Logic Programming

Datasets

Michael Genesereth
Computer Science Department
Stanford University
Datasets

**Dataset** - collection of facts about application area
- Facts in dataset are assumed to be true
- Facts not in dataset are assumed to be false

**Datasets as logic programs**
- simplest forms of logic programs
- used by selves as standalone databases
- used in combination with rules to form complex programs

**Datasets as basis for semantics of logical rules**
Conceptualization

**Objects** - e.g. people, companies, cities
- concrete (*person*) or abstract (*number, set, justice*)
- primitive (*computer chip*) or composite (*circuit*)
- real (*earth*) or fictitious (*Sherlock Holmes*)

**Relationships**
- properties of objects or relationships among objects
  - e.g. Joe *is a person*
  - e.g. Joe *is the parent of* Bill
  - e.g. Joe *likes* Bill *more than* Harry
Natural Language
  e.g. Joe is a person.
  e.g. Joe works for Apple.
  e.g. Joe is in office B-122.

But natural language is ambiguous and difficult to process

Formal Language
  person(joe)
  worksfor(joe,apple)
  office(joe,b122)
Constants are strings of lower case letters, digits, underscores, and periods or strings of arbitrary ascii characters within double quotes.

Examples:
    joe, bill, cs151, 3.14159
    person, worksfor, office
    the_house_that_jack_built,
    “Mind your p’s & q’s!”

Non-examples:
    Art, p&q, the-house-that-jack-built

A set of constants is called a vocabulary.
Types of Constants

**Symbols** represent objects.
- joe, bill, harry, a23, 3.14159
- the_house_that_jack_built
- “Mind your p’s & q’s!”

**Constructors** represent functions.
- pair, triple, set

**Predicates** represent relations.
- person, worksfor, office
The **arity** of a constructor or a predicate is the number of arguments that can be associated with the constructor or predicate in writing complex expressions in the language.

**Unary** predicate (1 argument):  
person

**Binary** predicate (2 arguments):  
parent

**Ternary** predicate (3 arguments):  
prefers

In defining vocabulary, we sometimes notate the arity of a constructor or predicate by annotating with a slash and the arity, e.g. `male/1`, `parent/2`, and `prefers/3`. 
In some logic programming languages (e.g. Prolog), types and arities determine syntactic legality; and they are enforced by interpreters and compilers.

In other languages (e.g. Epilog), types and arities suggest their intended use. However, they do not determine syntactic legality, and they are not enforced by interpreters and compilers.

In our examples, we use Epilog; but we specify types and arities where appropriate and we try to adhere to them.
A **ground term** is either a symbol or a compound name.

A **compound name** is an expression formed from an $n$-ary constructor and $n$ ground terms enclosed in parentheses and separated by commas.

Symbols: $a, b$
Constructor: `pair/2`
Ground term: `pair(a,a)`
Ground term: `pair(a,b)`
Ground term: `pair(pair(a,b),b)`
Ground term: `pair(pair(a,pair(b,a)),b)`

The adjective “ground” here means that the term does not contain any “variables” (which we discuss in later lessons).
The **Herbrand universe** for a vocabulary is the set of all ground terms that can be formed from the symbols and constructors in the vocabulary.

NB: For a finite vocabulary without constructors, the Herbrand universe is finite (i.e. just the symbols).

NB: For a finite vocabulary *with* constructors, the Herbrand universe is infinite (i.e. the symbols and all compound names that can be formed from those symbols).

\[
\text{pair}(a, b), \text{pair}(a, \text{pair}(b, c)), \text{pair}(a, \text{pair}(b, \text{pair}(c, d))), \ldots
\]
A **datum / factoid / fact** is an expression formed from an \( n \)-ary predicate and \( n \) ground terms enclosed in parentheses and separated by commas.

Symbols: \( a, b \)
Constructor: \( f/1 \)
Predicate: \( p/2, q/1 \)
Sample Datum: \( p(a, b) \)
Sample Datum: \( p(f(a), f(b)) \)
Sample Datum: \( q(a) \)
Sample Datum: \( q(f(b)) \)

The **Herbrand base** for a vocabulary is the set of all factoids that can be formed from the vocabulary.
A **dataset** is a set of factoids that can be formed from a vocabulary, i.e. a subset of the Herbrand base.

