I will spend time motivating the problem

I will define what I mean by declarative programming
  - Key: RELATIONAL PL

I will define probabilistic programming as an application of optimization
  - Key: ESO

Goal: transfer motivation and intuition to motivate you to dig in
WORKING AT THE INTERSECTION OF 3 COMMUNITIES

PL

LogicBlox

DB

ANALYTICS (ML; OR; AI)
OUR DOMAIN: BUSINESS
WHO USES US?

All clients are in production. Primary use is in support of core and mission critical processes – majority of our clients’ revenues are impacted by high value applications.
April 11, 2014

Mr. Benny Kimelfeld
LogicBlox, Inc.
1900 Addison St, Ste 200
Berkeley, CA 94704

Dear Mr. Kimelfeld:


Your proposal was reviewed by a panel of experts in accordance with the criteria set forth in that announcement. I am pleased to inform you that your proposal has been selected for a potential award. A government agent will contact you in the near future to
“Fast becoming a leader in the RAMA market, Predictix moved from the Challengers quadrant in 2014 to the Leaders quadrant this year.”

Magic Quadrant for Merchandise Assortment Management Applications, July 2015
OUR SOLUTIONS
MODEL BASED SOLUTIONS THAT ARE CONFIGURED ITERATIVELY
COMPLEXITY IN BUSINESS

- Businesses are big systems with overwhelming complexity, variability, and detail
- To manage the complexity of business we build simplifying models:
  - Financial models
  - Data models
  - Supply chain network models
  - Consumer demand models
  - Price elasticity models
  - Process models
- Typical models are limited/small/coarse/lo-fidelity
  - Spreadsheets: typically 1000’s of little spreadsheets
  - Enterprise systems: you lose the forest for the trees
- The most useful models are the ones that can be built and maintained by domain experts who can improve them over time
DRIVING BUSINESS VALUE WITH PROFIT MODELING

\[
\text{profit} = \text{revenue} - \text{costs}
\]

Focused on increasing revenues and gross margin: 20% ROS

Focused on decreasing expense and cost of capital: 2% ROA
BUILDING A STATISTICAL MODEL OF DEMAND

- **Size of the problem**
  - 234 Billion weekly forecasts (500K skus X 9,000 stores X 52 weeks)
  - Many TB’s of data
  - 3,000 computing cores elastically provisioned

- **Forecast accuracy**
  - Measured 25% to 50% reduction in MAPE
  - The harder the problem the better the improvement
  - Measured reduction of bias in forecasts

- **Benefit Quantification**
  - $96M to $149M from inventory reductions alone
Problem size:
- 40,000 SKUs
- 100 stores
- Number of inventory variables alone is:
  \[ 40,000 \times 100 \times 435 \approx 1.7 \text{ billion variables} \]
- 100’s of cores provisioned elastically, each running modern mathematical solvers

45 pages for the entire model
- Number of constraints = \(~500\)
- Objective = Maximize Profit

Benefits quantification
- $30M of estimated inventory reductions
- Improved service levels
Starting inventory was too small for these 3 SKUS, leading to initial lost sales. Optimization engine had to order more to avoid this situation from repeating, thereby increasing ending inventory.
Bob Glass, the Merchant for door locks, recently received the “Hogs for Cogs” award from Ted Decker, Chief Merchant, for the assortment results (reduced COGS) done in the PDX AP tool. The PDX AP tool provided analysis for Bob to wring more "cost out" from the vendors than any other category! $XX.X Million benefits for next year, just in Door Locks.
ENTERPRISE COMPUTING
Enter the Cloud
### Top500 List - November 2011

R\text{max} and R\text{peak} values are in TFlops. For more details about other fields, check the TOP500 description.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>R\text{max} (TFlop/s)</th>
<th>R\text{peak} (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIII fx 2.0GHz, Tohu interconnect</td>
<td>705024</td>
<td>10510.0</td>
<td>11280.4</td>
<td>12860</td>
</tr>
<tr>
<td>2</td>
<td>National Supercomputing Center in Tianjin China</td>
<td>Tianhe-1A - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT</td>
<td>186368</td>
<td>2565.0</td>
<td>4701.0</td>
<td>4040</td>
</tr>
<tr>
<td>3</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Jaguar - Cray XT5-HE Opteron 6-core 2.6 GHz Cray Inc.</td>
<td>224162</td>
<td>1759.0</td>
<td>2331.0</td>
<td>6950</td>
</tr>
<tr>
<td>4</td>
<td>National Supercomputing Center in Shenzhen (NSCS) China</td>
<td>Calcul Canada/Calcul Québec/Université de Sherbrooke Canada</td>
<td>Rackable C2112-4G3 Cluster, Opteron 12 Core 2.10 GHz, Infiniband QDR SGi</td>
<td>37726</td>
<td>240.3</td>
<td>316.9</td>
</tr>
<tr>
<td>5</td>
<td>GSIC Center, Tokyo institute of Technology Japan</td>
<td>Amazon Web Services United States</td>
<td>Amazon EC2 Cluster Compute Instances - Amazon EC2 Cluster, Xeon 8C 2.60GHz, 10G Ethernet Self-made</td>
<td>17624</td>
<td>240.1</td>
<td>354.1</td>
</tr>
<tr>
<td>6</td>
<td>DOE/NNSA/LANL/SNL United States</td>
<td>Grand Equipement National de Calcul Intensif - Centre Informatique National de l’Enseignement Supérieur (GECI-CINES) France</td>
<td>Jade - SGI Altix ICE 8200EX, Xeon E5472 3.0/X5660 2.8 GHz SGI</td>
<td>23040</td>
<td>237.8</td>
<td>267.9</td>
</tr>
<tr>
<td>44</td>
<td>KTH - Royal Institute of Technology Sweden</td>
<td>Lindgren - Cray XE6, Opteron 12 Core 2.10 GHz, Custom Cray Inc.</td>
<td>36384</td>
<td>237.2</td>
<td>305.6</td>
<td></td>
</tr>
</tbody>
</table>
THE CLOUD HAS FREED US FROM TYRANNY

