DOOD:
FROM WISHFUL THINKING
TO (VIRTUAL) REALITY

Michael Kifer
University at Stony Brook, U.S.A.
DOOD Progress Timeline

1995
- The 4th DOOD, Singapore
  - DOOD Conference coming of age
  - Theoretical base well-developed
  - Further prototypes with various degrees of DOODness
  - BULL still talking ...

1993
- The 3d DOOD in Phoenix
  - LDL++, Quixote, F-logic interpreter
  - BULL talking about a commercial product

1991
- The 2nd DOOD in Munich
  - progress in formalizing both structure and behavior of OO

1989
- The 1st DOOD Conference in Kyoto

1988
- The idea of DOOD conference conceived
  - disparate theories
  - serious doubts about feasibility

Around 1975
- Minsky’s "Frames"
- Declarative vs. Procedural knowledge debates

... DOOD Mutants break out from Gov AI Labs...
DOOD Religions

- **prolog++** (cross between Prolog and C++)
  
e.g., LDL++, Coral++

- **message-passing prologs**
  
e.g., Quintus Objects, Prolog++ (Logic Programming Associates, UK)

- **logic-based religions**
  
  *The main contenders:*
  
  - Linear logic based approaches
  - Dynamic logic based approaches
  - Rewrite logic based works
  - Altered classical logic
  - New logics (usually upward compatible with classical logic)

  ◇ **F-logic/Transaction logic** logic suite is the most developed and comprehensive in this category.
The Grand Scheme of DOOD

DOOD

Structure
(complex objects, queries, types
ISA hierarchy, inheritance)

Behavior
(methods with side-effects,
state changes)

F-Logic
(Kifer, Lausen, Wu)

Transaction Logic
(Bonner, Kifer)
F-logic: Complex Structure the Easy Way

Object description:

john[name→"John Doe"; spouse→mary; children→{bob, alice}]
mary[name→"Mary Doe"; spouse→john; age→30; children→{bob, alice}]

ISA hierarchy:

john : student  (john is a student)
mary : person  
student :: person  (student is a subclass of person)

- Can combine the above using \&, \lor, \neg, etc.
  In particular, attributes and even class hierarchies can be defined via logical rules:

\[ \begin{align*}
nil & : list(T) \\
< X \mid L > & : list(T) \leftarrow X : T \land L : list(T) \\
list(T) & :: list(S) \leftarrow T :: S
\end{align*} \]
**F-logic (contd.)**

**Methods** (i.e., functions that take objects as arguments)

\[
P[\text{ageAsOf} @Y \rightarrow Y - B] \leftarrow P : \text{person} \land P[\text{yearOfBirth} \rightarrow B] \\
L[\text{append} @\text{nil} \rightarrow L] \leftarrow L : \text{list}(T) \\
< X|L > [\text{append} @M \rightarrow < X|N >] \leftarrow L[\text{append} @M \rightarrow N] \land X : T \land L : \text{list}(T)
\]

**Queries**

?— \textit{john}[\text{ageAsOf} @1989 \rightarrow Y \land \text{children} \rightarrow C] \land C[\text{yearOfBirth} \rightarrow B] \land B > 1980

?— \textit{< 1, 2, 3 >} [\text{append} @ < 5, 6, 7 > \rightarrow A]

- F-logic methods can only query no updates
  (which is why Transaction Logic is here!)
F-logic (contd.)

**Typing** (actually, type signatures):

\[
\text{person}[\begin{array}{l}
\text{ageAsOf} \Rightarrow \text{int}; \\
\text{name} \Rightarrow \text{string}; \\
\text{yearOfBirth} \Rightarrow \text{int}; \\
\text{spouse} \Rightarrow \text{person}; \\
\text{children} \Rightarrow \text{person}
\end{array}]
\]

\[
\text{list}(T)[\begin{array}{l}
\text{append}@\text{list}(T) \Rightarrow \text{list}(T)
\end{array}]
\]

- Can define signatures via deductive rules, query the type structures, etc.
- Type correctness has formal meaning
F-logic’s Bullet List

- Logical semantics (General and Herbrand)
- Proof theory
- Upward compatible with classical logic
- Complex objects
- Semantics for type-correctness
- Semantics for structural and behavioral inheritance
- Semantics for encapsulation

♠ Details: JACM, July 1995
Transaction Logic: A Conquest of Change

Why new logic?

