Programming
with
Logic and Objects

Michael Kifer
Stony Brook University
Outline

• Introduction: About FLORA-2
• F-logic
• HiLog
• Example of an application of FLORA-2
• Transaction Logic (if time permits)
Introduction
What’s Wrong with Classical Programming with Logic?

• Precisely that it is based on *classical* logic:
  – Essentially flat data structures (relations with *structs*)
  – Awkward meta-programming
  – Ill-suited for modeling side effects (state changes, I/O)
What is FLORA-2?

• **F-Logic tRAnslator** (the next generation)
  – FLORA-2 programs are translated into XSB & executed by the XSB tabling inference engine

• Language for knowledge-based applications
  – *Declarative* – much more so than Prolog
  – *Object-oriented* (frame based)

• Overcomes most of the usability problems with Prolog

• *Practical & usable* programming environment based on
  – *F-logic* (Frame Logic) ≡ objects + logic (+ extensions)
  – *HiLog* – high degree of *truly declarative* meta-programming
  – *Transaction Logic* – database updates + logic

• Builds on earlier experience with implementations of F-logic: FLORID, FLIP, FLORA-1 (which don’t support HiLog & Transaction Logic)

Applications of FLORA-2

• Ontology management (Semantic Web)
• Information integration
• Software engineering
• Agents
• Anything that requires manipulation of complex structured (especially semi-structured) data
Other F-logic Based Systems

- **No-name system** (U. Melbourne – M. Lawley) – early 90’s; first Prolog-based implementation
- **FLORID** (U. Freiburg – Lausen et al.) – mid-late 90’s; the only C++ based implementation
- **FLIP** (U. Freiburg – Ludaescher) – mid 90’s; first XSB based implementation. Inspired the FLORA effort
- **TFL** (U. Valencia – Carsi) – mid 90’s; first attempt at F-logic + Transaction Logic
- **SILRI** (Karlsruhe – Decker et al.) – late 90’s; Java based
- **TRIPLE** (Stanford – Decker et al.) – early 2000’s; Java

✓ **FLORA-2** – most comprehensive and general purpose of all these
F-Logic
Usability Problems with Flat Data Representation

Typical result of translation from the E-R diagram:

<table>
<thead>
<tr>
<th>Person</th>
<th>SSN</th>
<th>Name</th>
<th>PhoneN</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111-22-3333</td>
<td>Joe Public</td>
<td>516-123-4567</td>
<td>222-33-4444</td>
</tr>
<tr>
<td></td>
<td>111-22-3333</td>
<td>Joe Public</td>
<td>516-345-6789</td>
<td>222-33-4444</td>
</tr>
<tr>
<td></td>
<td>111-22-3333</td>
<td>Joe Public</td>
<td>516-123-4567</td>
<td>333-44-5555</td>
</tr>
<tr>
<td></td>
<td>111-22-3333</td>
<td>Joe Public</td>
<td>516-345-6789</td>
<td>333-44-5555</td>
</tr>
<tr>
<td></td>
<td>222-33-4444</td>
<td>Bob Public</td>
<td>212-987-6543</td>
<td>444-55-6666</td>
</tr>
<tr>
<td></td>
<td>222-33-4444</td>
<td>Bob Public</td>
<td>212-987-1111</td>
<td>555-66-7777</td>
</tr>
<tr>
<td></td>
<td>222-33-4444</td>
<td>Bob Public</td>
<td>212-987-6543</td>
<td>555-66-7777</td>
</tr>
<tr>
<td></td>
<td>222-33-4444</td>
<td>Bob Public</td>
<td>212-987-1111</td>
<td>444-55-6666</td>
</tr>
</tbody>
</table>

*Problem*: redundancy due to dependencies

\[
\text{Person} = (SSN,Name,PhoneN) \bowtie (SSN,Name,Child)
\]

\[
\text{SSN} \quad \text{—} \quad \text{Name}
\]
Normalization That Removes Redundancy

