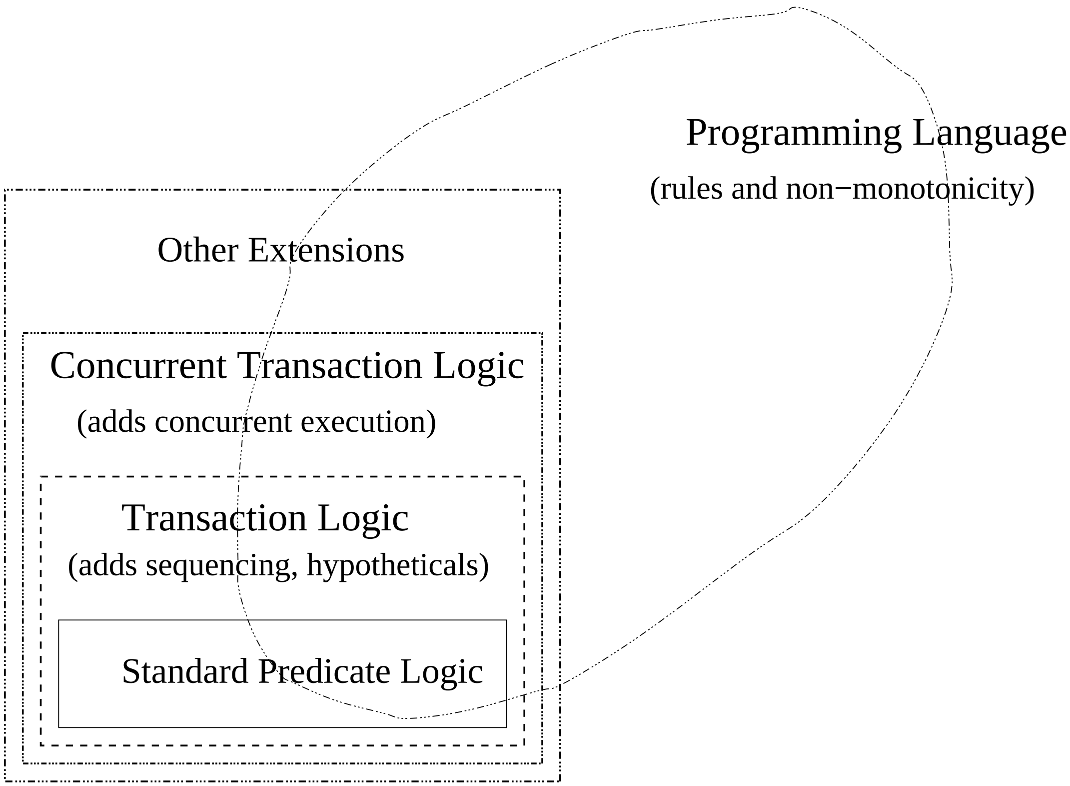
Concurrent Transaction Logic by Example
(with emphasis on stuff deemed to be relevant to SWS)

Michael Kifer

Stony Brook University, NY, USA

Transaction Logic Extends Predicate Calculus



What Is It?

1. Logic for defining "procedures" for querying and **updating** the underlying logical theory (database).
2. Not an ontology – unlike PSL.
3. Formulas are executable specifications that actually perform actions.
4. Programming language – "Prolog done right" if you will.
5. Does not specify properties of processes – at least not directly.
6. Orthogonal to things like Description Logic, F-logic, etc. Therefore, these can be used in conjunction.
7. Can be thought of as orthogonal to the ontology part of PSL.

What Is It? (Contd.)

1. Has model theory.
2. Proof theory for the programming part of the logic.
3. Formulas are viewed as programs (transactions) that perform operations that query and modify the database.
4. As the proof theory proves a formula, it finds the *execution path* of the formula, *i.e.*, a sequence of states that would result if the transaction executed.
   Transactions are actually executed as proofs get constructed.

Process Modeling with Concurrent Transaction Logic

1. Sequential composition of processes
2. Parallel composition of processes
3. Alternative executions (transactions can be non-deterministic)
4. Pre/post conditions
5. Constraints on execution (state constraints, temporal, etc.)
6. Workflow modeling and reasoning
7. Planning
8. Communication through Messages
9. ...

