20 years of Transaction Logic

Michael Kifer

Stony Brook University

August 28, 2013
Outline

How it all began

What is Transaction Logic?

What’s wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
The paper

The paper

- Never actually presented: Tony got stuck in a hospital in Prague on the way to Budapest 😁
How the work started

• 1991: I was invited by Alberto Mendelzon to spend a sabbatical year at U. of Toronto
How the work started

- 1991: I was invited by Alberto Mendelzon to spend a sabbatical year at U. of Toronto
- ... but Tony was the only one left around by the time I would get to the office
Motivation

\[ \mathcal{F}\text{-logic} \quad (\text{Kifer\&Lausen 1989}) \]

\[ \equiv \]

logic of objects

Person[name*=>string, spouse*=>person].
john:Person[age=>21, child=>{bill,bob}].

Logic for state-changing methods in objects

Knowledge base dynamics

even = select(X),odd[add:b(X)].
odd = select(X),even[add:b(X)].

Expressive language for state-changing actions

Hypothetical Datalog \quad (\text{Bonner 1989})

\[ \equiv \]

LP + hypotheticals
Outline

How it all began

What is Transaction Logic?

What’s wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
What is Transaction Logic?

- A logic for actions
  - Programming complex actions
  - Executing them
  - Reasoning about their effects
- Conservative extension of predicate calculus
- General model theory
  - Can do both monotonic and non-monotonic reasoning
- Proof theory (sound and complete)
What is Transaction Logic?

- A logic for actions
  - Programming complex actions
  - Executing them
  - Reasoning about their effects
- Conservative extension of predicate calculus
- General model theory
  - Can do both monotonic and non-monotonic reasoning
- Proof theory (sound and complete)
What is Transaction Logic?

- A logic for actions
  - Programming complex actions
  - Executing them
  - Reasoning about their effects
- Conservative extension of predicate calculus
- General model theory
  - Can do both monotonic and non-monotonic reasoning
- Proof theory (sound and complete)
What is Transaction Logic?

- A logic for actions
  - Programming complex actions
  - Executing them
  - Reasoning about their effects
- Conservative extension of predicate calculus
- General model theory
  - Can do both monotonic and non-monotonic reasoning
- Proof theory (sound and complete)
What is Transaction Logic (cont’d)

- A general framework that can be instantiated to many different interesting logics
- Underlying database states can be virtually anything
  - relational DBs
  - logic programs
  - first-order theories
  - with monotonic and nonmonotonic semantics
- Complex actions are composed out of elementary actions
- Elementary actions
  - can be arbitrary state changes viewed as “black boxes”
  - or they can be axiomatized
What is Transaction Logic (cont’d)

• A general framework that can be instantiated to many different interesting logics

• Underlying database states can be virtually anything
  • relational DBs
  • logic programs
  • first-order theories
  • with monotonic and nonmonotonic semantics

• Complex actions are composed out of elementary actions

• Elementary actions
  • can be arbitrary state changes viewed as “black boxes”
  • or they can be axiomatized
What is Transaction Logic (cont’d)

• A general framework that can be instantiated to many different interesting logics

• Underlying database states can be virtually anything
  • relational DBs
  • logic programs
  • first-order theories
  • with monotonic and nonmonotonic semantics

• Complex actions are composed out of *elementary actions*

• Elementary actions
  • can be arbitrary state changes viewed as “black boxes”
  • or they can be axiomatized
Why Transaction Logic?

- No *(back in 1991)* acceptable *programming* logic that integrates transactional updates with queries
- No acceptable logical account for methods with side-effects in object-oriented languages (e.g., in F-logic)
- No logic of actions became the basis for updates in LP
Why Transaction Logic?

- No (back in 1991) acceptable *programming* logic that integrates transactional updates with queries
- No acceptable logical account for methods with side-effects in object-oriented languages (e.g., in F-logic)
- No logic of actions became the basis for updates in LP
Why Transaction Logic?

- No (back in 1991) acceptable *programming* logic that integrates transactional updates with queries
- No acceptable logical account for methods with side-effects in object-oriented languages (e.g., in F-logic)
- No logic of actions became the basis for updates in LP
Why Transaction Logic?

