

Transaction Logic and Active Rules

Michael Kifer

*State University of New York
at Stony Brook
USA*

What is Transaction Logic?

- A logic for programming change
- Designed both for programming *and* reasoning
 - Other logics, e.g., situation calculus, temporal, dynamic, and process logics are designed for reasoning only
 - They typically lack such basic facility as subroutines
- Applications:
 - Workflow/process modeling
 - Planning
 - And *active rules*

Bonner&Kifer, An Overview of Transaction Logic, in *Theoretical Computer Science*, 1995.

Bonner&Kifer, A Logic for Programming Database Transactions, in *Logics for Databases and Information Systems*, Chomicki&Saake (eds), Kluwer, 1998.

Bonner&Kifer, Results on Reasoning about Action in Transaction Logic, in *Transactions and Change in Logic Databases*, LNCS 1472, 1998

Problems with State Dynamics in Logic Programming

- *assert/retract* have no logical semantics
- Non-backtrackable, e.g.,
 - ? – *assert(p), fail.*

leaves *p* around

- Prolog actions are *not atomic* in the database sense
- Prolog programs with updates are the hardest to write, debug, and understand

Transaction Logic: Basic Idea

- Make all state-changing actions *atomic*
 - either execute to completion or leave no effect
- This makes state-changing actions compositional and easy to specify declaratively
 - Leads to a wide range of applications: active rules, process modeling, etc.
- Model-theoretic semantics
- Proof theory *executes* actions

ECA as a Rule in Transaction Logic

```
ECARule(...) :- eventDetected(...),  
                condition(...),  
                action(...),  
                postcondition(...)
```