Logic Programming
View Definitions

Michael Genesereth
Computer Science Department
Stanford University

Kinship Dataset

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

More Comprehensive Dataset

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)

grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

Views as Named Queries

```
goal(X,Z) :- parent(X,Y) & parent(Y,Z)
```

$$\downarrow$$

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

Deduction

```
parent(art,bob)
parent(art,bea)
parent(bob,cal)
parent(bob,cam)
parent(bea,cat)
parent(bea,coe)
```

Base relation

+

```
grandparent(X,Z) :- parent(X,Y) & parent(Y,Z)
```

=

```
grandparent(art,cal)
grandparent(art,cam)
grandparent(art,cat)
grandparent(art,coe)
```

View relation

Benefits

Economy - fewer facts need to be stored

Less chance of getting out of sync

View definitions work for any number of objects

* More expressive (e.g. recursive definitions) *

Syntax

Constants and Variables

A **constant** is a string of lower case letters, digits, underscores, and periods *or* a string of ascii characters within double quotes.

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

Same as before

A **variable** is either a lone underscore or a string of letters, digits, underscores beginning with an upper case letter.

```
X    Y23    Somebody    _
```

Terms

Symbols

```
art
bob
```

Variables

```
X
Y23
```

Same as before

Compound Terms

```
pair(art,bob)
pair(X,Y23)
pair(pair(art,bob),pair(X,Y23))
```

Atoms, Negations, and Literals

Atoms

```
p(a,b)
p(a,X)
p(Y,c)
```

Negations

```
~p(a,b)
```

Literals (atoms or negations of atoms)

```
p(a,Y)
~p(a,Y)
```

Same as before

An atom is a *positive literal*.

A negations is a *negative literal*.

Ground Rules

$$\underbrace{\texttt{r(a,b)}}_{\textit{head}}\ \texttt{:-}\ \underbrace{\overbrace{\texttt{p(a,b)}}^{\textit{subgoal}}\ \texttt{\&}\ \overbrace{\texttt{\textasciitilde q(b)}}^{\textit{subgoal}}}_{\textit{body}}$$

Rules

$$\underbrace{\texttt{r(X,Z)}}_{\textit{head}} \;\texttt{:-}\; \underbrace{\overbrace{\texttt{p(X,Y)}}^{\textit{subgoal}} \;\texttt{\&}\; \overbrace{\texttt{\~{}q(Y)}}^{\textit{subgoal}}}_{\textit{body}}$$

(1) *Named queries (not just* `goal`*).*

(2) *Body may mention view relations as well as base relations.*

Rulesets

A **ruleset** is a set of rules.

Example:

```
r(X,Y) :- p(X,Y) & ~q(Y)

p(X,Y) :- f(X,Y)
p(X,Y) :- m(X,Y)
```

A ruleset by itself is sometimes called an **open logic program**.

Features of Rulesets

Multiple Relations

```
f(X,Y) :- p(X,Y) & q(X)
m(X,Y) :- p(X,Y) & ~q(X)
```

View Relations in Subgoals

```
g(X,Z) :- f(X,Y) & p(Y,Z)
```

Recursive Relations

```
a(X,Z) :- p(X,Z)
a(X,Z) :- p(X,Y) & a(Y,Z)
```

Closed Logic Program

A **closed logic program** is a dataset together with a ruleset.

Ruleset

```
s(X,Y) :- p(X,Y)
s(X,Y) :- q(X,Y)
t(X,Y) :- s(X,Y) & ~r(Y)
```

Dataset

```
p(a,b)
p(a,c)
q(b,c)
r(a)
r(c)
```

Well-Formed Closed Logic Programs

Three requirements:

Compatibility

Global safety

Stratification

Requirement #1 - Compatibility

A ruleset is **compatible** with a dataset if and only if

(1) all words shared between the dataset and the ruleset are of the same type (symbol, constructor, predicate)

(2) all constructors and predicates have the same arity

(3) no predicate in the *dataset* appears in the *head* of a rule.

The **vocabulary** of a basic logic program is the union of the vocabularies of the dataset and the logic program.

