Datalog for Enterprise Applications: from Industrial Applications to Research

Datalog 2.0 Workshop
March 16, 2010
Talk Objective

Discuss industrial influence on datalog research
So What Is LogicBlox?

LogicBlox is a platform-as-a-service for rapid development of enterprise SaaS applications in the domains of decision automation, analytics, and planning.

LB is based on datalog and is 100% declarative. It is intended to be used by domain experts who are either Excel or modeling tool savvy.
Enterprise Decision Automation Nightmare

9+ Tech Stacks with 12+ different programming languages = HAIRBALL
Oracle Fusion
Figure 1: SAP NetWeaver: Powering all SAP Solutions
Pep Talk

• Please forget anything you know about Enterprise software
• The way things are usually done does not matter

➤ We are here to invent the future 🦁

• Forget about the hairball. The hairball is dumb
• Some things need to be believed to be seen.

“The more ambitious plan may have more chances of success…”
G. Polya, How To Solve It, 1973
One programming/modeling language that is **declarative** and expressive for:
- Business Logic – Visual ORM Notation, Natural Language, Mathematical/Rules
- Workflow – Visual Statechart Notation, Mathematical/ECA rules
- User Interface – Spreadsheet, Web, Visualization, Multi-device(Desktop, Browser, Handheld, Voice), Machine Learning, Statistics, Data Mining
- Optimization – LP, IP, SAT, SMT

Throw in DB stuff (out-of-core, parallelization, concurrency, recovery)
Make schema changes easy so we can get spreadsheets and get our end-users involved
Some Popular Declarative Languages

Excel: The world’s most popular IDE with the world’s most popular programming language – a mostly declarative functional rule language:

\[ A1 = B1 + C1 \]

RDBMS: The world’s second most popular declarative programming language (SQL) – a mostly declarative logic rule language:

\[
\text{Select * from Sales, Returns}
\text{where ...}
\]

OLAP & Reporting tools: KPI editors

\[ \text{Netsales} = \text{Sales} - \text{Returns}. \]

Also: HTML, XML, Optimization languages
The Real World

- End Users
- Power Users
- Consultants
- Scientists & Statisticians
- Application Developers
- System Devs

NOT our target audience
Why Datalog?

- **Usability**
  - Executable modeling/specification language
  - Declarative by birth and first order by the grace of god
    - So is Excel – passing functions to functions is too hard for most
    - Cons() considered harmful – first order and first normal form
  - Guaranteed expressivity with “controllable” power
  - Skins – give people the syntax they like.

- **Safety**
  - Turing completeness considered harmful

- **Performance**
  - Memory hierarchy friendly
  - Parallelizable fragment
  - Jujutsu – let’s us use the power of the machine against itself

- **Large body of (mostly) un-commercialized research in Computer Science.**
“There is a flaw in the very foundations of Logic Programming: Prolog is nondeclarative”
-David S Warren (see www.cs.sunysb.edu/~warren/xsbbook/node2.html)

- In datalog conjunction is commutative!
- In datalog rule order doesn’t matter!
- In datalog the meaning of a program doesn’t depend on any one way of evaluating it!
Why Datalog Now? A Tale of Two Transitions

- A transition from owning your own computers to utility computing: SaaS, PaaS, and Cloud computing (IaaS).
  - It’s hard to change a mindset within 1 technology era
  - All bets are off in the new era. A transition usually means a re-write!!!
  - Many think that we need a rewrite to support multi-core and cloud-computing.
  - The giants of one era are the dinosaurs of the next
    - Mainframe: Sperry, Burroughs, CDC, Amdahl?
    - Mini: DEC, Wang, Data General?
    - DOS: Lotus, Wordperfect, dBase?
    - Client-Server: Oracle, SAP, JDA, Lawson, Manhattan?
    - SaaS: Salesforce.com, Netsuite, Workday?
  - The transition to client-server and open system economics helped establish the SQL era
  - Client-Server to SaaS transition in the next 5 years?
Why Datalog Now? A Tale of Two Transitions

- A transition from book keeping to decision automation
  - OLTP in “open” systems & client server is hard enough
  - OLTP, OLAP, spreadsheets, machine learning, & optimization is damn near impossible
  - Enterprises can no longer differentiate with book keeping (i.e. ERP) systems
    - IBM buys SPSS and ILOG
    - Accenture partners with SAS Institute
  - Transition to decision automation in the next 10 years?
“I’m perfectly okay if somebody says, ‘Very nice work, but if you can change it a little bit, maybe it will better fit my needs,’ rather than ‘You’re wasting our time, you’re wasting your time, stop bothering us,’ which is, I have to say, very often what we were hearing.”

– Moshe Vardi, (see www.sigmod.org/record/issues/0603/p56-column-marianne.pdf)

• We have a new generation of systems people who:
  – Aren’t scared of theory
  – Aren’t scared of Michael Stonebraker 😊

• Let’s not miss this window
What Kind of Datalog?
What Kind of Datalog?

- Classic datalog as a foundation
- Datalog with Integrity Constraints
- Datalog with State and Incremental Update
- Datalog with Defaults
- Datalog with Higher Order Predicates
  - But didn’t you say “first order by the grace of god”? 
- Datalog with Existentially Quantified Head Variables
- Datalog with Constraint Stratification
- Datalog with IC’s on Schemas (Schema Schemas)
- Datalog with language skins
- Datalog evaluated in different ways – not just semi-naive
I Am My Own Grandpa (1)

T() ← person(x).
F() ← !person(x), man(x).
F() ← !person(x), woman(x).

There are person entities (no-op)
All men and women are persons
I Am My Own Grandpa (1)

```
person(x) ->.
man(x)  ->  person(x).
woman(x)  ->  person(x).
```

There are person entities (no-op)

We flipped the arrows to distinguish IC’s from derivation rules
person(x) ->.
man(x) -> person(x).
woman(x) -> person(x).

There are person entities (no-op)

man(x) -> man(x);
woman(x) -> woman(x).
!(man(x), woman(x)).

All men and women are persons

person(x) -> man(x); woman(x).
!

All persons are either men or women

!(man(x), woman(x)).

No person is both a man and a woman

father(x, v) -> person(x), man(v).
father(x, v), father(x, v') -> v=v'.

Fathers are men and everyone has at most one

father(x, v) -> person(x), man(v).
father(x, v), father(x, v') -> v=v'.

Fathers are men and everyone has at most one
person(x) -> .
man(x) -> person(x).
woman(x) -> person(x).

person(x) -> man(x); woman(x).
!(man(x), woman(x)).

father[x] = v -> person(x), man(v).

There are person entities (no-op)
All men and women are persons
All persons are either men or women
No person is both a man and a woman
We added a “skin” for functions (however, we lose semantic stability)
I Am My Own Grandpa (1)

\[
\begin{align*}
\text{person}(x) & \rightarrow . \\
\text{man}(x) & \rightarrow \text{person}(x). \\
\text{woman}(x) & \rightarrow \text{person}(x). \\
\text{person}(x) & \rightarrow \text{man}(x); \text{woman}(x). \\
!(\text{man}(x), \text{woman}(x)). \\
\text{father}[x] = y & \rightarrow \text{person}(x), \text{man}(y). \\
\text{mother}[x] = y & \rightarrow \text{person}(x), \text{woman}(y). \\
\text{wife}[x] = y & \rightarrow \text{man}(x), \text{woman}(y). \\
\text{husband}[x] = y & \rightarrow \text{woman}(x), \text{man}(y).
\end{align*}
\]

There are person entities (no-op)
All men and women are persons
All persons are either men or women
No person is both a man and a woman
Fathers are men and everyone has at most one
Mothers are women and everyone has at most one
Wives are women and every man has at most one
Husbands are men and every woman has at most one
bioparent(x, y) -> person(x), person(y).  A biological parent is either a mother or a father

bioparent(x, y) <-
  mother[x] = y;
  father[x] = y.

bioanc(x, y) -> person(x), person(y).  A biological ancestor is a biological parent or the biological ancestor of a biological parent

bioanc(x, y) <- bioparent(x, y);
  bioparent(x, t), bioanc(t, y).

selfbioanc() <- bioanc(p, p).  No person is his or her own biological ancestor

!selfbioanc().

wife[x] = y -> husband[y] = x.  A man’s wife has that man as a husband

!(wife[x] = y, bioanc(x, y))  A man’s wife is not a biological ancestor

!(husband[x] = y, bioanc(x, y)).  A woman’s husband is not a biological ancestor
parent(x, y) <- bioparent(x, y);
    wife[father[x]] = y;
    husband[mother[x]] = y.

