20 years of Transaction Logic

Michael Kifer

August 28, 2013

Outline

The paper

- Anthony Bonner & MK: *Transaction Logic Programming*, ICLP 1993.

The paper

- Anthony Bonner & MK: *Transaction Logic Programming*, ICLP 1993.
- Never actually presented: Tony got stuck in a hospital in Prague on the way to Budapest 

How the work started

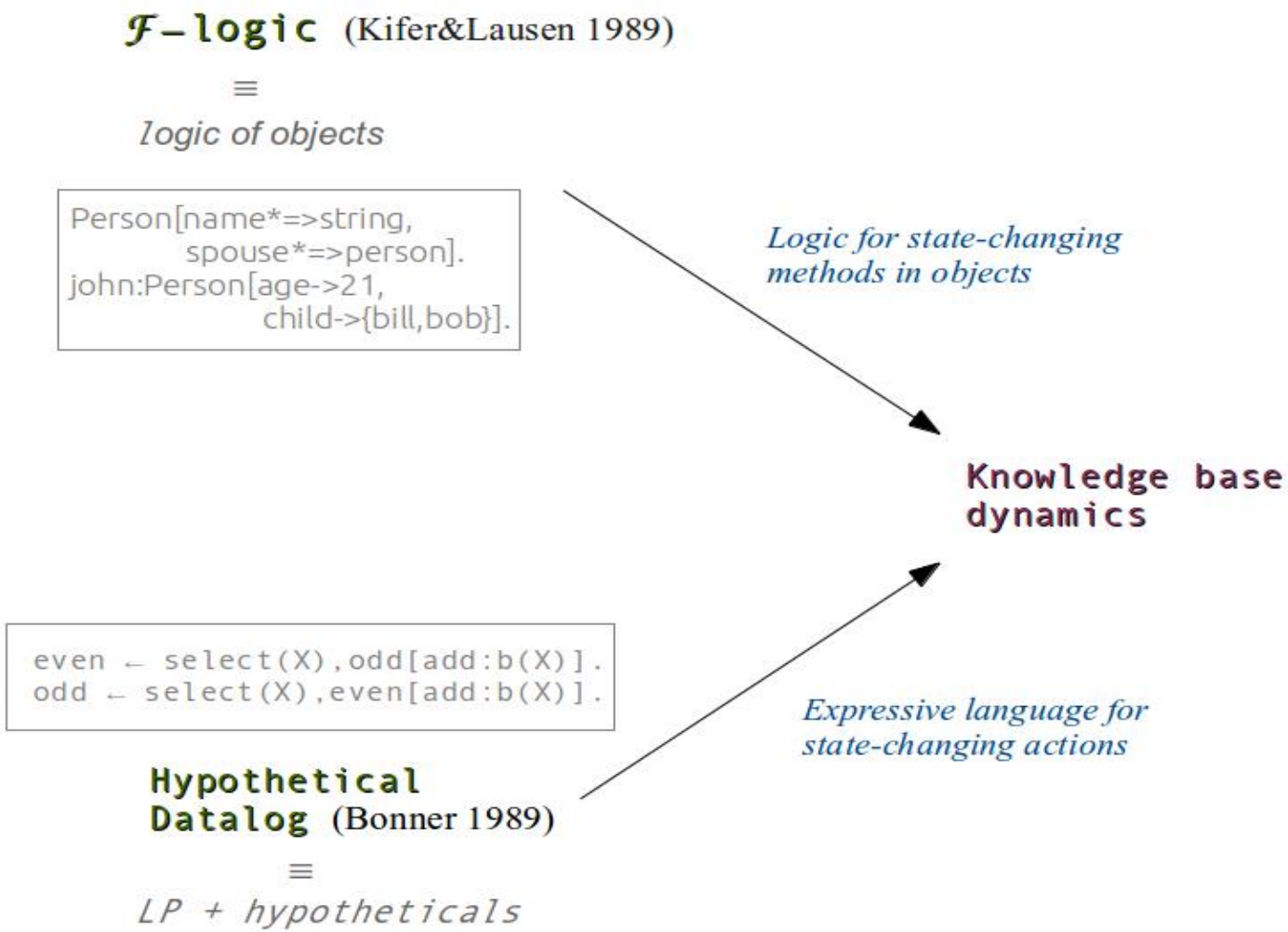
- 1991: I was invited by Alberto Mendelzon to spend a sabbatical year at U. of Toronto

