Logic Programming

Datasets

Michael Genesereth
Computer Science Department
Stanford University
Datasets

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

Role #1 - Datasets as logic programs
simplest forms of logic programs
used by selves as standalone databases
used in combination with rules to form complex programs

Role #2 - Datasets as basis for semantics of logic programs
Conceptualization

Objects - e.g. people, companies, cities
  concrete \((person)\) or abstract \((number, set, justice)\)
  primitive \((computer chip)\) or composite \((car)\)
  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
Graphical Representation

- art
  - bob
    - cal
  - bea
    - cat
    - coe
<table>
<thead>
<tr>
<th>parent</th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
</tr>
<tr>
<td>art</td>
</tr>
<tr>
<td>bob</td>
</tr>
<tr>
<td>bob</td>
</tr>
<tr>
<td>bea</td>
</tr>
<tr>
<td>bea</td>
</tr>
</tbody>
</table>
Natural Language

Art is the parent of Bob.
Art is the parent of Bea.
Bob is the parent of Cal.
Bob is the parent of Cam.
Bea is the parent of Cat.
Bea is the parent of Coe.

Formal Language

parent(art, bob)
parent(art, bea)
parent(bob, cal)
parent(bob, cam)
parent(bea, cat)
parent(bea, coe)
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.occupant
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.
Symbols / object constants represent objects.
- joe, bill, harry, a23, 3.14159
- the_house_that_jack_built
- “Mind your p’s & q’s!”

Constructors / function constants represent functions.
- pair, triple, set

Predicates / relation constants represent relations.
- person, parent, prefers
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): \texttt{person(joe)}

**Binary** predicate (2 arguments): \texttt{parent(art,bob)}

**Ternary** predicate (3 arguments): \texttt{prefers(art,bob,bea)}

In defining vocabulary, we sometimes notate the arity of a constructor or predicate by annotating with a slash and the arity, e.g. \texttt{male/1}, \texttt{parent/2}, and \texttt{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, in this course, we specify types and arities where appropriate and we try to adhere to them.
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:  \(f/1, g/1\)
Ground terms:  \(f(a), f(a), g(a), g(b)\)
Ground term:  \(f(f(a)), f(g(a)), g(f(a)), g(g(a))\)

The adjective “ground” here means that the term does not contain any “variables” (which we discuss in later lessons).

A **ground term** is either a symbol or a compound name.
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).

\[ a, f(a), f(f(a)), f(f(f(a))), \ldots \]
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 any 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?
Vocabulary

Symbols: \( a, b \)
Constructors: \( f, g \)
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).

\[
\begin{align*}
\text{parent}(art, bob) \\
\text{parent}(bob, cal) \\
\text{p}(a,b) \\
\text{p}(b,c) \\
\text{coulish}(widget, gadget) \\
\text{coulish}(gadget, framis)
\end{align*}
\]

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   bea
 /     /
cal cam cat coe
```
Degenerate Relations

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

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. old(art) and young(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>

**Predicate:**

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

**Dataset:**

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

Leave out fields (syntactically illegal):

\[
\text{student}(\text{aaron},,,\text{freshman})
\]

Add suitable values to vocabulary (new symbol):

\[
\text{student}(\text{aaron},\text{undeclared},\text{orphan},\text{freshman})
\]

Database nulls (new linguistic feature):

\[
\text{student}(\text{aaron},\text{null},\text{null},\text{freshman})
\]
Suppose a student has *two* majors.

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

- `student(calvin,english,donna,junior)`
- `student(calvin,physics,donna,junior)`

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

- `student(calvin,english,physics,donna,junior)`
- `student(george,physics,physics,frank,senior)`

**Use compound names:**

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

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

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

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)
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
Subtlety

Missing information
- there is a value but we do not know it.
  e.g. Aaron has an advisor but we do not know who it is.

Non-existent value
- there is no value
  e.g. Aaron does not have an advisor.

For now, in talking about datasets, we assume full info. If a value is missing, there is no value.
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.
Real Estate Ledger

People:        Properties:        Years:        Money:
    art       arborhouse       2015       1000000
    bob       pelicanpoint     2016       2000000
    carl      ravenswood       2017       3000000
    dan

Relation Constant:  
    sale(Year,Seller,Property,Buyer,Amount)

Dataset:      
    sale(2015,art,arborhouse,bob,1000000)  
    sale(2016,art,pelicanpoint,bob,2000000)  
    sale(2016,carl,ravenswood,dan,2000000)  
    sale(2017,dan,arborhouse,art,3000000)
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.
### 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)  \textit{Duplicate factoid}!
- sale(2017, art, framis, bob, 30)
<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
- Transforming datasets
- Interpreter (for view definitions, action definitions)
- Trace capability (useful for debugging rules)
- Analysis tools (error checking and optimizing rules)

http://epilog.stanford.edu/homepage/sierra.php
\( p(c,d) \)
\( p(a,b) \)
\( p(b,c) \)
\[ p(c,d) \\
p(a,b) \\
p(b,c) \]
\begin{align*}
  p(a, b) \\
  p(b, c) \\
  p(c, d)
\end{align*}
\( p(a, b) \)
\( p(b, c) \)
\( p(c, d) \)
\( p(a, b) \)
\( p(b, c) \)
\( p(c, d) \)
\( p(e, f) \)
Syntax error.
<table>
<thead>
<tr>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>p(a, b)</td>
</tr>
<tr>
<td>p(b, c)</td>
</tr>
<tr>
<td>p(c, d)</td>
</tr>
</tbody>
</table>
\[ p(a, b) \\
p(b, c) \\
p(c, d) \]
New Dataset
Assignments
Readings

Required:
   Reading 2.1 - Datasets

Recommended:
   Reading 2.1 - Programs with Common Sense
   Reading 2.1 - Logic Programming
The goal of this exercise is for you to familiarize yourself with the updates mechanism of Sierra. As always, go to http://epilog.stanford.edu and click on the Sierra link.

In a separate window, open the documentation for Sierra. To access the documentation, go to http://epilog.stanford.edu, click on Documentation, and then click on the Sierra item on the resulting drop-down menu.

Read Sections 1-3 of the documentation and reproduce the examples in the Sierra window you opened earlier. Read section 9 and play around with saving and loading data and configurations.
Assignment - Game State
Consider a vocabulary that includes the following relations.

\textbf{movie.instance}(x) means that \(x\) is a movie.
\textbf{actor.instance}(x) means that \(x\) is an actor.
\textbf{director.instance}(x) means that \(x\) is a director.
\textbf{year.instance}(x) means that \(x\) is a year.
\textbf{title.instance}(x) means that \(x\) is a title.

\textbf{movie.star}(x,y) means that movie \(x\) stars actor \(y\).
\textbf{movie.director}(x,y) means that movie \(x\) was directed by \(y\).
\textbf{movie.year}(x,y) means that movie \(x\) was released in year \(y\).
\textbf{movie.title}(x,y) means that movie \(x\) has the title \(y\).

Choose symbols for a few movies, actors, directors, years, and titles, and encode the relevant data about these entities using this vocabulary.
Term Project

Criteria:
- Inherent interest of application (25%)
- Difficulty (25%)
- Appropriate use of Logic Programming (50%)

Topics:
- May be same as others or coordinate with others
- May be same as example in class or new and different

Deliverables:
- running code / Sierra configuration / etc.
- project presentation (last two class meetings)
- final report (end of quarter)

http://logicprogramming.stanford.edu/assignments/project.html