• No (back in 1991) acceptable *programming* logic that integrates transactional updates with queries
• No acceptable logical account for methods with side-effects in object-oriented languages (e.g., in F-logic)
• No logic of actions became the basis for updates in LP
• This is still the case today!!!
Why Transaction Logic?

- No *(back in 1991)* acceptable *programming* logic that integrates transactional updates with queries
- No acceptable logical account for methods with side-effects in object-oriented languages (e.g., in F-logic)
- No logic of actions became the basis for updates in LP
- This is still the case today!!
- But these days this is mostly out of ignorance 😐
What Transaction Logic does

- Transactional state changes
  - actions are atomic
  - can be nested, hypothetical, isolated
  - deterministic and/or non-deterministic

- Control:
  - subroutines
  - serial and parallel composition of processes
  - recursion, conditionals
  - communication and synchronization among processes

- Methods for object-oriented LP (e.g., F-logic)
- Integrates declarative and procedural knowledge
What Transaction Logic does

- Transactional state changes
  - actions are atomic
  - can be nested, hypothetical, isolated
  - deterministic and/or non-deterministic

- Control:
  - subroutines
  - serial and parallel composition of processes
  - recursion, conditionals
  - communication and synchronization among processes

- Methods for object-oriented LP (e.g., F-logic)
- Integrates declarative and procedural knowledge
What Transaction Logic does

- Transactional state changes
  - actions are atomic
  - can be nested, hypothetical, isolated
  - deterministic and/or non-deterministic

- Control:
  - subroutines
  - serial and parallel composition of processes
  - recursion, conditionals
  - communication and synchronization among processes

- Methods for object-oriented LP (e.g., F-logic)
  - Integrates declarative and procedural knowledge
What Transaction Logic does

- Transactional state changes
  - actions are atomic
  - can be nested, hypothetical, isolated
  - deterministic and/or non-deterministic

- Control:
  - subroutines
  - serial and parallel composition of processes
  - recursion, conditionals
  - communication and synchronization among processes

- Methods for object-oriented LP (e.g., F-logic)
- Integrates declarative and procedural knowledge
Outline

How it all began

What is Transaction Logic?

What's wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
Prolog as an action language
Prolog as an action language

- What Prolog does right:
  - The programming style
  - Actions composed serially
  - Actions can be built hierarchically out of subroutines
Prolog as an action language

• What Prolog does right:
  • The programming style
  • Actions composed serially
  • Actions can be built hierarchically out of subroutines

• What Prolog does wrong:
  • Non-logical (of course)
Prolog as an action language

- **What Prolog does right:**
  - The programming style
  - Actions composed serially
  - Actions can be built hierarchically out of subroutines

- **What Prolog does wrong:**
  - Non-logical (of course)
  - Non-compositional:
    - Perfectly working actions may not work when combined
    - Queries and actions may not be combinable
Prolog as an action language

- What Prolog does right:
  - The programming style
  - Actions composed serially
  - Actions can be built hierarchically out of subroutines

- What Prolog does wrong:
  - Non-logical (of course)
  - Non-compositional:
    - Perfectly working actions may not work when combined
    - Queries and actions may not be combinable
  - No hypothetical actions, concurrency, dynamic constraints, but these are secondary
Example: Graph coloring

colorNode :-   %% color one node correctly
    node(N), not colored(N, _),
    color(C),
    not (adjacent(N, N2), colored(N2, C)),
    assert(colored(N, C)).

colorGraph :- not uncoloredNodesLeft.
colorGraph :- colorNode, colorGraph.
Example: Graph coloring

colorNode :- % color one node correctly
    node(N), not colored(N, _),
    color(C),
    not (adjacent(N, N2), colored(N2, C)),
    assert(colored(N, C)).

colorGraph :- not uncoloredNodesLeft.
colorGraph :- colorNode, colorGraph.

- Seems “logical,” but won’t work:
  bad choices of asserts will be stuck in the database
Graph Coloring: Illustration of the problem

Available colors:

Current choices:

False: $\exists C \in \text{color} \ (\text{not} \ \exists N2(\text{adjacent}(3, N2), \text{colored}(N2, C)))$

$\Rightarrow$ Backtrack ...

... but the wrong asserts $\text{colored}(4, \text{green})$ or $\text{colored}(1, \text{blue})$ or $\text{colored}(1, \text{blue})$ will stay in the DB!
Graph Coloring: Illustration of the problem

Available colors:

Current choices:

False: $\exists C \in color \ (not \ \exists N2(\text{adjacent}(3, N2), \text{colored}(N2, C)))$

$\implies$ Backtrack ...