- Cloud has been a catalyst for breathtaking change and innovation in infrastructure
- Infrastructure evolving at very rapid rate
- The on-premise stack is no longer sacrosanct
- Several communities being formed
  - NoSQL – scalability at the expense of queries and ACID support
  - In-memory – speed in exchange for full mem hierarchy support
  - Column Store – mem hierarchy optimization, bad for TRX’s
  - NewSQL – elasticity for transactional workloads
- Some are trying to create a new category of hybrid database:
  - OLTP+ OLAP = OLTAP (but still not other interesting categories)
  - Use in-memory “feature” to help achieve performance goals – but expensive.
  - Combine column and row store technology under the covers – hairball on the inside
  - Programming model still a hairball – e.g. HANA: SQLscript, SQL, MDX, RDL, R, ABAP, etc.
IMPLICATION OF THE HAIRBALL – MORE TIME & $$$, LESS VALUE

- Harder to develop
- Harder to debug
- Harder to scale
- Harder to achieve quality objectives
- Harder to operate
- Harder to tailor
- Harder to evolve
- Harder to support
- Harder to use
  - Different UI’s for different parts
  - Data doesn’t flow naturally and in (near) real-time
  - Impossible for user to self serve meaningful changes
- Harder to fund
  - DB and app server software licenses,
  - Hardware or cloud fees
  - More and different types of developers for each tech silo
PEP TALK

We are here to invent the future
IN JUNE 2007 THE SMARTPHONE UNIFIED CONSUMER DEVICES

THE WHOLE IS GREATER THAN THE SUM OF THE PARTS
IT’S JUNE 2015 – CAN WE BUILD A “SMART” DATABASE?

CAN WE BUILD A “SMART” DATABASE THAT SIMPLIFIES THE MESS AND ENABLES APPLICATIONS THAT CAN’T BE WRITTEN ON DIFFERENT SPECIALIZED DB?
DECLARATIVE PROGRAMMING
POPULAR DECLARATIVE LANGUAGES

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

\[ A1 = B1 + C1 \]

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

```
Select * from Sales, Returns where ...
```

OLAP & Reporting tools: KPI editors

```
Netsales = Sales - Returns.
```

Also: HTML, XML, Mathematical Optimization languages
SIMPLE, DECLARATIVE, AND UNIFORM LANGUAGE – LogiQL

- Based on datalog with highly normalized schema (6NF)
- Usability
- Expressivity & Safety
- Performance
SIMPLE, DECLARATIVE, AND UNIFORM LANGUAGE – LogiQL

- **Usability**
  - Clear and easy to understand semantics:
    - 6NF -> Sets & two-valued logic (No NULLS and no duplicate rows)
  - Semantics independent of a specific evaluation strategy
  - Declarative by birth and first order by the grace of god
    - So is Excel – passing functions to functions is too hard for most
    - Order considered harmful – unlike functional model where everything is built with Lists/Cons()
  - Skins – give people the syntax they like (including diagrams)
  - Automatic incremental computation (support live programming tools)
Expressivity & Safety

- Turing completeness considered harmful
  - But you can have it if you have permission
- Guaranteed expressivity with “controllable” power e.g.
  - Stratified recursion and order capture PTIME
  - Existential Second Order (ESO) Quantifier gives us NP
- Language and programming model give us efficient and always correct ACID
Performance

- Memory hierarchy friendly
- Parallelizable fragment
- Semantics independent of a specific evaluator/solver
- Jujutsu – let’s use the power of the machine against itself. Declarativity makes it possible to use the power of the cloud at compile time to go faster
CORE LogiQL

- **Derivation Rules**
  - Specify how things are calculated
  - Support for arithmetic and aggregations
  - Added support for constructors (skolems) and recursion through negation and aggregation
  - Standard semantics when program is stratified, “staged” partial fixpoint semantics otherwise

- **Event-Condition-Action Rules** ← Key Language Feature (NOT TODAY)
  - Specify state change by introducing limited notion of trxn time (@now and @prev)
  - Used for edits & triggers and to model collaboration and workflow
  - All materialized views are maintained incrementally

- **Integrity Constraints** ← Key Language Feature
  - TGD’s and EGD’s
  - Specify the set of possible states – (with one exception) totally passive and don’t repair
  - Types and schema
  - With ESO, it generalizes all problems that can be thought of as instance (aka model) finding
    - Mathematical programming
    - Probabilistic programming
    - Verification
    - Test data generation
DERIVATION RULES

LogiQL

\[
\text{grandparent}(x, y) \leftarrow \text{parent}(x, t), \text{parent}(t, y).
\]

// basic join (boolean matmult)

\[
\text{ancestor}(x, y) \leftarrow \text{parent}(x, y); \text{parent}(x, t), \text{ancestor}(t, y).
\]

// basic recursion

\[
\text{ancestor}(x, y) \leftarrow \text{parent}(x, y); \text{ancestor}(x, t), \text{parent}(x, y).
\]

