So What Is LogicBlox?

LogicBlox is a platform-as-a-service for rapid development of adaptive 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.
The first rule of the datalog club: you do not talk about datalog
Enterprise Decision Automation Nightmare

9+ Tech Stacks with 12+ different programming languages = HAIRBALL

Planning (future)
- Excel
  - Formulas: A3 = B2 – D17
- UI
- Planning App Server
  - Hyperion RPAS
  - ETL – Many vendors & programming models

BI (past)
- Browser
- BI App Server
  - Microstrategy
  - Business Objects
  - Cognos

OLTP (now)
- Browser
  - AJAX- JavaScript, HTML, XML
  - Flash – Actionscript
  - Silverlight - .Net languages
  - JavaFX – JavaFX and Java
- App Server
  - J2EE – WebLogic, WebSphere, JBoss, etc in Java
  - Microsoft - .Net languages
  - Netweaver – ABAP + Rules engine
- Transaction DB
  - Oracle
  - IBM DB2
  - Microsoft SQL Server
  - Sun MySQL
  - PostgreSQL

Data Mining & Stats
- SAS
- Matlab
- SPSS
- R

Optimization Modeling
- OPL
- AMPL
- GAMS

Optimization Solver
- CPLEX
- DASH
- COIN-OR
- 'homegrown'

ETL – Many vendors & programming models
- Many vendors & programming models
- ESB – Tibco, Seebeyond, etc.

Queries & Views – SQL with DDL, DML, etc.
- Stored Procedures – PL/SQL, TSQL, Java, C#, etc.
Oracle Fusion
SAP NetWeaver

Figure 1: SAP NetWeaver: Powering all SAP Solutions
IT Landscape – Supply Chain (2% of footprint)
We are here to invent the future

“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
Declarative Languages in the Enterprise
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:

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

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

target audience

Contents Proprietary & Confidential © 2011 LogicBlox Inc.
### Area of interest

<table>
<thead>
<tr>
<th>Programs</th>
<th>Programmers</th>
</tr>
</thead>
<tbody>
<tr>
<td>future</td>
<td>HERE</td>
</tr>
<tr>
<td>existing</td>
<td></td>
</tr>
<tr>
<td>existing</td>
<td></td>
</tr>
</tbody>
</table>
Why Datalog?

No apriori bias!
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
  - Skins – give people the syntax they like.

• Expressivity
  - Guaranteed expressivity with “controllable” power

• 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
“There is a flaw in the very foundations of Logic Programming: Prolog is nondeclarative”

-David S Warren  (see www.cs.sunysb.edu/~warren/xsbbbook/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 rewrite!!!
  - 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?
## Economics & Risk Transference Driving Transition to SaaS

<table>
<thead>
<tr>
<th>Costs</th>
<th>SaaS</th>
<th>Traditional</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Up Front</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Software Licenses</td>
<td>0</td>
<td>$$$</td>
</tr>
<tr>
<td>• 3rd Party Licenses</td>
<td>0</td>
<td>$$$</td>
</tr>
<tr>
<td>• Hardware</td>
<td>0</td>
<td>$$$</td>
</tr>
</tbody>
</table>

| Implementation | $.$.$.           | $$$$$       |

<table>
<thead>
<tr>
<th>Ongoing</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Subscription/Maintenance</td>
<td>$ (shared risk)</td>
<td>$</td>
</tr>
<tr>
<td>• 3rd Party Maintenance</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>• Operation</td>
<td>0</td>
<td>$$$$$</td>
</tr>
<tr>
<td>• Minimum Commitment</td>
<td>0</td>
<td>3-5 Years</td>
</tr>
</tbody>
</table>
Why Datalog **Now**? A Tale of Two Transitions

- A transition from book-keeping to decision automation & decision sciences
  - 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?
Why Datalog Now?

“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
An Aside: Joe Hellerstein – A Man of Peace

EXPERIENCE

No practical applications of recursive query theory ... have been found to date.

... I find it sad that the theory community is so disconnected from reality that they don’t even know why their ideas are irrelevant.