... but the wrong asserts \text{colored}(4, \text{green}) or \text{colored}(1, \text{blue}) or \text{colored}(1, \text{blue}) will stay in the DB!
Graph Coloring: Illustration of the problem

Available colors: 🟢  🔴  🔵

Current choices: 🟢  🔴  🔵

False: ∃C ∈ color (not ∃N2(\text{adjacent}(3, N2), \text{colored}(N2, C)))

⇒ Backtrack ...

... but the wrong asserts \text{colored}(4, green) or \text{colored}(1, blue) or \text{colored}(1, blue) will stay in the DB!
Example: Building pyramids

stack(N,X) :- N>0, move(Y,X), stack(N-1,Y)
stack(0,X).
move(X,Y) :- pickup(X), putdown(X,Y)
pickup(X) :- clear(X), on(X,Y),
            retract(on(X,Y)), assert(clear(Y))
putdown(X,Y) :- wider(Y,X), clear(Y),
               assert(on(X,Y)), retract(clear(Y))

• Same thing: seems logical, but won’t work due to possible bad non-deterministic choices.
Example: Building pyramids

\[
\text{stack}(N,X) :\text{ }- \text{ }N>0, \text{ move}(Y,X), \text{ stack}(N-1,Y)
\]
\[
\text{stack}(0,X).
\]
\[
\text{move}(X,Y) :\text{ }- \text{ pickup}(X), \text{ putdown}(X,Y)
\]
\[
\text{pickup}(X) :\text{ }- \text{ clear}(X), \text{ on}(X,Y), \text{ retract(on}(X,Y)), \text{ assert(clear}(Y))
\]
\[
\text{putdown}(X,Y) :\text{ }- \text{ wider}(Y,X), \text{ clear}(Y), \text{ assert(on}(X,Y)), \text{ retract(clear}(Y))
\]

- Same thing: seems logical, but won’t work due to possible bad non-deterministic choices.
Both examples are correct in Transaction Logic

- In the Flora-2 syntax (http://flora.sourceforge.net).
  E.g., coloring — almost identical to Prolog:

```
%colorNode :- // color one node correctly
  node(?N), naf colored(?N,?),
  color(?C),
  naf exists(?N2)(adjacent(?N,?N2),colored(?N2,?C)),
  tinsert\{colored(?N,?C)\}.
%colorGraph :- naf uncoloredNodesLeft.
%colorGraph :- %colorNode, %colorGraph.
```
Both examples are correct in Transaction Logic

- In the Flora-2 syntax (http://flora.sourceforge.net).
  E.g., coloring — almost identical to Prolog:

%colorNode :- // color one node correctly
    node(?N), naf colored(?N,?),
    color(?C),
    naf exists(?N2)(adjacent(?N,?N2),colored(?N2,?C)),
    tinsert{colored(?N,?C)}.
%colorGraph :- naf uncoloredNodesLeft.
%colorGraph :- %colorNode, %colorGraph.

Has procedural flavor, but not an algorithm! — is declarative
Outline

How it all began

What is Transaction Logic?

What's wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
Basic Transaction Logic

- Extends normal LP programs (with negation, etc.) with serial conjunction $\otimes$ in rule bodies.
- $\textit{foo} \otimes \textit{bar}$ means: execute $\textit{foo}$ then execute $\textit{bar}$.
- Example:

  \[
  \text{colorNode} \leftarrow \\
  \quad \text{(node(N) } \land \text{ naf colored(N,_,) } \land \text{ color(C) } \land \\
  \quad \text{naf (adjacent(N,N2) } \land \text{ colored(N2,C)))}
  \otimes \text{ insert(colored(N,C)).}
  \]

  \[
  \text{colorGraph} \leftarrow \text{colorNode } \otimes \text{colorGraph.}
  \]

- For queries, $\land$ and $\otimes$ happen to boil down to the same connective (classical conjunction)
  
  ... so, in the above, we could have used $\otimes$ everywhere.
Basic Transaction Logic