// same as above

\[
\text{ancestor}(x, y) \leftarrow \text{parent}(x, t), \text{ancestor}(t, y); \text{parent}(x, y).
\]

// same as above

// alternative “functional” notation when abs(x, y), abs(x, y') -> y = y'

\[
\text{abs}(x, x) \leftarrow x \geq 0.
\]

\[
\text{abs}(x, -x) \leftarrow x < 0.
\]

\[
\text{abs}[x] = x \leftarrow x \geq 0.
\]

\[
\text{abs}[x] = -x \leftarrow x < 0.
\]

// value invention

\[
\text{phone}[p] = t, \text{telephone}(t) \leftarrow \text{person}(p)
\]

// t is Existentially quantified

// phone[p] is constructor. Implemented via skolemization (i.e hash cons)
DERIVATION RULES – GRAPH QUERIES - CLIQUES

LogiQL – 3 Clique

cliques(a,b,c)
<- edge(a,b),
edge(a,c),
edge(b,c),
a < b < c.

LogiQL – 4 Clique

cliques(a,b,c,d)
<- edge(a,b),
edge(a,c),
edge(a,d),
edge(b,c),
edge(b,d),
edge(c,d),
a < b < c < d.
CLIQUE BENCHMARK

Graph with 100K links:
LB is **33X** faster than Redshift
LB is **57X** faster than HANA

Graph with 10M links:
LB is **227X** faster than RedShift
HANA **fails** for graphs with more than 100K links
NEW MULTI-WAY RA OPERATOR

- Leapfrog Triejoin: A Simple Worst-Case Optimal Join Algorithm. T. Veldhuizen
  ICDT 2014 (best newcomer)

- Really a “select-project-union-difference-join” algorithm, i.e. $\exists 1$ queries

- Proven worst-case optimal in the sense of (Ngo, Porat, Re, Rudra, PODS’12 – Best Paper)
Why are simultaneous joins more efficient than pairwise joins?

- Suppose A has 1M records for Jan and 1M for Feb, B has 1M for Feb and 1M for Mar, etc.
  - Any pairwise join generates 500K results.
  - Simultaneous join → no results, examine < 10 records.
- Asymptotically better for complex queries
DERIVATION RULES – GRAPH QUERIES

Context sensitive program analysis – querying the call graph

- “Exception Analysis and Points-To Analysis - Better Together” (ISSTA'09)
- “Strictly Declarative Specification of Sophisticated Points-to Analyses” (OOPSLA'09)
- “Pick Your Contexts Well – Understanding Object-Sensitivity” (POPL’11)
- “Efficient and Effective Handling of Exceptions in Java Points-To Analysis” (CC’13)
- Hybrid Context Sensitivity for Points-To Analysis” (PLDI’13)
- “Set-based pre-processing for points-to analysis” (OOPSLA’13)
DERIVATION RULES – RECURRENCE THROUGH NEGATION

LogiQL – Computing Pi

\[
\begin{align*}
\pi[] &= 3.0f - !pp[]=_. \\
pp[] &= \pi[] + \sin[\pi[]]. \\
\pi[] &= pp[].
\end{align*}
\]

LogiQL - PageRank

\[
\begin{align*}
d[] &= 0.85f. \\
tolerance[] &= 0.01f.
\end{align*}
\]

\[
\begin{align*}
pr[p] &= r & \leftarrow r &= 1.0f / \text{node_count[]}[, \text{node}(p), !pr[p] = _]. \\
pr[p] &= r & \leftarrow r &= (1.0f - d[]) + (d[] \ast \text{sum}[p]), \ \text{abs}[r - pr[p]] > \text{tolerance[]}[]. \\
pr[p] &= pr[p] & \leftarrow r &= (1.0f - d[]) + (d[] \ast \text{sum}[p]), !\text{abs}[r - pr[p]] > \text{tolerance[]}[]. \\
pr[p] &= pr[p] & \leftarrow !\text{sum}[p] = _.
\end{align*}
\]

\[
\begin{align*}
\text{sum}[n] &= t & \leftarrow \text{agg}<<t = \text{total}(r)>> \ \text{edge}(p, n), \ r &= \text{pr}[p] / \text{out_count}[p].
\end{align*}
\]
CORE LogiQL

- **Derivation Rules**
  - Specify how things are calculated
  - Support for arithmetic and aggregations
  - Added support for constructors (skolems) and recursion through negation and aggregation
  - Standard semantics when program is stratified, “staged” partial fixpoint semantics otherwise

- **Event-Condition-Action Rules** ← Key Language Feature (NOT TODAY)
  - Specify state change by introducing limited notion of trxn time (@now and @prev)
  - Used for edits & triggers and to model collaboration and workflow
  - All materialized views are maintained incrementally

- **Integrity Constraints** ← Key Language Feature
  - TGD’s and EGD’s
  - Specify the set of possible states – (with one exception) totally passive and don’t repair
  - Types and schema
  - With ESO, it generalizes all problems that can be thought of as instance (aka model) finding
    - Mathematical programming
    - Probabilistic programming
    - Verification
    - Test data generation
CORE LogiQL

- **Derivation Rules**
  - Specify how things are calculated
  - Support for arithmetic and aggregations
  - Added support for constructors (skolems) and recursion through negation and aggregation
  - Standard semantics when program is stratified, “staged” partial fixpoint semantics otherwise