A parent is a biological parent, or a father’s wife, or a mother’s husband

grandpa(x, y) -> person(x), man(y).
grandpa(x, y) <- parent(x, t),
    parent(t, y), man(y).

A grandpa is a parent’s parent who is a man

ownGrandpa(m) -> man(x).
ownGrandpa(m) <- grandpa(m, m).

An own grandpa is the same man who is a grand child and a grand pa in the same entry of grandpa

ownParent(m) <- parent(m, m).

An own parent is the same person who appears twice in the same entry of parent
Datalog with State (based on Statelog)

name[p] = s -> person(p), string(s).

person(p) -> name[p] = _.

name(p, s), name(p’, s) -> p=p’.

+person(p), +name[p] = “fred” <-.

+person(p), +name[p] = “tom” <-.

+parent(x, y) <- name[x] = “tom”,
               name[y] = “fred”.

A name is a function from person to string.
Every person has a name.
Name is injective
Assert that there is a person called “fred”.
Assert that there is a person called “tom”.
Assert that “fred” is the parent of “tom”
A name is a function from person to string.

Every person has a name.

Name is injective

Assert that there is a person called “fred”.

Assert that there is a person called “tom”.

If you enter this, we’ll do the lookups for you.
Statelog in a nutshell

- Every predicate has “+” and “-” shadow predicates
- Every predicate has “now” and “next” shadow predicates
- These are all related via frame rules:
  \[
  \text{?R@next}(x) \leftarrow +?R(x).
  \]
  \[
  \text{?R@next}(x) \leftarrow ?R@now(x), !-?R(x).
  \]
- Run-time is optimized for these rules.
- “+” & “-” shadow predicates are “events” if in the body and “actions” if in the head:
  \[
  +\text{act}(x, y) \leftarrow +e1(y), -e2(x), \text{cond}(x, y).
  \]
An Observation in the Retail Industry

Decision-Support or Decision-Automation

Execution

Retail Operations

Excel (the good):
- Light
- Iterative
- User friendly
- Evolves
- Flexible
- “Comfort Zone”
- Tailored Practice

Excel (the bad):
- Outside of IT support
- Less reliable
- Incomplete information
- Many versions
- Hard to scale
- Hard to create group learning
- No science

Traditional (the bad):
- “Best Practice”
- One Size Fits All
- Rigid
- Heavy
- Black-box science
- Generic science
- Oversold automation

Traditional (the good):
- Sophisticated Science
- IT Friendly
- Reliable
- One version of the truth
- Centralize Management
- Workflow friendly
- Scalable

Goal:
- IT Friendly
- Science On-Demand
- Reliable
- One version of the truth
- Easy to administer
- Centralize Management
- Workflow friendly
- Scalable
- Tailored Practice
- Flexible
- Iterative
- Evolve … “Do-Learn-Do”
- Group learning

Excel (the good):
- Light
- Iterative
- User friendly
- Evolves
- Flexible
- “Comfort Zone”
- Tailored Practice

Excel (the bad):
- Outside of IT support
- Less reliable
- Incomplete information
- Many versions
- Hard to scale
- Hard to create group learning
- No science
What is the Enterprise Version of Excel?

<table>
<thead>
<tr>
<th>Enterprise</th>
<th>Oracle, DB2, etc.</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop</td>
<td><strong>Access</strong></td>
<td><strong>Excel</strong></td>
</tr>
<tr>
<td></td>
<td>Book Keeping</td>
<td>Decision Making</td>
</tr>
</tbody>
</table>
## One Way To Think About LogicBlox

<table>
<thead>
<tr>
<th>Enterprise</th>
<th>Oracle, DB2, etc.</th>
<th>LogicBlox</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(with good support for book-keeping)</td>
</tr>
<tr>
<td>Desktop</td>
<td>Access</td>
<td>Excel</td>
</tr>
<tr>
<td></td>
<td>Book Keeping</td>
<td>Decision Making</td>
</tr>
</tbody>
</table>
Quick Demo
Evolution of Enterprise Software

TODAY

Traditional Solution

Partly Configured
Partly Hard Coded
Disparate Applications

TOMORROW

100% Configured
No Hard Coding
Unified Platform
Research Roadmap

- **Point free notation**
  - Enhances semantic stability of derivation rules
  - e.g. net = sales – returns.

- **Incremental compiler & optimizer**
  - Compiler specified in LB in order to benefit from built-in incremental evaluation

- **Data exchange and migration when schema changes**

- **Visualization**

- **View update**
UIBlox User Interface Construction

- 100% Declarative – including controller logic
- Modeled on W3C XForms standard
- GUI as view on the application model
- Sophisticated browser based Portal and UI
- Sophisticated Java Applet for read-write multi-dimensional spreadsheet component
- Drag-and-drop, browser based UI construction
### Promotion Info
- Promotion: HD TV Bundle
- Core Event: March Madness
- Ad Zone: Mid-West
- Mechanic: Menu Offer $x

### Cost of Marketing
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Flyer</td>
<td>25000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>iPhone</td>
<td>10000</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Social Media</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### Products

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Regular Price</th>
<th>Effective Discount</th>
<th>Vendor Funding</th>
</tr>
</thead>
<tbody>
<tr>
<td>SKU 1</td>
<td>HD TV</td>
<td>$399.00</td>
<td>$0.25</td>
<td>$75.00</td>
</tr>
<tr>
<td>SKU 2</td>
<td>Digital Box</td>
<td>$69.99</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
<tr>
<td>Service 1</td>
<td>Netflix Monthly Gold Subscription</td>
<td>$25.00</td>
<td>$0.00</td>
<td>$0.00</td>
</tr>
</tbody>
</table>
Sample Code – ½ UI Description

```xml
- !Form
  name: PromoApproval
  label: Promotion Approval
  contextPredicates: ['PAF:offer']
  children:
    - !Group
      layout: !Rows {borderWidth: 0, deviceId: browser}
      children:
        - !Group
          layout: !Pane {defaultSize: 67%, position: top, deviceId: browser}
          children:
            - !Group
              layout: !Columns {name: paPageCols, borderWidth: 0, deviceId: browser}
              children:
                - !Group
                  name: paLeftGrp
                  layout: !Pane {defaultSize: 25%, deviceId: browser, position: left, splitter: False}
                  children:
                    - !Label {name: promoInfo, importance: 2, value: Promotion Info}
                    - !Output {name: promoLabel, label: Promotion, value: !Predicate 'Offer:name'}
                    - !Output {name: coreEvent, label: Core Event, value: !Predicate 'Event:name'}
                    - !Output {name: adZone, label: Ad Zone, value: !Predicate 'AdZone:name'}
                    - !Output {name: mechanic, label: Mechanic, value: !Predicate 'Mechanic:name'}
```

...
Sample Code – ½ Application Schema

Type(t), Type:name(t:name) -> string(name).