An Aside: Joe Hellerstein – A Man of Peace

MORE EXPERIENCE

In the last 7 years we have built

- distributed crawlers [Coo04,Loo04]
- network routing protocols [Loo05a,Loo06b]
- overlay networks (e.g. Chord) [Loo06a]
- a full-service embedded sensornet stack [Chu07]
- network caching/proxying [Chu09]
- relational query optimizers (System R, Cascades, Magic Sets) [Con08]
- distributed Bayesian inference (e.g. junction trees) [Atul09]
- distributed consensus and commit (Paxos, 2PC) [Alv09]
- a distributed file system (HDFS) [Alv10]
- map-reduce job scheduler [Alv10]
What Kind of Datalog?

- Classic datalog as a foundation
- Datalog with Aggregation
- Datalog with Integrity Constraints
- Datalog with State and Incremental Update
- Datalog with Defaults
- Datalog with (optional) Higher Order(s)
- Datalog with Existentially Quantified Head Variables
- Datalog with (constraint) Stratification
- Datalog with non-Determinism & Choice
- Datalog with language skins
- Datalog evaluated in different ways – not just semi-naive
I Am My Own Grandpa (1)

\[
\begin{align*}
T() & \leftarrow \text{person}(x). \\
F() & \leftarrow \neg \text{person}(x), \text{man}(x). \\
F() & \leftarrow \neg \text{person}(x), \text{woman}(x).
\end{align*}
\]

There are person entities (no-op)

All men and women are persons
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).

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

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

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

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) – 50’s version

\texttt{person}(x) \rightarrow .
\texttt{man}(x) \rightarrow \texttt{person}(x).
\texttt{woman}(x) \rightarrow \texttt{person}(x).

\texttt{person}(x) \rightarrow \texttt{man}(x); \texttt{woman}(x).
!(\texttt{man}(x), \texttt{woman}(x)).

\texttt{father}[x] = y \rightarrow \texttt{person}(x), \texttt{man}(y).
\texttt{mother}[x] = y \rightarrow \texttt{person}(x), \texttt{woman}(y).

\texttt{wife}[x] = y \rightarrow \texttt{man}(x), \texttt{woman}(y).
\texttt{husband}[x] = y \rightarrow \texttt{woman}(x), \texttt{man}(y).

\textbf{There are person entities (no-op)}
\textbf{All men and women are persons}
\textbf{All persons are either men or women}
\textbf{No person is both a man and a woman}
\textbf{Fathers are men and everyone has at most one}
\textbf{Mothers are women and everyone has at most one}
\textbf{Wives are women and every man has at most one}
\textbf{Husbands are men and every woman has at most one}
I Am My Own Grandpa (1) – HBO Version

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

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

father[x] = y -> person(x), man(y).
mother[x] = y -> person(x), woman(y).

wife(x, y) -> man(x), woman(y).
husband[x] = y -> woman(x), man(y).

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
A person has 0, 1, or more wives
Every person has at most one husband
person(x) ->.
man(x) -> person(x).
woman(x) -> person(x).

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

father[x] = y -> person(x), man(y).
mother[x] = y -> person(x), woman(y).

Fathers are men and everyone has at most one
Mothers are women and everyone has at most one

wife[x] = y -> person(x), person(y).
husband[x] = y -> person(x), person(y).

Every person has at most one wife
Every person has at most one husband
I Am My Own Grandpa (2)

\[
\text{bioparent}(x, y) \rightarrow \text{person}(x), \text{person}(y). \quad \text{A biological parent is either a mother or a father}
\]

\[
\text{bioparent}(x, y) \leftarrow \text{mother}[x] = y; \quad \text{father}[x] = y.
\]

\[
\text{bioanc}(x, y) \rightarrow \text{person}(x), \text{person}(y). \quad \text{A biological ancestor is a biological parent or the biological ancestor of a biological parent}
\]

\[
\text{bioanc}(x, y) \leftarrow \text{bioparent}(x, y);
\text{bioparent}(x, t), \text{bioanc}(t, y).
\]

\[
\text{selfbioanc}() \leftarrow \text{bioanc}(p, p). \quad \text{No person is his or her own biological ancestor}
\]

\[
\text{!selfbioanc}().
\]

\[
\text{wife}[x] = y \rightarrow \text{husband}[y] = x. \quad \text{A person’s wife has that person as a husband}
\]