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

Our idea of nightlife back then

Motivation


F-logic (Kifer&Lausen 1989)
≡
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 (Bonner 1989)
≡
LP + hypotheticals

Outline

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

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

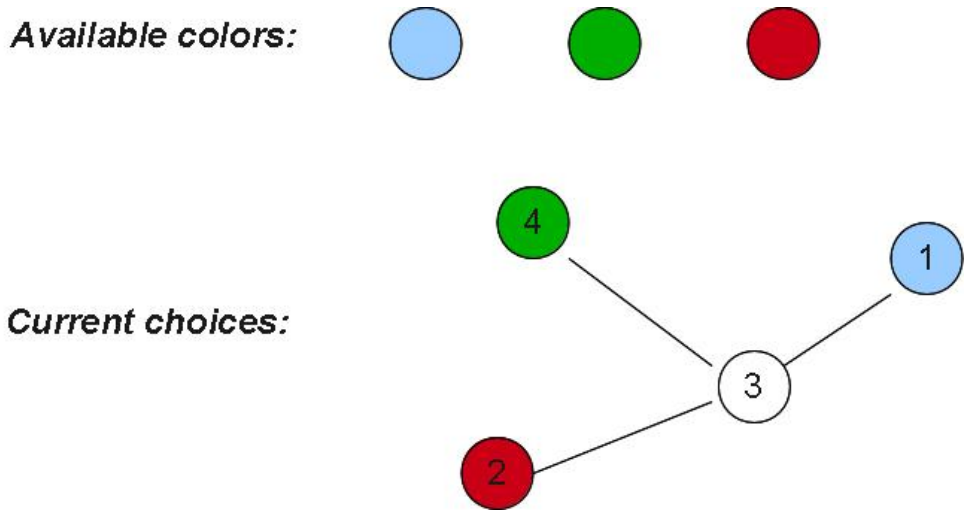
```
colorNode :-   %% color one node correctly
        node(N), not colored(N,_),
        color(C),
        not (adjacent(N,N2), colored(N2,C)),
        assert(colored(N,C)).
colorGraph :- not uncoloredNodesLeft.
colorGraph :- colorNode, colorGraph.
```

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

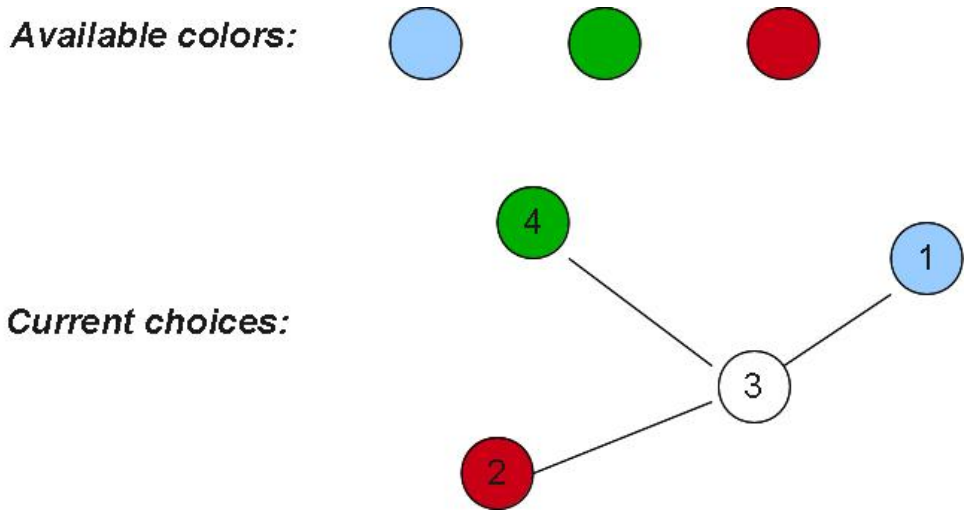


False: $\exists C \in color\,(not\,\exists N2(adjacent(3, N2), colored(N2, C)))$

$=>$ Backtrack ...