Product(p), Product:name(p:name) -> string(name).
Product:description[p] = d -> Product(p), string(d).
Product:margin[p] = v -> Product(p), float[32](v).
Product:cost[p] = v -> Product(p), float[32](v).
Product:type[p] = t -> Product(p), Type(t).

AdZone(a), AdZone:name(a:name) -> string(name).
Channel(c), Channel:name(c:name) -> string(name).
Location(l), Location:name(l:name) -> string(name).
Location:adZone[l] = a -> Location(l), AdZone(a).
Location:channel[l] = c -> Location(l), Channel(c).

Month(m), Month:name(m:name) -> string(name).
Month:weeks(w,m) <- Week:month[w] = m.
Week(w), Week:name(w:name) -> string(name).
Week:month[w] = m -> Week(w), Month(m).
Week:days(d,w) <- Day:week[d] = w.
Day(d), Day:name(d:name) -> string(name).
Day:week[d] = w -> Day(d), Week(w).

Event(e), Event:name(e:name) -> string(name).

...
Research Roadmap

• Data validation
  - Reg Expressions too crude
  - Schemas for legal inputs
  - Use the same schemas to generate sample data for testing

• View update
  - Bidirectional Exchange

• Distributed computation
  - Do in the browser as much as can be done

• Point free notation
• Joint work with Terry Halpin and Matt Curland (now at LB)
• Object Role Modeling (ORM) conceptual modeling notation developed by Terry Halpin over the last 20 years.
• Tools support visual and structured natural language interfaces
• 100’s of papers and several books published
VisualBlox Domain Modeling Environment
Cinema Model

* Cinema is multiplex iff Cinema has N-Theaters > 1.
Movie Model
Rich Constraint Language

- Unary and Binary Fact Types
- Ternary Fact Types
- Inclusive-Or Constraints
- External Uniqueness Constraints
- Value Constraints
- Derived Fact Types
- Objectification
- Independent Object Types
- Set-Comparison Constraints (single-role and multi-role)
- Frequency Constraints
- Ring Constraints
- Subtype Constraints
- User Specified Textual Constraints & Derivation rules
I Am My Own Grandpa in ORM
Syntax Skins
Free Syntax – Avoiding Syntax Holy Wars

Diagram

Rules

Tabular

Structured NL

LB-VM
Research Roadmap

- **Rules**
  - Hierarchical syntax – XML & YAML

- **Diagrams**
  - Statelog & State Chart
  - Generics & Templates
  - Euler, Spider, Constraint Diagrams
  - User Interface Annotations
  - Petri Nets & Activity Diagrams
  - Message Sequence Charts
  - Live Sequence Charts

- **Structured Natural Language**
  - FORML2 - for verbalizing rules

- **Tabular Notation**
Client Business Requirements

- Build a competitively differentiated application that supports multiple programs.
  - Never been done before with packaged apps
  - Need for EUP/self service

- Quick scale – client didn't want to spend months building a team.
  - Home grown option was problematic. Resources stretched thin. Dev not seen as core competency
  - Homegrown and traditional consulting options had long maintenance tail

- Make good build vs. buy decisions -- build what is differentiated and buy the commodity
  - Argued against package solution that is available to fast followers.

- Painfully aware of multiple peers
  - Spending 2 to 4 years and many million $$ without success
  - Failure led to lawsuit in at least one case

- Solution to include rating, billing, CRM, document management, workflow, MC simulation to assess portfolio “correlated” risk
## Analyst’s Assessment of Policy Admin Vendors

<table>
<thead>
<tr>
<th>Vendor 1</th>
<th>Vendor 2</th>
<th>Vendor 3</th>
<th>Vendor 4</th>
<th>Vendor 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># of Clients</strong></td>
<td>10 - 13</td>
<td>2 in US</td>
<td>12 carriers + 2 MGAs</td>
<td>3 in US</td>
</tr>
<tr>
<td><strong>Rules &amp; Code Based</strong></td>
<td>9</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td><strong>UI Integration</strong></td>
<td>9</td>
<td>?</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td><strong>Integrated Rating Tool</strong></td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td><strong>Self Service</strong></td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td><strong>Commercial Focus</strong></td>
<td>10</td>
<td>10</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td><strong>Portal 2.0</strong></td>
<td>7</td>
<td>9</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td><strong>License</strong></td>
<td>$.6 - $1.8 mm</td>
<td>.75 - $3.0 mm</td>
<td>$.3 - $2.0 mm</td>
<td>$.5 - $1.0 mm</td>
</tr>
<tr>
<td><strong>Implementation Cost</strong></td>
<td>$1.2 - $6.0 mm</td>
<td>$2.0 - $4.0 mm</td>
<td>$1.0 - $8.0 mm</td>
<td>$.25 - .75 mm</td>
</tr>
<tr>
<td><strong>Implementation Time</strong></td>
<td>5 - 18 months all lines</td>
<td>???</td>
<td>6 - 8 months multi-line</td>
<td>6 - 12 months</td>
</tr>
<tr>
<td><strong>Bench Depth</strong></td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td><strong>Technology</strong></td>
<td>Oracle, PL2</td>
<td>Any DB, Java server</td>
<td>Any DB, Java server</td>
<td>Any DB, C++</td>
</tr>
<tr>
<td><strong>Content</strong></td>
<td>Some</td>
<td>Workmans comp</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
## Application Stats

### Straight-Through Processing Policy App: Rules by App Component and Type of Rule

<table>
<thead>
<tr>
<th>Component</th>
<th>Rules</th>
<th>Constraints</th>
<th>Seed Rules</th>
<th>Seed Constraints</th>
<th>Total By Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foundation</td>
<td>1,129</td>
<td>1,470</td>
<td>182</td>
<td>64</td>
<td>2,845</td>
</tr>
<tr>
<td>Rating &amp; Pricing</td>
<td>708</td>
<td>1,105</td>
<td>84</td>
<td>36</td>
<td>1,933</td>
</tr>
<tr>
<td>Policy Core</td>
<td>2,626</td>
<td>3,307</td>
<td>56</td>
<td>12</td>
<td>6,001</td>
</tr>
<tr>
<td>Billing</td>
<td>453</td>
<td>1,028</td>
<td>136</td>
<td>176</td>
<td>1,793</td>
</tr>
<tr>
<td>Doc / Forms Mgt</td>
<td>126</td>
<td>347</td>
<td>186</td>
<td>100</td>
<td>759</td>
</tr>
<tr>
<td>Recruiter Portal</td>
<td>46</td>
<td>193</td>
<td>220</td>
<td>288</td>
<td>747</td>
</tr>
<tr>
<td>Product Config</td>
<td></td>
<td></td>
<td>46</td>
<td>199</td>
<td></td>
</tr>
<tr>
<td>Workflow</td>
<td>1,908</td>
<td>370</td>
<td></td>
<td></td>
<td>2,278</td>
</tr>
<tr>
<td>User Interface</td>
<td>3,692</td>
<td>3,912</td>
<td></td>
<td></td>
<td>7,604</td>
</tr>
<tr>
<td><strong>Total by Type</strong></td>
<td>10,688</td>
<td>11,732</td>
<td>910</td>
<td>875</td>
<td>24,205</td>
</tr>
</tbody>
</table>

### Complex Policy App: Rules by App Component and Type of Rule

<table>
<thead>
<tr>
<th>Component</th>
<th>Rules</th>
<th>Constraints</th>
<th>Seed Rules</th>
<th>Seed Constraints</th>
<th>Total By Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backend</td>
<td>3,317</td>
<td>1,626</td>
<td>60</td>
<td>206</td>
<td>5,209</td>
</tr>
<tr>
<td>User Interface</td>
<td>2,080</td>
<td>1,247</td>
<td>3</td>
<td>52</td>
<td>3,382</td>
</tr>
<tr>
<td><strong>Total by Type</strong></td>
<td>5,397</td>
<td>2,873</td>
<td>63</td>
<td>258</td>
<td>8,591</td>
</tr>
</tbody>
</table>
## Application Stats (cont’)