Compatibility

Compatible Ruleset and Dataset:

```
g(X,Z) :- p(X,Y) & p(Y,Z)

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

Non-compatible Ruleset and Dataset:

```
g(X,Z) :- p(X,Y) & p(Y,Z)

p(a,b)
g(b,c)
```

Requirement #2 - Safety

A rule is *locally safe* if and only if every variable in the head appears in some positive subgoal in the body and every variable in a negative subgoal appears in a prior positive subgoal.

Safe Rule:

```
r(X,Z) :- p(X,Y) & p(Y,Z) & ~q(X,Z)
```

Unsafe Rules:

```
r(X,Z) :- p(X,Y) & p(Y,X)
r(X,Y) :- p(X,Y) & ~q(Y,Z)
r(X,Z) :- p(X,Y) & ~q(Y,Z)
```

Global Safety

In Logic Programming, it is common to divide relations into three types - *input relations* (i.e. base relations), *intermediate relations*, and *output relations* (query relations).

Locally Unsafe Rule:

```
r(X,Z) :- p(X,Y) & ~q(Y,Z)
```

Globally Safe Program:

```
goal(X) :- r(X,b) & r(Y,c) & r(X,Y)
r(X,Z) :- p(X,Y) & ~q(Y,Z)
```

Requirement #3 - Stratification

We say that a set of view definitions is *stratified* if and only if its rules can be partitioned into *strata* in such a way that (1) every stratum contains at least one rule, (2) the rules defining relations that appear in positive goals of a rule appear in the same stratum as that rule *or* in some lower stratum, (3) the rules defining relations that appear in negative subgoals of a rule occur in some *lower* stratum (not the same stratum).

What???

Understanding Stratification

Two Stages:

Semipositive Programs

Stratified Programs

Semantics of Semipositive Programs*

**The Easy Case*

Semipositive Programs

A **semipositive** program is one in which negations apply only to base relations, i.e. there are no subgoals with negated views.

Example:

```
r(X) :- p(X,Y) & q(Y)
q(Y) :- m(Y,Z) & ~n(Z)
```

Non-Example:

```
r(X) :- p(X,Y) & ~q(Y)
q(Y) :- m(Y,Z) & ~n(Z)
```

Instances

An **instance of a rule** is a rule in which all variables have been consistently replaced by ground terms.

Rule

```
r(X,Y) :- p(X,Y) & ~q(Y)
```

Herbrand Universe

```
{a,b}
```

Instances

```
r(a,a) :- p(a,a) & ~q(a)
r(a,b) :- p(a,b) & ~q(b)
r(b,a) :- p(b,a) & ~q(a)
r(b,b) :- p(b,b) & ~q(b)
```

Rule Application

The **result of applying a rule** r **to a dataset** Δ (written $v(r,\Delta)$) is the set of all ψ such that

1) ψ is the head of an arbitrary instance of r,

(2) every positive subgoal in the instance is a member of Δ,

(3) no negative subgoal in the instance is a member of Δ.

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Result

```
p(a)
p(b)
p(c)
p(d)
```

Closure

The **closure** of a semipositive program Ω on a dataset Δ (written C(Ω,Δ)) is the result of *repeatedly* applying the rules in Ω to Δ as follows.

$\Delta_0 = \Delta$
$\Delta_{n+1} = \cup\, v(r, \Delta_0 \cup \ldots \cup \Delta_n)$ for all r in Ω

$C(\Omega,\Delta) = \cup \Delta_i$.

In other words, C(Ω,Δ) is the fixpoint of v.

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
p(a)
p(b)
p(c)
p(d)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)     q(a,b)
edge(b,c)     q(b,c)
edge(c,d)     q(c,d)
edge(d,c)     q(d,c)
p(a)          q(b,a)
p(b)          q(c,b)
p(c)
p(d)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)     q(a,b)
edge(b,c)     q(b,c)
edge(c,d)     q(c,d)
edge(d,c)     q(d,c)
p(a)          q(b,a)
p(b)          q(c,b)
p(c)          r(c,d)
p(d)          r(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)     q(a,b)     s(a,b)
edge(b,c)     q(b,c)     s(b,c)
edge(c,d)     q(c,d)     s(c,d)
edge(d,c)     q(d,c)     s(d,c)
p(a)          q(b,a)
p(b)          q(c,b)
p(c)          r(c,d)
p(d)          r(d,c)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)     q(a,b)     s(a,b)
edge(b,c)     q(b,c)     s(b,c)
edge(c,d)     q(c,d)     s(c,d)
edge(d,c)     q(d,c)     s(d,c)
p(a)          q(b,a)     s(a,c)
p(b)          q(c,b)     s(b,d)
p(c)          r(c,d)     s(c,c)
p(d)          r(d,c)     s(d,d)
```