Symbols: \( a, b \)
Predicates: \( p/2, q/1 \)
Dataset: \( \{ p(a,b), p(b,a), q(a) \} \)
Dataset: \( \{ \} \)
Dataset: \( \{ p(a,a), p(a,b), p(b,a), p(b,b), q(a), q(b) \} \)

We use datasets to characterize states of the world. The facts in a dataset are assumed to be true and those that are not in the dataset are assumed to be false.
Exercise

**Vocabulary**
- Symbols: $a, b$
- Predicates: $p/2, q/1$

**Questions**
- How many elements in the Herbrand universe?
- How many elements in the Herbrand base?
- How many possible datasets?
Spelling carries no meaning in logic programming (except as informal documentation for programmers).

parent(art,bob)
parent(bob,cal)

p(a,b)
p(b,c)

coulish(widget,gadget)
coulish(gadget,framis)

The meaning of a constant in logic programming is determined solely by the sentences that mention it.
The order of arguments in an instance of a relation is determined by one’s understanding of the relation.

Example:

\[
\text{prefers(art,bea,bob)}
\]

For me, this sentence means that Art prefers Bea to Bob. Other interpretations are possible; the important thing is to be consistent - once you choose, stick with it.
Kinship
Parentage

art

bob

cal

bea

cam

cat

coe
Degenerate Relations

\[
\begin{align*}
\text{art} \\
\text{bob} & \quad \text{bea} \\
\text{cal} & \quad \text{cam} & \quad \text{cat} & \quad \text{coe}
\end{align*}
\]
parent(art,bob)
parent(art,bud)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
\[
\text{grandparent}(\text{art}, \text{cal}) \\
\text{grandparent}(\text{art}, \text{cam}) \\
\text{grandparent}(\text{art}, \text{cat}) \\
\text{grandparent}(\text{art}, \text{coe})
\]
sibling(bob, bea)
sibling(bea, bob)
sibling(cal, cam)
sibling(cam, cal)
sibling(cat, coe)
sibling(coe, cat)
ancestor(art,bob)
ancestor(art,bea)
ancestor(art,cal)
ancestor(art,cam)
ancestor(art,cat)
ancestor(art,coe)
ancestor(bob,cal)
ancestor(bob,cam)
ancestor(bea,cat)
ancestor(bea,coe)
Unary Relations:

male(art)
male(bob)
male(cal)
male(cam)
female(bea)
female(cat)
female(coe)

Ternary Relations:
prefers(art, bob, bea)
prefers(bob, cam, cal)
prefers(bea, cat, coe)
Some relations definable in terms of others
  e.g. we can define grandparent in terms of parent
  e.g. we can define sibling in terms of parent
  e.g. we can define ancestor in terms of parent
  e.g. we can define parent in terms of ancestor
See upcoming material on view definitions

Some combinations of arguments do not make sense
  e.g. parent(art,art)
  e.g. parent(art,bob) and parent(bob,art)
  e.g. male(art) and female(art)
See upcoming material on constraints
Blocks World
Blocks World
Symbols: a, b, c, d, e

Unary Predicates:
  clear - blocks with no blocks on top.
  table - blocks on the table.

Binary Predicates:
  on - pairs of blocks in which first is on the second.
  above - pairs in which first block is above the second.