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

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

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

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

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

A parent is a biological parent, or a father’s wife, or a mother’s husband.
A grandpa is a parent’s parent who is a man.
An own grandpa is the same man who is a grand child and a grand pa in the same entry of grandpa.
An own parent is the same person who appears twice in the same entry of parent.
Datalog with State (similar to Bertram Ludascher’s Statelog)

\[
\text{name}(p) = s \rightarrow \text{person}(p), \text{string}(s).
\]

\[
\text{person}(p) \rightarrow \text{name}(p) = \_.
\]

\[
\text{name}(p, s), \text{name}(p', s) \rightarrow p = p'.
\]

\[
+\text{person}(p), +\text{name}(p) = "fred" \leftarrow.
\]

\[
+\text{person}(p), +\text{name}(p) = "tom" \leftarrow.
\]

\[
+\text{bioparent}(x, y) \leftarrow \text{name}(x) = "tom", \text{name}(y) = "fred".
\]

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”. ‘p’ is existentially quantified.

Assert that there is a person called “tom”. ‘p’ is existentially quantified.

Assert that “fred” is the parent of “tom”
Datalog with State

\[
\text{name}[p] = s \rightarrow \text{person}(p), \text{string}(s).
\]

\[
\text{person}(p) \rightarrow \text{name}[p] = _.
\]

\[
\text{name}(p, s), \text{name}(p', s) \rightarrow p = p'.
\]

\[
+\text{person}(p), +\text{name}[p] = "fred" \leftarrow.
\]

\[
+\text{person}(p), +\text{name}[p] = "tom" \leftarrow.
\]

\[
+\text{bioparent}("tom", "fred") \leftarrow.
\]

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”. ‘p’ is existentially quantified.

Assert that there is a person called “tom”. ‘p’ is existentially quantified.

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

- Every predicate has “+” and “-” shadow predicates
- Every predicate has “now” and “next” shadow predicates
- These are all related via frame rules:
  \[ ?R@\text{next}(x) \leftarrow +?R(x). \]
  \[ ?R@\text{next}(x) \leftarrow ?R@\text{now}(x), !-?R(x). \]
- Run-time is optimized for these rules.
- Paying attention to “@future” ideas in Dedalus
- “+” & “-” shadow predicates are “events” if in the body and “actions” if in the head:
  \[ +\text{act}(x, y) \leftarrow +e_1(y), -e_2(x), \text{cond}(x, y). \]
An Observation in the Retail Industry

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
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>Access</td>
<td>Excel</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 (with good support for book-keeping)</th>
</tr>
</thead>
<tbody>
<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
Research Roadmap

- **Point free notation**
  - Enhances semantic stability of derivation rules
  - e.g. $\text{net} = \text{sales} - \text{returns}$. 

- **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**
Visual and Language Syntax Skins
Alternative Modeling 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
Cinema Model

* Cinema is multiplex iff Cinema has NTheaters > 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
Automatic verbalization of a selected model fragment (one fact type and two constraints):

The verbalization of the uniqueness constraint is clear but verbose. This is fine for use as an output language because it is auto-generated. More compact forms may be used as an input language (e.g. append m:n) because the verbose form is always available for validation with the client.
Structured Natural Language as Input

Entering the text below automatically generates the diagram shown.