<table>
<thead>
<tr>
<th>Straight-Through Processing Policy App Development Days by Version and Component</th>
<th>V1.0</th>
<th>V1.1</th>
<th>V1.2 (est)</th>
<th>Total by Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy Core</td>
<td></td>
<td></td>
<td>291</td>
<td>291</td>
</tr>
<tr>
<td>Rating &amp; Pricing</td>
<td></td>
<td></td>
<td>180</td>
<td>180</td>
</tr>
<tr>
<td>Doc / Forms Mgt</td>
<td></td>
<td></td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Product 1 Config</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product 2 Config</td>
<td></td>
<td></td>
<td>371</td>
<td>371</td>
</tr>
<tr>
<td>Total Policy</td>
<td>2,467</td>
<td>429</td>
<td>993</td>
<td>3,889</td>
</tr>
<tr>
<td>Billing</td>
<td>335</td>
<td>123</td>
<td>10</td>
<td>468</td>
</tr>
<tr>
<td>Recruiter Portal</td>
<td>190</td>
<td></td>
<td></td>
<td>190</td>
</tr>
<tr>
<td>EUP / EUM / Common</td>
<td>997</td>
<td>184</td>
<td>334</td>
<td>1,515</td>
</tr>
<tr>
<td><strong>Total by Version</strong></td>
<td><strong>3,989</strong></td>
<td><strong>736</strong></td>
<td><strong>1,337</strong></td>
<td><strong>6,062</strong></td>
</tr>
</tbody>
</table>
Policy Admin Benchmark

With heavy simulated load on 1 Large EC2 instance

- 25% of pages load in < 1.8 seconds
- 50% of pages load in < 3.3 seconds
- 75% of pages load in < 5.2 seconds
- 95% of pages load in < 8.7 seconds
High Productivity Computing on GPUs

- Joint work with
  - Yalamanchili and Clark at GT (faculty)
  - D. Choudhary, G. Diamos, A. Kerr (graduate students)

- Conceive, implement, demonstrate, execution models for multi-GPU systems
  - Embodied in back-end system SW
- Integrate front-end for LB
  - Embodied in front-end tool chains
- Goal: Demonstrate ability to scale to large scale machines (Petaflops).
Financial Monte Carlo Simulation

- Client used C to implement a Monte Carlo simulation
  - Ran in 1 hour on client machine
- Client migrated from C to Python in order to improve analyst productivity
  - Provided us with ~1,000 line Python code
  - Python code ran 8 times slower (8 hours on client machine)
- We rewrote Python code to C and verified 8x performance difference
- We rewrote Python code to ~300 lines of datalog
  - Samples generated and materialized in LB
  - Results of simulation are aggregations of materialized predicates
  - Datalog on CPU run-time = Python on CPU run-time
  - BUT!!!!
Benchmark Results

Python ran 8 times slower than this in 656 minutes for 1M points.

Harmony distributes the work across 4 CPU cores, 2 slow GPU boards, and 1 fast GPU board.
Projected Problem Scaling

For 1M points/min

<table>
<thead>
<tr>
<th>Language</th>
<th>#Processors</th>
<th>Cost (processors)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>356 CPUs</td>
<td>$30616</td>
</tr>
<tr>
<td>C</td>
<td>83 CPUs</td>
<td>$7138</td>
</tr>
<tr>
<td>C/CUDA</td>
<td>20 GPUs</td>
<td>$2925</td>
</tr>
</tbody>
</table>

Using per processor costs for web vendors
Research Roadmap

• Implemented relational algebra “kernels” for GPUs
  – Have NOT implemented fix-point operator yet
• Automate compilation of datalog to GPU based run-time
  – Add support for datasets larger than GPU memory. Potentially use same program partitioning strategy referred to earlier.
• Performance Prediction
• Kernel Scheduling
• Dynamic Code Generation
• Kernel deconstruction and recombination
• Collaboration with GT and Oak Ridge on building a programming model for Petaflop machine.
  – $12M NSF grant to build a smaller prototype of the machine with ~1000 GPUs
LogicBlox
Optimization and
Constraint Satisfaction
Optimization and Constraint Satisfaction

- Joint work with Diego Klabjan and Bob Fourer at Northwestern
- Paper submitted to INFORMS Journal on Computing
- 2 Presentations at ICS ‘09
  - “Algebraic Modeling in a Deductive Database Language”
  - “Using Optimization Services in Datalog”
- Currently supports linear and integer programming
“I want to spend the least amount of money but still buy enough food that let’s me meet my daily minimums for important nutrients”

Minimize: \[ \sum_{j \in FOOD} cost_j \times Buy_f \]
Subject to: \[ \sum_{f \in FOOD} amt_{n,f} \times Buy_f \geq nutrLow_n, \forall n \in NUTR \]
\[ Buy_f \geq 0, \forall f \in FOOD \]
A Textbook Example

\[
\begin{align*}
\text{NUTR}(x), \text{NUTR}:\text{name}[x]=n & \rightarrow \text{string}(n). \\
\text{FOOD}(x), \text{FOOD}:\text{name}[x]=n & \rightarrow \text{string}(n).
\end{align*}
\]

\[
\begin{align*}
\text{amt}[n, f] = a & \rightarrow \text{NUTR}(n), \text{FOOD}(f), \text{float}[32](a), (a \geq 0). \\
\text{nutrLow}[n] = nL & \rightarrow \text{NUTR}(n), \text{float}[32](nL), (nL \geq 0). \\
\text{cost}[f] = c & \rightarrow \text{FOOD}(f), \text{float}[32](c), (c \geq 0).
\end{align*}
\]

\[
\begin{align*}
\text{Buy}[f] = b & \rightarrow \text{FOOD}(f), \text{float}[32](b), (b \geq 0). \\
\text{FOOD}(f) & \rightarrow \text{Buy}[f] = \_.
\end{align*}
\]

\[
\begin{align*}
\text{TotalCost}[] & += \text{cost}[j] \times \text{Buy}[j]. & \text{//objective}
\end{align*}
\]

\[
\begin{align*}
\text{sumAmtBuy}[n] & += \text{amt}[n, f] \times \text{Buy}[f]. & \text{//constraints}
\end{align*}
\]

\[
\begin{align*}
\text{NUTR}(n) & \rightarrow (\text{sumAmtBuy}[n] \geq \text{nutrLow}[n]).
\end{align*}
\]

\[
\begin{align*}
\text{lang:}\text{solve:variable(`Buy)}. & \text{ //solve}
\end{align*}
\]

\[
\begin{align*}
\text{lang:}\text{solve:min(`TotalCost).}
\end{align*}
\]
A small factory produces three kinds of items

- Coils
- Bands
- Plates

We want to help them decide how much of each item to produce in order to maximize their profit

- Must make a minimum amount of each item
- Must not use more manufacturing capacity than is available
Is Written As This Linear Program

// Givens
PRODUCT(x), PRODUCT:name(x:n) -> string(n).
tonsPerHr[p]=v -> PRODUCT(p), float[64](v).

// Constraints
profitPerTon[p]=v -> PRODUCT(p), float[64](v).
minTons[p]=v -> PRODUCT(p), float[64](v).
maxTons[p]=v -> PRODUCT(p), float[64](v).
maxHrs[]=v -> float[64](v).