Syntax

$\wedge, \vee, \neg$ — "classical" connectives

$\otimes, \mid$ — new connective

Also hypothetical operators (will not discuss)

- $\alpha \wedge \beta$ – execute α so that it'll 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 <u>or</u> β (alternatives, non-determinism).
- $\neg\alpha$ – execute in any way, provided that the resulting execution is <u>not</u> a valid execution of α.
- $\alpha \otimes \beta$ – Execute α <u>then</u> execute β (*serial conjunction*).
- $\alpha \mid \beta$ – Execute α and β in parallel (*parallel conjunction*).
- $\exists X \alpha(X)$ – Execute α for <u>some</u> X.

Simple Example: Money Transfer

Transfer Amt from account `a123` to account Acct (variables are uppercased, $h \longleftarrow b \equiv h \vee \neg b$):

$$
\begin{aligned}
payTo(Amt, Acct) &\longleftarrow withdraw(Amt, \mathtt{a123}) \otimes deposit(Amt, Acct).\\
withdraw(Amt, Acct) &\longleftarrow balance(Acct, B) \otimes B \geq Amt \otimes\\
&\qquad \boldsymbol{delete}(balance(Acct, B)) \otimes\\
&\qquad \boldsymbol{insert}(balance(Acct, B - Amt)).\\
deposit(Amt, Acct) &\longleftarrow balance(Acct, B) \otimes\\
&\qquad \boldsymbol{delete}(balance(Acct, B)) \otimes\\
&\qquad \boldsymbol{insert}(balance(Acct, B + Amt)).
\end{aligned}
$$

$$\text{Query}: \ ?- \ payTo(100, mortgage) \otimes payTo(100, creditCard).$$

- What would Prolog do if account `a123` had only $150?

Actions That Change State

- Certain predicates can be defined as state-changing updates using the so-called *Transition Oracle*
- Transition oracle is a parameter to the logic.
  - Transition oracles can define very different state-changing operations (*e.g.*, insert/delete facts, insert/delete rules, etc.)
  - Different oracles make different Transaction Logics. (But the model theory and proof theory do not change – they are defined modulo the oracle.)
  - In the above (and other examples here) we assume that we are dealing with relational states (simple sets of facts).
  - Assume that the oracle defines the following updates:
    $delete(fact)$ in state $\mathbf{D}$ returns state $\mathbf{D} - \{fact\}$.
    $insert(fact)$ in state $\mathbf{D}$ returns state $\mathbf{D} \cup \{fact\}$.

Example: A Recursively Defined Transaction

Stacking blocks:

$$stack(0, X).$$

$$stack(N, X) \longleftarrow N > 0 \otimes move(Y, X) \otimes stack(N - 1, Y)$$

$$move(X, Y) \longleftarrow pickup(X) \ \otimes \ putdown(X, Y)$$

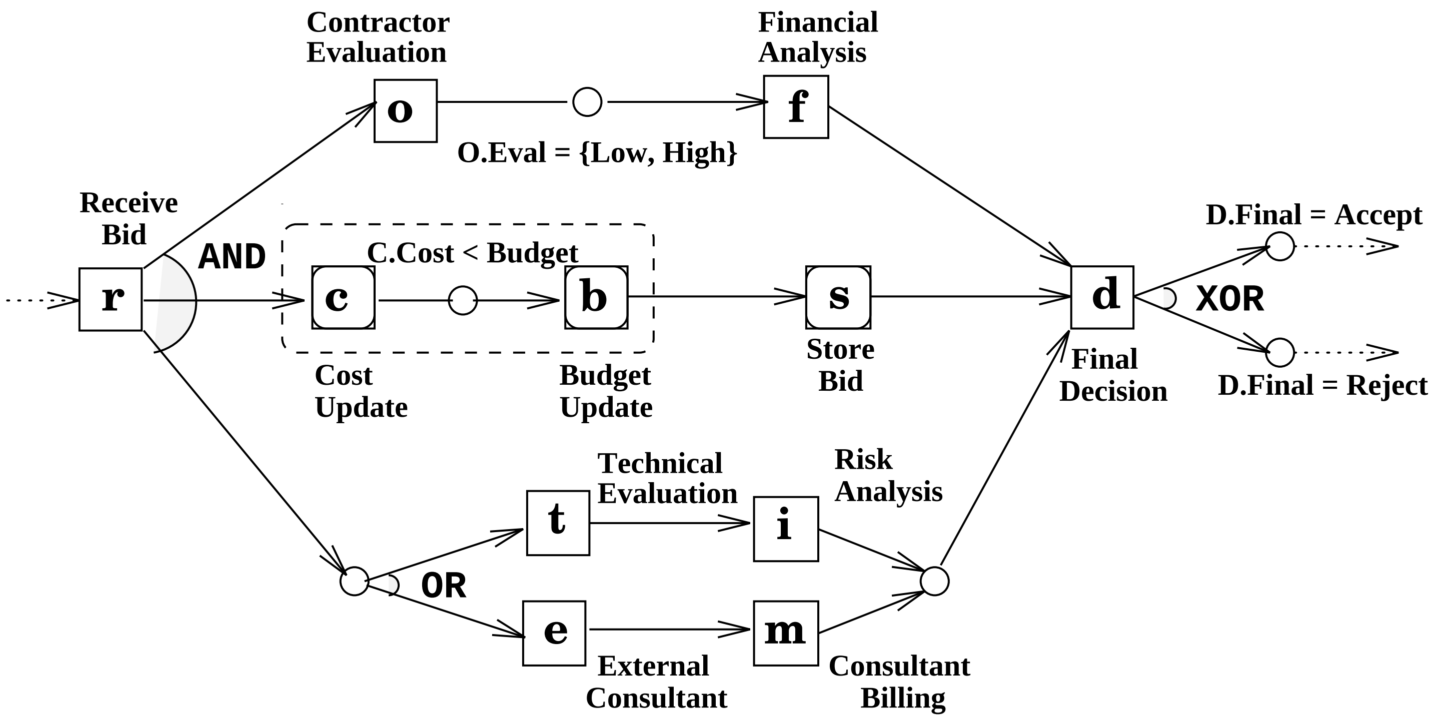
$$pickup(X) \longleftarrow clear(X) \ \otimes \ on(X, Y) \ \otimes \ delete(on(X, Y)) \ \otimes \ insert(clear(Y))$$

$$putdown(X, Y) \longleftarrow wider(Y, X) \ \otimes \ clear(Y) \ \otimes \ insert(on(X, Y)) \ \otimes \ delete(clear(Y))$$

Note: *stack* is non-deterministic and recursive

Bid Evaluation Workflow

❑ **Control Flow Graph:**



❑ **Global Coordination Dependencies:**

1. **IF o.eval = low THEN not e**
2. **IF occurs (e) THEN o before e**
3. **IF occurs (t) AND occurs (e) THEN e before i**
4. **c before f**

Capturing Bid Evaluation Workflow

- **Control-flow graphs** translates straightforwardly into logic programming style rules (in Concurrent Transaction Logic).
  $\odot$ - isolated (non-interleaved) execution – not discussed previously.

$$
\begin{array}{lll}
\mathit{bid\_eval} & \leftarrow & \mathbf{r} \otimes (\mathit{financial} \mid \mathit{db\_updates} \mid \mathit{technical}) \otimes \mathit{rest} \\
\mathit{financial} & \leftarrow & \mathbf{o} \otimes ([o.eval = \text{“}high\text{”}] \otimes \mathbf{f}) \vee (\mathit{low} \otimes \mathbf{f}) \\
\mathit{db\_updates} & \leftarrow & \odot(\mathbf{c} \otimes [c.cost < budget] \otimes \mathbf{b}) \otimes \mathbf{s} \\
\mathit{technical} & \leftarrow & (\mathbf{t} \otimes \mathbf{i}) \vee (\mathbf{e} \otimes \mathbf{m}) \vee (\mathbf{t} \otimes \mathbf{i} \mid \mathbf{e} \otimes \mathbf{m}) \\
\ldots.
\end{array}
$$

- **Global Coordination Dependencies** can be specified as well:

$$
\begin{array}{llll}
1. & \nabla low \rightarrow \neg \nabla \mathbf{e} & 3. & \nabla \mathbf{t} \wedge \nabla \mathbf{e} \rightarrow \nabla e \otimes \nabla \mathbf{i} \\
2. & \nabla \mathbf{e} \rightarrow (\nabla \mathbf{o} \otimes \nabla \mathbf{e}) & 4. & \nabla \mathbf{c} \otimes \nabla \mathbf{f}
\end{array}
$$