- **Event-Condition-Action Rules** ← Key Language Feature (NOT TODAY)
  - Specify state change by introducing limited notion of trxn time (@now and @prev)
  - Used for edits & triggers and to model collaboration and workflow
  - All materialized views are maintained incrementally

- **Integrity Constraints** ← Key Language Feature
  - TGD’s and EGD’s
  - Specify the set of possible states – (with one exception) totally passive and don’t repair
  - Types and schema
  - With ESO, it generalizes all problems that can be thought of as instance (aka model) finding
    - Mathematical programming
    - Probabilistic programming
    - Verification
    - Test data generation
INTEGRITY CONSTRAINTS

// IC’s expressed as derivation rules that derive FALSE
student(x) -> person(x)                // FALSE <- student(x), !person(x).

age(p, a) -> person(p), int(a)        // IC1(p,a) <- age(p,a), !(person(p),int(a)).
age(x, y), age(x, y’) -> y = y’         // IC2(x,y) <- age(x,y), age(x,y’), y != y’.
age[p] = a -> person(p), int(a)       // syntactic sugar for predicates with FD’s

parent(x, y) ->
    person(x), person(y)
2^36 possibilities: ~64 billion

parent2(x, y) ->
    person(x), person(y),
x != y
2^30 possibilities: ~1 billion

parent3(x, y) ->
    person(x), person(y),
age[x] > age[y].
2^15 possibilities: ~32 thousand
UNARY IC’S AS TYPES AND DERIVATION RULES

IC’s (upper bound)

\[
\begin{align*}
\text{person}(x) & \rightarrow . \\
\text{man}(x) & \rightarrow \text{person}(x) . \\
\text{parent}(x, y) & \rightarrow \text{person}(x), \text{person}(y) \\
\text{grandpa}(x, y) & \rightarrow \text{person}(x), \text{man}(y).
\end{align*}
\]

Rules (lower bound)

\[
\begin{align*}
\text{grandpa}(x, y) & \leftarrow \text{parent}(x, t), \\
& \text{parent}(t, y), \\
& \text{man}(y).
\end{align*}
\]

\[
\begin{align*}
\text{person}(x), \text{man}(y) & \leftarrow \text{grandpa}(x, y) \text{ and } \\
& \text{grandpa}(x, y) \leftarrow \text{parent}(x, t), \text{parent}(t, y), \text{man}(y).
\end{align*}
\]

\[
\begin{align*}
\text{person}(x), \text{man}(y) & \leftarrow \text{parent}(x, t), \text{parent}(t, y), \text{man}(y).
\end{align*}
\]

\[
\begin{align*}
[\text{person}(x), \text{man}(y)] \quad \text{CONTAINS/} \\
& \text{IS LOGICALLY IMPLIED BY} \\
& [\text{parent}(x, t), \text{parent}(t, y), \text{man}(y)].
\end{align*}
\]
**UNARY IC’S AS TYPES AND DERIVATION RULES**

**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).

**Unnecessary Check**
- grandpa (person x man)
UNARY IC’S AS TYPES AND DERIVATION RULES

IC’s

\[ \text{person}(x) \rightarrow . \]
\[ \text{man}(x) \rightarrow \text{person}(x). \]
\[ \text{parent}(x, y) \rightarrow \text{person}(x), \text{person}(y) \]
\[ \text{grandpa}(x, y) \rightarrow \text{person}(x), \text{man}(y). \]

Rules

\[ \text{grandpa}(x, y) \leftarrow \text{parent}(x, t), \]
\[ \text{parent}(t, y). \]

ERROR!!!
ESO PROBABILISTIC PROGRAMMING
ESO FOR SOLVING AND OPTIMIZATION

- First Example: Graph coloring

\[ \exists Col \ \forall XY \ Col(X,Y) \rightarrow node(X) \land color(Y) \land \]  
\[ \forall X \ \exists Y \ node(X) \rightarrow Col(X,Y) \land \]  
\[ \forall XYZ \ Col(X,Y) \land Col(X,Z) \rightarrow Y = Z \land \]  
\[ \forall XYZ \ X \neq Y \land Col(X,Z) \land Col(Y,Z) \rightarrow \neg edge(X,Y). \] (1.2)

- Variable predicates
  - The value of variable predicates is unknown in advance and inferred by LB so that:
    - Any integrity constraints mentioning those variable predicates hold,
    - We pick one of the possible worlds that satisfy the constraints by maximizing/minimizing an objective function
  - Variable predicates are often functional, i.e. have domain (key) and codomain (value)

- Dual but consistent semantics:
  - As integrity constraints the system checks that constraints are not violated
  - `lang:solver:variable('Col)` as syntactic sugar for second order existential quantifier. It tells the system that the predicate is to be treated as a variable predicate – the rest is up to the platform and hidden from the modeler (second order semantics, first order syntax)
SECOND EXAMPLE – ESO FOR ASSORTMENT OPTIMIZATION

- Similar to text book smugglers problem
- A small assortment has three kinds of items
  - Whiskey
  - Cigarettes
  - Perfume

(I don’t write the textbooks)

- We want to decide how much of each item to stock in order to maximize profit
- Must stock a minimum and maximum amount of each item
- Must not use more shelf space than is available
IS WRITTEN AS THIS LINEAR PROGRAM

PRODUCT(x), PRODUCT:name(x:n) -> string(n).

spacePerProd[p]=v -> PRODUCT(p), float(v).
profitPerProd[p]=v -> PRODUCT(p), float(v).

minStock[p]=v -> PRODUCT(p), float(v).
maxStock[p]=v -> PRODUCT(p), float(v).
maxShelf[]=v -> float[64](v).

