

# Modeling Processes Using Transaction Logic

A presentation for RuleML 2013 Doctoral Consortium

Reza Basheda

Stony Brook University (SUNY)

[rbasheda@cs.stonybrook.edu](mailto:rbasheda@cs.stonybrook.edu)

July 13, 2013

- 1 Introduction
- 2 Transaction Logic ( $\mathcal{TR}$ )
- 3 Methodology
- 4 Experiments
- 5 Conclusion

# Motivation

- Procedural knowledge in scientific knowledge bases
- Given a scientific knowledge base including processes
  - Example: A biological knowledge base including cell development processes
  - How to specify and query the procedural knowledge
  - Example Queries:
    - A researcher treats cells with a chemical that prevents DNA synthesis from starting. This treatment traps the cells in which part of the cell cycle?
    - How many chromosomes does a rat cell have when it is formed?
    - In mitosis, what step follows Metaphase?
  - Changing states while a process is being executed

# Background

- Integration of dynamic behavior and static components
- Declarative programming languages and Imperative specification languages
- Different frameworks can represent processes
  - Process Algebra
  - Action Modeling Languages
  - Semantic Web Frameworks
- Different challenges for above mentioned frameworks
  - Not a knowledge representation language
  - Lack of proper implementations

# Transaction Logic ( $\mathcal{TR}$ )

- $\mathcal{TR}$ 
  - A general logic of state changes
  - A Horn-like fragment: supports logic programming
- Every transaction: A sequence of knowledge base state changes(execution path)
- Executorial entailment:  $P, D_0, \dots, D_n \models \psi$

# Types of Transactions

- Elementary updates:  $P, D_1, D_2 \models u$ 
  - A truth value
  - A side effect on the knowledge base
  - Example:  $P, D_1, D_2 \models u$
- Complex transaction:
  - $\mathcal{TR}$  connectives
  - Example: Serial conjunctions ( $\psi \otimes \phi$ )
    - $P, D_2, D_3 \models v$
    - $P, D_1, D_2, D_3 \models u \otimes v$
- Executorial entailment:  $P, D_0, \dots, D_n \models \psi$

# Semantics and Oracles

- Data oracle ( $\mathcal{O}^d$ )
  - *Static* semantics of states
- Transition oracle ( $\mathcal{O}^t$ )
  - Specifies a set of primitive knowledge base updates

## Example: A simple financial database system

- $balance(Act, Amt)$
- Elementary updates:
  - $balance.ins(Act, Amt)$ 
    - $balance.ins(c_1, m_1) \in \mathcal{O}^t(D_0, D_0 \cup \{balance(c_1, m_1)\})$
  - $balance.del(Act, Amt)$ 
    - $balance.del(c_1, m_1) \in \mathcal{O}^t(D_0, D_0 - \{balance(c_1, m_1)\})$
- Complex transaction:
  - $balance.change(Act, Bal, Bal')$
  - $withdraw(Amt, Act)$
  - $deposit(Amt, Act)$
  - $transfer(Amt, Act, Act')$
- If we have two accounts:  $c_1$  and  $c_2$ , with balance amount of  $m_1$  and  $m_2$  respectively, then  $D_0 = \{balance(c_1, m_1), balance(c_2, m_2)\}$  is a valid state



## Example: A simple financial database system (Cont.)

$$\begin{aligned}
 \text{transfer}(\text{Amt}, \text{Act}, \text{Act}') &\longleftarrow \text{withdraw}(\text{Amt}, \text{Act}) \otimes \text{deposit}(\text{Amt}, \text{Act}') \\
 \text{withdraw}(\text{Amt}, \text{Act}) &\longleftarrow \text{balance}(\text{Act}, B) \otimes B \geq \text{Amt} \otimes \\
 &\quad \text{balance.change}(\text{Act}, B, B - \text{Amt}) \\
 \text{deposit}(\text{Amt}, \text{Act}) &\longleftarrow \text{balance}(\text{Act}, B) \otimes \\
 &\quad \text{balance.change}(\text{Act}, B, B + \text{Amt}) \\
 \text{change}(\text{Act}, B, B') &\longleftarrow \text{balance.del}(\text{Act}, B) \otimes \text{balance.add}(\text{Act}, B')
 \end{aligned}$$

# Types of Processes

- Complex processes:
  - A sequence of complex processes
- Primitive processes:
  - A single step of execution

