(\{d,b\},\{d,b,c\}) \models c.ins$, hence $M, \langle\{d,b\},\{d,b,c\}\rangle \models c.ins$
  $$\{d,b\} \;\text{—}\; c.ins \longrightarrow \{d,b,c\}$$

$\Rightarrow$ the definition of $\otimes$ implies that then $M, \pi \models \phi_1$

Semantics — Example (contd.)

More generally, let $\mathbf{P} = \{p \leftarrow a.del \otimes b.ins \otimes d \otimes c.ins\}$ (a *transaction program*).
As before: $\pi = \langle\{d,a\},\{d\},\{d,b\},\{d,b,c\}\rangle$.

We can show that if M is a path structure where $\mathbf{P}$ is true over *every* path, then also

$$M, \pi \models p$$

In fact, $M, \pi \models a.del \otimes b.ins \otimes d \otimes c.ins$ implies $M, \pi \models p$ in such path structures.
Read: $\mathbf{P}$ defines the subroutine p.

Are there M's where the above is not true? — No!

In contrast, in some path structures $M_1, \pi \not\models a.del \otimes b.ins \otimes d \otimes c.ins \otimes e$
and in some $M_2, \pi \models a.del \otimes b.ins \otimes d \otimes c.ins \otimes e$

- This leads to the notion of executional entailment.

Execution as Logical Entailment

Let $\mathbf{P}$ be a *transaction program* — a bunch of formulas (transaction definitions).

- $\mathbf{M}$ is a *model* of $\mathbf{P}$ iff $\mathbf{M}, \pi \models \phi$ for every path π and every $\phi \in \mathbf{P}$.
- If ϕ is a formula, and $\mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n$ is a sequence of database state ids, then ***executional entailment*** is a statement of the form:

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

  It means:

  $$\mathbf{M}, \langle \mathbf{D}_0, \mathbf{D}_1, \ldots, \mathbf{D}_n \rangle \models \phi$$

  for every model $\mathbf{M}$ of $\mathbf{P}$.

Proof Theory

A simple SLD-style procedure for *Concurrent Horn Clauses*.

Just 4 inference rules:

- An SLD-like rule.
- A rule for dealing with queries to states.
- A rule for executing state transitions.
- A rule for isolated execution.

Concurrent Horn Clauses:

- Rules of the form: $atom \leftarrow ConcurrentSerialGoal$
- *Concurrent Serial Goal*:
  - An atomic formula; or
  - $(\phi_1 \otimes ... \otimes \phi_k)$, where each ϕ_i is a concurrent serial goal; or
  - $(\phi_1 \mid ... \mid \phi_k)$, where each ϕ_i is a concurrent serial goal; or
  - $\odot\, \phi$, where ϕ is a concurrent serial goal.

Proof Theory (contd.)

- Uses **sequents** of the form: $\mathbf{P}, \mathbf{D} \text{---}\vdash \phi$
  meaning: ϕ can execute starting from state $\mathbf{D}$, given the transaction definitions in $\mathbf{P}$.
- Inference rules are of the form: $\mathtt{Condition}$, $\frac{\mathtt{sequent}_1}{\mathtt{sequent}_2}$
  meaning: if $\mathtt{Condition}$ is true and $\mathtt{sequent}_1$ has been proven then derive $\mathtt{sequent}_2$.
- Proves statements of the form: $\mathbf{P}, \mathbf{D} \text{---}\vdash \phi$
  and finds the execution path along the way.

Axiom: $\mathbf{P}, \mathbf{D} \text{---} \vdash (\,)$
where ( ) is the *empty* concurrent serial goal.

Proof Theory — Example

A top-down proof of $\mathbf{P}, \{c, d\} \text{---}\vdash p \mid (a \otimes c.del \otimes d.del)$
where $\mathbf{P} = \{p \leftarrow a.ins \otimes b.ins\}$.