Ternary Predicates:
  stack - triples of blocks arranged in a stack.
clear(a)
clear(d)

table(c)
table(e)

on(a,b)
on(b,c)
on(d,e)

above(a,b)
above(b,c)
above(a,c)
above(d,e)

stack(a,b,c)
University
<table>
<thead>
<tr>
<th>Students:</th>
<th>Departments:</th>
<th>Faculty:</th>
<th>Years:</th>
</tr>
</thead>
<tbody>
<tr>
<td>aaron</td>
<td>architecture</td>
<td>alan</td>
<td>freshman</td>
</tr>
<tr>
<td>belinda</td>
<td>computers</td>
<td>cathy</td>
<td>sophomore</td>
</tr>
<tr>
<td>calvin</td>
<td>english</td>
<td>donna</td>
<td>junior</td>
</tr>
<tr>
<td>george</td>
<td>physics</td>
<td>frank</td>
<td>senior</td>
</tr>
</tbody>
</table>

**Predicat:**

\[
\text{student}(\text{Student, Department, Advisor, Year})
\]

**Dataset:**

\[
\begin{align*}
\text{student}(\text{aaron, architecture, alan, freshman}) \\
\text{student}(\text{belinda, computers, cathy, sophomore}) \\
\text{student}(\text{calvin, english, donna, junior}) \\
\text{student}(\text{george, physics, frank, senior})
\end{align*}
\]
Suppose a student has not declared a major. What if a student does not have an advisor?

Leave out fields (syntactically illegal):

```plaintext
student(aaron,,,freshman)
```

Add suitable values to vocabulary (new symbol):

```plaintext
student(aaron,undeclared,orphan,freshman)
```

Database nulls (new linguistic feature):

```plaintext
student(aaron,null,null,freshman)
```
Suppose a student has *two* majors.

**Multiple Rows (storage, update inconsistencies):**

student(calvin,english,junior)
student(calvin,physics,junior)

**Multiple fields (storage, extensibility?):**

student(calvin,english,physics,junior)
student(george,physics,physics,senior)

**Use compound names:**

student(calvin,pair(english,physics),junior)
Represent wide relations as collections of binary relations.

**Wide Relation:**
\[
\text{student}(\text{Student}, \text{Department}, \text{Advisor}, \text{Year})
\]

**Binary Relations:**
\[
\begin{align*}
\text{student.major}(\text{Student}, \text{Department}) \\
\text{student.advisor}(\text{Student}, \text{Faculty}) \\
\text{student.year}(\text{Student}, \text{Year})
\end{align*}
\]

Always works when there is a field of the wide relation (called the **key**) that uniquely specifies the values of the other elements. If none exists, possible to create one.
student.major(aaron,architecture)
student.advisor(aaron,alan)
student.year(aaron,freshman)

student.year(belinda,sophomore)

student.major(calvin,english)
student.major(calvin,physics)
student.advisor(calvin,donna)
student.year(calvin, senior)

student.major(george,physics)
student.advisor(george,frank)
student.year(george, senior)
Terminology

Classes

student, department, faculty, year

Attributes (binary relations associated with a class):

student.major(Student, Department)
student.advisor(Student, Faculty)
student.year(Student, Year)

Properties of Attributes:

domain is class of objects in first position (arguments)
range is class of objects in second position (values)
unique if at most one value for each argument
total if at least one value for each argument
Sales
In 2015, Art sold Arborhouse to Bob for $1000000.
In 2016, Bob sold Pelicanpoint to Carl for $2000000.
In 2016, Carl sold Ravenswood to Dan in $2000000.
In 2017, Dan sold Ravenswood to Art for $3000000.

In 2015, Art sold Bob a widget for $10.
In 2016, Art sold Bob a gadget for $20.
In 2016, Art sold Bob another gadget for $20.
In 2017, Art sold Bob a framis for $30.
<table>
<thead>
<tr>
<th>People</th>
<th>Properties</th>
<th>Years</th>
<th>Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>arborhouse</td>
<td>2015</td>
<td>1000000</td>
</tr>
<tr>
<td>bob</td>
<td>pelicanpoint</td>
<td>2016</td>
<td>2000000</td>
</tr>
<tr>
<td>carl</td>
<td>ravenswood</td>
<td>2017</td>
<td>3000000</td>
</tr>
</tbody>
</table>

**Relation Constant:**

\[ \text{sale}(\text{Year, Seller, Property, Buyer, Amount}) \]