- Extends normal LP programs (with negation, etc.) with serial conjunction $\otimes$ in rule bodies.
- $\textit{foo} \otimes \textit{bar}$ means: execute $\textit{foo}$ then execute $\textit{bar}$.
- Example:

  ```prolog
colorNode <-
  (node(N) ∧ naf colored(N,_) ∧ color(C) ∧
   naf (adjacent(N,N2) ∧ colored(N2,C)))
  \otimes insert(colored(N,C)).
colorGraph <- colorNode \otimes colorGraph.
```

- For queries, $\land$ and $\otimes$ happen to boil down to the same connective (classical conjunction)
  ... so, in the above, we could have used $\otimes$ everywhere.
Basic Transaction Logic

- Extends *normal* LP programs (with negation, etc.) with serial conjunction $\otimes$ in rule bodies.
- $\text{foo} \otimes \text{bar}$ means: execute $\text{foo}$ then execute $\text{bar}$.
- Example:
  
  \[
  \text{colorNode} \leftarrow \\\n  \quad (\text{node}(N) \land \text{naf colored}(N,_) \land \text{color}(C) \land \text{naf} (\text{adjacent}(N,N2) \land \text{colored}(N2,C))) \\\n  \quad \otimes \text{insert}(\text{colored}(N,C)).
  \]
  
  \[
  \text{colorGraph} \leftarrow \text{colorNode} \otimes \text{colorGraph}.
  \]
- For queries, $\land$ and $\otimes$ happen to boil down to the same connective (classical conjunction)
  
  ... so, in the above, we could have used $\otimes$ everywhere.
Informal semantics

- Formulas have truth values on *paths*, which are sequences of states. Path of length $M+1$:

\[\text{Path: } \quad \text{state } 0 \rightarrow \text{state } i_1 \rightarrow \text{state } i_2 \rightarrow \ldots \rightarrow \text{state } i_M\]

*State* can be anything: relational, rule sets, KBs with nonmon semantics, etc.

- Path of length 1 is a sequence having one state only:

\[\text{Path of length 1: } \quad \text{state } 0\]

- Queries have truth values over paths of length 1; actions typically involve paths of length $> 1$.

- Truth of formula over path $\equiv$ execution over that path.

- Execution is *atomic*: a formula either executes in full or not at all—exactly as database transactions.
Informal semantics

- Formulas have truth values on *paths*, which are sequences of states. Path of length $M+1$:

  ![Path Diagram](image)

  **State** can be anything: relational, rule sets, KBs with nonmon semantics, etc.

- Path of length 1 is a sequence having one state only:

  ![Path of Length 1](image)

- Queries have truth values over paths of length 1; actions typically involve paths of length $>1$.

- **Truth of formula over path** $\equiv$ *execution* over that path.

- Execution is *atomic*: a formula either executes in full or not at all—exactly as database transactions.
Informal semantics

• Formulas have truth values on *paths*, which are sequences of states. Path of length $M+1$:

  ![Path diagram](image)

  **State** can be anything: relational, rule sets, KBs with nonmon semantics, etc.

• Path of length 1 is a sequence having one state only:

  ![Path of length 1](image)

• Queries have truth values over paths of length 1; actions typically involve paths of length $> 1$.

• Truth of formula over path $\equiv$ *execution* over that path.

• Execution is *atomic*: a formula either executes in full or not at all—exactly as database transactions.
⊗: Informal semantics

$\text{action}_1 \otimes \text{action}_2$

* $\text{action}_i$ is a state-changing action or a query

Path: state 0 -- -- -- -- \rightarrow state i -- -- -- -- \rightarrow state M
⊗: Informal semantics

\[ action_1 \otimes action_2 \]

Path: state 0 \[ \rightarrow \] state i \[ \rightarrow \] state M

action_1
⊗: Informal semantics

\[ \text{action}_1 \otimes \text{action}_2 \]

Path:

\[ \begin{array}{cccc}
\text{state } 0 & - & - & - & \rightarrow & \text{state } i & - & - & - & \rightarrow & \text{state } M \\
\text{action}_1 & & & & \text{action}_2 & & & & \\
\end{array} \]
Informal semantics (rules)

\[ \text{action} \leftarrow \text{action}_1 \otimes \cdots \otimes \text{action}_M \]

* \text{action}_i \; \text{is a state-changing action or a query}
Informal semantics (rules)

\[ \text{action} \leftarrow \text{action}_1 \otimes \cdots \otimes \text{action}_M \]