ORM Fact Editor

*Person*.nr reviewed *Book*(ISBN)

ORM Fact Editor

*Person*.nr wrote *Book*(ISBN)
I Am My Own Grandpa in ORM
Person has Person_id.
Each Person_id is of at most one Person.
This association with Person_id provides the preferred identification scheme for Person.
Each Person has at most one Person_id.
Each Person has some Person_id.
Each Male is an instance of Person.
Each Male is at most one Person.
Each Person is at most one Male.
Each Male is some Person.
Each Female is an instance of Person.
Each Female is at most one Person.
Each Female is some Person.
For each Person, exactly one of the following holds:
that Person is some Male;
that Person is some Female.
Male is married to Female.
Each Male is married to at most one Female.
Each Female is married to at most one Male.
For each Male and Female, at most one of the following holds:
that Male is married to that Female;
that Person has that biological ancestor Person.
For each Female and Male, at most one of the following holds:
that Male is married to that Female;
that Person has that biological ancestor Person.
Person has Male parent.
Each Person has at most one Male parent.
Person has Female parent.
Each Person has at most one Female parent.
Person has parent Person.
It is possible that more than one Person has the same parent Person and that the same Person has more than one parent Person.
In each population of Person has parent Person, each Person, Parent combination occurs at most once.
This association with Person, Parent provides the preferred identification scheme for PersonHasParentPerson.
Person has Male grandparent.
It is possible that more than one Person has the same Male grandparent and that the same Person has more than one Male grandparent.
In each population of Person has Male grandparent, each Person, Male combination occurs at most once.
This association with Person, Male provides the preferred identification scheme for PersonHasMaleGrandparent.
Male is involved in PersonHasMaleGrandparent.
Each PersonHasMaleGrandparent involves at most one Male.
Each PersonHasMaleGrandparent involves some Male.
Person has biological parent Person.
It is possible that more than one Person has the same biological parent Person and that the same Person has more than one biological parent Person.
In each population of Person has biological parent Person, each Person, Parent combination occurs at most once.
This association with Person, Parent provides the preferred identification scheme for PersonHasBiologicalParentPerson.
Person has biological ancestor Person.
It is possible that more than one Person has the same biological ancestor Person and that the same Person has more than one biological ancestor Person.
In each population of Person has biological ancestor Person, each Person, Parent combination occurs at most once.
This association with Person, Parent provides the preferred identification scheme for PersonHasBiologicalAncestorPerson.
No Person may cycle back to itself via one or more traversals through Person, has biological ancestor Person.
Person is involved in PersonHasBiologicalParentPerson.
Each PersonHasBiologicalParentPerson involves at most one Person.
Each PersonHasBiologicalParentPerson involves some Person.
Hierarchical Syntax Skin
Hierarchical Syntax

- It is common in our Datalog programs to use several predicates to relate data to an entity:

  +person(p),
  +firstname[p]="John",
  +lastname[p]="Doe",
  +address(a),
  +home[p]=a,
  +street[a]="Main Street"
  +city[a]="Atlanta"

- The variable for person has to be named.
- Repetitive use of the name for linking the data.
- Can be difficult to read in complex rule with nesting.
Hierarchical Syntax

- Our solution: hierarchical syntax

```plaintext
+person(_) {
  +firstname[]="John",
  +lastname[]="Doe",
  +home[]= +address(a) {
    +street[]="Main Street",
    +city[]="Atlanta"
  }
}
```

- No need to name variable for person
- Uses integrity constraints to determine where to insert the missing references to the person
- Nesting is allowed to any depth
LogicBlox
Generics and Modularity
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 101

Levels defined by arrow length:

\[ p(x) \leftarrow q(x), r(x). \quad // \text{level 0} \]

\[ p(x) \leftarrow q(x), r(x). \quad // \text{level 1} \]

\[ p(x) \leftarrow q(x), r(x). \quad // \text{level 2 (not implemented yet)} \]

Same for Integrity constraints:

\[ \text{int}[x] = p \rightarrow \text{uint8}(x), \text{unary_predicate}(p). \]

You can refer from lower level to higher level:

\[ \text{int}[32] = \text{`int32} \leftarrow . \]
\[ \text{rank}[l] = r \rightarrow \text{language}(l), \text{int}[32](r). \]
\[ \text{rank}[	ext{“datalog”}] = 1 \leftarrow . \]
Datalog in the Small

ancestor(x,y)  <-  parent(x,y).
ancestor(x,y)  <-  parent(x,z), ancestor(z,y).