PRODUCT(p), Tons[p]=v1, maxTons[p]=v2 -> v1 <= v2.
PRODUCT(p), Tons[p]=v1, minTons[p]=v2 -> v1 >= v2.

totalProductHrs[]=v -> float[64](v).
totalProductHrs[]=+ Tons[p] / tonsPerHr[p] <-.
totalProductHrs[]=v1, maxHrs[]=v2 -> v1 <= v2.

// Objective
TotalProfit[]=v -> float[64](v).
TotalProfit[]=+ Tons[p]*profitPerTon[p] <-.

// Solve for
Tons[p]=v -> PRODUCT(p), float[64](v).
PRODUCT(p) -> Tons[p]=_.

lang:solve:variable(`Tons).
lang:solve:max(`TotalProfit).
// Givens
PRODUCT(x), PRODUCT:name(x:n) -> string(n).
tonsPerHr[p]=v -> PRODUCT(p), float[64](v).

// Constraints
profitPerTon[p]=v -> PRODUCT(p), float[64](v).
minTons[p]=v -> PRODUCT(p), float[64](v).
maxTons[p]=v -> PRODUCT(p), float[64](v).
maxHrs[] = v -> float[64](v).

PRODUCT(p), Tons[p]=v1, maxTons[p]=v2 -> v1 <= v2.
PRODUCT(p), Tons[p]=v1, minTons[p]=v2 -> v1 >= v2.

totalProductHrs[] = v -> float[64](v).
totalProductHrs[] += Tons[p] / tonsPerHr[p] <-.
totalProductHrs[] = v1, maxHrs[] = v2 -> v1 <= v2.

// Objective
TotalProfit[] = v -> float[64](v).
TotalProfit[] += Tons[p]*profitPerTon[p] <-.

// Solve for
Tons[p]=v -> PRODUCT(p), int[32](v).  ← A one line change
PRODUCT(p) -> Tons[p]=_.

lang:solve:variable(`Tons).
lang:solve:max(`TotalProfit).
Or Drawn as this Integer Program
Multiple Variables

- Each variable is a predicate
  - A function predicate whose result is integer or float
  - Arbitrary dimensionality and multiple variables in the same problem

\[
\text{Make}[o,p]=v \rightarrow \text{ORIG}(o), \text{PROD}(p), \text{float}[64](v), v\geq 0.
\]

\[
\text{ORIG}(o), \text{PROD}(p)\rightarrow \text{Make}[o,p]=_.
\]

\[
\text{Trans}[o,d,p]=v \rightarrow \text{ORIG}(o), \text{DEST}(d), \text{PROD}(p), \text{float}[64](v), v\geq 0.
\]

\[
\text{ORIG}(o), \text{DEST}(d), \text{PROD}(p)\rightarrow \text{Trans}[o,d,p]=_.
\]

- \text{lang:solver:variable(`Make`).}
- \text{lang:solver:variable(`Trans`).}
Constraints can mix variables of different types

\[
\text{totalShipOut}[o,p] = v \rightarrow \text{ORIG}(o), \text{PROD}(p), \text{float}[64](v).
\]
\[
\text{totalShipOut}[o,p] += \text{Trans}[o,_,p].
\]
\[
\text{ORIG}(o), \text{PROD}(p), \text{totalShipOut}[o,p] = v_1, \text{Make}[o,p] = v_2 \rightarrow v_1 = v_2.
\]

\[
\text{totalDelivery}[d,p] = v \rightarrow \text{DEST}(d), \text{PROD}(p), \text{float}[64](v).
\]
\[
\text{totalDelivery}[d,p] += \text{Trans}[_,d,p].
\]
\[
\text{DEST}(d), \text{PROD}(p), \text{totalDelivery}[d,p] = v_1,
\]
\[
\text{demand}[d,p] = v_2 \rightarrow v_1 = v_2.
\]
Optimization in LB allows us to specify a family of problems indexed by some set. Each problem is solved separately

- Automated subproblem detection
  - Write one (large) mixed integer problem and the compiler will split it up into smaller ones which are easier to solve individually

- Incremental solving
  - Given a (large) group of indexed subproblems, datalog incremental evaluation will invoke the solver only to solve subproblems for which parameters and data have changed
A Note on Disjunction

- Disjunctive constraints not usually supported by LP/MIP
  - LB compiler automatically applies a convex hull algorithm to solve disjunctive constraints
  - Can be useful, but may be too computationally intensive

x[s]=v -> p(s), int[32](v), v<=5, v>=0.
p(s)->x[s]=_.
lang:solver:variable(`x).
y[s]=v -> p(s), int[32](v), v<=5, v>=0.
p(s)->y[s]=_.
lang:solver:variable(`y).
p(s), p:id(s:n) -> string(n).
objective[]+=x[s]+y[s].
lang:solver:minimal(`objective).
p(s)-> constr(s).
lang:isEntity[`constr]=false.
constr(s)->p(s).
constr(s)<-x[s]+y[s]<=2.
constr(s)<-x[s]-y[s]>=2.

x+y<=2
x-y>=2
Research Roadmap

- In production
  - Linear and Mixed integer programming
  - Disjunctive constraints
  - Dual values
    - A declarative specification of dual variables – useful in real world applications of LP
  - Ability to invoke different solvers
    - Gurobi, COIN-OR, Cplex, lp_solve

- In near future
  - Quadratic programming
  - SAT Solver

- In planning stage
  - User defined constraint solvers – working with Tom Schrijvers
  - Meta-heuristics modeled on Comet system
  - Approximation algorithms based on Grecco papers
  - Scalable Theorem Proving using datalog based on Hinrichs papers
  - Planning and Scheduling solvers
    - Using LB as config system for LB
LB Machine Learning

• Joint work with Alex Gray et al. at Georgia Tech
• Several papers published at NIPS ‘08, ’09
• LB features a powerful set of machine learning methods, designed for:
  • Comprehensive analytics capability
  • State-of-the-art statistical methodology
  • Scalability for massive datasets
• ML cast as an optimization problem:
  • Minimize error of fit to data
Machine Learning Methods

Advanced Queries
- Range search
- Single-query nearest-neighbors
- All-nearest-neighbors, all-farthest-neighbors

Density Estimation
- Histograms
- Kernel density estimation

Regression
- Linear regression with VIF
- Ridge regression
- LASSO regression
- Stepwise regression
- Kernel regression

Classification
- Naïve Bayes classifier
- Nonparametric Bayes classifier
- K-nearest-neighbor classifier
- Support vector machine
- Classification decision tree

Clustering
- K-means
- Spectral clustering

Dimension Reduction
- Principal component analysis
- Kernel principal component analysis
- Non-negative matrix factorization
- Maximum variance unfolding
Research Roadmap

- Reformulate ML algorithms as optimization problems
  - Let’s us experiment with different cost functions
  - Let’s us experiment with different optimizers
- Inductive Logic Programming & Relational Statistics
  - Avoids the need to reformulate or “flatten” the problem
  - Takes advantage of known structure
Integrity
Constraints

Blurring the Line Between Statically and Dynamically Typed Languages
Static and Dynamic Languages

- We think of IC’s as a descriptive type system
  - i.e. The IC’s don’t change the meaning of the derivation rules
- If we can statically prove that a derivation rule is not contained in an IC, then we signal a type error.
  - This improves safety
- If we can statically prove that a derivation rule is contained in an IC, then we can remove the IC from the running program. We can also avoid evaluating rule fragments.
  - This improves performance
- If containment check is not feasible, then IC remains in the program to catch errors at run-time.
  - This allows us to trade-off compile-time sophistication for run-time performance
- Bridging the divide between static and dynamic languages
**IC’s**

<table>
<thead>
<tr>
<th>IC’s</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>person(x) -&gt;.</td>
<td>grandpa(x, y) &lt;- parent(x, t), parent(t, y), man(y).</td>
</tr>
<tr>
<td>man(x) -&gt; person(x).</td>
<td></td>
</tr>
<tr>
<td>parent(x, y) -&gt; person(x), person(y)</td>
<td></td>
</tr>
<tr>
<td>grandpa(x, y) -&gt; person(x), man(y).</td>
<td></td>
</tr>
</tbody>
</table>

person(x), man(y) <- grandpa(x, y) <- parent(x, t), parent(t, y), man(y).

person(x), man(y) <- parent(x, t), parent(t, y), man(y).