\textbf{IF:}

\[ \text{Path:} \quad \text{state } 0 \rightarrow \text{state } i_1 \rightarrow \text{state } i_2 \rightarrow \cdots \rightarrow \text{state } i_M \]

\[ \text{action}_1 \rightarrow \text{action}_2 \rightarrow \text{action}_3 \rightarrow \cdots \rightarrow \text{action}_M \]
Informal semantics (rules)

\[ \text{action} \leftarrow \text{action}_1 \otimes \cdots \otimes \text{action}_M \]

THEN:

Path: 

\[
\begin{align*}
\text{state } 0 & \rightarrow \text{state } i_1 \rightarrow \text{state } i_2 \rightarrow \cdots \rightarrow \text{state } i_M \\
\text{action}_1 & \rightarrow \text{action}_2 \rightarrow \text{action}_3 \rightarrow \cdots \rightarrow \text{action}_M
\end{align*}
\]
Informal semantics (rules)

\[
action \leftarrow action_1 \otimes \cdots \otimes action_M
\]

The semantics can be intuitively understood in Prolog terms:

- *action* is a procedure one of whose definitions is the sequence of actions *action*$_1$, ..., *action*$_M$
- multiple definitions lead to non-determinism in execution
Advanced connectives

- **foo | bar**: do foo and bar concurrently.
- **◊foo**: check if foo is doable, but don’t do it.
- **⋄foo**: do foo in isolation (don’t allow it to interleave with other actions).
- **∧, ∨, ¬**: extend the corresponding classical connectives, but are more general. (They reduce to classical connectives over paths of length 1.)
- **∀, ∃**: ditto.
Advanced connectives

- **foo | bar**: do foo and bar concurrently.
- **◊foo**: check if foo is doable, but don’t do it.
- **⊙foo**: do foo in *isolation* (don’t allow it to interleave with other actions).
- **∧, ∨, ¬**: extend the corresponding classical connectives, but are more general. (They reduce to classical connectives over paths of length 1.)
- **∀, ∃**: ditto.
Planning example

• Natural for planning problems:

\[
\text{step} \leftarrow \text{action}_1. \\
... \\
\text{step} \leftarrow \text{action}_n. \\
\text{stepseq} \leftarrow. \\
\text{stepseq} \leftarrow \text{step} \otimes \text{stepseq}. \\
\text{plan} \leftarrow \text{stepseq} \otimes \text{planning_goal}. \\
\]

• To find a plan: issue the transaction \( ? \leftarrow \text{plan} \).

• Of course, the above is naive and impractical.

• But efficient planning strategies (STRIPS, hierarchical, etc.) are naturally & concisely expressible as Transaction Logic rules.
Planning example

- Natural for planning problems:

  \[
  \text{step} \leftarrow \text{action}_1.
  \]
  
  \[
  \ldots
  \]
  
  \[
  \text{step} \leftarrow \text{action}_n.
  \]
  
  \[
  \text{stepseq} \leftarrow .
  \]
  
  \[
  \text{stepseq} \leftarrow \text{step} \text{ } \otimes \text{ } \text{stepseq}.
  \]
  
  \[
  \text{plan} \leftarrow \text{stepseq} \text{ } \otimes \text{ } \text{planning}\_\text{goal}.
  \]

- To find a plan: issue the transaction \( ? - \text{plan} \).

- Of course, the above is naive and impractical.

- But efficient planning strategies (STRIPS, hierarchical, etc.) are naturally & concisely expressible as Transaction Logic rules.
Planning example

- Natural for planning problems:
  
  \[
  \text{step} \leftarrow \text{action}_1. \\
  \ldots \\
  \text{step} \leftarrow \text{action}_n. \\
  \text{stepseq} \leftarrow. \\
  \text{stepseq} \leftarrow \text{step} \otimes \text{stepseq}. \\
  \text{plan} \leftarrow \text{stepseq} \otimes \text{planning_goal}. 
  \]

- To find a plan: issue the transaction \( ? \leftarrow \text{plan}. \)
- Of course, the above is naive and impractical.