$\mathbf{P}, \{c, d\} \text{---}\vdash (a.ins \otimes b.ins) \mid (a \otimes c.del \otimes d.del)$	unfold with $p \leftarrow a.ins \otimes b.ins$
$\mathbf{P}, \{c, d, a\} \text{---}\vdash b.ins \mid (a \otimes c.del \otimes d.del)$	executed $a.ins$; changed state
$\mathbf{P}, \{c, d, a\} \text{---}\vdash b.ins \mid (c.del \otimes d.del)$	tested and discarded a; same state
$\mathbf{P}, \{d, a\} \text{---}\vdash b.ins \mid d.del$	executed $c.del$; changed state
$\mathbf{P}, \{a\} \text{---}\vdash b.ins$	executed $d.del$; removed the empty conjunction ( )
$\mathbf{P}, \{a, b\} \text{---}\vdash (\,)$	executed $b.ins$; changed state

Ended up with an axiom $\Rightarrow$ done!

Extract execution path from the proof:

$$\{c, d\},\ \{c, d, a\},\ \{d, a\},\ \{a\},\ \{a, b\}$$

Final state: $\{a, b\}$.

Proof Theory—Inference rules

- No variables, to simplify exposition.

1. *Applying transaction definitions:* Let $b \leftarrow \beta \in \mathbf{P}$.

$$\frac{\mathbf{P},\mathbf{D}\,\text{---}\vdash (\beta \otimes \alpha) \mid \gamma}{\mathbf{P},\mathbf{D}\,\text{---}\vdash (b \otimes \alpha) \mid \gamma}$$

2. *Querying the database:* If $\mathcal{O}^d(\mathbf{D}) \models^c d$:

$$\frac{\mathbf{P},\mathbf{D}\,\text{---}\vdash \alpha \mid \beta}{\mathbf{P},\mathbf{D}\,\text{---}\vdash (d \otimes \alpha) \mid \beta}$$

3. *Executing elementary updates:* If $\mathcal{O}^t(\mathbf{D}_1,\mathbf{D}_2) \models^c u$:

$$\frac{\mathbf{P},\mathbf{D}_2\,\text{---}\vdash \alpha \mid \beta}{\mathbf{P},\mathbf{D}_1\,\text{---}\vdash (u \otimes \alpha) \mid \beta}$$

4. *Isolated execution of transactions:*

$$\frac{\mathbf{P},\mathbf{D}\,\text{---}\vdash \alpha \otimes (\beta \mid \gamma)}{\mathbf{P},\mathbf{D}\,\text{---}\vdash (\odot(\alpha) \otimes \beta) \mid \gamma}$$

More Examples: Blocks World

$$stack(N, X) \leftarrow N > 0 \otimes move(Y, X) \otimes stack(N-1, Y)$$
$$stack(0, X) \leftarrow$$
$$move(X, Y) \leftarrow pickup(X) \otimes putdown(X, Y)$$

$$pickup(X) \leftarrow clear(X) \otimes on(X, Y) \otimes on.del(X, Y) \otimes clear.ins(Y)$$
$$putdown(X, Y) \leftarrow wider(Y, X) \otimes clear(Y) \otimes on.ins(X, Y) \otimes clear.del\ (Y)$$

Note: ***stack*** is non-deterministic.

Can go beyond specification of actions: it is easy to declaratively specify a planning strategy (*e.g.*, STRIPS), crank the proof theory — and out comes a plan!

Summary

- A logic for specifying, executing, and reasoning about transactions.
- Syntax:
  - ***Serial logic:*** first-order plus $\otimes$, $\Diamond$
  - ***Concurrent logic:*** serial plus $|$, $\odot$.
- Parameterized by data and transition oracles

  Can "plug in" different oracles and get different logics, tailored to specific applications.
- Model theory, proof theory.
- Uniformly integrates queries, updates, and transactions.

Applications

1. Transactional updates in logic programming and deductive databases.
2. Active databases.
3. Consistency maintenance.
4. Hypothetical reasoning.
5. Planning.
6. Object-oriented databases.
7. Workflow management systems.

All for the price of (1)!

Further Info

One implementation of the serial part of Transition Logic, one more forthcoming.

Tony Bonner maintains a page at

`http://www.cs.toronto.edu/~bonner/transaction-logic.html`

I also maintain related info at

`http://www.cs.sunysb.edu/~kifer/dood/`

(will put this tutorial there soon).