... but the wrong asserts $colored(4, green)$ or $colored(1, blue)$ or $colored(1, blue)$ will stay in the DB!

Graph Coloring: Illustration of the problem

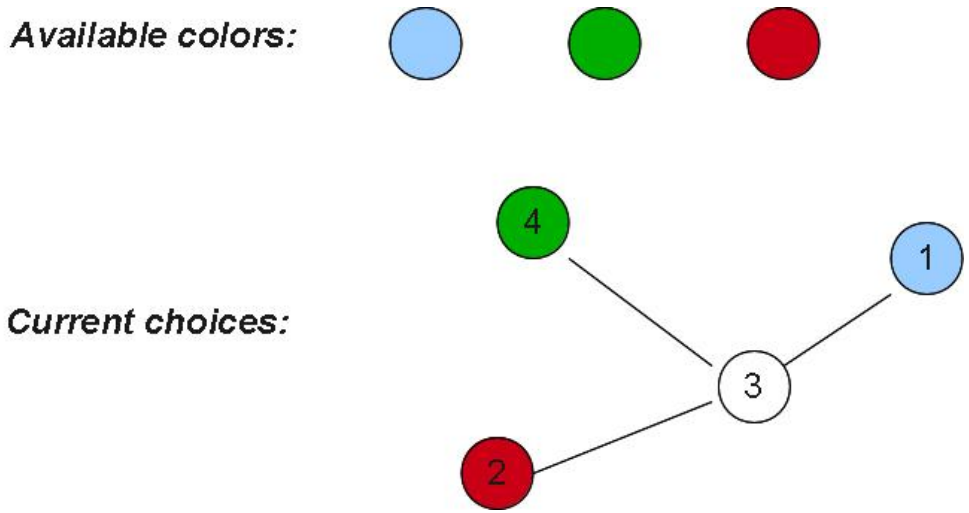


False: $\exists C \in color\,(not\,\exists N2(adjacent(3, N2), colored(N2, C)))$

$=>$ Backtrack ...

... but the wrong asserts $colored(4, green)$ or $colored(1, blue)$ or $colored(1, blue)$ will stay in the DB!

Graph Coloring: Illustration of the problem



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

$=>$ Backtrack ...