// Constraints
PRODUCT(p)-> Stock[p] >= minStock[p].
PRODUCT(p)-> Stock[p] <= maxStock[p].

totalShelf[]=v -> float(v).
totalShelf[]+= Stock[p] * spacePerProd[p] <-.
          -> totalShelf[] <= maxShelf[].

// Objective
TotalProfit[]=v -> float(v).
TotalProfit[] += Stock[p]*profitPerProd[p] <-.

// Solve for
Stock[p]=v -> PRODUCT(p), float(v).
PRODUCT(p)-> Stock[p]=_.
lang:solve:variable(`Stock).
lang:solve:max(`TotalProfit).
IS WRITTEN AS THIS INTEGER PROGRAM

PRODUCT(x), PRODUCT:name(x:n) -> string(n).

spacePerProd[p]=v -> PRODUCT(p), float(v).
profitPerProd[p]=v -> PRODUCT(p), float(v).

minStock[p]=v -> PRODUCT(p), float(v).
maxStock[p]=v -> PRODUCT(p), float(v).
maxShelf[] =v -> float[64](v).

// Constraints
PRODUCT(p) -> Stock[p] >= minStock[p].
PRODUCT(p) -> Stock[p] <= maxStock[p].

totalShelf[] =v -> float(v).
totalShelf[] += Stock[p] * spacePerProd[p] <= totalShelf[] <= maxShelf[].

// Objective
TotalProfit[] =v -> float(v).
TotalProfit[] += Stock[p]*profitPerProd[p] <=-

// Solve for
Stock[p]=v -> PRODUCT(p), int(v). ← A one line change
PRODUCT(p) -> Stock[p]=_.
lang:solve:variable(`Stock).
lang:solve:max(`TotalProfit).
THIRD EXAMPLE: SUPPLY CHAIN OPTIMIZATION
SUPPLY CHAIN NETWORK

Network Flow Model
- Each node represents SKU-location-time period
- Each arc represents merchandise flowing:
  - Red: demand
  - Grey: inventory

Our objective is to find flows that meet demand, maximize profit, and satisfy business constraints, eg:
- various network structures
- various modes of transportation
- min & max order quantities
- space constraints
- perishability
- display stock
- demand prioritization
- back-orders, etc.

Complete real world models ~500 integrity constraints
EXAMPLE – BASIC CONSTRAINTS

- Two variable predicates (dc_inventory and ship_from_vendor) indicating amount of a product available at time t, and the amount of product that should be ordered to arrive at a distribution center on time t.

- LogiQL integrity constraints recognized by the system as part of an optimization problem (e.g., any values picked for inventories or shipments must not be negative).

- Model can mix LogiQL derivation rules and integrity constraints for mathematical programming problem specification – has_shipment constrains the domain of ship_from_vendor

\[
\text{dc}_{\text{inventory}}[j,i,t]=v \rightarrow \text{dc}(j), \text{sku}(i), \text{time}(t), \text{float}[64](v). \quad \text{// IC (static)}
\]

\[
\text{dc}(j), \text{sku}(i), \text{time}(t), \text{dc}_{\text{inventory}}[j,i,t]=v \rightarrow v \geq 0. \quad \text{// IC}
\]

\[
\text{ship}_{\text{from_vendor}}[j,i,t]=v \rightarrow \text{dc}(j), \text{sku}(i), \text{time}(t), \text{float}[64](v). \quad \text{// IC (static)}
\]

\[
\text{has}_{\text{shipment}}(j,i,t), \text{ship}_{\text{from_vendor}}[j,i,t]=v \rightarrow v \geq 0. \quad \text{// IC}
\]

\[
\text{has}_{\text{shipment}}(j,i,t) \leftarrow \text{is}_{\text{ordering}} \text{dc}(j,i), \text{mod}[\text{time}_{\text{id}}[t], 7]=1. \quad \text{// Derivation rule}
\]
Aggregation is used to represent sums of unknown values.

Integrity constraints: for any time \( t \), \( \text{sku} \, i \) and distribution center \( j \), total amount shipped from \( j \) to all stores on day \( t \) must not exceed some maximum quantity per-sku stored in predicate \( \text{max\_quantity} \).

\[
\text{total\_to\_stores}[j, i, t] = v \iff \text{agg}\langle v = \text{total}(z) \rangle
\]

\[
\text{has\_shipment\_to\_store}(j, i, s, t\_arrive),
\text{ship\_to\_store}[j, i, s, t\_arrive] = z.
\]

\[
dc(j), \text{sku}(i), \text{time}(t), \text{total\_to\_stores}[j, i, t] = v \implies
v \leq \text{max\_quantity}[i].
\]
GROUNDING

- Translating constraints over variable predicates into a representation that can be consumed by the solvers

  parent3(x, y) -> person(x), person(y), age[x] > age[y].

