



Transaction Logic

Formalising database updates

presented by

Eyal Oren
DERI, Galway





Transaction Logic:

- ▶ logic for state change
- ▶ accounts declaratively for update-related phenomena
- ▶ difference to other formalisms:
 - ▶ pure logical formalism
 - ▶ allows programming (unlike action logics)
 - ▶ allows subroutine specifications





Why Logic Again?

- ▶ different databases
- ▶ different data models
- ▶ different query languages
- ▶ logic is a unifying framework to
 - ▶ understand semantics
 - ▶ compare expressiveness





What is the Problem?

- ▶ query languages: based on predicate logic (relational algebra)
- ▶ database updates: no declarative semantics (in databases or logic programming)
- ▶ no logical framework cleanly accounts for updates in databases
 - ▶ action logics not designed for database programming
 - ▶ action logics *reason* about programs
 - ▶ action logics are hypothetical
 - ▶ action logics suffer from the frame problem





How About Prolog?

- ▶ solves some problems: (i) real updates (ii) composable named procedures (iii) predicates with truth value and side effects (iv) frame problem not an issue
- ▶ no transaction mechanisms: no rollbacks
- ▶ updates non-logical
 - ▶ what does $assert(Y) \vee assert(X)$ or $\neg assert(X)$ mean
 - ▶ order of rules/updates relevant
 - ▶ not sound and complete





What is Transaction Logic (\mathcal{TR})

- ▶ *specifying* and *executing* logical procedures that update and change the database
- ▶ accounts in clean and declarative way for state changes
- ▶ combining simple actions, controlling their execution
- ▶ allows for programming transactions
- ▶ hypothetical and committed updates
- ▶ dynamic constraints on transaction execution
- ▶ nondeterminism
- ▶ bulk updates





TR Overview – Implication

- ▶ subprocedures¹ :
 - ▶ $newMember(X) \leftarrow member.ins(X)$
 - ▶ $removeMember(X) \leftarrow member.del(X)$
 - ▶ if you execute `?- newMember(piet)`, `member.ins(piet)` will get executed
- ▶ nondeterministic subprocedure²:
 - ▶ $flipCoin() \leftarrow coins.ins(tail)$
 - ▶ $flipCoin() \leftarrow coins.ins(head)$
 - ▶ `?- flipCoin` will result in either `coins.ins(tail)` or `coins.ins(heads)`



¹if β is true, then α is true; β is one possible way to execute α
² $p.ins(X)$, $p.del(X)$ is defined by transition oracle



TR Overview – Serial Conjunction

- ▶ do α and then do β :
 - ▶ $flipAndNewMember \leftarrow flipCoin() \otimes newMember(piet)$
 - ▶ $doSomething \leftarrow p.ins(a, b, c) \otimes p.del(d, e, f) \otimes q.ins(z)$
- ▶ preconditions and postconditions:
 - ▶ $sweet \otimes busy.ins$
 - ▶ $won.ins \otimes happy$
 - ▶ if happy is true in final state transaction commits; otherwise transaction fails and database is rolled back to initial state





\mathcal{TR} Overview – Non Determinism

- ▶ Classical Disjunction

- ▶ $game \leftarrow won.ins \vee lost.ins$

- ▶ Negation

- ▶ $\neg\alpha$

- ▶ do anything but α

- ▶ Existential Quantification

- ▶ ?- $\exists X has.ins(X)$

- ▶ inserts has.ins(c) for some (non-determ. chosen) c





Hypothetical Reasoning

- ▶ two modal operators, for reasoning about modality of sentences: \diamond and \square
 - ▶ $\diamond\phi$ means execution of ϕ is *possible* in present state
 - ▶ $\square\phi$ means execution of ϕ is *necessary* in present state
- ▶ two model formula for reasoning in retrospection:
 - ▶ $\phi\diamond$ tests whether we *could* have executed ϕ to reach the current state
 - ▶ $\phi\square$ tests whether we *must* have executed ϕ to reach the current state
- ▶ example: **if-then-else** in \mathcal{TR}
 - ▶ $if_a_b_c \leftarrow (\diamond a) \otimes b$
 - ▶ $if_a_b_c \leftarrow (\square \neg a) \otimes c$





Example: Bank Account

- ▶ $transfer(Amt, Acct1, Acct2) \leftarrow withdraw(Amt, Acct1) \otimes deposit(Amt, Acct2)$
- ▶ $withdraw(Amt, Acct) \leftarrow balance(Acct, Bal) \otimes Bal \geq$
 $Amt \otimes change_balance(Acct, Bal, Bal - Amt)$
- ▶ $deposit(Amt, Acct) \leftarrow$
 $balance(Acct, Bal) \otimes change_balance(Acct, Bal, Bal + Amt)$
- ▶ $change_balance(Acct, Bal1, Bal2) \leftarrow$
 $balance.del(Acct, Bal1) \otimes balance.ins(Acct, Bal2)$





\mathcal{TR} is Orthogonal to Elementary Operations

- ▶ emphasis is not on specification of elementary updates but on their logical combination
- ▶ \mathcal{TR} can accommodate any set of elementary operations
- ▶ database is collection of datatypes, with methods for accessing them
- ▶ two oracles “plugged into” \mathcal{TR}
 - ▶ state oracle defines states and queries
 - ▶ transition oracle defines transitions and updates
- ▶ \mathcal{TR} : programs transactions using any theory of updates
- ▶ \mathcal{TR} : accommodates wide variety of database semantics





Limitations of Commercial Databases

- ▶ in standard SQL one cannot combine/nest transactions (in contrast to nesting views)
- ▶ solution: embedding SQL in procedural language: harder to optimise
- ▶ I'm not sure about this ...





Application: View Updates I

- ▶ $takes(Stud, Crs, Sec)$
records students enrolled in section of a course
- ▶ $instructs(Prof, Crs, Sec)$
records professors who teach section of a course
- ▶ $enrolled(Crs, N)$: total number of students in a course
- ▶ $load(Prof, N)$: total number of classes a professor teaches
- ▶ $drop(Stud, Crs, Sec) \leftarrow$
 $takes(Stud, Crs, Sec) \otimes takes.del(Stud, Crs, Sec) \otimes decr_enrolled(Crs)$
- ▶ ...





Application: View Updates II

- ▶ now we define a view on who teaches whom:
 $teaches(Prof, Stud, Crs) \leftarrow takes(Stud, Crs, Sec) \otimes instructs(Prof, Crs, Sec)$
- ▶ what happens when we delete tuples to *teaches*? should it be deleted from *takes*, from *instructs* or both?
- ▶ the transition oracle does not specify *teaches.del* (it is not elementary):
make a non-deterministic transaction for deleting, or two distinct ones





Application: View Updates III

- ▶ $del_teaches(Prof, Stud, Crs) \leftarrow takes(Stud, Crs, Sec) \otimes drop(Stud, Crs, Sec)$
either drop the student
- ▶ $del_teaches(Prof, Stud, Crs) \leftarrow takes(Stud, Crs, Sec) \otimes relieve(Prof, Crs, Sec)$
or relieve the teacher
- ▶ now the choice is non-deterministic; user may want to restrict it:
- ▶ $load(stefan, N) \otimes del_teaches(stefan, S, cs100) \otimes load(stefan, N)$
- ▶ now $del_teaches$ is executed only if the load of stefan stays the same: if possible a student is dropped





What I did not cover

- ▶ concurrency
- ▶ transaction isolation
- ▶ partial commits
- ▶ difference between Horn-like subset and full \mathcal{TR}
- ▶ bulk updates
- ▶ (did not mention explicitly) paths and multipaths