... but the wrong asserts $colored(4, green)$ or $colored(1, blue)$ or $colored(1, blue)$ 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

```
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.

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

Basic Transaction Logic

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

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

- For queries, $\wedge$ 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.
- *foo* $\otimes$ *bar* means: execute *foo* then execute *bar*.
- Example:

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

- For queries, $\wedge$ 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

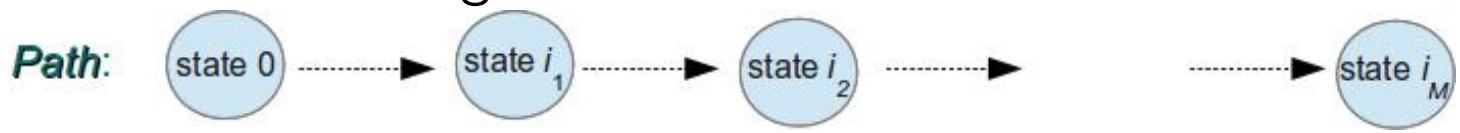
- Extends *normal* LP programs (with negation, etc.) with serial conjunction $\otimes$ in rule bodies.
- *foo* $\otimes$ *bar* means: execute *foo* then execute *bar*.
- Example:

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

- For queries, $\wedge$ 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:



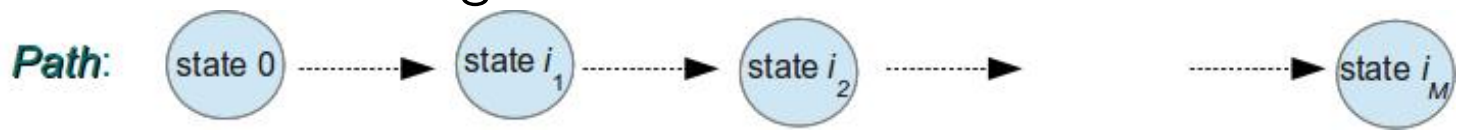
  State can be anything: relational, rule sets, KBs with nonmon semantics, etc.

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

- 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:



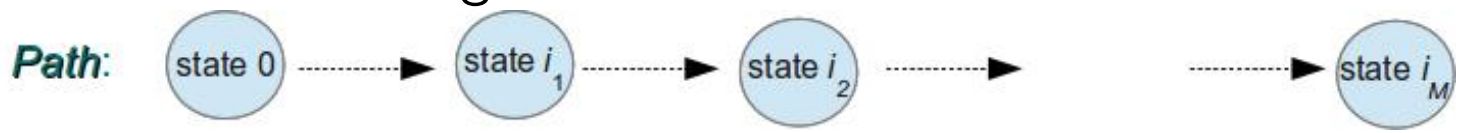
  State can be anything: relational, rule sets, KBs with nonmon semantics, etc.

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

- 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:



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