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Closure

```
edge(a,b)     q(a,b)     s(a,b)
edge(b,c)     q(b,c)     s(b,c)
edge(c,d)     q(c,d)     s(c,d)
edge(d,c)     q(d,c)     s(d,c)
p(a)          q(b,a)     s(a,c)
p(b)          q(c,b)     s(b,d)
p(c)          r(c,d)     s(c,c)
p(d)          r(d,c)     s(d,d)
                         s(a,d)
```

Semantics of Stratified Programs*

**Buckle your seat belts.*

Potential Problem

Dataset

```
p(a,b)
p(b,a)
```

Rule

```
r(X) :- p(X,Y) & ~r(Y)
```

Instances

```
r(a) :- p(a,a) & ~r(a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)
r(b) :- p(b,b) & ~r(b)
```

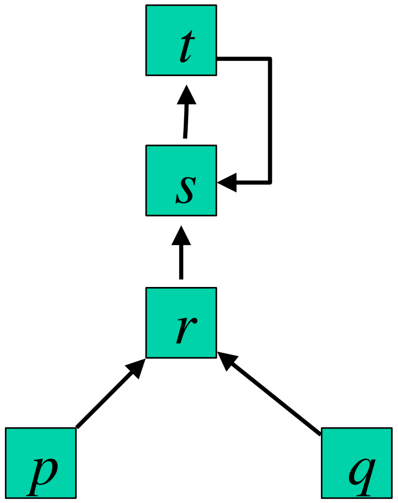
Possible Closures (depending on order of application)

```
p(a,b)      p(a,b)
p(b,a)      p(b,a)
r(a)        r(b)
```

Dependency Graph

The *dependency graph* for a set of rules is a directed graph in which (1) the nodes are the relations mentioned in the head and bodies of the rules and (2) there is an arc from a node *p* to a node *q* whenever *p* occurs with the body of a rule in which *q* is in the head.

```
r(X,Y) :- p(X,Y) & q(X,Y)
s(X,Y) :- r(X,Y)
s(X,Z) :- r(X,Y) & t(Y,Z)
t(X,Z) :- s(X,Y) & s(Y,X)
```



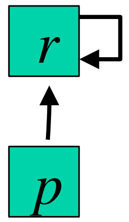
A set of rules is *recursive* if it contains a cycle. Otherwise, it is *non-recursive*.

Stratified Negation

The negation in a set of rules is said to be *stratified* if and only if there is no recursive cycle in the dependency graph involving a negation.

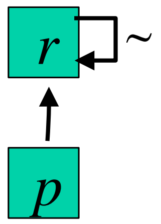
Stratified Negation:

```
        r(X,Z) :- p(X,Y)
        r(X,Z) :- r(X,Y) & r(Y,Z)
```



Negation that is not stratified:

```
        r(X,Z) :- p(X,Y)
        r(X,Z) :- p(X,Y) & ~r(Y,Z)
```



All negations ***must*** *be stratified.*

Alternative Definition

We say that a set of view definitions is *stratified* if and only if its rules can be partitioned into *strata* in such a way that

(1) every stratum contains at least one rule

(2) the rules defining relations that appear in positive goals of a rule appear in the same stratum as that rule *or* in some lower stratum

(3) the rules defining relations that appear in negative subgoals of a rule occur in some *lower* stratum (not the same stratum)

Examples

Example

```
r(X,Y) :- q(X,Y)
r(X,Z) :- q(X,Y) & r(Y,Z)
s(X,Y) :- p(X) & p(Y) & ~r(X,Y)
```

Stratum	Rules
2	`s(X,Y) :- p(X) & p(Y) & ~r(X,Y)`

1	`r(X,Y) :- q(X,Y)`
	`r(X,Z) :- q(X,Y) & r(Y,Z)`

Non-example

```
r(X,Y) :- p(X) & p(Y) & q(X,Y)
s(X,Y) :- r(X,Y) & ~s(Y,X)
```

Extension

The *extension* of a logic program Ω with strata $\Omega_1, \ldots, \Omega_k$ on a dataset Δ (written $E(\Omega,\Delta)$) is the result of repeatedly applying the rules in $\Omega_1, \ldots, \Omega_k$ to Δ as follows.