[person(x), man(y)] CONTAINS [parent(x, t), parent(t, y), man(y)]
IC’s

person(x) ->.
man(x) -> person(x).
parent(x, y) -> person(x), person(y)
grandpa(x, y) -> person(x), man(y).

Rules

grandpa(x, y) <- parent(x, t),
parent(t, y),
man(y).

grandpa (person x man)
I Am My Own Grandpa - Static

**IC’s**

- `person(x) -> .`  
- `man(x) -> person(x).`  
- `parent(x, y) -> person(x), person(y)`  
- `grandpa(x, y) -> person(x), man(y)`

**Rules**

- `grandpa(x, y) <- parent(x, t), parent(t, y), man(y).`
I Am My Own Grandpa - ERROR

**IC’s**
- person(x) ->.
- man(x) -> person(x).
- parent(x, y) -> person(x), person(y)
- grandpa(x, y) -> person(x), man(y).

**Rules**
- grandpa(x, y) <- parent(x, t), parent(t, y).
Example: Datalog with Complex Type Hierarchies

- Joint work with Oege de Moor, Max Schafer, & Semmle. “Joint” in the sense that we motivated it, they did all the work, and we plan on using it 😊.

- Uses IC’s defined in datalog with monadic extensionals including statements of disjointness, implication, and equivalence

- Paper POPL ’10:
  - “Type Inference for Datalog with Complex Type Hierarchies”

- See Max’s presentation
Inspired by Alloy

Software Abstractions
Logic, Language, and Analysis
Daniel Jackson
Basic Idea

- Observation about design analysis
  - Most flaws have small counterexamples
  - “small scope hypothesis”

*testing: a few cases of arbitrary size*

*scope-complete: all cases within a small bound*
Can constraints be satisfied (or not)?

- Solve for configuration with at most 4 people in which a man is his own grandfather
  
  
  `lang:solve:variable(`ownGrandpa`).

- Check command instructs LB to search for counter example to `ownParent` within scope of at most 3 persons
  
  -> count[`person`] = 3.
  
  `lang:check:variable(`ownParent`).`
LogicBlox Implementation

- Joint work with Tim Sheard
- Datalog -> Extended RA -> SAT
- Extends Alloy & KodKod in several dimensions.
  - Arbitrary arity relations (instead of just binary relations)
  - Notions of both projection and selection
  - A generalized notion of join (that doesn't throw away the join column)
  - A notion of complement
  - A fix point operator (translation to a SAT is parameterized by an index n, that unfolds the fixpoint n times) setting n to the size of the universe should be exact, but for many problems, smaller sizes might work.
  - Working on using the SAT solver to cut off the iteration when (fix n term) == (fix (n+1) term). This can be tested with the SAT solver
Combinatorial Testing
What if the Model/Program is Too Big?

- Joint work with Kurt Stirewalt, Matt McGill, Laura Dillon at Michigan State. Kurt Stirewalt now at LB
- Can generate populations from one part of the model (the inputs) to validate the rest of the model
- Generated populations distributed ~uniformly
- Submitted for publication
  - “Generating Combinatorial Test Suites of Structurally Complex Inputs for Model Transformers”
Combinatorial Testing

More uniform than ad hoc testing

combinatorial testing:

Bigger Scope than Alloy-like methods

testing:
a few cases of arbitrary size

scope-complete:
all cases within a small bound
Eating Our Own Dog Food: VisualBlox Compiler
## Scope Selection

### ATIG - RC-cardcats®

<table>
<thead>
<tr>
<th>Options</th>
<th>Scopes</th>
<th>Test Parameters</th>
<th>Alloy Specification</th>
<th>Forbidden Sets</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Scope:</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitwidth:</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Type</th>
<th>Scope</th>
<th>Override Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>FactType</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FactType is binary with distinct role players</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FactType is implied</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>FactType is subtype fact</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Formula</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Formula is negated</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Join</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>JoinType</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>ObjectType</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>ObjectifiedType</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>PredCreationOption</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Query</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Role</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>RoleSeq</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>RoleSeq is same fact</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>RoleVar</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>Split</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>SplitOp</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>
Defining Test Parameters – Scope Partitions
Inferred Invalid Value Combinations (IC’s)

```
<table>
<thead>
<tr>
<th>Forbidden Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 &lt;=</td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4 &lt;=</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>4 &lt;=</td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td>2 &lt;=</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>7 &lt;=</td>
</tr>
</tbody>
</table>
```
Combinatorial Test Generation

- We found 6 distinct bugs in our VisualBlox implementation that were not discovered by ad-hoc tests.
- T-way test suite guarantees coverage of all feasible size T combinations of features.
  - 2-way test of 10 parameters of 3 values each: 19 tests
  - 3-way test of 10 parameters of 3 values each: 65 tests
- Incremental algorithm that uses model instances to automatically infer so-called forbidden sets (or IC’s for that scope) during the generation of a T-way test suite
- Doesn’t currently support aggregation, external uniqueness constraints, ring constraints
  - Because current implementation translates to Alloy first
- Adding support for more expressive scope partitioning
Combinatorial Testing
What About Scalable Test Data Generation?

- Joint work with Yannis Smaragdakis at UMass
- Can generate very large sample databases from conceptual models based on subset of constraints
- Best paper award at ASE 2007

---

Scalable Automatic Test Data Generation from Modeling Diagrams

Yannis Smaragdakis
University of Groningen
Eindhoven, The Netherlands
yaniss@cs.rug.nl

Christoph Caillier
Georgia Tech
Atlanta, GA 30332-0815
caillier@cc.gatech.edu

Ranjith Subramaniam
College of Computing
Georgia Tech
Atlanta, GA 30332-0815
ranjith@cc.gatech.edu

ABSTRACT

We explore the automatic generation of test data that respect constraints expressed in the Object-Role Modeling (ORM) language. ORM is a popular conceptual modeling language,specially targeting database applications, with applications in its practice. The general problem of query data that respect ORM constraints is undecidable in general, and even in restricted settings. ORM constraints can be expressed in first-order logic, and any ORM query that can be expressed can be validated. In this paper, we present a scalable and efficient algorithm to generate sample databases that respect ORM constraints. The key idea is to use a set of automatic test data generation techniques that generate sample databases that respect ORM constraints. The key idea is to use a set of automatic test data generation techniques that generate sample databases that respect ORM constraints.

Categories and Subject Descriptors
D.4.4 [Software Engineering]: Testing and Debugging—testing under Object-Role Modeling (ORM) Languages

General Terms
Design Languages

1. INTRODUCTION

ORM languages offer a concise way to capture design intentions and express dependencies at a level of abstraction that is lower than concrete DML. The highest level of abstraction offers fewer constraints or constraints that are more functional than ORM constraints. Therefore, a compelling trend is that of adding more and more functionality to ORM languages, in order to bridge the gap between ORM and traditional database design. ORM languages offer a concise way to capture design intentions and express dependencies at a level of abstraction that is lower than concrete DML. The highest level of abstraction offers fewer constraints or constraints that are more functional than ORM constraints. Therefore, a compelling trend is that of adding more and more functionality to ORM languages, in order to bridge the gap between ORM and traditional database design.
• Constraints Supported:
  - internal uniqueness or EGD’s (multiple can exist, can span multiple roles, but cannot overlap)
  - mandatory (no disjunctive mandatory)
  - frequency (like uniqueness)
  - value and cardinality
  - subtype

• Limitations chosen to reflect commonly used IC’s.
  - note no exclusion (used in NP-hardness result), ring, subset, or user-defined constraints
Can we use as a preprocessing step to the chase? Use the chase to produce satisfying instance for richer set of IC’s.