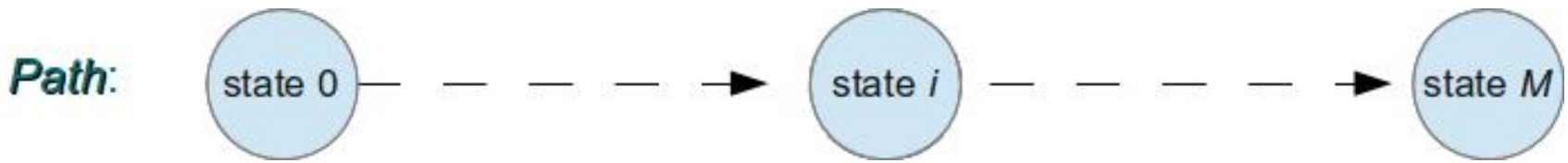
- Path of length 1 is a sequence having one state only:

- 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.

$\otimes$: Informal semantics

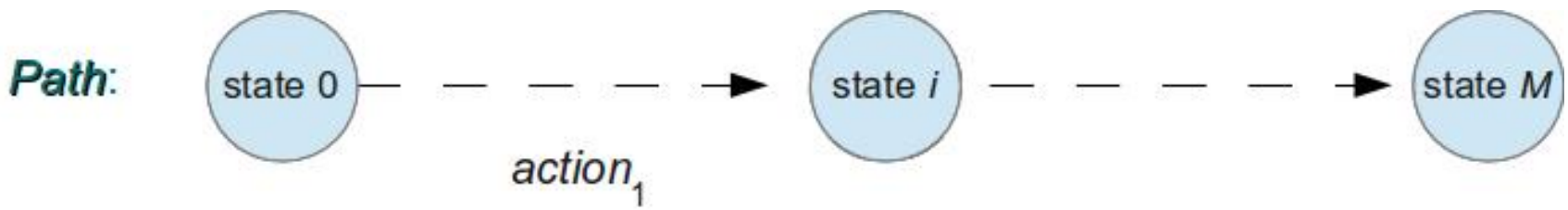
$$action_1 \otimes action_2$$

* $action_i$ is a state-changing action or a query



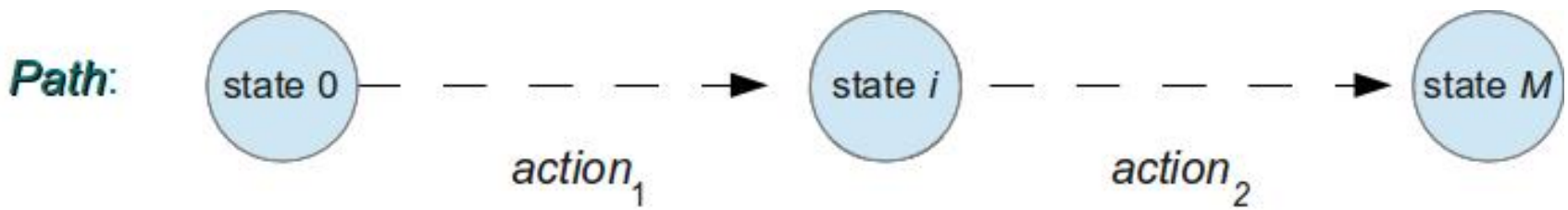
$\otimes$: Informal semantics

$$action_1 \otimes action_2$$



$\otimes$: Informal semantics

$$action_1 \otimes action_2$$



Informal semantics (rules)

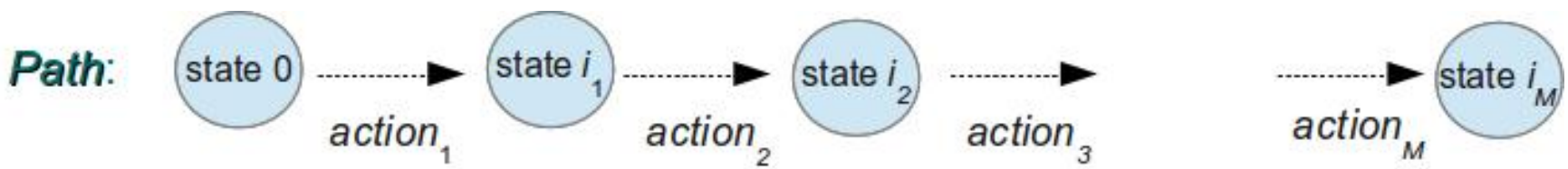
$$action \leftarrow action_1 \otimes \cdots \otimes action_M$$

* $action_i$ is a state-changing action or a query

Informal semantics (rules)

$$action \leftarrow action_1 \otimes \cdots \otimes action_M$$

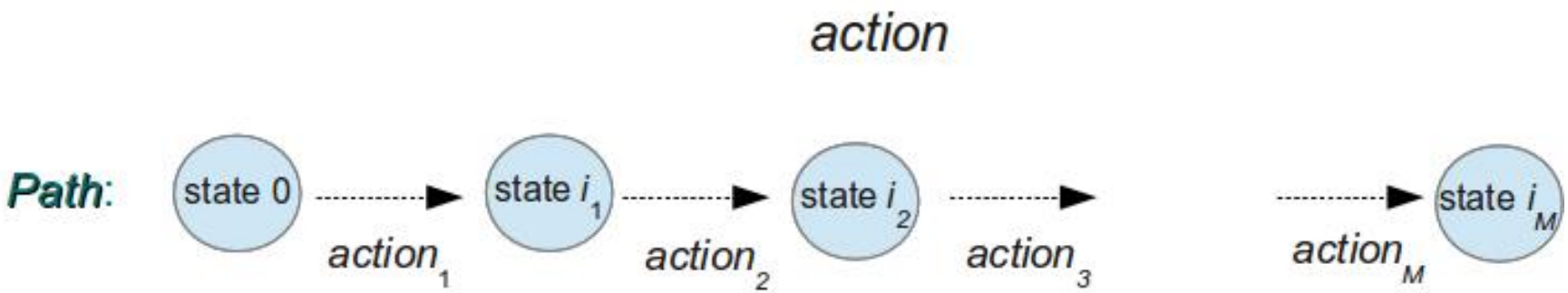
IF:



Informal semantics (rules)

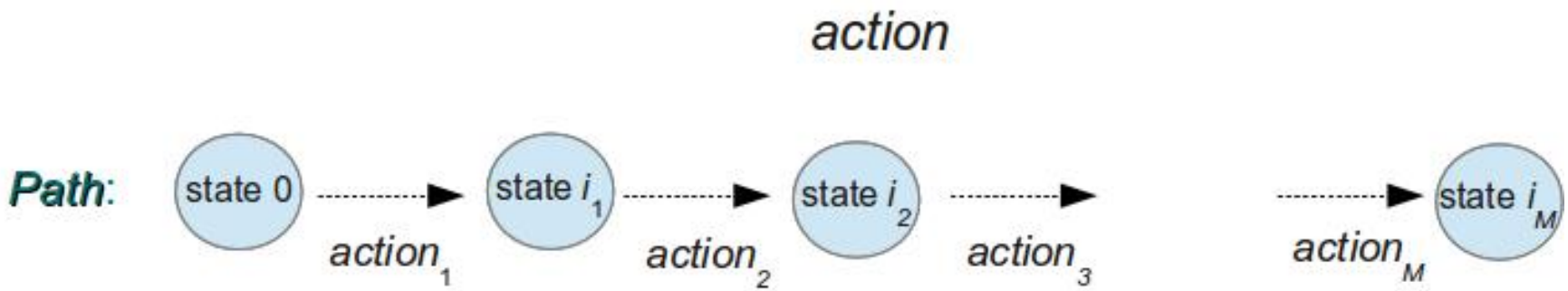
$$action \leftarrow action_1 \otimes \cdots \otimes action_M$$

THEN:



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

- $foo \mid bar$: do *foo* and *bar* concurrently.
- $\diamond foo$: check if *foo* is doable, but don't do it.
- $\odot foo$: do *foo* in *isolation* (don't allow it to interleave with other actions).
- $\wedge$, $\vee$, $\neg$: extend the corresponding classical connectives, but are more general. (They reduce to classical connectives over paths of length 1.)
- $\forall$, $\exists$: ditto.

Advanced connectives

- $foo \mid bar$: do *foo* and *bar* concurrently.
- $\diamond foo$: check if *foo* is doable, but don't do it.
- $\odot foo$: do *foo* in *isolation* (don't allow it to interleave with other actions).