<table>
<thead>
<tr>
<th>SSN</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>Joe Public</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>Bob Public</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>PhoneN</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>516-345-6789</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>516-123-4567</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>212-987-6543</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>212-135-7924</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>Child</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td>222-33-4444</td>
</tr>
<tr>
<td>111-22-3333</td>
<td>333-44-5555</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>444-55-6666</td>
</tr>
<tr>
<td>222-33-4444</td>
<td>555-66-7777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>ChildOf</th>
</tr>
</thead>
<tbody>
<tr>
<td>111-22-3333</td>
<td></td>
</tr>
<tr>
<td>111-22-3333</td>
<td></td>
</tr>
<tr>
<td>222-33-4444</td>
<td></td>
</tr>
<tr>
<td>222-33-4444</td>
<td></td>
</tr>
</tbody>
</table>
But querying is still cumbersome:

*Get the phone#'s of Joe’s grandchildren.*

Against the **original** relation –– complex:

```
SELECT  G.PhoneN
FROM      Person  P,  Person  C,  Person  G
WHERE   P.Name = 'Joe Public'
        AND  P.Child = C.SSN
        AND  C.Child = G.SSN
```

Against the **decomposed** relations –– even more so:

```
SELECT   N.PhoneN
FROM      ChildOf  C,  ChildOf  G,  Person1  P,  Phone  N
WHERE   P.Name = 'Joe Public'
        AND  P.SSN = C.SSN
        AND  C.Child = G.SSN
        AND  G.SSN = N.SSN
```
O-O approach: rich types and better query language

**Schema:**
Person(SSN: String,
    Name: String,
    PhoneN: {String},
    Child: {Person} )

-No need to decompose in order to eliminate redundancy

**Query:**
SELECT  P.Child.Child.PhoneN
FROM      Person  P
WHERE   P.Name = ‘Joe Public’

-Much simpler query formulation

Set data types
Path expressions
Basic Ideas Behind F-Logic

• Take complex data types as in object-oriented databases
• Combine them with logic
• Keep it clean – no ad hoc stuff
• Use the result as a programming/query language
F-Logic Features

- Objects with complex internal structure
- Class hierarchies and inheritance
- Typing
- Encapsulation

Background:
- Basic theory: [Kifer & Lausen SIGMOD-89], [Kifer, Lausen, Wu JACM-95]
- Powerful path expression syntax: [Frohn, Lausen, Uphoff VLDB-84]
- Semantics for non-monotonic inheritance: [Yang & Kifer, ODBASE 2002]
- Meta-programming + other extensions: [Yang & Kifer, ODBASE 2002]
F-logic: simple examples

Object description:

john\[name\rightarrow ‘John Doe’, phones\rightarrow\{6313214567, 6313214566\}, children\rightarrow\{bob, mary\}\]

mary\[name\rightarrow ‘Mary Doe’, phones\rightarrow\{2121234567, 2121237645\}, children\rightarrow\{anne, alice\}\]

Structure can be nested:

sally\[spouse \rightarrow john\[address \rightarrow ‘123 Main St.’\]\]
Examples (contd.)

ISA hierarchy:

john : person - class membership
mary : person
alice : student

student :: person - subclass relationship
Examples (Contd.)

Methods: like attributes, but take arguments

\[ P[ageAsOf(Year) \rightarrow Age] :-- P : person, P[born \rightarrow B], \text{Age is Year-B}. \]

Queries:

?- john[born \rightarrow Y, \text{children-\rightarrow}C],
C[born \rightarrow B], \text{Z is Y+30, B>Z}. \\

John’s children who were born when he was over 30.
Examples (Contd.)

Type signatures:

\[
\text{person}[\text{born} \rightarrow \text{integer}, \\
\text{ageAsOf}(\text{integer}) \rightarrow \text{integer}, \\
\text{name} \rightarrow \text{string}, \\
\text{children} \rightarrow \rightarrow \text{person}].
\]

Can define signatures as facts or via deductive rules; Signatures can be queried. Type correctness has logical meaning (as “runtime” constraints).
Syntax

• ISA hierarchy:
  • O:C -- object O is a *member* of class C
  • C::S -- C is a *subclass* of S

• Structure:
  • O[M→S] -- *scalar* (single-valued) invocation of method
  • O[M→>>S] -- *set*-valued invocation of method

• Type (signatures):
  • Typeobj[Meth => Resulttype] -- a scalar method signature
  • Typeobj[Meth =>>> Resulttype] -- signature for a set-valued method

• Combinations of the above: ∨, ∧, negation, quantifiers

• O,C,M,Typeobj, ... -- usual first order function terms, e.g.,
  *john*, *AsOf(Y)*, *foo(bar,X)*.
More Examples

Browsing ISA hierarchy:

?- john : X.
?- student :: Y

Virtual (view) class:

X : redcar :- X:car, X[color -> red].