contains(x,y)  <-  form_comp(x,y).
contains(x,y)  <-  form_comp(x,z), contains(z,y).

reachable(x,y)  <-  neighbor(x,y).
reachable(x,y)  <-  neighbor(x,z), reachable(z,y).
Transitive Closure in MoReBlox

// Level 1
tc[?PRED]=?TP,
`{
   tc[?PRED](?x,?y) <- ?PRED(?x,?y).
   tc[?PRED](?x,?y) <- ?PRED(?x,?z), tc[?PRED](?z,?y).
} <-- suitable(?PRED).

// Level 0
+suitable(parent).
+suitable(neighbor).

nearby_anc(?x,?y) <- tc[parent](?x,?y),
                  tc[neighbor](?x,?y).
MoReBlox in a Nutshell

- Captures reusable functionality
  - Ability to write libraries!
- “Level 1” derivation rules
  - Computations over “level 0” programs as data
- Deriving/constructing new (level 0) programs
- Modular well-formedness guarantee
  - Level 1 rules are guaranteed to derive only well-formed level 0 programs
MoReBlox Use Cases

• Generics are the most frequently requested language enhancement by application teams
  – Product Lines

• Other uses:
  – Modularity
  – Security & Trust
  – “Cube” operator
  – Units of Measure
  – Parallelization
  – IDE support
Instead of inventing new language for modules, we define modules to be level 1 programs using hierarchical syntax:

```plaintext
Block(people) {
    export(person),
    export(firstName),
    export(lastName),
    clauses[] = `{
        person(?p) -> .
        id[?p] = ?n -> person(?p), int[64](?n).
        firstName[?p] = ?s, person(?p), string(?s).
        lastName[?p] = ?s, person(?p), string(?s).
    }
}
```

We will soon support linking and composition of modules as level 1 programs.
MoReBlox Security & Trust Management Use Case

- 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”
MoReBlox Research Roadmap

- MoReBlox is implemented using LB.
- Level 2+ support for units of measure and other use cases
- Variable argument support
  - Looking into datalog and regular expressions
- Code quoting in rule body
  - Syntactic
  - Semantic
- Dynamic code generation
  - Level 1 code can refer to level 0 @prev values
Syntax Skins Revisited
Free Syntax – Avoiding Syntax Holy Wars

Diagram

Rules

Tabular

Structured

NL

LB-VM
Research Roadmap

- **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**
  - Extend Output Verbalization to cover advanced derivation rules
  - Extend Input Verbalization with rich text support
  - Add Pluralization support
  - Add Statelog support
  - Add Conceptual Queries support
  - Add support for non-English languages

- **Tabular Notation**
  - Virtual Finite State Machines
  - Software through Pictures (StP)
  - Parnas Tables
  - Software Cost Reduction (SCR)
  - Requirements State Machine Language (RSML)

- **Rules**
  - Logic syntax
  - Functional syntax
  - Hierarchical syntax – XML & YAML
User Interface
Blox
• Browser based Portal and UI
• Modeled on W3C XForms standard
• GUI as view on the application model
  – Each view has a schema
  – Each view populates view-meta model
  – Each view has declarative styling via CSS
• Much simplified event-management
  • Clean separation of UI state vs. Application state
  • State management done mostly for application state
• Reactive/Incremental value propagation eliminates the need for code to move edits between application layers
• Unified semantics mean that validation and integrity checking are defined once for entire application
Define Form Schema

UI Meta Model

Form (Id) -- has -- Session (Id) -- has -- Claim (Nr)

Form Schema

ProvideLossInfo

hasSelected -- CauseOfLoss (Code)

shows

canContinue
Populate Meta Model Instance (hierarchical syntax is handy)