- $\wedge$, $\vee$, $\neg$: extend the corresponding classical connectives, but are more general. (They reduce to classical connectives over paths of length 1.)
- $\forall$, $\exists$: ditto.

Planning example

- Natural for planning problems:

  step <- $action_1$.
  ...
  step <- $action_n$.
  stepseq ← .
  stepseq ← step ⊗ stepseq.
  plan ← stepseq ⊗ *planning_goal*.

- To find a plan: issue the transaction ? – *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:

  step <- $action_1$.
  ...
  step <- $action_n$.
  stepseq ← .
  stepseq ← step ⊗ stepseq.
  plan ← stepseq ⊗ *planning_goal* .

- To find a plan: issue the transaction ? − *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:

  $\texttt{step} \leftarrow \mathit{action}_1 .$

  ...

  $\texttt{step} \leftarrow \mathit{action}_n .$

  $\texttt{stepseq} \leftarrow .$

  $\texttt{stepseq} \leftarrow \texttt{step} \otimes \texttt{stepseq}.$

  $\texttt{plan} \leftarrow \texttt{stepseq} \otimes \mathit{planning\_goal} .$
- To find a plan: issue the transaction $? - \ \mathit{plan}.$
- 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:

  step <- $action_1$.
  ...
  step <- $action_n$.
  stepseq $\leftarrow$.
  stepseq $\leftarrow$ step $\otimes$ stepseq.
  plan $\leftarrow$ stepseq $\otimes$ *planning_goal* .

- To find a plan: issue the transaction $? - \ plan.$
- Of course, the above is naive and impractical.
- But efficient planning strategies (STRIPS, hierarchical, etc.) are naturally & concisely expressible as Transaction Logic rules.