Schema browsing:

O[attrs(Class) ->> A] :-
   (O[A -> V; A ->> V]), V:Class.

Parameterized classes:

[]:list(T).
[X|L]:list(T) :- X:T, L:list(T).

E.g., list(integer), list(student)
Semantics

**Herbrand universe:** $\text{HB} – \text{set of all ground terms}

**Interpretation:** $I = (\text{HB}, I_-, I_\rightarrow, \in, <)$

where $< : \text{partial order on HB}$

$\in : \text{binary relationship on HB}$

$I_- : \text{HB} \rightarrow (\text{HB} \rightarrow \text{HB})$

$I_\rightarrow : \text{HB} \rightarrow (\text{HB} \rightarrow \text{powersetOf}(\text{HB}))$

$I_\rightarrow : \text{HB} \rightarrow (\text{HB} \rightarrow \text{powersetOf}(\text{HB}))$

$I | o \rightarrow v \iff I_-(m)(o) = v$

$I | o \rightarrow\rightarrow v \iff v \in I_\rightarrow(m)(o)$

$I | o : c \iff o \in c$

$I | c : : s \iff c < s$

• Won’t discuss typing
Proof Theory

- Resolution-based
- Sound & complete w.r.t. the semantics
Inheritance in F-logic

Should conclude:

- fred [color $\rightarrow$ grey]
- clyde [color $\rightarrow$ white]
The Problem with Rules

- Inheritance is hard to even define properly in the presence of rules.

Several other non-obvious cases exist
Inheritance (Contd.)

• Hard to define semantics to multiple inheritance + overriding + deduction; several semantics might be “reasonable”

• The original semantics in [Kifer, Lausen, Wu JACM-95] was quite problematic

• Problem solved in [Yang & Kifer ODBASE 2002]
HiLog
HiLog

• Allows certain forms of logically clean meta-programming
• Syntactically appears to be higher-order, but semantically is first-order and tractable
• Has sound and complete proof theory
• [Chen, Kifer, Warren JLP-93]
Examples of HiLog

Variables over predicates and function symbols:
\[ p(X,Y) : - X(a,Z), Y(Z(b)). \]

Variables over atomic formulas:
\[ \text{call}(X) : - X. \]

HiLog in FLORA-2 (e.g., method browsing):
\[ O[\text{unaryMethods}(\text{Class}) \rightarrow M] :--
   O[M(_) \rightarrow V; M(_) \rightarrow V], V:\text{Class}. \]

\[ \text{john}[\text{believes} \rightarrow \${\text{mary}[\text{likes} \rightarrow \text{bob}} \} \} \]
Applications
Applications

• Web information extraction agents (XSB, Inc.’s prototype; FLORA-1)
• Info integration in Neurosciences (San Diego Supercomputing Institute; FLORA-1)
• Ontology management (Daimler-Chrysler; FLORA-2)
• CASE tool (U. Valencia; FLORA-2)
• Stony Brook CS Grad Program Manager (FLORA-2)
SBCS Graduate Program Manager

• Need to keep track of lots of special cases
  • MS, PhD status over time; with/without support
  • Types of support over time (RA/TA/fellowships, permanent/temporary)
  • PhD examinations (with history of failures, conditions); N/A to MS
  • Teaching history
  • Advisors over time

• TA assignments
  • 35+ courses
  • 70 TAs; ~50 guaranteed, ~50 wannabees (waitlist)
    – Preferences/skills
    – English proficiency test results, etc., etc.

• Need complex aggregate reports

• Very complicated
  – Hard to figure out the right database schema (still evolving)
  – Data highly semistructured
SBCS Grad Manager (Contd.)