- But efficient planning strategies (STRIPS, hierarchical, etc.) are naturally & concisely expressible as Transaction Logic rules.
Planning example

- Natural for planning problems:
  
  \[
  \text{step} \leftarrow \text{action}_1.
  \]

  \[
  \ldots
  \]

  \[
  \text{step} \leftarrow \text{action}_n.
  \]

  \[
  \text{stepseq} \leftarrow .
  \]

  \[
  \text{stepseq} \leftarrow \text{step} \otimes \text{stepseq}.
  \]

  \[
  \text{plan} \leftarrow \text{stepseq} \otimes \text{planning\_goal}.
  \]

- To find a plan: issue the transaction \( ? \leftarrow \text{plan} \).

- Of course, the above is naive and impractical.

- But efficient planning strategies (STRIPS, hierarchical, etc.) are naturally & concisely expressible as Transaction Logic rules.
Outline

How it all began

What is Transaction Logic?

What’s wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
Other applications

- Transaction logic and its variants have been applied in many domains by various people:
  - Process/workflow modeling
  - Web service choreography, contracts, discovery
  - Robotics, sensors
  - Production rules
  - Security policy analysis
  - Database view updates
  - Active databases
Further developments

- **1993**: *the original paper + much more (proof theory, LP semantics: local stratification, etc.)*
- **1996**: Concurrent transaction logic (*Bonner & K*)
  - Late 90’s: Applications: active databases, workflow modeling, reasoning about actions (*Bonner, K, Consens*)
  - Early 2000’s - Applications: robotics (*Santos*), workflow modeling (*Davulcu, Karagoz,*...*), security policy modeling and reasoning (*Becker*)
- **2004**: CTR-S — Concurrent Transaction Logic for Services (*Davulcu,*...*)
  (an extension for modeling Web services, has game-theoretic flavor).
- Late 2000’s - Applications: Web service modeling (*Roman*)
Further developments

• 1993: the original paper + much more (proof theory, LP semantics: local stratification, etc.)
• 1996: Concurrent transaction logic (Bonner & K)
• Late 90’s: Applications: active databases, workflow modeling, reasoning about actions (Bonner, K, Consens)
• Early 2000’s - Applications: robotics (Santos), workflow modeling (Davulcu, Karagoz,...), security policy modeling and reasoning (Becker)
• 2004: CTR-S — Concurrent Transaction Logic for Services (Davulcu,...)
  (an extension for modeling Web services, has game-theoretic flavor).
• Late 2000’s - Applications: Web service modeling (Roman)
Further developments (cont’d)

- 2010: Tabling for Transaction Logic (*Fodor*)
- 2011: Defeasible Transaction Logic, well-founded semantics (*Fodor*)
- 2012: Transaction Logic with partially defined actions (*Rezk*) (an extension that provides the missing pieces to enable reasoning about actions à la Gelfond/Lifschitz).
- 2012: Modeling RIF production rules and more (*Rezk*)
- This conference yesterday: adding external actions (*Gomes*)

Tabling and partially defined actions are probably the most significant developments from the practical point of view since the time concurrency was added in 1996.
Further developments (cont’d)

- 2010: Tabling for Transaction Logic (*Fodor*)
- 2011: Defeasible Transaction Logic, well-founded semantics (*Fodor*)
- 2012: Transaction Logic with partially defined actions (*Rezk*)
  (an extension that provides the missing pieces to enable reasoning about actions a la Gelfond/Lifschitz).
- 2012: Modeling RIF production rules and more (*Rezk*)
- This conference yesterday: adding external actions (*Gomes*)

Tabling and partially defined actions are probably the most significant developments from the practical point of view since the time concurrency was added in 1996.
Implementations

- Part of the Flora-2 system. Supports $\otimes$ (sequence), $\diamond$ (hypothetical, possible), $\sim \diamond$ (not possible), further extensions

- Toronto (Bonner): Supports $\otimes$, $\diamond$, $|$ (concurrency), exception handling

- Stony Brook (Fodor): $\otimes$, $\diamond$ + tabling:

- Only the Flora-2 version is really usable for building serious applications, as here Transaction Logic is integrated into a complete LP system, which, in addition, supports F-logic, HiLog, defeasible reasoning, and much more

- But the other two are useful for advanced experiments

- There were other, now defunct, implementations (e.g., University of Valencia)
Implementations