$\Gamma_0 = \Delta$
$\Gamma_{n+1} = \Gamma_n \cup C(\Omega_{n+1},\Gamma_n)$

The extension of a program with k strata is Γ_k.

Example

Ruleset

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
t(X,Y) :- p(X) & p(Y) & ~s(X,Y)
```

Dataset

```
edge(a,b)
edge(b,c)
edge(c,d)
edge(d,c)
```

Example

Stratum 2 Rules

```
t(X,Y) :- p(X) & p(Y) & ~s(X,Y)
```

Stratum 1 Rules

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Example

Stratum 1 Rules

```
p(X) :- edge(X,Y)
q(X,Y) :- edge(X,Y)
q(X,Y) :- edge(Y,X)
r(X,Y) :- edge(X,Y) & edge(Y,X)
s(X,Y) :- edge(X,Y)
s(X,Z) :- edge(X,Y) & s(Y,Z)
```

Closure

```
edge(a,b)     q(a,b)     s(a,b)
edge(b,c)     q(b,c)     s(b,c)
edge(c,d)     q(c,d)     s(c,d)
edge(d,c)     q(d,c)     s(d,c)
p(a)          q(b,a)     s(a,c)
p(b)          q(c,b)     s(b,d)
p(c)          r(c,d)     s(c,c)
p(d)          r(d,c)     s(d,d)
                         s(a,d)
```

Example

Stratum 2 Rule

```
t(X,Y) :- p(X) & p(Y) & ~s(X,Y)
```

Closure

```
edge(a,b)     q(a,b)     s(a,b)     t(a,a)
edge(b,c)     q(b,c)     s(b,c)     t(b,a)
edge(c,d)     q(c,d)     s(c,d)     t(b,b)
edge(d,c)     q(d,c)     s(d,c)     t(c,a)
p(a)          q(b,a)     s(a,c)     t(c,b)
p(b)          q(c,b)     s(b,d)     t(d,a)
p(c)          r(c,d)     s(c,c)     t(d,b)
p(d)          r(d,c)     s(d,d)
                         s(a,d)
```

Properties

The extension of every stratified logic program is unique. Multiple stratifications are sometimes possible, but all produce the same result.

Extension of stratified logic program *without constructors* is finite. *Without* recursion it can be computed in polynomial time. *With* recursion, can be computed in exponential time.

Extension of a stratified logic program *with constructors* may be infinite. The extension is still well-defined but not completely computable.

Alternative Semantics*

Warning: Complicated*

*** But beautiful!*

Definitions

The **Herbrand universe** for a closed logic program is the set of all *ground terms* that can be formed from the symbols and constructors in the vocabulary.

The **Herbrand base** for a closed logic program is the set of all *ground atoms* that can be formed from the vocabulary.

An **interpretation** of a closed logic program is an arbitrary *subset* of the program's Herbrand base.

A **model** of a closed logic program is an interpretation that *satisfies* the program (definition to follow).

Instances

An **instance of a rule** is a rule in which all variables have been consistently replaced by ground terms.

Rule

```
r(X,Y) :- p(X,Y) & ~q(Y)
```

Herbrand Universe

```
{a,b}
```

Instances

```
r(a,a) :- p(a,a) & ~q(a)
r(a,b) :- p(a,b) & ~q(b)
r(b,a) :- p(b,a) & ~q(a)
r(b,b) :- p(b,b) & ~q(b)
```

Semantics

Suppose Γ is an interpretation.

Γ **satisfies** a *ground atom* ϕ if and only if ϕ is in Γ.

Γ **satisfies** a *ground negation* $\sim\phi$ if and only if ϕ is *not* in Γ.

Γ **satisfies** a *ground rule* ϕ :- ϕ_1 & ... & ϕ_n if and only if Γ satisfies ϕ whenever it satisfies $\phi_1, \ldots, \phi_n$.

An interpretation Γ **satisfies** a *closed logic program* (i.e. a dataset + a ruleset) if and only if (1) all of the elements of the constituent dataset are included in Γ and (2) Γ satisfies all *instances* of all of the rules in the constituent ruleset.

Example

Dataset

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

Ruleset