  $2^{15}$ possibilities: represented with 15 boolean vars

- Use LogiQL derivation rules to preform grounding efficiently and at scale
  - Automatically compile constraints to LogiQL derivation rules
- The grounding program is automatically synthesized by the LogiQL compiler from a set of constraints
- The computed predicates coefficient, bounds (etc) are the exact, low-level representations needed by most modern/efficient LP, IP solvers
- LogicBlox maintenance algorithms automatically maintain the problem instance thus represented in a consistent state and allow incremental evaluation
  - If `max_shipment` changed from 2 to 5 for one `sku` out of thousands, only very few rows will be recomputed, significantly amortizing cost of grounding in interactive/persistent applications
constraint_column(col), ship_from_vendor ctor[i,j,t]=col <- has_shipment(i,j,t).

constraint_column(col), ship_from_vendor ctor[i,j,t]=col <- has_shipment(i,j,t),
max_shipment[i]=_ // only in context where max_shipment[i] is defined

constraint_row(row), row ctor_1[i,j,t]=row <- has_shipment(i,j,t).

constraint_row(row), row ctor_2[i,j,t]=row <- has_shipment(i,j,t),
max_shipment[i]=_.

lower_bound[row]=0.0 <- row ctor_1[i,j,t]=row.


STATISTICAL INFERENCE
BUILT IN SCALABLE MACHINE LEARNING

- **Regression:** predict a number
  - Linear regression with VIF
  - LASSO regression
  - Multi-time series prediction
  - Nonparametric regression
  - Mixture of experts
  - Factorization machines

- **Clustering:** find natural groups
  - K-means
  - Spectral clustering
  - Mean shift clustering

- **Search:** find similar object
  - Nearest neighbors
  - Orthogonal range search
  - Dynamic time warping search

- **Density Estimation:** find likelihood of objects
  - Histograms
  - Kernel density estimation

- **Classification:** predict a category
  - Naive Bayes classifier
  - Non-parametric Bayes classifier
  - K-nearest neighbor classifier
  - Support vector machine
  - Decision tree
  - Hidden Markov model

- **Dimension Reduction:** combine features
  - Singular value decomposition
  - Maximum variance unfolding
  - Non-negative matrix factorization
  - Kernel principal component analysis
  - (Ensemble) Singular value decomposition
  - Grouse
  - Random projections
  - Tensor factorization
  - PARFAC/CANDECOMP

Equivalent in scalability and capability to Skytree machine learning (LogicBlox supported academic precursor)
BUILT IN SCALABLE MACHINE LEARNING (good start)

- Good start, but not very satisfying
- Machine learning methods are extra-logical
- Statistical model is described “imperatively”.
  - Transform data
  - Feature extraction
  - Call machine learning method
  - Repeat
- Statistical model doesn't directly reflect the domain being modeled

Observation:
- Machine learning methods can be defined as optimization problems where we are trying to minimize the distance between data and some compressed representation of the data

Key idea:
- Use soft or weighted integrity constraints to model the domain of interest
- Integrity constraints are features
PROBABILISTIC PROGRAMS AS WEIGHTED IC’s

Given: Relation \text{friends(person,person)}, partial \text{votesFor(person,person)}
Predict: Remaining \text{votesFor(person,person)} using weighted rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>\text{Rule}</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>votesFor(A,C) \land friends(A,B) \rightarrow votesFor(B,C)</td>
<td>0.3</td>
</tr>
<tr>
<td>R2</td>
<td>votesFor(A,C) \land C \neq D \rightarrow \neg votesFor(A,D)</td>
<td>5.0</td>
</tr>
</tbody>
</table>

- **Weights**: cost for IC violation
- **Semantics**: - a distribution over possible worlds
  - Each possible world \text{pw} has Cost(\text{pw})
  - Different cost models give different semantics (e.g. MLN’s and PSL’s)
  - Probability(\text{pw}) is proportional to Cost(\text{pw})
- **Inference**
  - Maximum a posteriori (MAP) inference is picking most likely world, i.e. the one with the lowest cost. This can be formulated as a mathematical programming problem.
  - Marginal inference is aggregating over subset of possible worlds that meet criteria (can be done via sampling and [approximate] lifted inference)
- **Weight learning** can be formulated as a mathematical programming problem as well
### PROBABILISTIC PROGRAMS AS WEIGHTED IC’s

Given: Relation `friends(person,person)`, partial `votesFor(person,person)`

Predict: Remaining `votesFor(person,person)` using weighted rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1 ( \text{votesFor}(A,C) \land \text{friends}(A,B) \rightarrow \text{votesFor}(B,C) )</td>
<td>0.3</td>
</tr>
<tr>
<td>Rule 2 ( \text{votesFor}(A,C) \land C \neq D \rightarrow \neg\text{votesFor}(A,D) )</td>
<td>5.0</td>
</tr>
</tbody>
</table>

Next, we will represent in LogiQL (with ESO) for 2 interpretations:

- **Probabilistic Soft Logic (PSL)** [Broecheler et al. 10]
- **Markov Logic Networks (MLN)** [Richardson & Domingos 06]
PROBABILISTIC PROGRAMS -> MATHEMATICAL PROGRAMS

Given: Relation \texttt{friends(person,person)}, partial \texttt{votesFor(person,person)}
Predict: Remaining \texttt{votesFor(person,person)} using weighted rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1 ( \text{votesFor}(A,C) \land \text{friends}(A,B) \rightarrow \text{votesFor}(B,C) )</td>
<td>0.3</td>
</tr>
<tr>
<td>R2 ( \text{votesFor}(A,C) \land C \neq D \rightarrow \neg \text{votesFor}(A,D) )</td>
<td>5.0</td>
</tr>
</tbody>
</table>

\textit{EDBs}

- \texttt{person(A)} \rightarrow \texttt{string(A)}.
- \texttt{friends[A,B]=t} \rightarrow \texttt{person(A), person(B), type(t)}.
- \texttt{votesFor\_given[A,C]=t} \rightarrow \texttt{person(A), person(B), type(t)}.