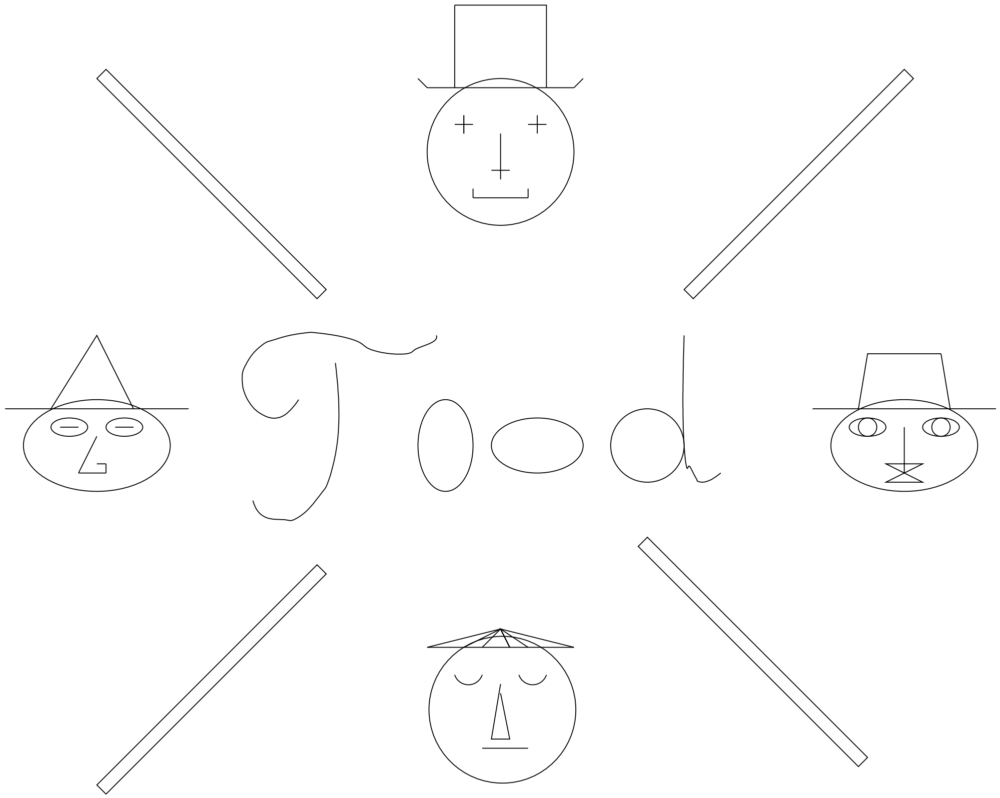
($\nabla\phi$ means "action ϕ *occurs* somewhere during execution" – can be expressed in CTR.)

Reasoning/Execution with Constraints

CTR proof theory can:

- Schedule workflows subject to constraints.
- Decide whether workflow is consistent with a set of constraints.
- Decide whether some constraints imply other constraints.

The Dining Philosophers: Communication and Messages



Dining Philosophers in CTR

N rounds of thinking & eating for a party of X philosophers:

$$\begin{aligned} thinkEat(Ph, X) \longleftarrow\ & think(Ph) \otimes take2Sticks(Ph, X) \\ & \otimes\ eat(Ph) \otimes put2Sticks(Ph, X). \\ thinkEatLoop(Ph, X, N) \longleftarrow\ & N > 0 \otimes thinkEat(Ph, X) \\ & \otimes\ thinkEatLoop(Ph, X, (N-1)\mathbf{mod}X). \\ thinkEatLoop(Ph, X, 1) \longleftarrow\ & \boldsymbol{send}(Ph, done). \end{aligned}$$

The Battle of the Chopsticks:

$$take2Sticks(Ph, X) \longleftarrow takeStick((Ph-1) \,\mathbf{mod}\, X) \otimes takeStick(Ph).$$

$$put2Sticks(Ph, X) \longleftarrow putStick((Ph-1)\,\mathbf{mod} X) \otimes putStick(Ph).$$

$$takeStick(Ph, St) \longleftarrow$$

$$\boldsymbol{send}(Ph, request(St)) \otimes \boldsymbol{receive}(Ph, grant(St)).$$

$$putStick(Ph, St) \longleftarrow \boldsymbol{send}(Ph, relinquish(St)).$$

$think(Ph) \longleftarrow$ ...definition of the thinking process.

$eat(Ph) \longleftarrow$ ...definition of the eating process.

