CS 4110
Programming Languages & Logics

Lecture 34
Logic Programming

Logic Programming: Origins

- Proposed in 1960s–1970s as logic-based approach to computation
- Usually based on first-order logic and resolution
- Examples:
  - Classic Languages: Prolog and Datalog
  - Modern Languages: Erlang, Verse
- Applications:
  - Database query languages
  - Program analysis
  - Various "niche" uses (e.g., Ericsson)

Facts, Rules, Queries

A logic program consists of:

- *Facts*: base truths
- *Rules*: implications
- *Queries*: goals to prove

Basic building blocks are Horn clauses of the form:

$$p_1 \wedge \cdots \wedge p_n \rightarrow h$$

In Prolog and Datalog, Horn clauses are usually written "backwards:"

$$h :- p_1, \ldots, p_n.$$

Prolog

- Expressive but operationally sensitive
- Uses depth-first, left-to-right resolution
- Allows functions, lists, arithmetic, control ops (`cut`)
- Rule order and subgoal order affect termination and correctness

Datalog

A disciplined subset of Prolog:

- No function symbols
- Rule safety required
- Negation must be stratified

Semantic guarantees:

- All programs terminate
- Finite set of derivable facts
- Evaluation does not depend on order

Datalog Syntax

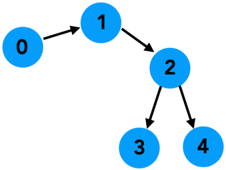
$$
\begin{array}{rcll}
r & ::= & h :- b. & \textit{rule} \\
 & | & h. & \\
h & ::= & p & \textit{head} \\
b & ::= & p_1, p_2, \ldots p_n & \textit{body} \\
p & ::= & p(t_1, \ldots t_n) & \textit{predicate} \\
t & ::= & x & \textit{term} \\
 & | & n &
\end{array}
$$

Definitions and Conventions

- An atomic predicate $p(n_1, \ldots, n_k)$ with no variables is called a ground atom.
- Each variable in the head of a rule must also appear in the body.

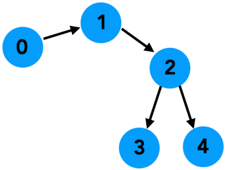
Example: Graph Reachability

Suppose we have a graph:



Example: Graph Reachability

Suppose we have a graph:



We can encode its edge relation as a collection of facts:

```
edge(0,1).
edge(1,2).
edge(2,3).
edge(2,4).
```

Example: Graph Reachability

We can express reachability in the graph using the following rules:

```
reachable(x,y) :- edge(x,y).
reachable(x,y) :- edge(x,z), reachable(z,y).
```

Example: Graph Reachability

We can express reachability in the graph using the following rules:

```
reachable(x,y) :- edge(x,y).
reachable(x,y) :- edge(x,z), reachable(z,y).
```

We can detect cycles using the following rule:

```
cycle(x) :- reachable(x,x).
```

Example: Graph Reachability

We can express reachability in the graph using the following rules:

```
reachable(x,y) :- edge(x,y).
reachable(x,y) :- edge(x,z), reachable(z,y).
```

We can detect cycles using the following rule:

```
cycle(x) :- reachable(x,x).
```

We can compute strongly-connected components using the following rule:

```
scc(x,y) :- reachable(x,y), reachable(y,x).
```

Demo

Let’s fire up Soufflé...

Herbrand Interpretation

The constants appearing in a program P form the Herbrand universe:

$$HU(P) = Const(P).$$

Herbrand Interpretation

The constants appearing in a program P form the Herbrand universe:

$$HU(P) = Const(P).$$

The ground atoms over predicates and constants form the *Herbrand base*:

$$HB(P) = \{\, p(n_1, \ldots, n_k) \mid n_i \in HU(P) \,\}.$$

Herbrand Interpretation

The constants appearing in a program P form the Herbrand universe:

$$HU(P) = Const(P).$$

The ground atoms over predicates and constants form the *Herbrand base*:

$$HB(P) = \{\, p(n_1, \ldots, n_k) \mid n_i \in HU(P) \,\}.$$

A Herbrand interpretation is any $I \subseteq HB(P)$.

Ground Instances and Consequence

Given a rule, suppose we substitute the variables with each constant in $HU(P)$.

Ground Instances and Consequence

Given a rule, suppose we substitute the variables with each constant in $HU(P)$.

Let $Ground(P)$ denote the set of all such rules, called ground instances.

Ground Instances and Consequence

Given a rule, suppose we substitute the variables with each constant in $HU(P)$.

Let $Ground(P)$ denote the set of all such rules, called ground instances.

Example

```
reachable(0,0) :- edge(0,0).
reachable(1,1) :- edge(1,1).
reachable(0,1) :- edge(0,1).
reachable(1,0) :- edge(1,0).
      ⋮
```

Ground Instances and Consequence

Given a rule, suppose we substitute the variables with each constant in $HU(P)$.

Let $Ground(P)$ denote the set of all such rules, called ground instances.

Example

```
reachable(0,0) :- edge(0,0).
reachable(1,1) :- edge(1,1).
reachable(0,1) :- edge(0,1).
reachable(1,0) :- edge(1,0).
      ⋮
```

We can define the immediate consequence operator as follows:

$$T_P(I) = \{\, h \mid (h : -p_1, \ldots, p_k) \in Ground(P) \text{ and } p_1, \ldots, p_k \in I \,\}.$$

Properties

The T_P operator is monotone:

$$I \subseteq J \Rightarrow T_p(I) \subseteq T_P(J)$$

Hence, because the Herbrand base is finite, $T_P(I)$ reaches a fixed point in finitely many iterations.

Formal Semantics

We can compute the meaning of a program by iterating the T_P operator to a fixed point, starting from the empty set:

$$\begin{aligned} I_0 &\triangleq \emptyset \\ I_{i+1} &\triangleq T_P(I_i) \end{aligned}$$

Formal Semantics

We can compute the meaning of a program by iterating the T_P operator to a fixed point, starting from the empty set:

$$\begin{aligned} I_0 &\triangleq \emptyset \\ I_{i+1} &\triangleq T_P(I_i) \end{aligned}$$

Definition (Meaning of Datalog Program)

$M(P) \triangleq \mathit{fix}(T_P)$

Negation

Pure Datalog is monotone: adding facts never invalidates conclusions.

Negation breaks monotonicity so we must be careful!

A well-behaved fragment is *stratified negation*, where negation is only used with previously-defined relations.

Stratification

A Datalog program is stratified if its predicates can be partitioned into layers (strata) $S_0, S_1, \ldots, S_n$ such that:

- If P depends *positively* on Q, then stratum$(P) \geq$ stratum(Q).
- If P depends *negatively* on Q, then stratum$(P) >$ stratum(Q).

Intuition:

- Lower strata computed first; higher strata may use negation on them.
- No predicate may depend *negatively* on itself, even indirectly.

Stratification Examples

Not Stratified:

```
p(x) :- q(x), not r(x).
r(x) :- s(x), not p(x).
```

We have a cycle, $p \rightarrow r \rightarrow p$, so no valid stratification exists.

Stratified:

```
b(x) :- a(x), c(x).
n(x) :- a(x), not b(x).
```

Here `a`, `b`, `c` are in S_0 while `n` is in S_1.

Other Extensions

Other extensions of Datalog include:

- Aggregation (min, count, etc.)
- Arithmetic (plus, times, etc.)
- Datatypes (lists, trees, etc.)