Outline

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 (*Davlcu*, *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 (*Davlcu*, *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 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.

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

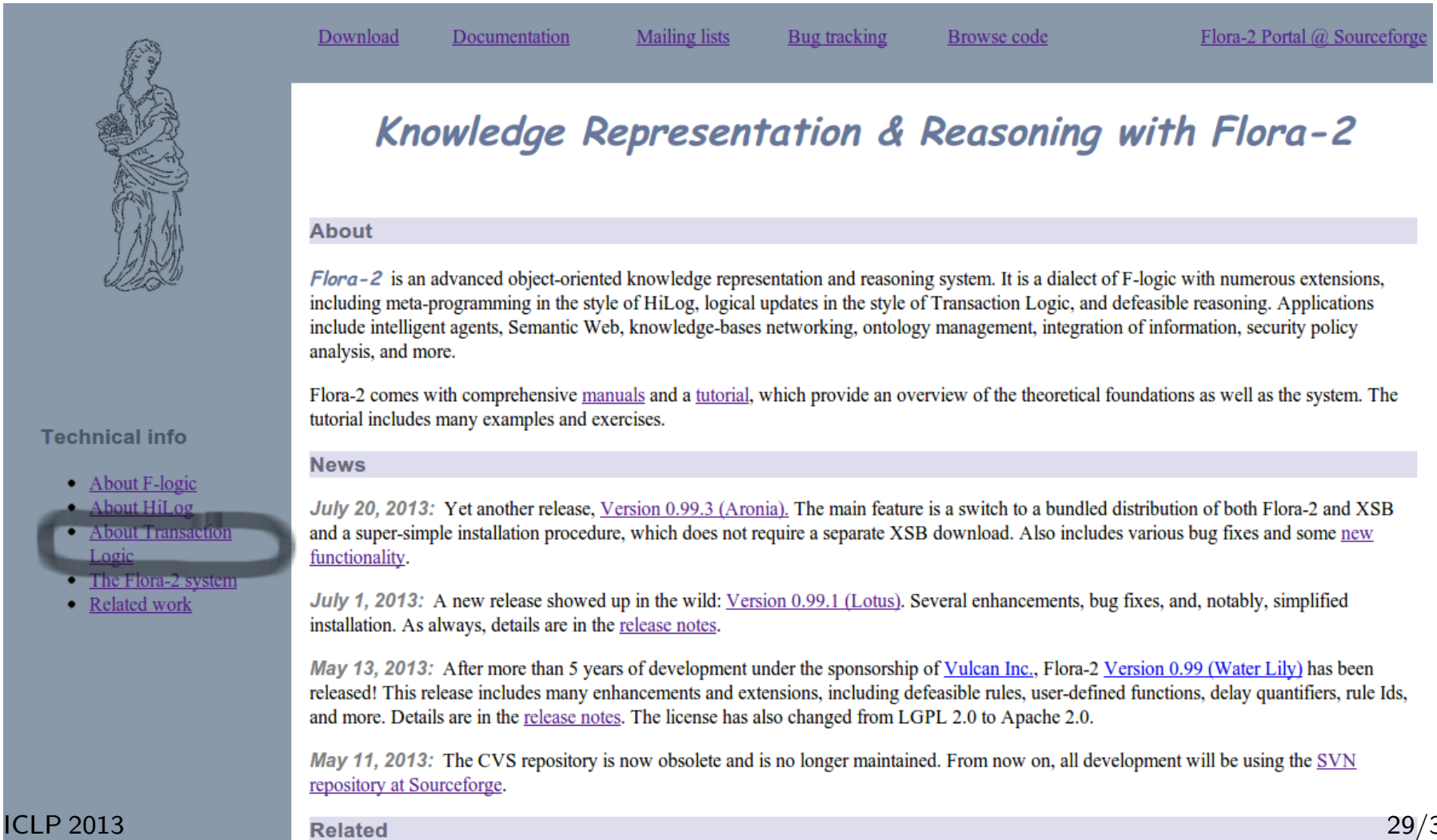
- Part of the Flora-2 system. Supports $\otimes$ (sequence), $\Diamond$ (hypothetical, possible), $\sim\Diamond$ (not possible), further extensions
  http://flora.sourceforge.net
- Toronto (Bonner): Supports $\otimes$, $\Diamond$, $|$ (concurrency), exception handling
  http://www.cs.toronto.edu/~bonner/transaction-logic.html
  http://www.cs.toronto.edu/~bonner/ctr/index.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
  http://www.cs.toronto.edu/~bonner/ctr/index.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

Other works

- SitCalc, Golog, $\mathcal{A}$ (GL), Event calculus, all kinds of deductive DB proposals: A. Bonner and MK (1995), *Transaction Logic Programming or A logic for Declarative and Procedural Knowledge*. Nov. 1995 U of Toronto CSRI Tech report. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.9935`
- (P)DDL, ASM, etc.: van Eck, P.A.T. and Engelfriet, J. and Fensel, D. and van Harmelen, F.A. and Venemaand, Y. and Willems, M. (2001), *A Survey of Languages for Specifying Dynamics: A Knowledge Engineering Perspective*. IEEE Transactions on Knowledge and Data Engineering, 13 (3). pp. 462-496. `http://eprints.eemcs.utwente.nl/942/`

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?