**Dataset:**

\[
\begin{align*}
\text{sale}(2015, \text{art, arborhouse, bob, 1000000}) \\
\text{sale}(2015, \text{art, arborhouse, bob, 1000000}) \\
\text{sale}(2015, \text{art, arborhouse, bob, 1000000}) \\
\text{sale}(2015, \text{art, arborhouse, bob, 1000000}) \\
\end{align*}
\]
### Sales Ledger

<table>
<thead>
<tr>
<th>People</th>
<th>Items</th>
<th>Years</th>
<th>Money</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>widget</td>
<td>2015</td>
<td>10</td>
</tr>
<tr>
<td>bob</td>
<td>gadget</td>
<td>2016</td>
<td>20</td>
</tr>
<tr>
<td>carl</td>
<td>framis</td>
<td>2017</td>
<td>30</td>
</tr>
<tr>
<td>dan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Relation Constant:**

\[
sale(\text{Year}, \text{Seller}, \text{Item}, \text{Buyer}, \text{Amount})
\]

**Dataset:**

- sale(2015, art, widget, bob, 10)
- sale(2016, art, gadget, bob, 20)
- sale(2016, art, gadget, bob, 20)  <--- Duplicate factoid!
- sale(2017, art, framis, bob, 30)
## Sales Ledger

<table>
<thead>
<tr>
<th>Sales:</th>
<th>People:</th>
<th>Items:</th>
<th>Years:</th>
<th>Money:</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>art</td>
<td>widget</td>
<td>2015</td>
<td>10</td>
</tr>
<tr>
<td>t2</td>
<td>bob</td>
<td>gadget</td>
<td>2016</td>
<td>20</td>
</tr>
<tr>
<td>t3</td>
<td>carl</td>
<td>framis</td>
<td>2017</td>
<td>30</td>
</tr>
<tr>
<td>t4</td>
<td>dan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Relation Constant:

\[
sale(Sale, Year, Seller, Item, Buyer, Amount)
\]

### Dataset:

\[
sale(t1, 2015, art, widget, bob, 10) \\
sale(t2, 2016, art, gadget, bob, 20) \\
sale(t3, 2016, art, gadget, bob, 20) \\
sale(t4, 2017, art, framis, bob, 30)
\]
Sierra
Sierra is browser-based IDE (interactive development environment) for Epilog.

- Saving and loading files
- Visualization of datasets
- Querying datasets
- Transformation tools for datasets
- Interpreter (for view definitions, constraints, transitions)
- Trace capability (useful for debugging rules)
- Analysis tools (error checking and optimizing rules)

http://epilog.stanford.edu/homepage/sierra.php
Load Sierra (http://epilog.stanford.edu/homepage/sierra.php)
Open a Data window
Enter and edit some data involving at least two relations
Open windows for the two relations
Edit the data in any window and notice changes in others

Query ground datum
Query with variables
Query with logical operators
Transform data and notice how everything automatically updates
Save p.hdf to /logicprogramming/examples/p.hdf
Delete p facts from lambda
Load p.hdf from /logicprogramming/examples/p.hdf

Quirks and idiosyncrasies
location of files and how to change (Safari - Preferences General)
renaming of files

Reload
Load college data from /logicprogramming/examples/college.hdf
Query a binary relation
Transform college to university ((x,y,z) —> faculty.y(x,y)
Save the result to /logicprogramming/examples/university.hdf
The goal of the **guided project** is to develop a program to allow users to play a board game - providing the users with a visualization of the state, the ability to make moves, and an automated opponent against which to play.

Examples:
- tictactoe.html
- nineboardtictactoe.html
- connectfour.html

Load from [http://logicprogramming.stanford.edu/examples/](http://logicprogramming.stanford.edu/examples/)
Assignment for this week:
  Select a board game (e.g. Chess, Pentago).
  Design a vocabulary to represent states of the game.
  Build a dataset representing the initial state of the game.
  Use Sierra to create, modify, save, load datasets for states.
  Use query to query one of your states.
  Use transform to simulate a move in one of your states.

Due next Thursday
  English description for your game
  State vocabulary (constants and arities)
  Sentences describing the initial state