- Will we need all dependencies for the chase or can we omit the ones used to generate the input instance?
- Can we use datalog with existentially quantified head variables instead of the real chase?

Can we support more dependencies and keep scalability?

- Smaragdakis thinks we can do a little bit better
### Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Language</th>
<th>Automation</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>Limited &amp; varies e.g. Unary UCQ</td>
<td>Full</td>
<td>Infinite Scope</td>
</tr>
<tr>
<td>“Lightweight”</td>
<td>Full</td>
<td>Full</td>
<td>Small Scope</td>
</tr>
<tr>
<td>Combinatorial</td>
<td>Full</td>
<td>Partial</td>
<td>Uniform samples from medium scope</td>
</tr>
<tr>
<td>Scalable</td>
<td>Partial</td>
<td>Full</td>
<td>Large Scope</td>
</tr>
<tr>
<td>Unit Test</td>
<td>Full</td>
<td>None</td>
<td>Sample from infinite scope</td>
</tr>
</tbody>
</table>

- Can build up model incrementally
- Since the model expressed in datalog is executable, we eliminate risk of bugs introduced by “programming” the model!!!
Research Roadmap

- More containment checks
- Give the programmer the opportunity to provide a containment proof if the compiler can’t do so.
  - Inspired by Pasalic-Taha work on Conqoction
- Learn about model checking and explore uses for verifying specification properties.
Automatic Programming & Program derivation
Program Derivation

• Joint work with Alex Gray, Ryan Riegel, Sooraj Bhat
  - Inspired by AutoBayes (mostly), SPIRAL, FLAME, Dyna
• “Schemas” are core notion
  - A schema is a rewrite rule that embodies a problem reformulation theorem. For example:
    
    real(a), real(b), a * b = 0 -> a = 0; b = 0.
    real(x) -> x + 0 = x, 0 + x = x.
    a * b = 0 -> a + b = a; a + b = b.

• Higher order schemas (or schema combinators) take schemas as input and return a new schema
  - Takes the first two “schemas” above and infers the third.
  - E.g. induction is a higher order schema.

• Goal is to achieve correctness and performance by construction
Example: Generalized N-Body Problems

• Class of problems which would naïvely be solved in order $O(n^{>2})$
  - Includes nested aggregations and all range search queries
• Dual Tree traversal algorithms
  - Use inherent spatial information to cut down number of computations
  - Ideally Suited for space partitioning trees like Kd-Trees
  - Significantly reduce the complexity to $O(n \log(n))$ once and $O(n)$ many times – for an $O(n^2)$ algorithm
• Do for a large class of programs what quicksort did for sort and the FFT did for the Fourier Transform
  - Am I the only person annoyed by how hard it is to do $O(n \log(n))$ sort in datalog. Has anyone done it?
• Early results are encouraging but we still have a very long way to go.
- **Range Search**

  \[
  \text{count}[] = v \leftarrow \text{agg} << v = \text{count}() >> \\
  x[p] > 5.0, x[p] < 10.0, y[p] > 2.0, y[p] < 3.0
  \]

- **All Nearest Neighbor**

  // using Euclidean distance
  \[
  \text{nnDist}[p] = md, \\
  \text{nnPoint}[p] = q \leftarrow \text{agg} << md = \text{min}(\text{dist}), q = \text{min}(\text{dist}, r) >> \\
  \text{dist} = ((x[p] - x[r])*(x[p] - x[r])) + \\
  \]

  // or using a different distance metric
  \[
  \text{nnDist}[p] = md, \\
  \text{nnProduct}[p] = q \leftarrow \text{agg} << md = \text{min}(\text{dist}), q = \text{min}(\text{dist}, r) >> \\
  \text{dist} = \text{abs}[	ext{ProfitRank}[p] - \text{ProfitRank}[r]] + \\
  \text{abs}[	ext{SalesRank}[p] - \text{SalesRank}[r]].
  \]
Performance Improvement

- **Naïve**
  - Slope = 2, $O(n^2)$

- **Dual Tree (new)**
  - Slope = 4.7/4, $O(n^{1.175})$
Speedup Relative to Native Code

- Dual Tree
- Naïve

**Graph:**
- **Y-axis:** Speedup (log scale)
- **X-axis:** Dataset size (log scale)

The graph shows the speedup relative to native code for different dataset sizes, with Dual Tree and Naïve methods compared.
We have implemented the following:
- Matrix multiplication (just like a join)
- LU decomposition
- Cholesky decomposition
- QR factorization
- Linear system solution
- Eigenvalue/Singular Value decomposition
- Simplex algorithm
• Global stratification made it hard to express some of these algorithms
  - Exploring use of constraint stratification to allow safe recursion through aggregation

• We will use these datalog programs to benchmark our implementation against state of the art dense and sparse linear algebra systems.
  - We expect to lose, but hopefully not by much

• Does datalog allow us to more naturally model real problems instead of having to collapse them to 2-d matrices?
Algorithms + Data Structures = Programs
-Niklaus Wirth
(Prentice Hall, 1976)

Algorithm = Logic + Control
-Bob Kowalski
(Communications of ACM, 1979)

Program = Logic + Control (aka search) + Data Structure
-LogicBlox
Program = Logic + Control + Data Structure

Program = Datalog$^{LB}$ + Semi-Naïve + B-Tree
Program = Logic + Control + Data Structure

Program = Datalog$^{LB}$ + Semi-Naïve + B-Tree
Program = Datalog$^{LB}$ + DPLL + ?
Program = Logic + Control + Data Structure

Program = Datalog\textsuperscript{LB} + Semi-Naïve + B-Tree
Program = Datalog\textsuperscript{LB} + DPLL + ?
Program = Datalog\textsuperscript{LB} + Simplex + ?
Program = Logic + Control + Data Structure

Program = Datalog$^{LB}$ + Semi-Naïve + B-Tree
Program = Datalog$^{LB}$ + DPLL + ?
Program = Datalog$^{LB}$ + Simplex + ?
Program = Datalog$^{LB}$ + Branch&Bound + ?
Program = Logic + Control + Data Structure

Program = Datalog\(^{LB}\) + Semi-Naïve + B-Tree
Program = Datalog\(^{LB}\) + DPLL + ?
Program = Datalog\(^{LB}\) + Simplex + ?
Program = Datalog\(^{LB}\) + Branch&Bound + ?
Program = Datalog\(^{LB}\) + Divide&Conquer + Kd-Tree
Program = Logic + Control + Data Structure

Program = Datalog\(^{LB}\) + Semi-Naïve + B-Tree
Program = Datalog\(^{LB}\) + DPLL + ?
Program = Datalog\(^{LB}\) + Simplex + ?
Program = Datalog\(^{LB}\) + Branch&Bound + ?
Program = Datalog\(^{LB}\) + Divide&Conquer + Kd-Tree
Program = Datalog\(^{LB}\) + ? + ?
Free Syntax and Platform Independence

