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*
<table>
<thead>
<tr>
<th>parent</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>art</td>
<td>bob</td>
</tr>
<tr>
<td>art</td>
<td>bea</td>
</tr>
<tr>
<td>bob</td>
<td>cal</td>
</tr>
<tr>
<td>bob</td>
<td>cam</td>
</tr>
<tr>
<td>bea</td>
<td>cat</td>
</tr>
<tr>
<td>bea</td>
<td>coe</td>
</tr>
</tbody>
</table>
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.

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
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 / 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}. 
Formality and Informality

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 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: \( f/1, g/1 \)
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).
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.
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).

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    bea  \\
 / \   / \   /  \
cal cam cat coe
```
Kinship Relations

Kinship Relations

Kinship Relations

Kinship Relations
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)
\texttt{grandparent(art, cal)}
\texttt{grandparent(art, cam)}
\texttt{grandparent(art, cat)}
\texttt{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(art, coe)
ancestor(bob, cal)
ancestor(bob, cam)
ancestor(bea, cat)
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. 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:**

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

**Dataset:**

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

Leave out fields (syntactically illegal):

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

Add suitable values to vocabulary (new symbol):

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

Database nulls (new linguistic feature):

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

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
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 none.
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.
<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>
<tr>
<td>dan</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Relation Constant:**

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

**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.
People: Items: Years: Money:
art widget 2015 10
bob gadget 2016 20
carl framis 2017 30
dan

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

Dataset:
sale(2015,art,widget,bob,10)
sale(2016,art,gadget,bob,20)
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, action definitions)
- Trace capability (useful for debugging rules)
- Analysis tools (error checking and optimizing rules)

http://epilog.stanford.edu/homepage/sierra.php
\[
\rho(c, d) \\
\rho(a, b) \\
\rho(b, c)
\]
\[ p(c,d) \]
\[ p(a,b) \]
\[ p(b,c) \]
\[
\begin{align*}
p(a, b) \\
p(b, c) \\
p(c, d)
\end{align*}
\]
\( p(a,b) \)
\( p(b,c) \)
\( p(c,d) \)
\( p(a, b) \)
\( p(b, c) \)
\( p(c, d) \)
\( p(e, f) \)
\[ p(a,b) \]
\[ p(b,c) \]
\[ p(c,d) \]
\[ p(a,b) \\
\[ p(b,c) \\
\[ p(c,d) \\

Lambda

Query

Pattern

Query

100 result(s)

Autorefresh
p(a,b)
p(b,c)
p(c,d)

Pattern: goal(X,Z)
Query: p(X,Y) & p(Y,Z)
p(a,b)
p(b,c)
p(c,d)

Pattern: goal(X,Z)
Query: p(X,Y) & p(Y,Z)

goal(a,c)
good(b,d)
p(a,b)
p(b,c)
p(c,d)
p(d,e)

Pattern: goal(X,Z)
Query: p(X,Y) & p(Y,Z)

100 result(s)

goal(a,c)
goal(b,d)
p(a,b)
p(b,c)
p(c,d)
p(d,e)

Pattern: \textit{goal}(x, z)

Query: \textit{p}(x, y) \land \textit{p}(y, z)

100 result(s)

\textit{goal}(a, c)
\textit{goal}(b, d)
\textit{goal}(c, e)
Lambda

\[ p(b,a) \]
\[ p(c,b) \]
\[ p(d,c) \]
\[ p(e,d) \]

Query

Pattern: \( \text{goal}(x,z) \)
Query: \( p(x,y) \land p(y,z) \)

Transformation

Condition: \( p(x,y) \)
Conclusion: \( \neg p(x,y) \land p(y,x) \)

Save Revert Sort

Show Next 100 results Autorefresh
Lambda

\p(b,a) 
\p(c,b) 
\p(d,c) 
\p(e,d) 

Query

Pattern: \text{goal}(X,Z) 
Query: \text{p}(X,Y) \land \text{p}(Y,Z) 

Transform

Condition: \text{p}(X,Y) 
Conclusion: \neg \text{p}(X,Y) \land \text{p}(Y,X) 

Expand on update: No 
Run on clock tick: No
Lambda

\begin{align*}
p(a, b) \\
p(b, c) \\
p(c, d) \\
p(d, e) \\
\end{align*}

Query

Pattern: \texttt{goal(X, Z)}

Query: \texttt{p(X, Y) \& p(Y, Z)}

Transform

Condition: \texttt{p(X, Y)}

Conclusion: \texttt{\neg p(X, Y) \& p(Y, X)}

Expand: \checkmark

Expand on update: \checkmark

Run on clock tick: \blank

\begin{align*}
\neg p(a, b) \\
p(b, a) \\
\neg p(b, c) \\
p(c, b) \\
\neg p(c, d) \\
p(d, c) \\
\neg p(d, e) \\
p(e, d) \\
\end{align*}
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