\textit{IDBs}

- \texttt{votesFor\_given[A,C]=t1, votesFor[A,C]=t2} \rightarrow t1=t2.
- \texttt{votesFor[A,C]=t} \rightarrow \texttt{person(A), person(B), type(t), 0<=t<=1}.
- \texttt{EQ[C,D] = t} \rightarrow \texttt{person(C), person(D)}.

- \texttt{EQ[C,D] = 1} \leftarrow \texttt{person(C), person(D), C=D}.
- \texttt{EQ[C,D] = 0} \leftarrow \texttt{person(C), person(D), C\neq D}.

MLN: \textit{type} = \textit{int}
PSL: \textit{type} = \textit{float}

\begin{itemize}
\item lang:solver: variable(`\texttt{votesFor}`)
\end{itemize}
Given: Relation \texttt{friends(person,person)}, partial \texttt{votesFor(person,person)}

Predict: Remaining \texttt{votesFor(person,person)} using weighted rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1: \texttt{votesFor(A,C) \land friends(A,B) \rightarrow votesFor(B,C)}</td>
<td>0.3</td>
</tr>
<tr>
<td>R2: \texttt{votesFor(A,C) \land C\neq D \rightarrow \neg votesFor(A,D)}</td>
<td>5.0</td>
</tr>
</tbody>
</table>

\texttt{Rule-satisfaction variables}

\begin{align*}
\text{R1}[A,B,C] &= a & \rightarrow \text{person}(A), \text{person}(B), \text{person}(C), \texttt{type}(a). \\
\text{R2}[A,C,D] &= a & \rightarrow \text{person}(A), \text{person}(C), \text{person}(D), \texttt{type}(a).
\end{align*}

\texttt{a} = \texttt{distance to satisfaction (PSL) / violation (MLN)}

\begin{align*}
\text{PSL} & \quad \text{R1}[A,B,C]=a, \text{votesFor}[A,C]=d, \text{friends}[A,B]=e, \text{votesFor}[B,C]=f \rightarrow a = \max[0, (d+e-1)-f]. \\
& \quad \text{R2}[A,C,D]=a, \text{votesFor}[A,C]=d, \text{EQ}[C,D]=e, \text{votesFor}[A,D]=f \rightarrow a = \max[0, ((d+1-e)-1)-(1-f)].
\end{align*}

\begin{align*}
\text{MLN (logic as ilp)} & \quad \text{R1}^\wedge[A,B,C]=b, \text{votesFor}[A,C]=d, \text{friends}[A,B]=e \rightarrow b \leq d, b \leq e, b \geq d+e-1. \quad \wedge \\
& \quad \text{R1}[A,B,C]=a, \text{R1}^\wedge[A,B,C]=b, \text{votesFor}[B,C]=e \rightarrow a \leq b, a \leq (1-e), a \geq b+(1-e)-1. \quad \wedge \neg \\
& \quad \text{R2}^\wedge[A,C,D]=b, \text{votesFor}[A,C]=d, \text{EQ}[C,D]=e \rightarrow b \leq d, b \leq (1-e), b \geq d+(1-e)-1. \quad \wedge \neg \\
& \quad \text{R2}[A,C,D]=a, \text{R2}^\wedge[A,B,C]=b, \text{votesFor}[B,C]=e \rightarrow a \leq b, a \leq e, a \geq b+e-1. \quad \wedge
\end{align*}
PROBABILISTIC PROGRAMS -> MATHEMATICAL PROGRAMS

Given: Relation \text{friends(person, person)}, partial \text{votesFor(person, person)}

Predict: Remaining \text{votesFor(person, person)} using weighted rules:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0.3</td>
</tr>
<tr>
<td>R2</td>
<td>5.0</td>
</tr>
</tbody>
</table>

\begin{align*}
\text{votesFor}(A, C) \land \text{friends}(A, B) \rightarrow & \text{votesFor}(B, C) \\
\text{votesFor}(A, C) \land C \neq D \rightarrow & \neg \text{votesFor}(A, D)
\end{align*}

**Weighted sum of rule distances/satisfactions**

\begin{align*}
\text{R1	extunderscore sum}[] & += 0.3 \times \text{R1}[A, B, C]. \\
\text{R2	extunderscore sum}[] & += 5.0 \times \text{R2}[A, C, D]. \\
\text{objective}[] & = \text{R1	extunderscore sum}[] + \text{R2	extunderscore sum}[] < . \\
\text{lang:} & \text{solver:minimal(`objective).}
\end{align*}

**Done!**

Weight learning can be formulated as a mathematical programming problem as well.
STILL NOT FULLY SATISFYING

- Minor: how do we do regression? Hybrid MLNs?

- Should the models be invariant under logical equivalence (e.g., a rule subsumed by another is redundant)?
  - A \rightarrow B, C & A \rightarrow B should be the same as A \rightarrow B, C
  - A \rightarrow B, C should be the same as A \rightarrow B, A \rightarrow C
  - A \rightarrow B, A \rightarrow B should be the same as A \rightarrow B.
  - MLN [Domingos+], PSL [Bröcheler+10]: factor per rule grounding
WHAT’S REAL AND WHAT’S ASPIRATIONAL?