- Was hard-pressed: didn’t have the time to do it in Java/JDBC (also: maintenance would have been a serious problem afterwards)

- FLORA-2 was ideal for this:
  - Objects don’t need to have exactly the same structure
  - Changes of object schema (usually) don’t require changes to old rules/queries – low maintenance overhead

- Took only 2 weeks for initial version including data entry and debugging FLORA-2 itself!
  - Had some fun doing the otherwise boring job
Student Data – Highly Semi-structured

#1: student

- last -> ‘Doe’, first -> ‘Mary’, email -> ‘marydoe@yahoo.com’,
- joined -> fall(1999), graduated -> futuredate,
- advisor -> #(#1)[who -> johndoe, since -> fall(1999)],
- support -> { #(#1)[type -> ra, since -> fall(2001)],
  #(#1)[type -> ta, until -> spring(2001)] },
- status -> { #(#1)[type -> phd, since -> spring(2002), remarks -> 'part time'],
  #(#1)[type -> phd, until -> summer2(2000)],
  #(#1)[type -> ms, since -> fall(2000), until -> fall(2001)] },
- quals -> #(#1)[passed -> date(2000, 10), history -> data(2000, 5)],
- defense -> #(#1)[passed -> futuredate],
- female,
- domestic,
- taught -> { #(#1)[course -> cse529, semester -> fall(2000), load -> 0.5],
  #(#1)[course -> cse310, semester -> fall(2000), load -> 0.5],
  #(#1)[course -> cse305, semester -> spring(2001)] },
- canteach -> { cse332, cse336, cse333, cse230, cse528 }

Anonymous oid

Variations in structure

Can be missing

Variations in structure

Hackery to improve indexing

Can be missing
Course Data – Also Semistructured

cse505 : course[
    name   -> 'Computing with Logic',
    offerings ->>> {  
        #[semester -> fall(2001),
           instructors ->> {cram},
           enrollment    -> 15,
        ],
        #[semester -> fall(2002),
           instructors ->> {warren},
           enrollment    -> 25,
           need_grad_ta    -> 0.5
        ]
    }
].

Variation in structure
cse334 : course[
    name  -> 'Introduction to Multimedia Systems',
    crosslisted -> ise334,
    offerings -> {
        #[semester  -> fall(2001),
           instructors  -> {tony, rong},
           enrollment  -> 182,
           waiting  -> 0,
           need_grad_ta  -> 2,
           need_ug_ta  -> 3,
           ug_ta  -> {
               'John, Public (jp@aol.com)',
               'Blow, Joe (jblow@ic.sunysb.edu)' } ]
    }
]
Instructor Data

ted:lecturer[name->'Ted Teng'].
robkelly:lecturer[name->'Rob Kelly'].

ari:faculty[name -> 'Ari Kaufman', section587 -> 19].
skiena:faculty[name -> 'Steve Skiena'].
kifer:faculty[name -> 'Michael Kifer', section587 -> 9].

Variation in structure
Main Meta-Query

%% Sorted report main entry. Arguments:
%%      PrintMethod (what info about students to print)
%%      SortSpec (how to sort output)
%%      QuerySpec (which students to retrieve)
Class[#sprintquery(PrintMethod,SortSpec,QuerySpec)] :-
    L = collectset{Var | (O:Class)@students,
        %% Bind Query/SortSpec to the same oid
        SortSpec = sortSpec(Path,O,Val),
        QuerySpec = querySpec(O,QueryCond),
        Path,
        QueryCond,
        Var = Val-O
    },
    keysort(L,SortedL)@prolog(),
    Class[#printlist(PrintMethod,SortedL)],
    length(SortedL,Count)@prolog(basics),
    format('Total ~w count: ~w~w', [Class, Count])@prolog().
Pragmatics

• Very flexible module system
  – Can load programs into modules on-the-fly
  – Can create modules at run time and put a program into it
  – Prolog environment with its own module system is viewed as a set of “prolog modules”
  – FLORA-2 can call Prolog modules and Prolog can call FLORA-2 modules

• Anonymous OIDs (also useful in RDF and the like)
• Input/Output – use Prolog’s
• Prolog cuts – non-logical, but useful
Transaction Logic
Transaction Logic

• A logic of change
• Unlike temporal/dynamic/process logics, it is also a logic for programming (but can be used for reasoning as well)
• In the object-oriented context:
  – A logic-based language for programming object behavior (methods that change object state)
• [Bonner & Kifer, TCS 1995 and later]
What’s Wrong with Logics of Change?