- State changes are notoriously hard to capture in logic.
- Reasoning about change vs. specifying change and executing it.
- Harmful (for O-O) distinction between actions and queries.
- Inefficient elementary updates.
- No subroutines!

♣ Transaction logic overcomes all of these and more in a simple, uniform way.
Transaction Logic’s Basic Ideas

• Syntax:
  – serial conjunction, $\otimes$:
    $$a \otimes b$$ - do $a$ then do $b$ (plus the “usual” $\land$, $\lor$, $\neg$, $\forall$, $\ldots$)

• Semantics:
  – Transaction execution paths (sequences of database states)
  – Elementary state transitions.
  – Truth on a path $\equiv$ execution along the path.

• Proof theory:
  – proves and executes actions.
Example

- stack a pyramid of $N$ blocks on top of block $X$:

  \[\begin{align*}
  & stack(0, X) \\
  & stack(N, X) \leftarrow N > 0 \otimes move(Y, X) \otimes stack(N - 1, Y)
  \end{align*}\]

  \[\begin{align*}
  & move(X, Y) \leftarrow pickup(X) \otimes putdown(X, Y) \\
  & pickup(X) \leftarrow clear(X) \otimes on(X, Y) \otimes on.del(X, Y) \otimes clear.ins(Y) \\
  & putdown(X, Y) \leftarrow wider(Y, X) \otimes clear(Y) \otimes on.ins(X, Y) \otimes clear.del(Y)
  \end{align*}\]

- Query: Stack a pyramid of 20 blocks.

  \[\begin{align*}
  ? & \leftarrow stack(20, blkC)
  \end{align*}\]
Transaction Logic’s House Specialties

- Active rules (what did you think?)
- Hypothetical reasoning (two new modal operators)
- Subjunctive and counterfactual queries
- Bulk updates
- Updates to arbitrary (not only relational) states
- Allen-style temporal constraints
- Planning
- Avoids the frame problem (for execution and planning)
Transaction F-logic: Easier than you may have thought

☐ Basic ideas:
  - serial conjunction applies to object formulas
  - elementary state transitions can be caused by executing methods

♠ Methods can query, change state, or both (i.e., they can be queries with side effect).
Easier than you may have thought (contd.)

Transaction Logic as a Framework
Example

- Stacking pyramids object-oriented way:

\[ R[stack@N, BaseBlk\rightarrow nil] \leftarrow N > 0 \otimes R[move@Frm, BaseBlk\rightarrow TmpBlk \otimes stack@N - 1, TmpBlk\rightarrow nil] \]

\[ R[move@Frm, To\rightarrow Blk] \leftarrow R[pickup@Blk\rightarrow Frm \otimes putdown@Blk\rightarrow To] \]

\[ R[pickup@Blk\rightarrow Frm] \leftarrow R : \text{robot} \otimes \text{Blk : block} \otimes \text{Frm : block} \]
\[ \otimes \text{Blk[top\rightarrow clear; bottom\rightarrow Frm} \otimes R[\text{state\rightarrow idle}] \]
\[ \otimes \text{Blk[bottom.del\rightarrow Frm} \otimes \text{Frm[top.ins\rightarrow clear}] \]
\[ \otimes R[\text{state.replace\rightarrow holding}] \]

\[ R[putdown@Blk\rightarrow To] \leftarrow R : \text{robot} \otimes \text{Blk : block} \otimes \text{To : block} \otimes \text{To[top\rightarrow clear]} \]
\[ \otimes \text{wider(To, Blk)} \otimes R[\text{state\rightarrow holding}] \]
\[ \otimes \text{Blk[bottom.ins\rightarrow To} \otimes \text{To[top.del\rightarrow clear}] \]
\[ \otimes R[\text{state.replace\rightarrow idle}] \]

♣ This program is longer than the one before because it does a bit more.
**Conclusion**

DOOD is within reach:

- Theory is there
- Early prototypes are out:
  - Mike Lawley’s F-logic interpreter.
  - Georg Lausen’s FL System (F-logic).
  - Dave Warren’s XSB (HiLog and C-logic, a subset of F-logic).
  - Tony Bonner’s Transaction Logic interpreter.

▷ All these are reachable via http://www.cs.sunysb.edu/~kifer/dood/

... and is exciting:

- Vast field for research (all those problems that I am afraid to touch :-)