# Example

*complex\_process(p).*  
*complex\_process(p<sub>2</sub>).*  
*primitive\_process(p<sub>1</sub>).*  
*primitive\_process(p<sub>3</sub>).**primitive\_process(p<sub>2</sub><sub>1</sub>).*  
*primitive\_process(p<sub>2</sub><sub>2</sub>).*  
*primitive\_process(p<sub>2</sub><sub>3</sub>).*  
*first\_step(p, p<sub>1</sub>).*  
*first\_step(p<sub>2</sub>, p<sub>2</sub><sub>1</sub>).*  
*next\_step(p, p<sub>1</sub>, p<sub>2</sub>).*  
*next\_step(p, p<sub>2</sub>, p<sub>3</sub>).*  
*next\_step(p<sub>2</sub>, p<sub>2</sub><sub>1</sub>, p<sub>2</sub><sub>2</sub>).*  
*next\_step(p<sub>2</sub>, p<sub>2</sub><sub>2</sub>, p<sub>2</sub><sub>3</sub>).*

## Sequential execution of subprocesses

To keep the track of a complex process execution, we need a structure maintaining the execution status of the complex process. The current step of a process,  $current\_step(P, SP)$ , is an example of such a structure keeping execution status. A primitive process does not have structure and it only updates the knowledge base state based on its definition.

$$execute(P) \leftarrow complex\_process(P) \wedge current\_step(P, CS) \otimes \\ execute(CS) \otimes advance(P, CS) \otimes execute(P).$$

$$execute(P) \leftarrow complex\_process(P) \wedge \\ current\_step(P, CS) \wedge \sim next\_step(P, CS, -).$$

$$advance(P, CS) \leftarrow complex\_process(P) \wedge current\_step(P, CS) \\ \wedge next\_step(P, CS, NS) \otimes current\_step.delete(P, CS) \\ \otimes current\_step.insert(P, NS).$$

# Primitive Processes

- Execution
  - Defined in terms of *insert* and *delete*
  - Extend the transition oracle and define a specific primitive process execution as a elementary transaction
- Conditional statements

# Fault-tolerant execution

- If transaction  $execute(CS)$  fails and returns false, the transaction  $execute(P)$  also fails and returns false.
- Hypothetical reasoning:

$$\begin{aligned} execute(P) &\leftarrow complex\_process(P) \wedge current\_step(P, CS) \otimes \\ &\quad \sim \diamond execute(CS) \otimes failed(CS) \otimes \\ &\quad advance(P, CS) \otimes execute(P). \end{aligned}$$

# A cell mitosis division process

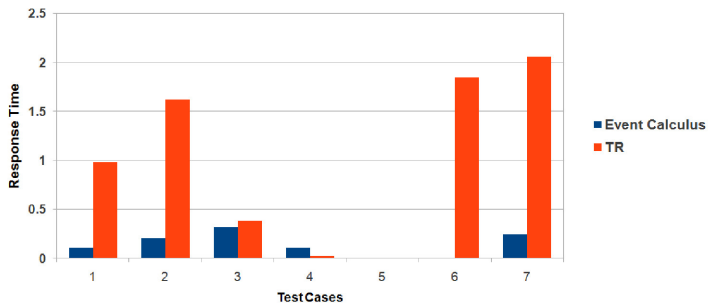
- A simple implementation of mitosis cell division process
  - Flora-2
  - An object oriented knowledge base language and application platform
- Comparison of
  - $\mathcal{AL}_d$  in SILK
  - An implementation based on the event calculus concepts in Flora2
  - $\mathcal{TR}$  in Flora2

# Lines of code

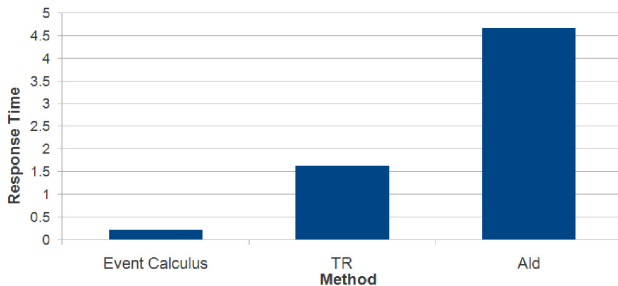
| Method           | Lines of Code |
|------------------|---------------|
| event calculus   | 1196          |
| $\mathcal{AL}_d$ | 707           |
| $\mathcal{TR}$   | 490           |



# Response Time



# Response Time (Cont.)



# Conclusion

- $\mathcal{TR}$ 
  - Allows definitions of processes as first class entities
  - Simplifies programs and makes them more extensible and reusable
  - Improves the response time of queries
- Future Directions
  - $\mathcal{TR}$ 
    - Scalability in terms of size and complexity of process descriptions
    - Expansion of elementary updates to domain specific updates
  - Consider other capabilities of  $\mathcal{TR}$  as a process representation tool
    - Concurrent behaviors???
    - Encode other process specification conventions such as process algebra.

Thank you