• Designed for reasoning, not programming
  • E.g., situation calculus, temporal, dynamic, process logics
• Typically lack such basic facility as subroutines
• None became the basis for a reasonably useful programming language
What’s Wrong with Prolog?

- `assert/retract` have no logical semantics
- Non-backtrackable
- Prolog programs with updates are the hardest to write, debug, and understand
Example: Stacking a Pyramid

stack(0,X).
stack(N,X) :- N>0, move(Y,X), stack(N-1,Y).

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)).

Action:
?- stack(18,block32). % stack 18-block pyramid on top of block 32

Note:
Prolog won’t execute this very natural program correctly!
Syntax

- **Serial conjunction, \( \otimes \)**
  - \( a \otimes b \) – do \( a \) then do \( b \)

- The usual \( \land, \lor, \neg, \forall, \exists \) (but with a different semantics)
  - \( a \lor (b \otimes c) \land (d \lor \neg e) \)

- \( a \leftarrow b \equiv a \lor \neg b \)
  - Means: to execute \( a \) must execute \( b \)
Semantics

• The basic ideas
  – *Execution path* ≡ sequence of database states
  – Truth values over paths, not over states
  – Truth over a path ≡ *execution* over that path
  – *Elementary state transitions* ≡ propositions that cause a priori defined state transitions
Semantics

$\mathbf{a}$ $\mathbf{\otimes}$ $\mathbf{b}$

$a \otimes b$

$a \land b$

$a, b$
Semantics

The semantics makes updates logical

If action is true, but postcondition false, then
action $\otimes$ postcondition is false on $p$.

In practical terms: updates are undone on backtracking.
Proof Theory

- To prove $\phi$, tries to find a path, $\pi$, where $\phi$ is true
- $\Rightarrow$ executes $\phi$ as it proves it (and changes the underlying database state from the initial state of $\pi$ to the final state of $\pi$)
Pyramid Building (again)

stack(0,X).
stack(N,X) :- N>0 ⊗ move(Y,X) ⊗ stack(N-1,Y).

move(X,Y) :- pickup(X) ⊗ putdown(X,Y).
pickup(X) :- clear(X) ⊗ on(X,Y) ⊗ delete(on(X,Y)) ⊗ insert(clear(Y)).
putdown(X,Y) :- wider(Y,X) ⊗ clear(Y) ⊗ insert(on(X,Y)) ⊗ delete(clear(Y)).

?- stack(18,block32). % stack 18-block pyramid on top of block 32

• Under the Transaction Logic semantics the above program does the right thing
Constraints

• Can express not only execution, but all kinds of sophisticated constraints:

\[ \text{?- stack(10, block43)} \]
\[ \land \ \forall X,Y \ ((\text{move}(X,Y) \otimes \text{color}(X,\text{red})) \Rightarrow \exists Z(\text{color}(Z,\text{blue}) \otimes \text{move}(Z,X))) \]

*Whenever a red block is stacked, the next block stacked must be blue*

• Has been shown useful for process modeling (Davulcu et. al. PODS-97, Thesis 2002, Senkul et. al. VLDB-02)
Reasoning

• Can be used to *reason* about the effects of actions  [Bonner&Kifer 1998]
Integration into FLORA-2

• FLORA-2 provides
  – btinsert{Template | Query}
  – btdelete{Template | Query}
  – bterase{Template | Query}
  – And other “elementary” updates that behave according to the semantics of Transaction Logic

• FLORA’s “,” then serves as ⊗ and “;” as ∨, which allows us to build larger and larger transactions
Pragmatics

• FLORA-2 also provides non-logical updates that are similar, but more powerful to Prolog’s

• Logical updates + Prolog cuts
  – can be used to implement “partial commit” of transactions
  – have perfect sense in databases, but (unfortunately) not in Transaction Logic
Conclusion

• FLORA-2
  \[ \equiv \text{F-logic} + \text{HiLog} + \text{Transaction Logic} + \text{XSB} \]
  \[ \equiv \text{Logic} + \text{Objects} + \text{Meta-programming} \]
  + State changes + Implementation
Future Work

• **XSB**: has a number of problems that spoil the party
  – Limitations on cuts (will be fixed in the future)
  – Problems with updates
  – Bad interaction between tabling and updates

• **FLORA-2**:  
  – Interfaces to databases, C, Java
  – Additional features: encapsulation, various optimizations