In production today
- Linear and Integer programming
- Incremental re-optimization where appropriate
- Ability to invoke different solvers
  - Gurobi by default
  - CPLEX and other solvers optional

Futures
- More OR Solvers – e.g. Quadratic programming
- SAT & SMT solvers – for verification use cases and for application uses cases (tax laws)
- Solvers based on Local Search/Meta-heuristics
- Domain Specific Solvers for test data generation and testing use cases
  - “Scalable Test Data Generation from Multidimensional Models”, Emina Torlak, FSE 2012 (Distinguished Paper)
  - “Scalable Automatic Test Data Generation from Modeling Diagrams”, Yannis Smaragdakis, ASE 2007 (Best Paper)
- Lazy grounding
  - Specific techniques such as column generation and model decomposition
- A framework to simplify solver integration
  - “Painless Programming Combining Reduction and Search”, Tim Sheard, ICFP 2012
- Integrating simulation and optimization
  - Automatically derive simulators from network layouts implied by constraints
  - Use LogiQL and LB to run very large scale simulation
  - Store detailed simulation traces in database
THANK YOU.
PHILOSOPHY: BRAINS BEFORE BRAWN

- **Algorithms**
  - New worst-case optimal join algorithm (patent filed)
  - Adaptive domain decomposition for parallelization
  - Incremental maintenance proportional to trace edit distance (patent filed)
  - Salient sampling for accurate query cost estimation (patent filed)
  - Transaction repair for lock-free, scalable transaction processing with full serializability (patent filed)

- **Data structures**
  - Compression: close to info-theoretic limit in some cases
  - I/O minimization, cache consciousness
  - Persistent data structures for strongest consistency guarantees, full serializability, built-in auditability, scalable distribution (patent application pending)

- **Unified, declarative programming model**
  - For complete freedom to evaluate the model in the most efficient way possible.

- **Brute Force**
  - Distribution across thousands of cores; in-memory; GPUs
A SMART JOIN ALGORITHM

Leapfrog Triejoin: A Simple, Worst-Case Optimal Join Algorithm
T. Veldhuizen  ICDT 2014

Really a “select-project-union-difference-join” algorithm, i.e. \( \exists 1 \) queries

Proven worst-case optimal in the sense of (Ngo, Porat, Re, Rudra, PODS’12 – Best Paper)
INTUITION BEHIND LFTJ

- Why are simultaneous joins more efficient than pairwise joins?

Suppose A has 1M records for Jan and 1M for Feb, B has 1M for Feb and 1M for Mar, etc. Any pairwise join generates 500K results. Simultaneous join → no results, examine < 10 records.
A SMARTER WAY TO PARALLELIZE QUERIES

- Paper in progress
- Common practice: Fixed domain decomposition (sharding), having $N$ jobs for $N$ cpus $\Rightarrow$ poor load balancing
- LogicBlox instead:
  - For each query, a custom decomposition in which each subdomain will require the same anticipated amount of work
  - Partitioning policy is automatic, dynamic, and adaptive
  - Have $k*N$ jobs, where $k$ is large:
    - scheduling/packing are NP-complete problems
    - fitting boxes into a shipping container $\Rightarrow$ hard
    - pouring sand into a dumptruck $\Rightarrow$ dead easy
    - sand, not boxes
SMARTER INCREMENTAL MAINTENANCE

- Replaced our implementation of classic Count and DRed algorithms [Gupta+ 93]
  - Count for non-recursive predicates
  - DRed ("delete and rederive") for recursive predicates
- Guarantees that the work done is proportional to the trace edit distance between the before and after
SMARTER TRANSACTIONS PROCESSING

- World first: lock-free and scalable transaction processing with full serializability.
- Combines:
  - Leapfrog Triejoin algorithm
  - Incremental maintenance of Leapfrog Triejoin
  - Object Store
  - Versioned data structures
  - Transaction repair circuits
- Possible because of simplified and declarative programming model
SMARTER META DATA MANAGEMENT

- Declarative and incremental maintenance of program state using tiny datalog engine
- Necessary for “live” end user programming and for handling large schemas
- Meta-rules operate on internal representation of LogiQL programs and specify how they can be changed
- Meta-engine built on top of object store
  - Meta-predicates are OODB sets/vectors/maps implemented as treaps that support union/diff efficiently
  - Branching meta-engine to support ad-hoc and precompiled queries

Recently presented at IBM Almaden. Paper and patent in progress.

Yo Dawg, We Heard You Like Datalog Engines

Dan Olteanu
LogicBlox

Joint work with T.J. Green, T. Veldhuizen

NorCal DB Day, IBM Almaden
April 23, 2014
LOGICBLOX FOR LANGUAGE AND SYSTEMS RESEARCH

- LB source available for community to experiment with
  - New language features
    - E.g. probabilistic programming
  - Join Algorithms
    - E.g. MINESWEEPER – a new instance optimal algorithm
  - Incremental maintenance algorithms
    - E.g. DBTOASTER
  - Query costing
  - Query optimization
    - E.g. Semantic query optimization
  - Query parallelization
  - Alternative hardware configurations (e.g. GPU or FPGA)
    - E.g. REDFOX, Datalog Hardware Description Language (DHDL)
  - Transaction support

- Benchmarking framework
  - Add new benchmarks
  - Add new databases and systems
  - Add new data sets
APPLICATIONS OF LOGICBLOX FOR RESEARCH

- Number of researchers to accept academic license: 100
  - Last 10 came from: Osaka, Athens, Zurich, Oracle Labs, Stanford, Michigan State, Wuhan, Waterloo, Indian Institute of Science, IMDEA

- Program Analysis
- Graph based applications
  - Social
  - Fraud
- Simulation
- Provenance
- Network analysis
- Security
- Entity resolution
“The more ambitious plan may have more chances of success” - George Polya, How To Solve It, 1973
THANK YOU.