The Stick Manager:

$$
\begin{aligned}
stickMngr(N) \leftarrow\ & \boldsymbol{receive}(Ph, request(St)) \\
& \otimes ontable(St) \otimes delete(ontable(St)) \\
& \otimes \boldsymbol{send}(Ph, granted(St)) \otimes stickMngr(N). \\
stickMngr(N) \leftarrow\ & \boldsymbol{receive}(Ph, relinquish(St)) \\
& \otimes insert(ontable(St)) \otimes stickMngr(N). \\
stickMngr(N) \leftarrow\ & \boldsymbol{receive}(Ph, done) \otimes stickMngr(N-1). \\
stickMngr(0). &
\end{aligned}
$$

Dinner for Three (100 rounds):

$$
\begin{aligned}
?-\ & thinkEatLoop(1,3,100) \mid thinkEatLoop(2,3,100) \\
& \mid thinkEatLoop(3,3,100) \mid stickMngr(3).
\end{aligned}
$$

Planning with Transaction Logic

- Main ideas illustrated using STRIPS.
- Easy to define much more sophisticated strategies.

STRIPS

A simple planning system. Actions have the form:

Name: unstack(X,Y)
Comment: Pick up block X from block Y
Precondition: handempty, clear(X), on(X,Y)
Delete: handempty, clear(X), on(X,Y)
Insert: clear(Y), holding(X)

- Uses an ad hoc algorithm to construct plans.
- Most AI planning systems use ad hoc algorithms.
- We can write planning strategies at the high level in Transaction Logic without worrying about the low-level details

The Planning Problem

- Given a set of primitive actions, $a_1, ..., a_n$
  - each a_i has a precondition and an effect (the definition of the change it makes)
- a goal, G (the condition on the final database state that we want to achieve)
- and the initial state $\mathbf{D}_0$

Find a sequence of the actions that starting at $\mathbf{D}_0$ leads to a state $\mathbf{D}$ that satisfies G.

Planning with TR

Naive planning is easy:

$$plan \longleftarrow action \otimes plan.$$

$$plan \longleftarrow action.$$

$$action \longleftarrow a_1.$$

$$\ldots$$

$$action \longleftarrow a_n.$$

Naive planning — just pose the query: $? - plan \otimes goal.$

For instance, $? - plan \otimes (on(b, c) \otimes on(c, d) \otimes clear(b)).$

will find a sequence of actions that puts b on c, c on d, and leaves b clear.

- **Problem:** inefficient, might search through all sequences.

Representing STRIPS in Transaction Logic

First, write a rule for each action — straightforward:

$$
\begin{aligned}
& unstack(X, Y) \ \longleftarrow \\
& \qquad handempty \otimes clear(X) \otimes on(X, Y) \\
& \qquad \otimes\ delete(clear(X)) \otimes delete(on(X, Y)) \\
& \qquad \otimes\ delete(handempty) \\
& \qquad \otimes\ insert(holding(X)) \otimes insert(clear(Y))
\end{aligned}
$$

Representing STRIPS in Transaction Logic (cont'd)

Second, show how to *achieve* each goal of interest:

$$achieveClear(Y) \longleftarrow achieveUnstack(X, Y).$$

$$achieveHolding(X) \longleftarrow achieveUnstack(X, Y).$$

$$\begin{aligned} achieveUnstack(X, Y) \longleftarrow \\ & (achieveClear(X) \mid achieveOn(X, Y) \mid achieveHandempty) \\ & \otimes\ unstack(X, Y). \end{aligned}$$

- To achieve a goal, achieve the precondition of an action that inserts that goal
- To achieve action precondition, achieve each of the subgoals in that precondition

Representing STRIPS in Transaction Logic (cont'd)

Base cases: if a goal is already true then it has been achieved.

$$achieveOn(X, Y) \longleftarrow on(X, Y).$$

$$achieveClear(Y) \longleftarrow clear(Y).$$

$$achieveHolding(X) \longleftarrow holding(Y).$$

$$achieveHandempty \longleftarrow handempty.$$

Representing STRIPS in Transaction Logic (cont'd)

A planning query: Stack c on d and b on c.

$$?-(achieveOn(b,c) \mid achieveOn(c,d)) \otimes on(b,c) \otimes on(c,d).$$

Finds a solution when one exists

- STRIPS was not based on a logic, so they had to develop an execution mechanism
- The original STRIPS was not complete. Was made complete after a series of papers
- Using an appropriate logic makes the whole problem trivial