Diagram

Rules

Tabular

Structured NL

LB-VM

RDBMS SQL

LogicBlox

Hadoop

GPU

CPU

FPGA
Provenance, Exchange, and Integration

- Joint work with Grigoris Karvounarakis at Penn (now at LB)
- Enables schema migration for use in ORM modeling, spreadsheets, and application upgrades.
- Enables declarative debugging and data forensics
- Enables GUI-as-View, workbook refresh and commit
- Enables integration with other apps.
- Requires support for existentially quantified head variables
- Elements of this research enables the Chase algorithm which can be used for query optimization and test data generation.
MoReBlox – Datalog as a Meta-language

- Joint work with Shan Shan Huang (now at LB) and others
- Morphing enables safe and rich modularity
- Requires higher-order, existentially quantified head variables, and code quoting
- Best Paper at GPCE ’04 and papers at AOSD ’07, ECOOP ’07, ’08, TOSEM ’08, PLDI ’08
  - “Generating AspectJ Programs with Meta-AspectJ”
  - “Expressive and Safe Static Reflection with MorphJ”
MoReBlox Use Cases

- Generics are the most frequently requested language enhancement by application teams
  - Product Lines
- Other uses:
  - Modularity
  - “Cube” operator
  - Units of Measure
  - Security & Trust
  - Parallelization
- The meta language is datalog and this feature is implemented using LB.
- See Shan Shan’s presentation
LogicBlox
Security and Trust
Mgmt Case Study
(showcase generics)
Security and Trust Management Case Study

- Joint work with Boon Thau Loo at Penn and Bill Marczak at Berkeley
- Bring 2 aspects of the system together to solve trust management:
  - Meta and generics (PL)
  - Distributed computing (systems)
- Unified many trust management systems in one framework using generics
- Papers CIDR ’09, SIGMOD ‘10:
  - “Declarative Reconfigurable Trust Management”
  - “SecureLB: Safe and Secure Distributed Data Processing”
LogicBlox
Partitioning & Parallelization
(will showcase generics)
Basic Idea

- Transform datalog program into $N$ datalog programs.
- Choose types (unary predicates) to partition.
  - Approach inspired by PLAPACK.
- Choose partitioning scheme for each type.
- Predicates with those types in key are split into fragments for each combination of type partitions.
- Fragments are placed on particular nodes.
- Clauses involving partitioned predicates are decomposed into sub-clauses.
- Sub-clauses are evaluated on specific nodes, according to a computed execution plan, which aims to optimize communication, time etc.
Example Partitioning

- **Store**
  - S1
  - S2
  - S3
  - S4
  - S5
  - S6
  - S7
  - S8

- **Item**
  - I1
  - I2
  - I3
  - I4
  - I5
  - I6
  - I7
  - I8

- **P(I,S,V)**

- **partition[Item]**
  - ip1
  - ip2

- **partition[Store]**
  - sp1
  - sp2
  - N1
  - N2

- **placement[P](P_i, P_s)**

- **item_ip1**
  - I1
  - I2
  - I3

- **store_sp1**
  - S1
  - S2
  - S3
  - S4

- **P[ip1,sp1](I,S,V)**

- **item_ip1**
  - I1
  - I2
  - I3

- **store_sp2**
  - S5
  - S6
  - S7
  - S8

- **P[ip1,sp2](I,S,V)**
Conforming Example

\[
\text{reg_sales}[w, p, l] = v \rightarrow \text{week}(w), \text{sku}(p), \text{store}(l), \text{int}[32](v).
\]

\[
\text{promo_sales}[w, p, l] = v \rightarrow \text{week}(w), \text{sku}(p), \text{store}(l), \text{int}[32](v).
\]

\[
\text{net_sales}[w, p, l] = v \rightarrow \text{week}(w), \text{sku}(p), \text{store}(l), \text{int}[32](v).
\]

\[
\text{net_sales}[w, p, l] = \text{reg_sales}[w, p, l] + \text{promo_sales}[w, p, l] \leftarrow.
\]
netsales_0[w, p, l] = reg_sales_0[w, p, l] + promo_sales_0[w, p, l] <-.

netsales_1[w, p, l] = reg_sales_1[w, p, l] + promo_sales_1[w, p, l] <-.

netsales_2[w, p, l] = reg_sales_2[w, p, l] + promo_sales_2[w, p, l] <-.

netsales_3[w, p, l] = reg_sales_3[w, p, l] + promo_sales_3[w, p, l] <-.
Conforming Benchmark 😊
Non-conforming Example

\[
\text{profit}[w,p,l]=v \rightarrow \text{week}(w), \text{sku}(p), \text{store}(l), \text{float}[32](v).
\]

\[
\text{profit}[w,p,l]= \text{reg\_sales}[w,p,l] \ast \text{price}[p] \ast \text{markup}[c] \leftarrow \text{skuCategory}[p]=c.
\]
profit_0[w,p,l] = \text{reg\_sales}_0[w,p,l] \times \text{price}_0[p] \times \text{markup}_0[c] \leftarrow \text{skuCategory}_0[p]=c.

profit_0[w,p,l] = \text{reg\_sales}_0[w,p,l] \times \text{price}_0[p] \times \text{markup}_1[c] \leftarrow \text{skuCategory}_0[p]=c.

profit_1[w,p,l] = \text{reg\_sales}_1[w,p,l] \times \text{price}_1[p] \times \text{markup}_0[c] \leftarrow \text{skuCategory}_1[p]=c.

profit_1[w,p,l] = \text{reg\_sales}_1[w,p,l] \times \text{price}_1[p] \times \text{markup}_1[c] \leftarrow \text{skuCategory}_1[p]=c.
Non-conforming Benchmark 😞
Research Roadmap

- Re-implement using MoReBlox
- Focus on shared memory scenario first
- Compare with approaches that parallelize the run-time.
- Use advanced optimizations on output of rewriter
  - Folding & Conditional Fact-bus expected to make a big difference
- How do we re-implement on top of a distributed implementation of datalog in the spirit of P2, Nlog, Dedalus, etc.?
Screaming Fast Program Analysis

- Joint work with Yannis Smaragdakis at UMass and Martin Bravenboer (now at LB)
- 10X faster than next best published result
- Papers at ISSTA ’09 and OOPSLA ‘09
  - “Exception Analysis and Points-To Analysis - Better Together”
  - “Strictly Declarative Specification of Sophisticated Points-to Analyses”
- See Yannis’ talk for more info
VarPointsTo(?var, ?obj) <-
    AssignObjectAllocation(?var, ?obj).

VarPointsTo(?to, ?obj) <-
    Assign(?from, ?to),
    VarPointsTo(?from, ?obj).

FieldPointsTo(?baseobj, ?field, ?obj) <-
    StoreField(?from, ?base, ?field),
    VarPointsTo(?base, ?baseobj),
    VarPointsTo(?from, ?obj).

VarPointsTo(?to, ?obj) <-
    LoadField(?base, ?field, ?to),
    VarPointsTo(?base, ?baseobj),
    FieldPointsTo(?baseobj, ?field, ?obj).
10 X Faster Than (previous) State-of-Art
Can Query Optimization Be Automated?

- The work on static program analyses has set the stage for automatic query optimization
- Joint work with Oege de Moor, Damien Sereni, Pavel Avgustinov, & Semmle
- PODS ‘08, SIGMOD ‘08:
  - “Type inference for datalog and its application to query optimisation”
  - “Adding magic to an optimising datalog compiler”
- 2 Semmle Patents Granted
- See Damien’s talk for more info
Can Query Optimization Be Automated?

- **Semmle Benchmark**
  - 150 complex queries
  - Not hand-written for LB Backend: queries run on SQL, LB and Semmle in-memory DB
  - Queries are generated using a large library of reusable abstractions
Semmle Benchmark

Query time (s)

- Optimised
- No optimisation

Time out without optimizations
Compile-time optimizations make these queries feasible.

Speedup: **8.8x** ignoring timeouts, **18.3x** with timeouts.

Time out without optimizations.