Modeling Processes Using Transaction Logic

A presentation for RuleML 2013 Doctoral Consortium

Reza Basseda

Stony Brook University (SUNY)

rbasseda@cs.stonybrook.edu

July 13, 2013
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
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 ($\mathcal{T} \mathcal{R}$)

- $\mathcal{T} \mathcal{R}$
  - A general logic of state changes
  - A Horn-like fragment: supports logic programming
- Every transaction: A sequence of knowledge base state changes (execution path)
- Executional entailment: $P, D_0, \cdots, D_n \models \psi$
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:**
  - $\mathcal{T}\mathcal{R}$ connectives
  - Example: Serial conjunctions $(\psi \otimes \phi)$
    - $P, D_2, D_3 \models v$
    - $P, D_1, D_2, D_3 \models u \otimes v$

- **Executional entailment:** $P, D_0, \cdots, D_n \models \psi$
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:
- $balance.ins(Act, Amt)$
  - $balance.ins(c_1, m_1) \in O^t(D_0, D_0 \cup \{balance(c_1, m_1)\})$
  - $balance.del(Act, Amt)$
    - $balance.del(c_1, m_1) \in O^t(D_0, D_0 - \{balance(c_1, m_1)\})$

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.)

\[
\begin{align*}
\text{transfer}(\text{Amt}, \text{Act}, \text{Act}') & \leftarrow \text{withdraw}(\text{Amt}, \text{Act}) \otimes \text{deposit}(\text{Amt}, \text{Act}') \\
\text{withdraw}(\text{Amt}, \text{Act}) & \leftarrow \text{balance}(\text{Act}, B) \otimes B \geq \text{Amt} \otimes \\
& \quad \text{balance.}\text{change}(\text{Act}, B, B - \text{Amt}) \\
\text{deposit}(\text{Amt}, \text{Act}) & \leftarrow \text{balance}(\text{Act}, B) \otimes \\
& \quad \text{balance.}\text{change}(\text{Act}, B, B + \text{Amt}) \\
\text{change}(\text{Act}, B, B') & \leftarrow \text{balance.}\text{del}(\text{Act}, B) \otimes \text{balance.}\text{add}(\text{Act}, B')
\end{align*}
\]
Types of Processes

- Complex processes:
  - A sequence of complex processes

- Primitive processes:
  - A single step of execution
Example

\[
\begin{align*}
\text{complex\_process}(p). \\
\text{complex\_process}(p_2). \\
\text{primitive\_process}(p_1). \\
\text{primitive\_process}(p_3). \text{primitive\_process}(p_{21}). \\
\text{primitive\_process}(p_{22}). \\
\text{primitive\_process}(p_{23}). \\
\text{first\_step}(p, p_1). \\
\text{first\_step}(p_2, p_{21}). \\
\text{next\_step}(p, p_1, p_2). \\
\text{next\_step}(p, p_2, p_3). \\
\text{next\_step}(p_2, p_{21}, p_{22}). \\
\text{next\_step}(p_2, p_{22}, p_{23}).
\end{align*}
\]
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, $\text{current\_step}(P, SP)$, 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.

\[
\text{execute}(P) \leftarrow \text{complex\_process}(P) \land \text{current\_step}(P, CS) \otimes \\
\text{execute}(CS) \otimes \text{advance}(P, CS) \otimes \text{execute}(P).
\]

\[
\text{execute}(P) \leftarrow \text{complex\_process}(P) \land \\
\text{current\_step}(P, CS) \land \sim \text{next\_step}(P, CS, \_).
\]

\[
\text{advance}(P, CS) \leftarrow \text{complex\_process}(P) \land \text{current\_step}(P, CS) \\
\land \text{next\_step}(P, CS, NS) \otimes \text{current\_step}.\text{delete}(P, CS) \\
\otimes \text{current\_step}.\text{insert}(P, NS).
\]
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 $\text{execute}(CS)$ fails and returns false, the transaction $\text{execute}(P)$ also fails and returns false.

- Hypothetical reasoning:

\[
\text{execute}(P) \leftarrow \text{complex \_ process}(P) \land \text{current \_ step}(P, CS) \\
\sim \Diamond \text{execute}(CS) \otimes \text{failed}(CS) \\
\text{advance}(P, CS) \otimes \text{execute}(P).
\]
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
  - $\mathcal{T}\mathcal{R}$ in Flora2
## Lines of code

<table>
<thead>
<tr>
<th>Method</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>event calculus</td>
<td>1196</td>
</tr>
<tr>
<td>$AL_d$</td>
<td>707</td>
</tr>
<tr>
<td>$TR$</td>
<td>490</td>
</tr>
</tbody>
</table>
Response Time

![Bar chart showing response time for different test cases. The chart compares Event Calculus (blue bars) and TR (red bars).]
Response Time (Cont.)

- Event Calculus
- TR Method
- Ald

The graph shows the comparison of response times for different methods.
**Conclusion**

- **$\mathcal{T}\mathcal{R}$**
  - 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**

- $\mathcal{T}\mathcal{R}$
  - Scalability in terms of size and complexity of process descriptions
  - Expansion of elementary updates to domain specific updates
  - Consider other capabilities of $\mathcal{T}\mathcal{R}$ as a process representation tool
    - Concurrent behaviors??
    - Encode other process specification conventions such as process algebra.
Thank you