- !Form
  name: ProvideLossInfo
  label: Provide Loss Info
  save: continue
  contextPredicates: ['ui:claim', 'demo:claim:site:for']
  children:
  - !Label {value: 'Please provide the details of your loss.', importance: 2}
  - !Select1
    label: Cause of Loss?
    idPredicate: 'demo:causeOfLoss:code'
    valuePredicate: 'demo:causeOfLoss:code'
    filterPredicate: 'ui:shownCauses'
    selectedIdPredicate: 'ui:chosenCauseId'
    writeTo: 'demo:claim:hasReportedCauseOfLossMod'
  - !CurrencyField {label: Your estimate of loss amount for contents, writeTo: 'hasEstimateContentsLossAmount'}
  - !CurrencyField {label: Your estimate of loss amount for structure, writeTo: 'hasEstimateStructureLossAmount'}
  - !DateField {label: When did loss occur, writeTo: 'demo:claim:eventOccuredOnDate'}
  - !Select1
    label: Has this loss been previously reported?
    idPredicate: 'demo:previouslyReportedStatus:name'
    valuePredicate: 'demo:previouslyReportedStatus:name'
    filterPredicate: 'ui:reportedStatuses'
    selectedIdPredicate: 'ui:reportedStatusId'
    writeTo: 'demo:claim:hasPreviouslyReportedStatusObj'
  - !TextField {label: If Other, please specify, enabled: !Predicate 'ui:otherSelected', writeTo: 'demo:claim:hasPreviouslyReportedStatusOtherText'}
  - !Trigger {label: Continue, onActivate: continue}
  - !FormRedirect {formType: ConfirmClaimInfo, condition: !Predicate continue}
UIBlox – Browser Based Form Builder
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
Blurring the Line Between Statically and Dynamically Typed Languages
## Containment

### 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).`

- `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)]`
I Am My Own Grandpa – Dynamic Check

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 Check

**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**
- person
- man
- parent (person x person)
- grandpa (person x man)
I Am My Own Grandpa – Static Check

**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).

**ERROR!!!**
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 relevant derivation rules are contained in an IC, then we can remove the IC from the running program.
  This improves performance
• If containment check is not feasible, then IC remains in the program to catch errors at run-time
  Trade-off compile-time sophistication for run-time performance
• Bridging the divide between static and dynamic languages
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”
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
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

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

spacePerProd[p]=v -> PRODUCT(p), float[64](v).
profitPerProd[p]=v -> PRODUCT(p), float[64](v).

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

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

totalShelf[]=v -> float[64](v).
totalShelf[]=+= Stock[p] * spacePerProd[p] <-.
-> totalShelf[] <= maxShelf[ ].

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

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

lang:solve:variable(`Stock).
lang:solve:max(`TotalProfit).
// Givens
PRODUCT(x), PRODUCT:name(x:n) -> string(n).

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

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

totalShelf[] = v -> float[64](v).
totalShelf[] += Stock[p] * spacePerProd[p] <=.
-> totalShelf[] <= maxShelf[].

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

// Solve for
Stock[p] = v -> PRODUCT(p), int[64](v). A one line change
PRODUCT(p) -> Stock[p] =_.
lang:solve:variable(`Stock).
lang:solve:max(`TotalProfit).

Or 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

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

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

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

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

\[
\text{lang:solver:variable(`Make}).
\]

\[
\text{lang:solver:variable(`Trans}).
\]
Multiple Constraints

- Constraints can mix variables of different types

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

\[
\text{ORIG}(o), \text{PROD}(p), \text{totalShipOut}[o,p]=v1, \text{Make}[o,p]=v2 \rightarrow v1=v2.
\]

\[
\text{totalDelivery}[d,p] = v \rightarrow \text{DEST}(d), \text{PROD}(p), \text{float}[64](v).
\text{totalDelivery}[d,p] += \text{Trans}[,d,p].
\]

\[
\text{DEST}(d), \text{PROD}(p), \text{totalDelivery}[d,p]=v1, \text{demand}[d,p]= v2 \rightarrow v1=v2.
\]
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