- Part of the Flora-2 system. Supports $\otimes$ (sequence), $\diamond$ (hypothetical, possible), $\sim \diamond$ (not possible), further extensions
  
  http://flora.sourceforge.net

- Toronto (Bonner): Supports $\otimes$, $\diamond$, $|$ (concurrency), exception handling
  
  http://www.cs.toronto.edu/~bonner/transaction-logic.html

- Stony Brook (Fodor): $\otimes$, $\diamond$, + tabling:
  
  http://flora.sourceforge.net/tr-interpreter-suite.tar.gz

- Only the Flora-2 version is really usable for building serious applications, as here Transaction Logic is integrated into a complete LP system, which, in addition, supports F-logic, HiLog, defeasible reasoning, and much more

- But the other two are useful for advanced experiments

- There were other, now defunct, implementations (e.g., University of Valencia)
Implementations

- Part of the Flora-2 system. Supports $\otimes$ (sequence), $\diamond$ (hypothetical, possible), $\sim \diamond$ (not possible), further extensions
  http://flora.sourceforge.net

- Toronto (Bonner): Supports $\otimes$, $\diamond$, $|$ (concurrency), exception handling
  http://www.cs.toronto.edu/~bonner/transaction-logic.html

- Stony Brook (Fodor): $\otimes$, $\diamond$ + tabling:
  http://flora.sourceforge.net/tr-interpreter-suite.tar.gz

- Only the Flora-2 version is really usable for building serious applications, as here Transaction Logic is integrated into a complete LP system, which, in addition, supports F-logic, HiLog, defeasible reasoning, and much more

- But the other two are useful for advanced experiments

- There were other, now defunct, implementations (e.g., University of Valencia)
Implementations

• Part of the Flora-2 system. Supports $\otimes$ (sequence), $\Diamond$ (hypothetical, possible), $\sim \Diamond$ (not possible), further extensions
  http://flora.sourceforge.net

• Toronto (Bonner): Supports $\otimes$, $\Diamond$, $|$ (concurrency), exception handling
  http://www.cs.toronto.edu/~bonner/transaction-logic.html

• Stony Brook (Fodor): $\otimes$, $\Diamond +$ tabling:
  http://flora.sourceforge.net/tr-interpreter-suite.tar.gz

• Only the Flora-2 version is really usable for building serious applications, as here Transaction Logic is integrated into a complete LP system, which, in addition, supports F-logic, HiLog, defeasible reasoning, and much more

• But the other two are useful for advanced experiments

• There were other, now defunct, implementations (e.g., University of Valencia)
More information

• flora.sourceforge.net
• www.cs.toronto.edu/~bonner/transaction-logic.html
• www.cs.toronto.edu/~bonner/ctr/index.html
Outline

How it all began

What is Transaction Logic?

What’s wrong with Prolog?

Transaction Logic basics

Later developments

Wrap up
Other works


Conclusions

- Transaction Logic is young and healthy @20.
  - Many extensions, interesting developments, applications
- Many people saw the light and wrote interesting stuff about it
- ... but too many still have not
- Many challenges, research problems remain: taming concurrency, more efficient implementations, partial actions+ASP, ...
- @40 it should look even better
Conclusions

• Transaction Logic is young and healthy @20.
  • Many extensions, interesting developments, applications
• Many people saw the light and wrote interesting stuff about it
  • ... but too many still have not
• Many challenges, research problems remain: taming concurrency, more efficient implementations, partial actions + ASP, ...
• @40 it should look even better
Conclusions

- Transaction Logic is young and healthy @20.
  - Many extensions, interesting developments, applications
- Many people saw the light and wrote interesting stuff about it
- ... but too many still have not 😊
- Many challenges, research problems remain: taming concurrency, more efficient implementations, partial actions+ASP, ...
- @40 it should look even better
Conclusions

• Transaction Logic is young and healthy @20.
  • Many extensions, interesting developments, applications
• Many people saw the light and wrote interesting stuff about it
• ... but too many still have not 😊
• Many challenges, research problems remain: taming concurrency, more efficient implementations, partial actions+ASP, ...
• @40 it should look even better
Conclusions

• Transaction Logic is young and healthy @20.
  • Many extensions, interesting developments, applications
• Many people saw the light and wrote interesting stuff about it
• ... but too many still have not 😊
• Many challenges, research problems remain: taming concurrency, more efficient implementations, partial actions+ASP, ...
• @40 it should look even better
Thanx! Questions?
Oh, and greetings from Tony!