```
r(X,Y) :- p(X,Y) & ~p(Y,X)
```

Model

```
p(a,b)
p(b,c)
p(c,d)
p(d,c)
r(a,b)
r(b,c)
```

Non-Examples

Dataset

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

Ruleset

```
r(X,Y) :- p(X,Y) & ~p(Y,X)
```

***Not* Models**

```
p(a,b)        p(a,b)        p(a,b)
p(b,c)        p(b,c)        p(b,c)
p(c,d)        p(c,d)        p(c,d)
p(d,c)        p(d,c)        p(d,c)
              r(a,b)        r(b,c)
```

Multiple Models

Dataset

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

Ruleset

```
r(X,Y) :- p(X,Y) & ~p(Y,X)
```

Models

```
p(a,b)      p(a,b)      p(a,b)
p(b,c)      p(b,c)      p(b,c)
p(c,d)      p(c,d)      p(c,d)
p(d,c)      p(d,c)      p(d,c)
r(a,b)      r(a,b)      r(a,b)
r(b,c)      r(b,c)      r(b,c)
            r(c,d)      r(d,c)
```

```
p(a,b)
p(b,c)
p(c,d)
p(d,c)
r(a,b)
r(b,c)
r(c,d)
r(d,c)
r(a,b)
r(b,c)
r(c,d)
r(d,c)
r(a,b)
r(b,c)
r(c,d)
r(d,c)
r(a,b)
r(b,c)
r(c,d)
r(d,c)
```

So What?

We want our definitions to be *if and only if*. We want to include among our conclusions only those facts that *must* be true given our data and rules.

All factoids in dataset *must* be true.
All factoids required by rules *must* be true.
Arbitrary factoids should be excluded.

What we want is logical entailment. A factoid is **logically entailed** by a closed logic program if and only if it is true in *every* model of the program, i.e. *the set of conclusions is exactly the intersection of all models of the program*.

Minimal Models

A model D of a logic program P is **minimal** if and only if no proper subset of D is a model of P.

Models

`p(a,b)`	`p(a,b)`	`p(a,b)`
`p(b,c)`	`p(b,c)`	`p(b,c)`
`p(c,d)`	`p(c,d)`	`p(c,d)`
`p(d,c)`	`p(d,c)`	`p(d,c)`
`r(a,b)`	`r(a,b)`	`r(a,b)`
`r(b,c)`	`r(b,c)`	`r(b,c)`
	`r(c,d)`	`r(d,c)`

If a program has just one minimal model, then every factoid true in that model is trivially true in *every* minimal model.

Uniqueness

A logic program that does not contain any negations has a *unique minimal model*.

If a program is *semipositive*, it has a unique minimal model.

Hooray!

Multiple Minimal Models

A logic program that does not contain any negations has a *unique minimal model*.

If a program is *semipositive*, it has a *unique minimal model*.

A logic program with negations that is *not* semipositive can have *more than one minimal model* (in addition to multiple non-minimal models).

Uh-oh!

Non Semipositive Programs

Dataset

```
p(a,b)
p(b,a)
```

Rule

```
r(X) :- p(X,Y) & ~r(Y)
```

Interpretations:

```
p(a,b)      p(a,b)      p(a,b)      p(a,b)
p(b,a)      p(b,a)      p(b,a)      p(b,a)
            r(a)        r(b)        r(a)
                                    r(b)
```

Multiple *Minimal* Models

Dataset

```
p(a,b)
p(b,a)
```

Rule

```
r(X) :- p(X,Y) & ~r(Y)
```

Instances

```
r(a) :- p(a,a) & ~r(a)
r(a) :- p(a,b) & ~r(b)
r(b) :- p(b,a) & ~r(a)
r(b) :- p(b,b) & ~r(b)
```

Interpretations:

```
p(a,b)     p(a,b)     p(a,b)     p(a,b)
p(b,a)     p(b,a)     p(b,a)     p(b,a)
           r(a)       r(b)       r(a)
                                 r(b)
```

So What?

Dataset

```
p(a,b)
p(b,a)
```

Rule

```
r(X) :- p(X,Y) & ~r(Y)
```

Minimal Models:

```
p(a,b)      p(a,b)      p(a,b)      p(a,b)
p(b,a)      p(b,a)      p(b,a)      p(b,a)
            r(a)        r(b)        r(a)
                                    r(b)
```

Is r(a) true or not? What about r(b)?

NB: The intersection of all models is not a model!

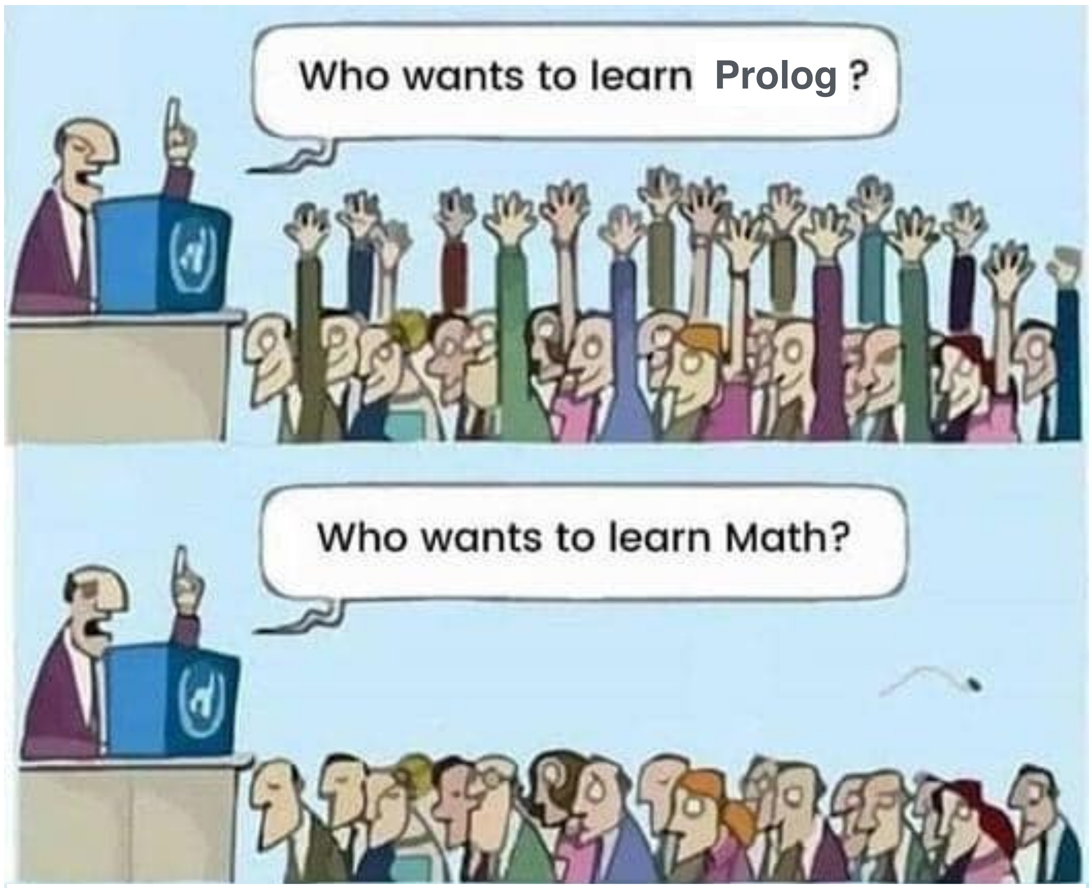
Stratified Negation

A logic program that does not contain any negations has a *unique minimal model*.

If a program is *semipositive*, it has a *unique minimal model*.

A logic program with negations that is *not* semipositive can have *more than one minimal model* (in addition to multiple non-minimal models).

If a logic program with negation is *stratified*, then it has a *unique minimal model*.


Who wants to learn Prolog ?
Who wants to learn Math?