\[
x[s] = v \rightarrow p(s), \text{int}[32](v), v \leq 5, v \geq 0.
p(s) \rightarrow x[s] = _.
\text{lang:solver:variable(`x`).}
\]
\[
y[s] = v \rightarrow p(s), \text{int}[32](v), v \leq 5, v \geq 0.
p(s) \rightarrow y[s] = _.
\text{lang:solver:variable(`y`).}
\]
\[
p(s), p:id(s:n) \rightarrow \text{string}(n).
\text{objective[]} = z \rightarrow \text{int}[32](z).
\text{objective[]} += x[s] + y[s].
\text{lang:solver:minimal(`objective`).}
p(s) \rightarrow \text{constr(s).}
\text{lang:isEntity[`constr]=false.}
\text{constr(s) -> p(s).}
\text{constr(s) <= x[s] + y[s] <= 2.}
\text{constr(s) <= -x[s] - y[s] >= 2.}
\]
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, CPLEX, SCIP, COIN-OR, 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 – working with Tim Hinrichs
  - Planning and Scheduling solvers
    - Using LB as provisioning system for LB
LB Machine Learning

• Joint work with N Vasiloglou 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

Dual-Tree Fast Gauss Transforms

QUIC-SVD: Fast SVD Using Cosine Trees

Ultrafast Monte Carlo for Kernel Estimators and Generalized Statistical Summations

Michael F. HIVens, Alexandre G. Gray, and Charles Lee Ishii, Jr.
College Of Computing
Georgia Institute Of Technology
Atlanta, GA 30332
mph, ishii@cc.gatech.edu

Abstract

Machine learning concerns many computational hurdles to be solved, such as noise reduction and smoothing. We present a new algorithm, similar to the method of moments, which can be applied to a broad class of high-dimensional data sets. We also provide an empirical analysis of the method's performance in comparison with other methods.

1 Introduction

Many machine learning methods face computational hurdles to be solved. In this paper we present a new method for the general class of random sampling and smoothing. We use a high-order polynomial basis function to represent the data and then compute the moments of the function. This method can be applied to a broad class of high-dimensional data sets.

Previous approaches are limited to data with a small number of components. Our method can be applied to data with many components. We provide an empirical analysis of the method's performance and compare it with other methods. Our method is simple to implement and can be applied to a wide range of problems.
Scalable 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
KDE Benchmark
Predictive Analytics – Fraud Example
constantCoeff[] = C,  
crimeCoeff[] = crimeCoeff,  
mCCoeff[] = mCCoeff,  
rICoeff[] = rICoeff,  
rCCoeff[] = rCCoeff,  
qPCoeff[] = qPCoeff,  
cLCoeff[causeOfLoss] = cLCoeff

<-  
regression << fraud ~ C + crimeCoeff * crimescore + mCCoeff * multipleClaim + rICoeff *  
    recentIncrease + rCCoeff * recentCvgInquiry + qPCoeff * questionableProducer + cLCoeff * causeOfLoss  
    where requiredFeatures = {causeOfLoss} >>

wasFraud[a] = fraud,  
hasCrimeScore[a] = crimescore,  
hasMultipleClaimsIndicator[a] = multipleClaim,  
hasQuestionableProducerIndicator[a] = questionableProducer,  
hasRecentCoverageIncreaseIndicator[a] = recentIncrease,  
hasRecentCvgInquiryIndicator[a] = recentCvgInquiry,  
hasReportedCauseOfLoss[a] = causeOfLoss.
hasFraudProbability[a] =
  constantCoeff[] +
  crimeCoeff[]*hasCrimeScore[a] +
  mCCoeff[]*hasMultipleClaimsIndicator[a] +
  rICoeff[]*BoolToNum[hasRecentCoverageIncreaseIndicator[a]] +
  rCCoeff[]*BoolToNum[hasRecentCvgInquiryIndicator[a]] +
  qPCoeff[]*BoolToNum[hasQuestionableProducerIndicator[a]] +
  clCoeff
<- 
  cLCoeff[hasReportedCauseOfLoss[a]] = clCoeff
;
hasReportedCauseOfLoss[a] = cl, not exists cLCoeff[cl], clCoeff=0.
fraudProbabilityAttribute[c] = v <-
val = hasFraudProbability[c],
(
  v = "likely_fraud", val > 0.75 ;
  v = "possible_fraud", val >= 0.30, val <= 0.75 ;
  v = "no_fraud", val < 0.30
).
# show red if likely fraud
.likely_fraud {
    background-color: red;
}

# show yellow if possible fraud
.possible_fraud {
    background-color: yellow;
}

# show green if no fraud
.no_fraud {
    background-color: green;
}
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
  - Markov Logic to unify classic AI with ML?
    • Integrity constraints with weights
  - Avoids the need to reformulate or “flatten” the problem
  - Takes advantage of known structure

• Develop Expert-System shell for LB
“Lightweight” Formal Methods
Inspired by Alloy

Alloy contributor Emina Torlak now at LB
Basic Idea

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

![Diagram showing testing and scope-complete cases](image-url)
Can constraints be satisfied (or not)?

- Solve for configuration with at most 4 people in which a man is his own grandfather
  
  -> count[\texttt{person}] \leq 4.
  
  -> count[\texttt{ownGrandpa}] \leq 1.

\texttt{lang:solve:variable(`ownGrandpa`).}

- Solve command instructs LB to search for counter example to ownParent within scope of at most 3 persons

  -> count[\texttt{person}] \leq 3.
  
  \texttt{lang:solve:variable(`ownParent`).}
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

Bigger Scope than Alloy-like methods

testing: a few cases of arbitrary size

scope-complete: all cases within a small bound
Combinatorial Test Generation

- We found 6 distinct bugs in our compiler 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
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”
Constraints Supported for Large Test Data

• 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
Research Roadmap

• 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
  - Stirewalt and McGill did a little bit better
Turning the Dial
For Quality
## Dimensions of Quality

<table>
<thead>
<tr>
<th>Expressiveness</th>
<th>Automation</th>
<th>Scope</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>None</td>
<td>Small</td>
<td>Random sparse</td>
</tr>
<tr>
<td>((&lt; monadic datalog)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>Partial</td>
<td>Medium</td>
<td>Uniform sparse</td>
</tr>
<tr>
<td>(monadic datalog)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>High</td>
<td>Full (fast)</td>
<td>Large</td>
<td>Dense</td>
</tr>
<tr>
<td>(dependency theory)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full</td>
<td>Infinite</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Full (slow)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Expressive</th>
<th>Automation</th>
<th>Scope</th>
<th>Density</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types I</td>
<td>Low</td>
<td>Full (fast)</td>
<td>Infinite</td>
<td>Dense</td>
</tr>
<tr>
<td>Types II</td>
<td>Medium</td>
<td>Full</td>
<td>Infinite</td>
<td>Dense</td>
</tr>
<tr>
<td>“Lightweight”</td>
<td>High</td>
<td>Full (slow)</td>
<td>Small</td>
<td>Dense</td>
</tr>
<tr>
<td>Combinatorial</td>
<td>High</td>
<td>Partial</td>
<td>Infinite</td>
<td>Uniform Sparse</td>
</tr>
<tr>
<td>Scalable</td>
<td>Low</td>
<td>Full</td>
<td>Large</td>
<td>Random Sparse</td>
</tr>
<tr>
<td>Unit Test</td>
<td>High</td>
<td>None</td>
<td>Infinite</td>
<td>Random Sparse</td>
</tr>
<tr>
<td>Auto Capture</td>
<td>High</td>
<td>Full</td>
<td>Infinite</td>
<td>Random Sparse</td>
</tr>
<tr>
<td>ITP</td>
<td>High</td>
<td>None</td>
<td>Infinite</td>
<td>Dense</td>
</tr>
</tbody>
</table>

**Biggest advantage of all:** Since the model expressed in datalog is executable, we eliminate risk of bugs introduced by “programming” the model!!!